Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València
Enginyeria Tècnica en Informàtica de Gestió

Memòria Projecte Final de Carrera

Implementació de Framework PHP per desenvolupar aplicacions web ràpidament

Autor: Andrés Ignacio Martínez Soto
Director: Pedro J. Valderas Aranda.

València, juliol de 2011

Índex

Part I

Introducció

Capítol 1

Introducció

1.1 Introducció

Aquest capítol introdueix al lector a les bases, objectius i motivacions que han conduit al estudi, disseny i construcció d'aquest sistema que presenta un marc de treball escrit en PHP per facilitar el desenvolupament d'aplicacions web.

1.2 Descripció del projecte

Aquest projecte implementa un **framework** o **marc de treball** per al desenvolupament ràpid d'aplicacions i serveis web escrit en llenguatge de programació *PHP*, **orientat a objectes i components** i preparat per a fer fàcil el treball de desenvolupament web 2.0. Utilitzant com a base el llenguatge de programació PHP i sense més requeriments que un servidor web i una base de dades compatible s'arriba a poder utilitzar amb la majoria de hostings comercials existents sense emprar biblioteques addicionals ni instal·lacions complexes.

Implementa per una part amb la que es poden construir aplicacions web dinàmiques, serveis web d'arquitectura *REST*, serveis web *SOAP* i execució de tasques automàtiques.

Per la part d'aplicacions web, es poden construir aplicacions web de tot tipus a les que se lis pot afegir interactivitat utilitzant *AJAX* mitjançant JavaScript o construir un *Frontend* tipus RIA (*Rich Internet Applications*) amb *Adobe Flex* o altra tecnologia RIA , demanant-li dades al servidor, processant-les en aquest i retornant les dades del processament al client.

En quant a la part de serveis web. Aquest projecte permet generar i consumir serveis web realitzats amb *arquitectura REST* i implementats amb l'estàndard *SOAP*. En els serveis web REST es proporciona un component de senzill us que ens permetrà consumir aquestos serveis web com serveis web de Twitter, Facebook i d'altres plataformes web 2.0.

Tant els serveis web REST com els SOAP es poden construir per fer-se servir en aplicacions empresarials (altres webs, aplicacions d'escriptori, intranets, ...) i moure l'arquitectura i dades d'un entorn personal al *núvol* ≪cloud-computing≫.

Per terminar, es poden implementar tasques automàtiques que realitzen operacions útils per al negoci de l'aplicació web.
Aquestes tasques automàtiques han de ser cridades per el programador del sistema (o execució de tasques del nostre hosting web) per a poder ser executades. Una tasca automàtica pot ser, el generar informes diaris, setmanals i mensuals de qualsevol dada d'interès, generar estadístiques, enviar i rebre missatges, enviar newsletters a tots els subscriptors, fer còpies de seguretat, ...

Totes aquestes coses - i més - es poden implementar amb aquest projecte, que pretén ser un marc de treball escrit en PHP i amb unes mínimes dependències externes; per possibilitat el desenvolupament i educar en el desenvolupament d'aplicacions web 2.0 d'una manera senzilla, ràpida i amena.

1.3 Motivació

La principal motivació que ha dut a aquest projecte és el adquirir experiència i competències en el terreny del desenvolupament de programari, és a dir, millorar el nivell de programació existent i aplicar els principis d'enginyeria del programari explicats al llarg de tota la carrera. També el adquirir coneixements sobre desenvolupament i disseny web, encaminats cap al que - avui en dia - és un dels sectors més importants de la informàtica; la web.

Segonament, els marcs de treball o framework comercials existents, són complicats d'aprendre, amb instal·lacions costoses, i resultats desesperants per el que comença a introduir-se en aquest món, d'això el desenvolupament d'aquest marc de treball en el que és senzill i ràpid desenvolupar una aplicació web.

Finalment, la darrera motivació, és el crear un marc de treball senzill d'aprendre per a poder ser aprés i utilitzat per tot tipus de programadors - estudiants d'informàtica especialment - i constituir una base sobre la que aprendre aquest apassionant mon que és l'enginyeria web.

1.4 Objectius

Aquest projecte té de principal objectiu la construcció d'un marc de treball complet per a desenvolupament d'aplicacions web d'una manera ràpida i senzilla, és a dir, de facilitat la construcció d'aplicacions web empresarials grans i escalables.

S'esmenten a continuació els objectius d'aquest projecte:

- Desenvolupament d'un marc de treball complet.
- Disseny de components eficients.
- Desenvolupament d'aplicacions Web 2.0.
- Proporcionar serveis web per els estàndards SOAP i REST.
- Proporcionar una manera senzilla d'estendre el marc de treball.
- Servir de marc de treball per a principiants i estudiants, és a dir, en educació.

El haver escrit aquest marc de treball des de zero - basant-se en projectes similars - ha donat una visió completa del panorama actual en el desenvolupament de programari, complementant les nocions ja vistes en les assignatures específiques d'intensificació d'Enginyeria del Programari, Desenvolupament de Programari per Components, Desenvolupament Avançat de Programari i en la resta d'assignatures de la carrera com Enginyeria del Programari, Disseny de Bases de Dades, Bases de Dades i per descomptat Programació. Aquesta experiència ha aportat - sense dubte - una espurna per a l'alliberació d'aquest Projecte Final de Carrera com a projecte de programari lliure, a la creació d'una comunitat al voltant d'aquest projecte i a la programació d'aplicacions web amb aquest projecte; sempre amb l'objectiu de millorar el projecte i innovar en el camp de la web i serveis web, segons vagen apareixent noves tecnologies i formes més eficients i ràpides de programar aplicacions web.

1.5 Principis utilitzats en aquest desenvolupament

El desenvolupament d'aquest marc de treball inclou els següents principis bàsics:

- Codi font obert - open source.
- Codi simple, principi KISS [1].
- Disseny multicapa en 3 capes i vàries subcapes.
- Disseny orientat a objectes i fortament estructurat en components.
- Aplicació de patrons de disseny en la construcció de components.
- Separació de lògica de negoci, fonts de dades i vistes.
- Extensibilitat (a través de plugins) i reutilització de codi.
- Facilitat de programació i corba d'aprenentatge reduïda.
- Interoperatibilitat amb serveis web.
- Us d'estàndars en el camp de la web (PHP, XML, JSON, AJAX).
- Algorismes senzills, ràpids i eficients.
- Us de metodologies àgils i estàndars de codificació.
- Servidors de codi o de control de versions per a treballs en grup.

1.6 Principals components i sistemes del framework

Aquest projecte inclou els següents components principals:

- Router o enrutador de peticions, facilita la creació d'URLs personalitzades i amigables als cercadors.
- Sistema MVC per desenvolupament dels diferents mòduls d'una aplicació web.

[1] Keep It Simple Stupid!

- Sistema Active Record per a obtenció i gestió de dades en bases de dades.
- Sistema de plugins, extensibilitat del marc de treball.
- Sistema de Caché que permet un major rendiment en llocs web de molt de trànsit.
- Sistema d'AJAX amb XML i JSON.
- Servidor de serveis web SOAP.
- Servidor de serveis web REST.
- Internacionalització i18n.
- Sistemes de plugins i widgets per a una major reutilització de codi.
- Validació de dades personalitzada.

1.7 Història i versions d'aquest projecte

Aquest projecte va començar després d'haver-se programat una aplicació web titulada ≪Gestió d'Activitats per a la Setmana de Benvinguda≫, una petita aplicació web que va gestionar el apuntats a les activitats de la mencionada setmana de benvinguda de la Universitat Jaume I de Castelló de la Plana des de 2006 a 2009. Mentre presentava el projecte web a les pràctiques de l'assignatura APW ≪Desenvolupament d'Aplicacions en Entorns Web≫, el professor de pràctiques em va suggerir si podria fer-lo més genèric i em va donar la clau per alguns components bàsics i consells per dissenyar la versió de l'aplicació web presentada al 2009. Les pràctiques d'aquesta assignatura van obrir-me la ment al món del desenvolupament de programari per components i van donar orígen a la primera versió d'aquest projecte.

Des de llavors, les versions han estat succeint-se poc a poc, amb la remodelació de components, reescriptures, refactoritzacions de components i nous components per resoldre problemes bàsics. Aquest projecte ha estat utilitzat en un parell d'ocasions per muntar aplicacions web, una d'elles muntà una aplicació web per enviament i gestió de missatges SMS per part d'una empresa, una aplicació web multiusuari que permet enviar missatges curts a telèfons mòbils amb agenda i gestió i classificació d'aquestos missatges; va ser programada en una estància en pràctiques en l'empresa *Open Xarxes Coop.V* des de setembre a desembre de l'any 2010.

1.7.1 Cronologia de versions

V 0.01 : Novembre de 2009, primeres proves del marc de treball.

V 0.05 : Gener de 2010, reescriptura total del projecte.

V 0.10 : Març de 2010, implementació del sistema d'enrutat, configuració i despatxament de peticions web.

V 0.11 : Juny de 2010, implementació d'Active Record.

V 0.5 : Setembre de 2010, refactorització del sistema d'enrutat i diferents components.

V 0.7 : Octubre de 2010, clients de REST i sistema de plugins.

V 0.8 : Desembre de 2010, refactorització del sistema de configuració i d'Active Record.

V 0.9 : Desembre de 2010, llançament d'aplicació per enviament d'SMS.

V 1.0 : Febrer de 2011, reescriptura total de vàries parts del projecte.

V 1.3 : Març de 2011, reescriptura d'Active Record, Front Controller i base de dades.

V 1.45 : Abril de 2011, reescriptura de l'API de plugins i configuració.

V 1.49 : Maig de 2011, implementació del sistema de ≪layouts≫, reescriptura de sistemes de SOAP i REST, refactorització de Front Controller, implementació de sistema ≪Filter≫, implementació de validacions.

V 1.50 : Juny de 2011, reescriptura del component de logging, autenticació, connexió amb bases de dades. Descripció i documentació de tot el projecte.

Capítol 2

Antecedents

2.1 Introducció

En aquesta secció es presenten una breu història i evolució del desenvolupament web al llarg del darrers anys.

2.2 Estat de l'art: evolució

2.2.1 CGI *Common Gateway Interface*

El *Common Gateway Interface* sigles de **CGI** és un estàndard (veure *RFC 3875*) que defineix com el servidor web pot delegar la generació de pàgines a un programa o fitxer executable.
Aquestos programes coneguts com *scripts CGI* poden ser escrits en qualsevol llenguatge de programació.
Hi va haver una època en que a través del CGI era la única manera d'executar contingut dinàmic. S'utilitzava molt per arreplegar valors en funcionaris de contacte.
El llenguatge de programació *Perl* estava de moda i era el que va començar a fer la revolució dels llenguatges dinàmics, malgrat això, les grans empreses o proveïdors generaven els seus CGI en llenguatges tant poc dinàmics i flexibles com C o C++.
Actualment totes les limitacions de CGI han estat superades i la majoria de plataformes funcionen de la mateixa manera que ho feia en l'època CGI: delegant l'execució en programes externs.

2.2.2 Llenguatges dinàmics

Després del boom de l'època del CGI, les grans companyies de programari començaren a desenvolupar els seus llenguatges de programació per programar CGIs. Apareixen les primeres versions de Microsoft ASP que permet fer aplicacions web amb l'ajuda de Visual Studio. Més tard apareixen de la ma de Sun Microsystems les primeres versions de Java per a la web (Java Server Pages) i J2EE ... Per part del programari lliure, el llenguatge de programació preferit per programar webs és Perl que poc a poc va deixant pas a un derivat seu que avui es coneix com a PHP. També sorgeixen alternatives com Python o la més moderna, Ruby on Rails.

Avui en dia el mercat està repartit entre Ruby on Rails, Microsoft ASP.NET, PHP i totes les implementacions corporatives de Java per a aplicacions web.

2.2.3 AJAX i la Web 2.0

AJAX són les sigles de Asynchronous Javascript And Xml, (JavaScript asíncron i XML), un conjunt de tecnologies que permeten actualitzar continguts web sense haver de tornar a carregar la pàgina, obrint la port a aplicacions web interactives.

Ajax és asíncron en tant que les dades addicionals són demanades i carregades en un segon pla, sense interferir en la presentació i el comportament de la pàgina.
Habitualment les funcions d'Ajax es criden des del llenguatge JavaScript. S'encarega d'enviar una petició HTTP al servidor i espera una resposta, que processarà i actualitzarà dades en la pàgina.
Ajax és multiplataforma i es pot usar en diversos sistemes operatius, arquitectures de computador i navegadors web ja que es basa en estàndards oberts com JavaScript i DOM. Hi ha implementacions open source de frameworks i llibreries, en especial els frameworks més utilitzats: (*jQuery*,*ExtJS*,*Yahoo User Interface*,*Mootools* i *Google Web Toolkit*).
Un framework de Java Script ajuda molt a desenvolupar una pàgina dinàmica i interactiva, ja que permet fer més fàcil la programació de les peticions AJAX, d'animacions sobre elements i de navegació i transformació sobre els nodes DOM de la pàgina.

Des del moment que va eixir la possibilitat d'utilitzar AJAX i aquest es va popularitzat, no han deixat d'aparèixer noves i curioses formes de programar la web i de fer que cada vegada s'assemble més a una aplicació d'escriptori que no pas a un formulari HTML amb programació dinàmica per darrere. L'evolució d'AJAX natural és cap a la utilització de formats d'intercanvi de dades més lleugers que XML (donat el temps de processament que tarda aquest format tant en el servidor com en el client) cap al format JSON (Java Script Object Notation) que al ser natiu de Java Script redueix els temps de manipulació i processament de la resposta en el propi navegador.

Capítol 3

Tecnologies

3.1 Introducció

Aquest projecte intenta emprar les darreres tecnologies i estàndars apareguts en el món del programari orientat a objectes, l'enginyeria web, les arquitectures orientades a serveis i els darrers avanços en web per a dispositius mòbils. El marc de treball proveïx una gran compatibilitat amb totes les plataformes en les que es pot fer servir, proporcionant una implementació allunyada de les característiques dels sistemes operatius sobre els quals funciona, i els servidors web en els quals pot funcionar donant servei. Així mateixa, al estar adaptat als estàndars de nova generació - per exemple l'XHTML 1.1 o l'HTML 5, SOAP, JSON ...- , i que són de molta difusió; es garantitza una interoperabilitat entre plataformes.

3.2 Tecnologies de costat del servidor

3.2.1 PHP: PHP HyperText Processor

PHP és un llenguatge de programació interpretat i orientat a objectes que s'utilitza per a generar pàginas web dinàmiques. S'executa en el servidor mitjançant un intèrpret (o màquina virtual) que s'encarrera de compilar en temps d'execució el codi font en un llenguatge intermig de la màquina virtual, i després en codi màquina del sistema operatiu i màquina sobre el que s'executa. PHP es distribueix sota la llicència PHP, que la Free Software Foundation qualifica com a programari lliure.

El llenguatge de programació PHP Com s'ha mencionat abans, PHP és un llenguatge de programació orientat a objectes i dèbilment tipat amb una sintaxi mescla entre C++,Perl i Java; dels quals agafa la majoria de conceptes. PHP permet - des de la versión 4.0 - declarar classes, objectes i interfícies, el qual el fa molt flexible, adés, la facilitat amb la que es poden gestionar els arrays o vectors; la senzillessa de connexió a bases de dades i l'absència de punters fa que PHP siga un llenguatge de programació molt emprat i senzill d'apendre, productiu a la vegada i fàcil de depurar.

L'Entorn de producció/execució de PHP

L'entorn de producció de PHP consisteix generalment en un intèrpret que junt a unes quantes biblioteques i extensions proporciona una API per a execució de guions d'or-

dres en mode CGI. Aquesta API de CGI és l'encarregada de comunicar-se amb el servidor web tant d'entrada com d'eixida de dades. Així doncs, el servidor web s'encarrega de cridar a PHP amb certes dades d'entrada, PHP produeix unes dades d'eixida en resposta a les entrades, i les torna al servidor, que a la seva vegada les tornarà al client que va demanar-les. Aquest entorn de producció és fortament depenent del sistema operatiu sobre el que s'execute tot, malgrat que PHP és multiplataforma - és a dir, funciona i s'executa sobre diversos sistemes operatius i arquitectures de computadors - i hi ha un gran nombre de servidors web sobre els quals funciona PHP que són també multiplataforma. PHP, per tal d'abstraure tota la implementació per als diferents sistemes operatius, encapsula sota el mateix nom mètodes i funcions que tornen els mateixos valors però implementats de diferent forma en cada plataforma.

Per a començar a produir pàgines PHP només es necessita un entorn mínim que és un intèrpret de PHP (o distribució), un servidor web - Apache2 és el recomanat i més emprat per la seva difusió - i una base de dades - MySQL és la més estàndar i multiplataforma - . Amb tot això i junt amb un editor de text o un entorn de programació integrat 1 es poden generar aplicacions web d'una manera ràpida i senzilla, sense disposar d'una gran i complexa instal·lació que sobrecarregue la màquina servidora.

Una manera ràpida de començar amb PHP és descarregar-se el paquet XAMPP des de l'adrecça `http://www.apachefriends.org/en/xampp.html` que conté tots els elements necessaris per a muntar un servidor web de pàgines dinàmiques en pocs minuts i sota qualsevol sistema operatiu.

Errors comuns, fallades de seguretat

Malgrat tot això, és comú que en PHP el desenvolupador novell - tal vegada per falta de pràctica - s'oblide de la seguretat que és un dels temes pendents en totes les plataformes de programació d'aplicacions web, i que degut a la globalització d'internet, i a la facilitat d'explotar aplicacions web de tot tipus a distància i d'amagat; i per això hi haja moltes "fallades"a causa del que s'anomena security flaws o injeccions de codi PHP, HTML, SQL o JavaScript. La seguretat en tota aplicació és un tema molt important, per això el marc de treball descrit en aquest projecte fa èmfasi en aquestes qüestions, forçant i ajudant al desenvolupador a que evite els errors de seguretat, i que l'aplicació web que amb aquest marc de treball programe siga fiable i segura en un alt percentatge. També és responsabilitat del programador el comprovar els tipus de les variable i objectes de PHP, ja que al ser un llenguatge dèbilment tipat, les variables i el seu tipus s'assignen en temps d'execució i sense comprovació del tipus; podent provocar errors inesperats.

3.2.2 SOAP: Simple Object Access Protocol

SOAP (Simple Object Access Protocol o Protocol Simple d'Accés a Objectes) és un protocol de comunicació dissenyat per intercanviar missatges en format XML entre aplicacions connectats mitjançant una xarxa, i sobre el protocol HTTP. Està pensat per facilitar la comunicació entre aplicacions, independentment de la plataforma on s'executen i del llenguatge de programació en què estiguen implementades. És senzill i fàcilment extensible.

El protocol SOAP es basa en RPC ≪Remote procedure Call≫ que és una tecnologia que permet a un programa fer que una subrutina o procediment s'execute en un altre espai d'adreces (habitualment en un altra màquina en una xarxa compartida) sense que

el programador haja de programar explícitament els detalls d'aquesta interacció remota. És a dir, el programador escriuria una cridada a un procediment que està en altra màquina sense haver de preocupar-se per els detalls de la invocació ni el transport de les dades des de i fins a la màquina on s'ofereix el procediment o subrutina a executar. A més a més, SOAP també conté:

- Informació adicional inclosa en el document XML que descriu el contingut, i les operacions que es poden executar i que ofereix el servidor remotament.
- Definició dels paràmetres d'entrada i eixida de cada operació que s'ofereix remotament.
- Definició de tipus de dades complexos i/o estructurats que permeten enviar i rebre dades complexes/objectes entre client i servidor.

WSDL: Web Services Description Language

WSDL són les inicials de ≪Web Services Description Language≫, un format XML que s'utilitza per descriure Serveis web. WSDL descriu la interfície pública dels serveis web. Està basat en XML i descriu la forma de comunicació, és a dir, els requeriments del protocol i els formats dels missatges necessaris per interactuar amb els serveis que es troben en la llista del seu catàleg. Les operacions i missatges que admet es descriuen de forma abstracta i s'enllacen després al protocol concret de xarxa i al format del missatge.

Un programa client que es connecta a un servei web pot llegir la descripció WSDL per determinar quines funcions es troben disponibles al servidor, quins són els seus paràmetres d'entrada i els d'eixida. A banda, WSDL també permet especificar tipus de dades complexos que poden emprar-se en les funcions del servidor o en les peticions del client. Actualment WSDL és la forma més utilitzada d'especificació de serveis web ja que amb la informació que conté un sol document WSDL el client és capaç de generar un pont d'accés a una operació remota amb tots els paràmetres necessaris per a que el programador puga utilitzar aquest pont com si estigués cridant a operacions dins del propi sistema malgrat que SOAP s'encarrega de processar la petició, encaminar-la al servidor remot, esperar l'execució de l'operació, retornar un resultat (o error) i per últim tornar-li les dades demanades al programa client que va invocar la operació. Tot això es fa de forma transparent al programador que s'abstrau dels detalls de com es transporten les dades del client al servidor remot i només ha de saber què paràmetres són d'entrada i quins d'eixida.

3.2.3 REST

REST, acrònim de **Representational State Transfer** és una arquitectura de programari per a continguts hipermèdia distribuïts sobre Internet, seguint la filosofia de la WWW. El terme es va originar l'any 2000, apareixent per primera vegada a una tesi doctoral sobre la web escrita per *Roy Fielding*, un dels principals autors de l'especificació de l'HTTP. El terme REST es refereix originalment a un conjunt de principis per al disseny d'arquitectures en xarxa. Aquests principis resumeixen com els recursos són definits. El terme freqüentment és emprat en el sentit de descriure qualsevol interfície que transmet dades específiques d'un domini sobre HTTP sense una capa addicional, com fa SOAP. Aquests dos significats poden xocar i fins i tot solapar-se. És possible

dissenyar un sistema de programari de gran volum, d'acord amb l'arquitectura proposada per Fielding sense emprar HTTP o sense interactuar amb la Web. O també es pot dissenyar una interfície XML sobre HTTP que no segueixi els principis REST, i en canvi seguir un model RPC. Cal destacar que REST no és un estàndard, és una arquitectura. Encara que REST no sigui un estàndard, està basat en la utilització dels següents estàndards com HTTP, URL, representació de recursos (XML/HTML/GIF/JPEG/...) i tipus MIME (text/xml, text/html, ...). La motivació de REST és de seguir les mateixes característiques de la web que l'han feta esdevenir una vertadera revolució a nivell mundial. La web és la única aplicació distribuïda que ha aconseguit ser escalable a nivell de tot internet. L'èxit és degut a l'ús de formats de missatges extensibles i estàndards, amb un adreçament global (estàndard i extensible a la vegada).

3.3 Tecnologies de costat del client

3.3.1 JavaScript

JavaScript és un llenguatge script basat en el concepte de prototip, implementat originàriament per Netscape Communications Corporation, i que va derivar en l'estàndard ECMAScript. És conegut sobretot pel seu ús en pàgines web, però també s'utilitza en altres aplicacions. Malgrat el seu nom, JavaScript no deriva del llenguatge de programació Java, però tots dos compartixen una sintaxi similar inspirada en el llenguatge C. Semànticament, JavaScript és més pròxim als llenguatges Self i ActionScript (basat també en l'ECMAScript).

Aquest projecte utilitza de forma massiva JavaScript en la part del client per a proporcionar-li una bona experiència d'usuari en el us de les funcions del navegador i els controls i aspectes visuals de la pàgina en sí. Donat que JavaScript té implementacions diferents en cada navegador, aquest projecte utilitza una abstracció muntada sobre JavaScript anomenada jQuery que és una biblioteca o framework de Javascript, creada inicialment per John Resig, que permet simplificar la manera de interactuar amb els documents HTML, manipular l'arbre DOM, gestionar events, desenvolupar animacions i afegir interacció amb la tecnologia AJAX en pàgines web. jQuery facilita molt la creació d'interfícies d'usuari amagant les diferències d'implementació entre navegadors i fent que l'aplicació web funcione en tots els navegadors amb un esforç mínim. Adés, jQuery és molt extensible i compta amb una gran quantitat de "plugins"que ajuden a crear funcionalitats i facilitar la creació d'interfícies gràfiques semblants a les que el usuari pot tindre en una aplicació d'escriptori. Per terminar,jQuery, facilita l'us d'AJAX que demana al marc de treball dades asíncronament i proporciona una veritable interacció en temps real sense recarregar cap pàgina; donant respostes als events de l'usuari en pocs milisegons.

3.3.2 CSS

Els fulls d'estil en cascada (Cascading Style Sheets, CSS) són un llenguatge formal usat per a definir la presentació d'un document estructurat escrit en HTML o XML (i per extensió en XHTML). El W3C (World Wide Web Consortium) és l'encarregat de formular l'especificació dels fulls d'estil que servirà d'estàndard per als agents d'usuari o navegadors.

La idea que es troba darrere del desenvolupament de CSS és separar l'estructura d'un document de la seva presentació. La informació d'estil pot ser adjuntada tant com

un document separat o en el mateix document HTML. En aquest document, podrien definir-se estils generals en la capçalera del document o en cada etiqueta particular mitjançant l'atribut "style". Els avantatges d'utilitzar CSS (o altre llenguatge d'estil) són:

- Control centralitzat de la presentació d'un lloc web complet amb el que s'agilitza de forma considerable l'actualització del mateix.
- Els navegadors permeten als usuaris especificar el seu propi full d'estil local que serà aplicada a un lloc web remot, amb el que augmenta considerablement l'accessibilitat. Per exemple, persones amb deficiències visuals poden configurar el seu propi full d'estil per a augmentar la grandària del text o remarcar més els enllaços.
- Una pàgina pot disposar de diferents fulles d'estil segons el dispositiu que la mostre o fins i tot a elecció de l'usuari. Per exemple, per a ser impresa, mostrada en un dispositiu mòbil, o ser llegida per un sintetitzador de veu.
- El document HTML en si mateix és més clar d'entendre i s'aconsegueix reduir considerablement la seva grandària.

Hi ha diverses versions : CSS1 i CSS2, amb CSS3 en desenvolupament pel World WideWeb Consortium (W3C). Els navegadors moderns implementen CSS1 correctament, encara que existeixen petites diferències d'implementació segons marques i versions dels navegadors. CSS2, no obstant això, està solament parcialment implementat en els més recents. És per això que a vegades cal utilitzar fulles d'estil diferents per a cada navegador que seran carregades segons una sèrie de regles condicionals per a cada navegador fent així posible que les pàgines web mantinguen un aspecte idèntic i uniforme en cadascun dels diferents navegadors malgrat les diferències en l'interpretació del CSS que fa cada fabricant i navegador.

3.3.3 HTML i XHTML

HTML, sigles *d'HyperText Markup Language* (Llenguatge de Marcat d'Hipertext),és el llenguatge de marcat predominant per a l'elaboració de pàgines web.
És usat per descriure l'estructura i el contingut en forma de text, així com per complementar el text amb objectes tals com a imatges.
HTML s'escriu en forma d'etiquetes, envoltades per claudàtors angulars (<,>). HTML també pot descriure, fins a un cert punt, l'aparença d'un document, i pot incloure un script (per exemple Javascript), el qual pot afectar el comportament de navegadors web i altres processadors d'HTML.

XHTML, acrònim en anglès d'extensible Hypertext Markup Language (llenguatge extensible de marcat d'hipertext), és el llenguatge de marcat pensat per substituir a HTML com a estàndard per a les pàgines web. En la seva versió 1.0, XHTML és solament la versió XML d'HTML, per la qual cosa té, bàsicament, les mateixes funcionalitats, però compleix les especificacions, més estrictes, de XML. El seu objectiu és avançar en el projecte del World Wide Web Consortium d'aconseguir una web semàntica, on la informació, i la forma de presentar-la estiguen clarament separades.

Part II

Arquitectura

Capítol 4

Arquitectura i funcionament del projecte

4.1 Introducció

Aquest projecte està creat sobre una arquitectura multicapa de tres capes, dividides a la vegada en múltiples subcapes.

4.1.1 Capa de persitència

La capa de persitència és l'encarregada de comunicar-se amb la base de dades, i proveir/emmagatzemar dades de les aplicacions web desenvolupades amb el marc de treball. En aquesta capa existeix un component anomenat *Database* que mitjançant un adaptador anomenat *Database_Driver* enllaça directament amb la base de dades mitjançant les funciones que té implementades PHP per a cada base de dades en concret.
Després d'obtindre les dades; les pot processar mitjançant una implementació del patró *Active Record*, que facilita l'us de la base de dades fent un mapeig de les taules a objectes de lògica de domini. Aquesta capa pot proveir objectes de domini, o resultats de consultes.
És la capa més important, ja que totes les aplicacions web funcionen per mitjà d'una o moltes bases de dades; i generen/processen gran quantitat de volum de dades, que han de ser emmagatzemats eficientment.
El objecte responsable d'invocar aquestos serveis anomenats abans és un objecte de tipus *BaseModel*, és l'encarregat de proveir de dades als controladors de cada mòdul dins d'una aplicació web. Aquestes dades després seran passades cap al controlador, que serà qui, mitjançant la lògica de negoci farà transformacions amb aquestes dades obtingudes de la base de dades. Cal comentar que en aquesta capa, dins del model es pot fer servir un client de serveis web (SOAP) per tal d'obtindre dades externes a l'aplicació web, i proporcionar-se-les al controlador.

4.1.2 Capa de lògica de negoci

La capa de lògica de negoci és l'eix clau de tot el sistema, doncs és l'encarregada d'obtindre la petició HTTP, entendre que es demana, executar la petició, i tornar-li la resposta al client. En aquesta capa podem trobar els següents elements:

Request : Objecte que encapsula tots els paràmetres de la petició HTTP, i els fa disponibles a tota l'aplicació web.

Router : Peça clau que s'encarrega de processar tots els fitxers de connexions inclosos en el sistema i que defineixen les accions,què mètodes s'han de cridar per dur a terme l'acció (executar-la), i quins paràmetres necessita aqueixa acció per a que siga efectiva l'execució.
Una vegada ha processat totes les connexions (o les ha carregades d'un fitxer serialitzades) i les té carregades en memòria passa a atendre la petició HTTP actual, cercant d'entre totes les connexions carregades quina és la que ha d'executar (la que la URL correspon a la URL especificada en la connexió). Aquest element proporciona una gran flexibilitat al sistema per a poder fixar adreces arbitràries i *SEO-friendly* [1] important per al posicionament de pàgines web en els principals cercadors.
Una vegada s'han obtingut tots els paràmetres que descriuen l'acció a executar, es passa al següent element **Front Controller**.

El Front Controller o despatxador de peticions és la peça del sistema que s'encarrega de que - a partir dels paràmetres obtinguts per el Router - trobar l'acció a executar, i executar-la. Per tal de fer flexible i ampli el marc de treball, el Front Controller divideix segons el tipus de petició el processament, cerca d'acció, i execució en diverses classes relacionades amb aquest component que s'encarreguen d'obtindre resultats per a cada tipus de petició ja siga del tipus que siga.

El controlador és un conjunt de classes (una o vàries per mòdul de l'aplicació web), que estan destinades a mantindre les accions que poden ser executades en el sistema. Cada controlador proveeix d'accés directe al corresponent model que és a través de qui obté les dades a processar per la lògica de negoci de la base de dades o de serveis web.
Adés, també proveeix de mètodes que permeten renderitzar vistes de la capa superior; la de presentació .

Per tant, aquesta capa és la més important de tot el sistema, i és peça central del patró MVC, Model-View-Controller (Model-Vista-Controlador), que permet desacoblar la lògica de negoci de la persistència i la presentació. Aquest projecte utilitza aquest patró de disseny junt al patró anomenat Front Controller per a proveir d'un potent sistema modularitzat i amb baix acoblament.

4.1.3 Capa de presentació

La capa de presentació és la que s'encarrega de mostrar al usuari el resultat de l'excució de les accions d'una manera visual, o de rebre el resultat d'una acció demanada mitjançant un servei web. En el cas que la presentació siga mitjançant un client web, s'empren diferents petites porcions de codi anomenades ≪views≫ o vistes que contenen codi xhtml, JavaScript,CSS i/o PHP. Adés, per a fer una separació de codi més efectiva, també es poden desacoblar de la vista els codis de JavaScript o de CSS ficantlos en fitxers a banda, i que són carregats a demanda en cada petició HTTP evitant que tota l'aplicació web tinga carregat estils CSS o pedaços de JavaScript no necessaris per la vista que actualment s'està mostrant, alleugerant així en la mida possible el pes i

[1] SEO: sigles de "Search Engine Optimization" - Optimització per a motors de recerca -

temps de càrrega d'una aplicació web. Per tal de produir vistes dinàmiques, es poden emprar diferents tipus de funcions ajudants *helpers* que poden ajudar a fer tasques repetitives (exemple: inserir un enllaç, inserir taules. També es poden construir *widgets* (un conjunt de codis xhtml, JavaScript i CSS que representen una petita funcionalitat dins una web), adés; també es poden instanciar plugins que mostren qualsevol resultat d'algún processament, que envien correus electrònics, sms, o que inserisquen un mapa dins una vista - sempre fent-ho amb el menor nombre de línies possibles i la major claredat - . Per a terminar, també és possible que vistes renderitzen altres vistes,o que criden a altres accions del controlador podent construir una vista complexa amb moltes vistes simples.

Capítol 5

Components del marc de treball

5.1 Introducció

En aquest capítol es fa una presentació de tots els components importants per el funcionament del marc de treball. Per a cada component, es descriu breument, es mostra la seva utilitat i un diagrama UML mostrant per a cada cas el component i els seus subcomponents - en cas de que els haguera - .
Es presenta també una breu referència al patró de disseny emprat per implementar cada component (si escau).

Tots els components ací presentats estan programats dins el directori `/framework/lib` amb la corresponent documentació (en idioma anglès), aquesta documentació es troba bé en el capítol corresponent del component dins d'aquesta memòria (en cas de components molt prioritaris) o dins la documentació que acompanya a aquesta memòria i que està generada automàticament per el programa *PHPDoc*.

Totes les classes, per necessitats del sistema afegeixen al nom del component el prefix **FW_** que indica al sistema que és una classe del marc de treball i li facilita la tasca al sistema de *Bootstrap* o càrrega de dades, donada aquesta diferència amb la resta de classes i el nom de cada classe (separades les paraules amb guions baixos «_»), el sistema de Bootstrap sap que ha de trencar el nom per els guions baixos, substituir-los per el caràcter separador de directoris (caràcter «/») i afegir-li el sufix *.class.php* .
Per tant, per trobar un fitxer d'un component, s'ha de fer el mateix que fa Bootstrap a partir del directori `/framework/lib`.

5.2 Active Record

Active Record engloba un conjunt de classes que serveixen per fer un *mapeig* de dades *Objecte-Relacional*, és a dir, que s'encarreguen de generar objectes a partir d'informació desada en una base de dades relacional. Per fer el procés de mapeig es necessita conèixer dades de l'objecte a representar i la seva correspondència en una taula d'una base de dades relacional. Aquest procés es fa automàticament mitjançant les API de reflexió d'objectes en PHP - que són capaces de fer instrospecció en un objecte i

obtindre les seves propietats - i obtenint informació d'una taula de la base de dades. [1]

Active Record proporciona una manera neta, ràpida i senzilla d'obtindre les dades d'una base de dades en forma d'objectes mitjançant la *implementació d'una versió del patró de disseny ActiveRecord* original de Ruby On Rails - utilitzat amb molt d'èxit en Ruby On Rails - i descrit per *Martin Fowler* en *Patterns of Enterprise Application Architecture*.

Abstrau al desenvolupador de gestionar directament SQL i redueix les possibilitats d'un atac mitjançant SQL-Injection. A més a més, al transformar les dades de la base de dades relacionals en objectes, el desenvolupador es beneficia de tot un món d'avantatges que permet el disseny de sistemes orientat a objectes ja que cada objecte pot encapsular lògica i regles de negoci segons les necessitats del client i a la vegada tindre una forma senzilla i elegant de poder guardar eixos canvis en l'objecte i els objectes relacionats en una base de dades relacional sense gaires esforços per al desenvolupador. Es pot dir, que ActiveRecord és el component més important i gran del marc de treball i dota a aquest de facilitat per construir aplicacions web o serveis web. En conclusió, un disseny més senzill, un desenvolupament més ràpid.

5.2.1 Components d'Active Record

Components generals

Active Record té els següents components principals:

Model : Proporciona les funcionalitats bàsiques per a implementar un objecte i que es puga fer servir amb ActiveRecord. Totes les classes del model de dades han d'estendre de la classe Model per a així poder utilitzar tots els avantatges que Active Record proporciona. També proporciona funcionalitats avançades per convertir dades a formats CSV, XML, JSON o importar-ne dades des d'aquestos formats.

Tree : Extensió de *Active_Record_Model* que proporciona una manera de gestionar les col·leccions de dades jeràrquiques com són els arbres de dades.

Active Record : Proporciona les funcionalitats comunes en una base de dades per al mapeig objecte-relacional, es suporta en *ActiveRecord_Metadata_Manager* i *ActiveRecord_Metadata_Schema* per obtindre informació sobre un objecte del model de domini i en *ActiveRecord_Query_Builder* per generar les consultes necessàries per obtindre les dades de la base de dades i generar així un objecte i totes les seves relacions. Utilitza el component Database per transferir-li les consultes SQL (sempre en estàndar SQL92/99) a la base de dades utilitzada per a cada cas.

Result : Proporciona una col·lecció d'objectes d'Active Record (resultats d'una consulta) i els mètodes per recórrer-la. També proporciona funcionalitats avançades per convertir dades a formats CSV, XML, JSON o importar-ne dades des d'aquestos formats.

Relation : Proporciona una estructura de dades en les que emmagatzemar les relacions amb altres objectes d'una propietat d'un model d'Active Record. En aquestes re-

[1] El procés està descrit amb més detall en el capítol corresponent al model de dades, en la part de desenvolupament

lacions es poden obtindre, filtrar i cercar les dades gràcies a que hereten mètodes de Result i d'Active Record.

Pagination : Proporciona funcionalitats de paginació de resultats i ajudants per mostrar en les vistes el número de pàgines, les pàgines i número de resultats disponibles.

Exception : Proporciona excepcions que llança el sistema *Active Record*.

Esquemes

El component Active Record ha de saber per a cada model amb el que tracta quines columnes té i quins tipus de dades tenen eixes columnes així com les relacions entre models.

Manager : Classe que funciona com a dipòsit general de tots els esquemes d'Active Record.

Schema : Classe que analitza els models i proporciona informacions sobre les seves columnes, tipus de dades i relacions amb altres models.

Generació de consultes

El component Active Record delega en vàries classes l'escriptura de consultes SQL o No-SQL de forma que aquestes siguen capaços d'escriure les consultes corresponents segons convinga i segons els models sobre els que s'utilitza. Aquesta part està implementada utilitzant el patró de disseny *Strategy* de forma que es puga desacoblar la generació de consultes segons el tipus de base de dades utilitzat, ja siga SQL o No-SQL. En concret per aquestes darreres bases de dades (les No-SQL) el sistema encara no funciona, però ja està preparat per emprar un generador específic per cada tipus de base de dades No-SQL majoritària existent (*CouchDB*,*Cassandra*, ...) donat que es considera que seran en breu una forma d'emmagatzemar dades. L'eixida d'aquestes consultes s'envia directament al component *Database* que les executa contra la base de dades utilitzada a cada moment i segons convinga.

QueryBuilder : Proporciona una classe base sobre la que fer les cridades a generació de consultes per executar-les en la base de dades. Gestiona els generadors de consultes (actualment només per a SQL 99 estàndard) i delega en ells la responsabilitat de generar la consulta corresponent.

QueryBuilder_Base : Classe bàsica i abstracta per implementar generadors de consultes.

QueryBuilder_SQL : Genera les consultes SQL necessàries per consultar a la base de dades.

Validació de models

El component Active Record també proporciona validacions de dades en totes les columnes d'un objecte de domini per evitar que s'inserix-ca informació incorrecta en la base de dades. Les validacions poden ser de cinc tipus:

- De tipus de dades en cada columna: Cada columna té un tipus de dades bàsic que ha de ser respectat.

- De llargària del contingut de cada columna: Per a cada columna es defineix a banda d'un tipus de dades una llargària màxima i mínima.

- De valors nuls: Una columna que no accepte valors *nuls* no podrà contindre mai un valor nul donat que provocaria un error en la base de dades.

- De clau primària: Assegurar-se de no repetir valors i no inserir valors nuls en aquestes columnes.

- De unicitat: Assegurar-se que el valor de la columna és únic i està inserit ja en la base de dades.

- Personalitzats: Es poden fer validacions personalitzades sobre qualsevol de les columnes del model.

Validator : Aquest objecte analitza el esquema del model i valida pas per pas cadascun dels valors de les propietats de l'objecte.

En quant a patrons de disseny Active Record implementa els següents patrons de disseny:

- Singleton a *ActiveRecord_Medatada_Manager*

- Active Record a *Model*,*Tree* i *ActiveRecord*

- Iterator a *ActiveRecord_Result*

- Registry a *ActiveRecord_Medatada_Manager*

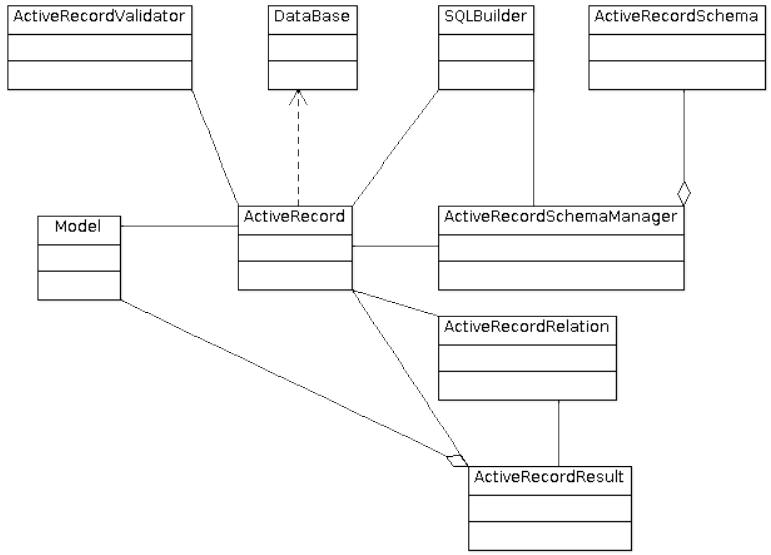


Figura 5.1: Diagrama UML del sistema ActiveRecord

5.3 Authentication

El component *Authentication* proveu de serveis d'autenticació d'usuaris al marc de treball. Authentication utilitza una sèrie de regles per determinar la forma d'autenticar usuaris.

Proveu autenticació via tres handlers (mecanismes):

database : Autenticació mitjançant la base de dades.

imap : Autenticació mitjançant servidors *IMAP*

htpasswd : Autenticació mitjançant fitxers *httpasswd* generats per el servidor web Apache.

Aquestos mecanismes poden ser configurats en forma de regles en el fitxer de configuració *authentication.php* situat en el directori de configuració. Es preveu poder donar suport a autenticació via protocol *OAuth* en un futur. OAuth permetrà una millor autenticació i integració amb les aplicacions web 2.0 que utilitzen aquest protocol per identificar els usuaris i intercanviar-ne dades.

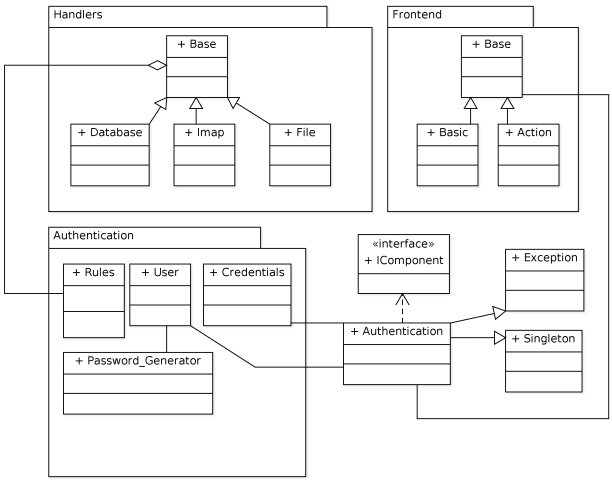


Figura 5.2: Diagrama UML del component Authentication

5.4 Browser

El component *Browser* s'encarrega de la detecció de les característiques de l'agent d'usuari (navegador) a través del qual es fa una petició http al marc de treball. *Browser* pot identificar - llegint la capçalera *HTTP_USER_AGENT* que envia el client en una petició *HTTP* - el tipus de navegador emprat per a la petició, la seva versió, i el sistema operatiu sobre el que funciona el navegador. *Browser* també implementa el patró de disseny *Singleton*.

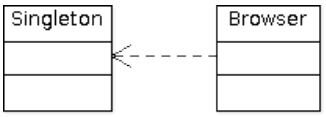


Figura 5.3: Diagrama UML del component *Browser*

5.5 Cache

El component *Cache* és un component essencial en tota aplicació web que vaja a suportar una càrrega de tràfic molt gran. *Cache* implementa un sistema de caching de dades que permet desar dades en el disc dur o en la base de dades (segons el *driver* de caché escollit) durant un temps definit i recuperar eixa informació per a posterior usos. Això evita que cada petició es processe una gran quantitat d'informació que pot durar prou temps fent que el sistema tinga una pobre resposta.
Es poden cachear els següents tipus de dades:

- **Eixida d'una acció**. Es desen en la *caché* les dades produïdes com a eixida de l'execució d'una acció.

- **Plantilles XHTML**. Es poden desar plantilles ja composades i amb les dades necessàries.

- **Dades de l'usuari**, dades intermèdies que es gastaran més tard , qualsevol informació que siga necessària i comporte un gran consum de processament de dades.

El component *Cache* té quatre formes de guardar les dades:

- En **fitxers** en el disc dur: Desa les dades en fitxers en el disc dur.

- En **bases de dades**: Desa les dades en una taula especialment dissenyada en la base de dades que utilitza el sistema.

- En base de dades **SQLite3** en memòria: Desa les dades en una base de dades en memòria en el gestor de bases de dades SQLite3, que escrit en C i molt optimitzada és una de les bases de dades més ràpides existents avui en dia.

- En **Memcache**: Utilitza una extensió de PHP per el sistema *Memcached*, que empra com a sistema d'emmagatzematge la memòria RAM, reduïnt així la necessitat d'accessos a disc dur o a bases de dades.

El sistema de caché desa dades identificades per id (habitualment es fa un md5 de la cadena *'Module—Controller—Action—Internal'* amb les dades corresponents a cada acció, de forma que l'identificador **és únic** i fàcil de generar, també s'identifiquen per una clau anomenada **namespace o espai de noms** i el **ttl o temps de vida** de les dades. Així doncs, unes dades tenen un identificador en un espai de noms i un temps de vida. Passat aquest temps de vida (si la data de les dades convertida a segons des de l'inici de *l'època UNIX* sumada al temps de vida és major o igual que la data i hora del sistema en el moment que es generà la petició) el sistema donarà per expirades les dades.

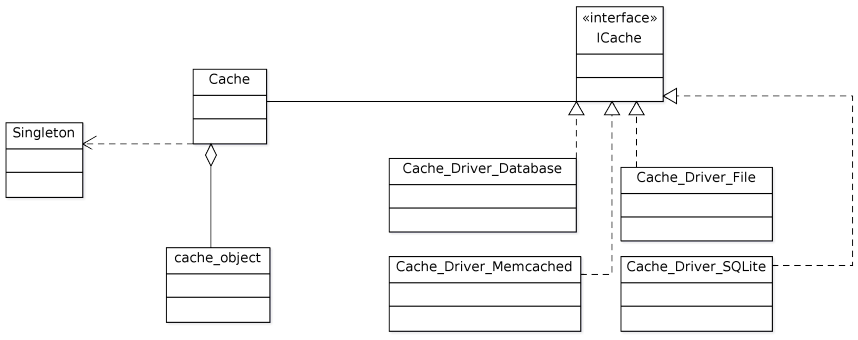


Figura 5.4: Diagrama UML del component *Cache*

Aquest component implementa els patrons de disseny *Singleton* i *Strategy*

5.6 Config

El component *Config* és el component més important del sistema. S'encarrega de gestionar tota la configuració del marc de treball així com les configuracions de les aplicacions web o d'usuari. Fa senzilla la càrrega, cerca i/o escriptura de valors en fitxers de configuració. Aquest component carrega tots els fitxers de configuració del sistema, organitza els seus valors i els deixa accessibles des de qualsevol punt del sistema (emprant el patró de disseny *Singleton*).

També disposa d'una *Caché* de dades (implementada utilitzant el patró de disseny *Registry*) per fer més ràpida la recuperació d'un valor de configuració, emmagatzemant-lo en Caché la primera vegada que es llegeix i evitant haver d'accedir-ne cada vegada a un fitxer de configuració complex i per tant estalviant temps al accedir a un lloc on està el valor i que es recupera immediatament.

A més a més, per fer més senzilla la tasca als desenvolupadors i als consumidors de la informació emmagatzemada en la configuració, és possible crear classes que accedeixen directament a la configuració i recuperen per el client els valors necessaris sense haver el client d'accedir-ne directament a la configuració. A aquestes classes ajudants se'ls coneix com *Config_Handler* i estenen de *Config_Handler_Base* que proporciona els mètodes necessaris per tractar amb la configuració.

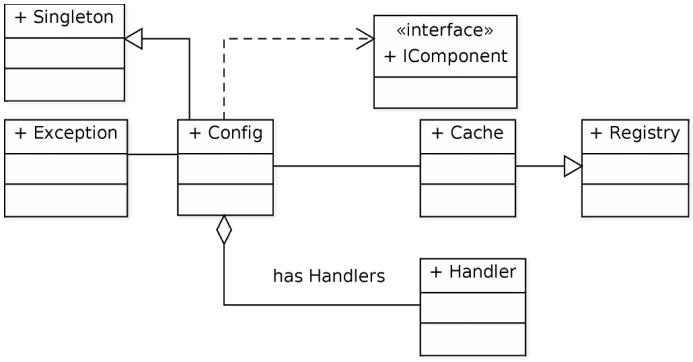


Figura 5.5: Diagrama UML del component *Config*

5.7 Context

El component *Context* té com a finalitat compartir dades entre components, és a dir, proveir les dependències dels diferents components sense haver-les de passar per mitjà dels constructors o d'altres mètodes. Les dades creades per aquest component només serveixen per una petició, és a dir, que no es mantenen constantment.

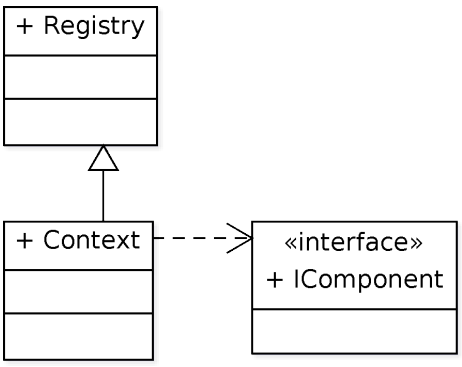


Figura 5.6: Diagrama UML del component *Context*

Aquest component implementa els patrons de disseny *Singleton* i *Registry*.

5.8 Database

El component *Database* és la principal i única interfície del sistema per a comunicar-se amb bases de dades. Aquest component implementa les funcionalitat més comunes d'una base de dades (connectar, consultar, recuperar resultats, iniciar i aturar transaccions, obtindre el número de files afectades per una consulta o del resultat i obtindre informació sobre una taula d'una base de dades). Per a poder maximitzar la compatibilitat amb bases de dades, el component implementa una separació de la lògica i algorismes necessaris per a cada tipus de base de dades, encapsulada aquesta en un component anomenat *Database_Driver* que implementa les operacions del driver bàsic de base de dades *Database_Driver_Base*. Això fa que tots els drivers de bases de dades tinguen els mateixos mètodes i aquestos amb la mateixa signatura, per poder fer servir d'una manera senzilla i transparent qualsevol tipus de base de dades suportada per PHP o les seves extensions.

Per tal d'obtindre informació sobre la configuració de la base de dades en us actualment, el component Database fa servir els serveis del component Environment el qual ajudant-se del serveis que proveeix el component encarrega de gestionar la configuració obté informació sobre la base de dades en us, el tipus d'aquesta i altres detalls com els missatges d'error o la configuració del component Log.

A més a més, Database permet gestionar les connexions a bases de dades, de manera que es poden fer servir vàries bases de dades ≪a la vegada≫. Això és bo si es pretenen fer migracions de bases de dades, però es desaconsella a menys que es controlen les operacions d'inserció i actualització, i la concurrència per tal de no fer malbé la integritat de les dades.

DataBase proveu compatibilitat amb les següents bases de dades:

- MySQL: Drivers MySQL i MySQL Improved.
- PostgreSQL: Driver de PostgreSQL.
- SQLite 2 i 3: Drivers respectius d'SQLite.
- PDO: Compatible amb qualsevol tipus de base de dades que és compatible PDO (PHP Data Objects)

Properament implementarà també possibilitat de connexió a bases de dades de tipus *NoSQL*, tipus de bases de dades - orientades en la seva majoria a documents - que estan guanyant força front a les bases de dades tradicionals o relacionals.

Database implementa un patró Singleton de forma que la única instància que funciona en el sistema és global i compartida amb tot el marc de treball.

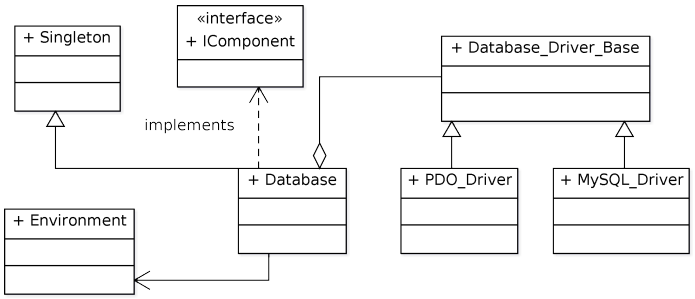


Figura 5.7: Diagrama UML del component Database

5.9 Environment

El component *Environment* s'encarrega de gestionar les diferents configuracions dels entorns de programació del sistema. Un entorn de programació és una sèrie de regles, que regeixen el funcionament del sistema.
Les regles són:

- Nivell de depuració: Determina si s'ha de depurar tota acció del marc de treball o només una part de l'execució de cada acció, així com el depurador a utilitzar.
- Nivell de log: Determina quins tipus de missatges han de ser loggejats i a quina profunditat de detalls o el driver del component *Log* a emprar.
- Caché: Determina si utilitzar o no caché de dades.
- Base de dades: Determina quina connexió a base de dades utilitzar.

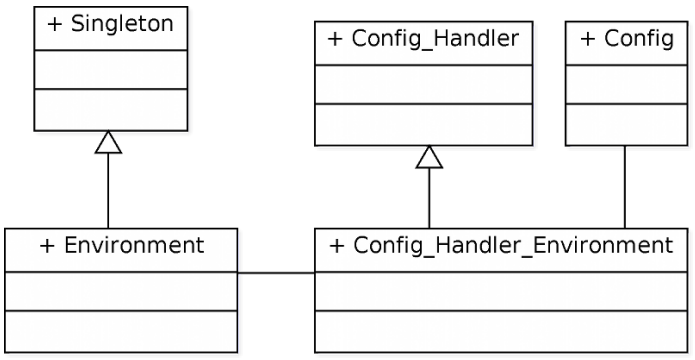


Figura 5.8: Diagrama UML del component Environment

Com a molts altres components, també implementa el patró de disseny Singleton.

5.10 Error_Handler

Error_Handler proveu un sistema que captura tots els errors del sistema i s'ajuda del component *Log* per desar informació sobre l'error. Aquest component dona accés ràpid a mostrar al client els tipus d'errors més comuns: 403 - accés denegat, 404 - recurs no trobat, 500 - error intern del servidor i gestiona tots els errors que pot llançar qualsevol component del marc de treball.

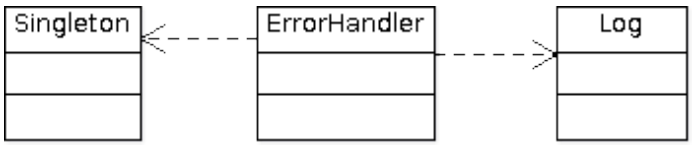


Figura 5.9: Diagrama UML del component *Error_Handler*

Com a molts altres components, també implementa el patró de disseny Singleton.

5.11 Filter

El component *Filter* s'encarrega de realitzar comprovacions i filtrar totes les peticions HTTP que arriben al sistema. Filter, mitjançant una cadena de filtres *Filter_Chain*, comprova que es donen totes les condicions de seguretat, autenticació i que es trobe implementada la funcionalitat que es vol executar. Aquesta cadena de filtres fa que tots els filtres s'executen un darrere de l'altre fins que algún filtre falle o es termine la cadena de filtres.

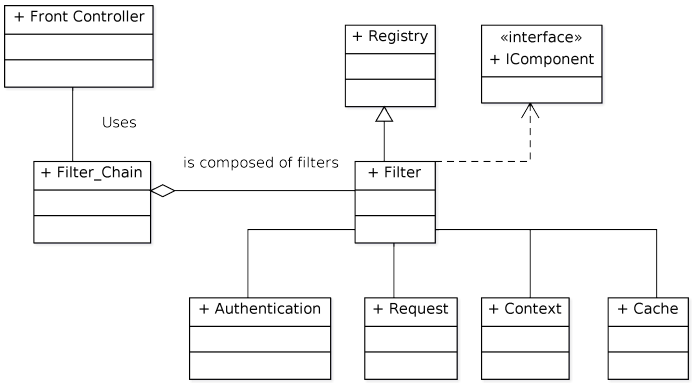


Figura 5.10: Diagrama UML del component Filter

Aquest component implementa els patrons de disseny *Registry* i *Intercepting Filter*.

5.12 Flash

Aquest component permet al desenvolupador el pas de missatges i dades entre peticions HTTP del marc de treball. És especialment útil per mostrar el resultat d'accions de creació, modificació o destrucció d'objectes de domini, en forma de missatges html

renderitzats en la vista de l'aplicació i que indiquen a l'usuari el resultat de la seva acció.

El component Flash està dissenyat utilitzant el patró de disseny *Singleton*

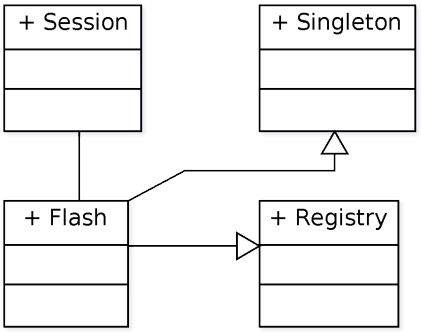


Figura 5.11: Diagrama UML del component Flash

5.13 Front Controller

El component *Front Controller* s'encarrega de la coordinació entre el procés d'enrutament fet en el component *Router* i l'execució de l'acció del resultat obtingut o el mostrar un error.

Rep la petició, la passa al component *Router* per a que l'analitze i quan obté el resultat, tracta la petició de la forma necessària per a cada tipus de ruta, o d'error. El resultat que proporciona el component Router després del procés d'enrutament d'una petició web, conté tots els paràmetres necessaris per executar una acció d'un determinat mòdul i controlador del sistema MVC. Aquesta informació es passa a un mètode del Front Controller que selecciona el tipus d'algorisme a utilitzar, per a això aquestos algorismes estan implementats en classes que conformen una estratègia per a cada tipus de ruta.

Després el component *Front Controller* delega en la cadena de filtres del component *Filter* comprovacions d'autenticació i control d'usuaris, si existeix l'acció implementada en el sistema, si existeixen dades en caché, ... Genera també les capçaleres de resposta HTTP pròpies de cada tipus de ruta amb l'ajuda del component *HttpResponse*, executa l'acció i per terminar renderitza el layout, en cas que l'acció siga de tipus 'aplicació'

Els component Dispatcher s'estructura en els següents subcomponents:

Dispatcher : Classe principal, el FrontController del marc de treball.

BaseEngine : Clase base que inclou funcionalitats comunes a totes les clases que implementen algorismes de despatxament

AppEngine : Algorisme de despatxament per a pàgines web dinàmiques.

AjaxEngine : Algorisme de despatxament per accions AJAX (que tornen dades en format XML).

jsonEngine : Algorisme de despatxament per accions AJAX (que tornen dades en format JSON).

CronEngine : Algorisme de despatxament per accions programables mitjançant el programador de tasques Cron.

MimeEngine : Algorisme de despatxament per accions que produeixen dades binàries, o d'algun format mime reconegut (pdf, sons, vídeos, postscript, imatges, ...).

RestEngine : Algorisme de despatxament per serveis web de tipus REST.

PluginEngine : Algorisme de despatxament per funcionalitats implementades en plugins.

Front Controller implementa els patrons de disseny *Singleton*, *Strategy* i *Front Controller*.

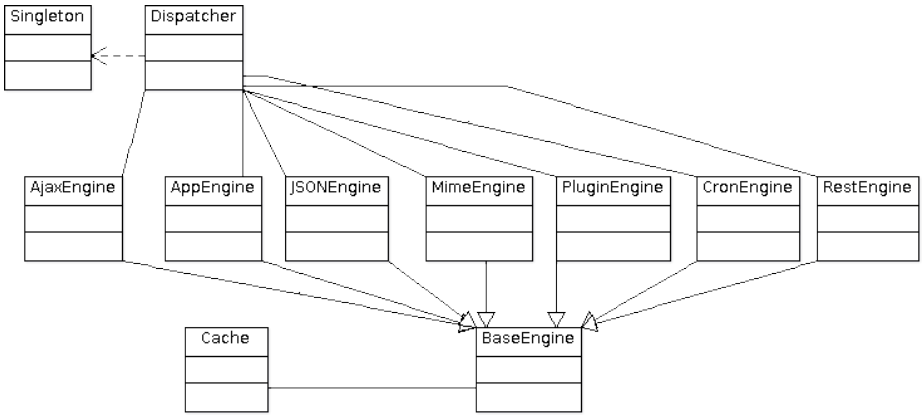


Figura 5.12: Diagrama UML del sistema Front Controller

5.14 Helpers

Un *Helper* o ajudant del marc de treball, és un conjunt de classes que contenen mètodes i funcions d'ajuda al desenvolupament de vistes o accions del sistema MVC.

Existeix un helper que proporciona funcions capaces de generar etiquetes xhtml amb els paràmetres que el desenvolupador necessite, per evitar que aquest escriga les etiquetes una i una altra vegada en les diferents vistes del sistema. A més a més, aquest helper proporciona funcions especials que ajuden a obtindre codi d'enllaços xhtml a les diferents accions del sistema, fent que es mantinga l'enllaç encara que canvie l'adreça URL, és a dir, que el desenvolupador enllaça peces del sistema MVC sense haver-se de preocupar per cada URL que pot ser decidida en temps d'execució i automàticament. També cal dir, que el desenvolupador pot desenvolupar els seus propis helpers per a tasques repetitives que necessite.

5.15 HttpResponse

Aquest component implementa una forma unificada i centralitzada de generar les capçaleres HTTP en resposta a cada petició HTTP. Permet enviar capçaleres de caché, de tipus de contingut, d'autenticació o personalitzades.

Implementa el patró de disseny *Singleton* de manera que només existeix una instància en tot el sistema d'aquest component que envia les capçaleres HTTP de resposta.

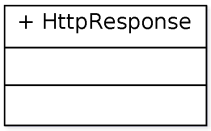


Figura 5.13: Diagrama UML del component HttpResponse

5.16 Locale - *I18N* (internacionalització)

El component *Locale* permet fer aplicacions multilingües amb el marc de treball. Proporciona serveis per a traduir-ne cadenes o textos i per a seleccionar el fitxer d'idioma - o locale - corresponent per a cada usuari traduint automàticament la web de forma totalment transparent a l'usuari.

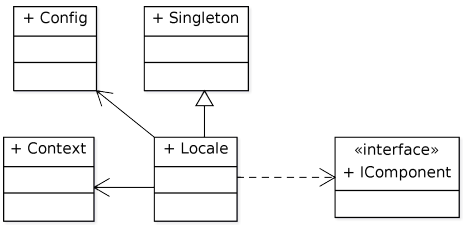


Figura 5.14: Diagrama UML del component Locale

5.17 Log

El component Log proveu d'un mecanisme de logging d'errors al marc de treball. Logger pot desar els missatges dels log de les següents formes:

- Text pla/fitxer .log : Desa els missatges de log dins un fitxer de text pla.
- Base de dades: Desa els missatges de log dins una taula en una base de dades.
- Correu electrònic: Envia cada log per correu electrònic.

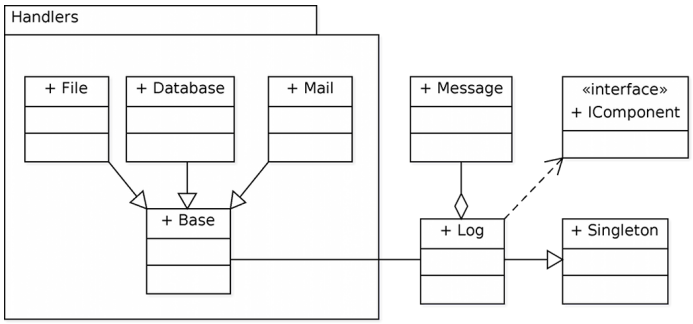


Figura 5.15: Diagrama UML del component Log

Log implementa els patrons de disseny *Singleton* i *Strategy*, de manera que amb el primer es fa accessible des de qualsevol punt del marc de treball i amb el segon desacobla la lògica i algorismes de desat dels logs per a cada tipus.

5.18 Mailer

Mailer és un component encarregat de l'enviament de correus electrònics. Fa us de la biblioteca externa *PHP Mailer* que proveu al marc de treball de capacitat per enviar tot tipus de correu electrònics, incloent-hi aquells en *format html* o amb *fitxers adjunts*.

Mailer implementa el patró de disseny *Wrapper* o *Adapter*, ja que és un embolcall de l'API de la biblioteca externa *PHP Mailer*

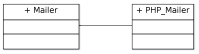


Figura 5.16: Diagrama UML del component Mailer

5.19 MVC

El sistema MVC és un component que permet al desenvolupador desenvolupar les funcionalitats de la seva aplicació web.
Consisteix en un sistema composat per:

BaseController : Classe base que representa la funcionalitat bàsica a utilitzar en la part del *controlador* del sistema MVC.

BaseModel : Classe que representa la funcionalitat bàsica a utilitzar en la part del *model* d'una aplicació escrita amb sistema MVC. A més a més, proveeix un accés a la base de dades.

Views : Fitxers que implementen *vistes*, és a dir codi XHTML mesclat amb un mínim de codi PHP.

Un mòdul del sistema MVC es composa com a mínim d'una classe controlador - que ha d'estendre de *BaseController* - , i una classe model - que ha d'estendre de la classe *BaseModel* - [2] .

Les aplicacions web que poden ser desenvolupades amb aquest marc de treball es basen en el sistema MVC, amb tal de desacoblar la lògica de negoci (Controller), de la presentació (View) i l'obtenció de dades (Model). Aquest component implementa els patrons de disseny *Model View Controller* i el patró HMVC o *Hierarchical Model View Controller*.

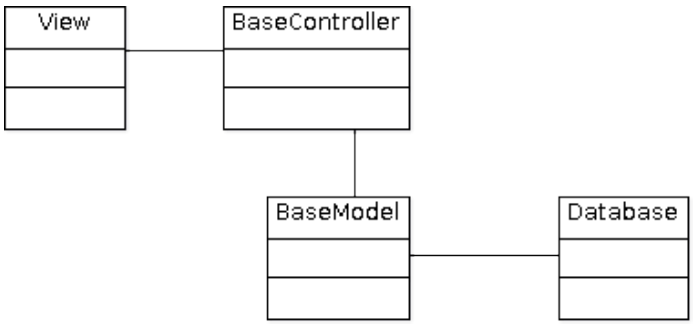


Figura 5.17: Diagrama UML del sistema MVC

[2] Es detalla amb més precisió el procés en el capítol corresponent en la part de desenvolupament

5.20 Plugins

El sistema de Plugins permet al desenvolupador estendre les funcionalitat d'aquest mitjançant llibreries externes, codis de tercers, o funcionalitats pròpies. Aquestes noves funcionalitats es poden agrupar en forma d'afegit o *Plugin*, de forma que siguen accessibles des de qualsevol acció del sistema MVC. El sistema de plugins del marc de treball es composa d'una classe anomenada *Plugin_Registry*, que carrega, gestiona i proporciona les funcionalitats necessàries per obtindre el Plugin que es necessite a cada moment. Plugin_Registry conté un registre (implementat mitjançant el patró de disseny *Registry*) de Plugins disponibles en el sistema i que poden utilitzar-se per realitzar qualsevol acció i obtindre un resultat a l'acció executada.

Cada Plugin escrit per aquest marc de treball ha d'estendre de la clase *Plugin_Base*, que proporciona funcionalitats de càrrega de fitxers o biblioteques, de càrrega dinàmica de models de dades (els plugins *també poden utilitzar el sistema ActiveRecord*), de gestió de propietats/opcions de cada plugin dins la base de dades i de càrrega i processament d'informació del fitxer de descripció de cada plugin `options.php` i `plugin.php` A més a més, proporciona mètodes que inicialitzen i configuren el plugin segons les opcions a utilitzar. [3]

Els plugins permeten encapsular funcionalitats externes al marc de treball - fent us dels patrons de disseny *Adapter* o *Proxy* - i accedir-ne a la funcionalitat externa a través de la interfície del plugin. D'aquesta forma, es poden incorporar funcionalitats, existents en altres biblioteques al marc de treball permetent reutilització de codi font i ampliació de funcionalitats més enllà de la pròpia aplicació web a desenvolupar amb el marc de treball. Està especialment recomanat per a utilització d'APIs web externes de manera que es cree un objecte que permeta fer-ne us d'aquestes d'una manera senzilla.

Aquest sistema de Plugins implementa els patrons de disseny *Registry* i *Singleton* en *Plugin_Registry*, i *Adapter* en *Plugin_Base*.

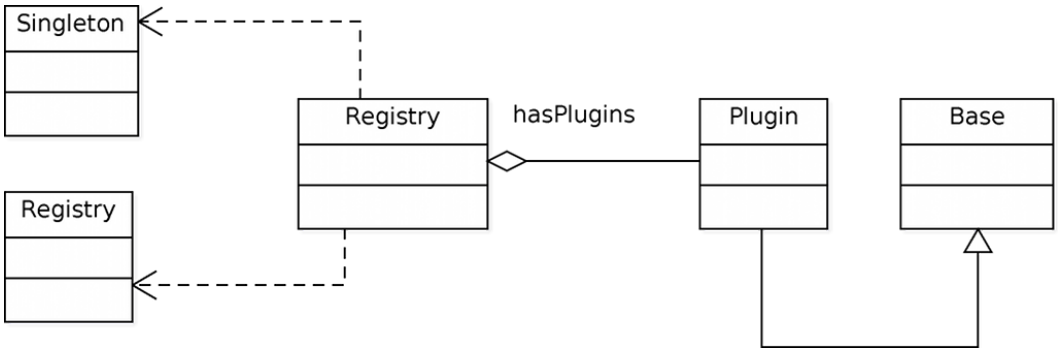


Figura 5.18: Diagrama UML del sistema de Plugins

5.21 Registry

El component *Registry* és una implementació del patró de disseny *Registry* que permet al desenvolupador desar dades en un magatzem de dades en el sistema i accedir-ne a ell en qualsevol moment del cicle de l'aplicació.

[3] Per més informació dels plugins, veure capítol dedicat al tema en la part de desenvolupament

Amb *Registry* es proporciona una manera senzilla i endreçada d'emmagatzemar i accedir-ne a aquestes dades. El component Registry està implementat com una classe abstracta, per tant, per fer us d'ella s'ha de crear una classe derivada amb les funcionalitats pròpies del tipus de registre que es necessite.

Figura 5.19: Diagrama UML del component *Registry*

5.22 Request

Request és un dels principals components del marc de treball. Request emmagatzema tots els paràmetres d'una petició web, tals com els paràmetres passats via els mètodes GET i POST, la URL de la petició, i els fitxers enviats mitjançant formulari. També serveix per obtindre dades de la petició HTTP (*http referrer* o des d'on vé la petició HTTP), saber si s'està produint la petició http des d'un entorn segur (HTTPS), si aquesta ve produïda per una *petició http ajax* o inclòs per obtindre certes informacions de les capçaleres de la petició http.

Aquest component és accessible des de qualsevol punt del marc de treball degut a que està implementat seguint el patró de disseny *Singleton*.

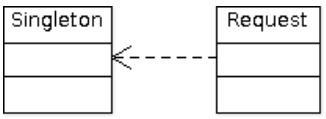


Figura 5.20: Diagrama UML del component *Request*

5.23 REST

Aquest projecte disposa d'una implementació completa per a proporcionar serveis web implementats mitjançant l'arquitectura REST (Representational State Transfer). Per una banda disposa del component *Rest_Server* que proporciona les funcionalitats de servidor de serveis web REST mentre que per l'altra, la part del client la proporcionen els components *Rest_Client* i *Rest_Response* amb els seus tipus de resposta *Rest_Response_JSON* i *Rest_Response_XML*.

5.23.1 Part del client

El client de serveis web basats en l'arquitectura REST és capaç de connectar-se a un servei web i obtindre dades mitjançant peticions HTTP amb els verbs que defineix l'arquitectura REST *(GET,POST,PUT i DELETE)*.

El client de REST està preparat per a consumir els dos formats més habituals de dades per a aquestos serveis web, XML i JSON. Així doncs, el client pot connectar-se i obtindre informació en algun d'eixos formats que després serà transformada dins el marc de treball a elements utilitzables senzillament com són els arrays en el cas de JSON i objectes de tipus *SimpleXMLElement* en el cas de XML. També és capaç d'interpretar les capçaleres de resposta HTTP i els errors que aquestes poden contindre.

Aquest component empra la biblioteca *CURL* de PHP per a poder fer connexions HTTP, així que és indispensable que la instal·lació de PHP compte amb aquesta biblioteca.

5.23.2 Part del servidor

El servidor REST està implementat com una extensió del sistema d'enrutament i Front Controller en l'especialització d'aquest últim anomenada *FrontController_Rest*.

S'encarrega de comprovar els tipus de paràmetres, el servei sol·licitat i el mètode HTTP de la petició per a despatxar-la, obtindre el resultat i enviar-lo de tornada al client. El servidor de rest suporta els formarts XML,JSON o qualsevol tipus de dades *MIME* per a respostes i peticions HTTP. Els serveis web desenvolupats per a REST han de retornar un array de dades que després serà convertit a XML,JSON o altres formats segons convinga.

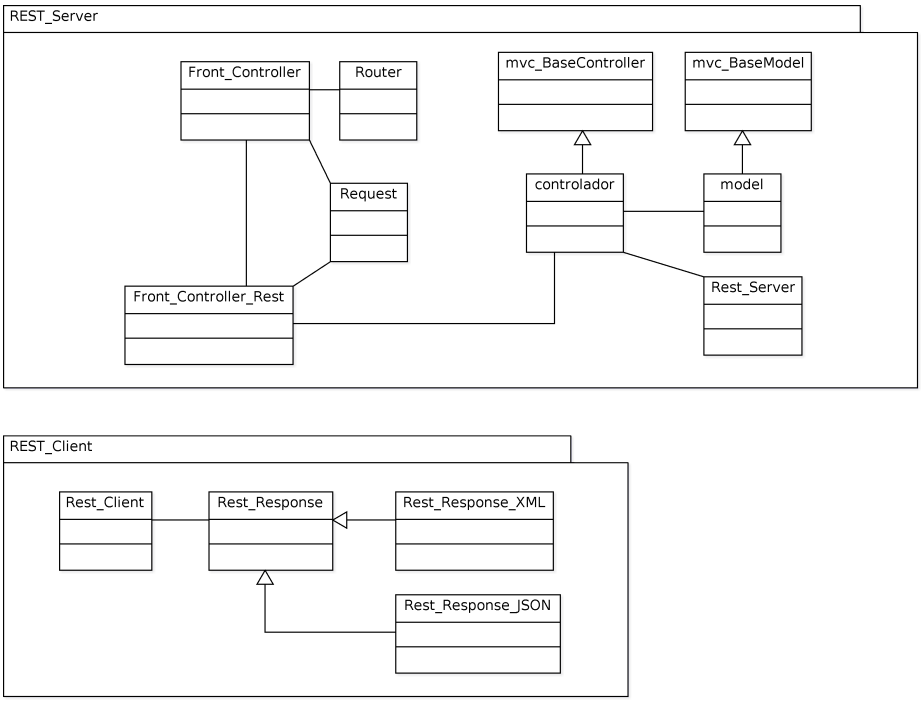


Figura 5.21: Diagrama UML del sistema de serveis web REST

5.24 Router

Router és el component del sistema que s'encarrega de processar la petició web (la url i paràmetres d'aquesta, en concret) i obtindre un resultat que indique la localització del codi - dins el sistema MVC - , que s'ha d'executar per satisfer la petició web.

Router basa el seu funcionament en l'existència de diferents tipus de rutes - una ruta es una sèrie de dades (url, dades del sistema MVC, paràmetres necessaris per execució de l'acció corresponent, dades d'autenticació, dades de caché, ...) organitzades en forma d'arrays - .
Cada ruta pot ser d'un dels següents tipus: application, ajax ,json ,mime ,plugin, cron, redirect, rest, static i soap .

Aquestes rutes resideixen en una estructura de dades que es carrega o genera en temps d'execució i per a cada execució del marc de treball. Per accelerar el processament d'aquestes rutes i guanyar temps i memòria, i evitar el processament de molts fitxers, primer es carreguen les rutes en memòria (processant tots els fitxers PHP que calguen) i quan l'estructura de dades necessària està formada i ordenada, es desa (serialitzada) en un fitxer binari que conté les dades necessàries per a que després el marc de treball siga capaç d'obtindre la mateixa informació però sense hacer de processar fitxers de rutes, la qual cosa fa un estalvi important de temps.

Si aquest fitxer amb les dades serialitzades no existeix, es genera; per altra banda, si aquest fitxer existeix, només s'ha de carregar, tenint ja totes les dades en memòria, el component *Router* ja pot processar i *enrutar* la URL corresponent a la petició web que ha entrat al sistema.

Router és el component més crític de tot el marc de treball, ja que és el motor que analitza cada petició web i que obté la informació de l'acció que ha d'executar en cada moment el marc de treball.

El component Router implementa el patró de disseny *Singleton*.

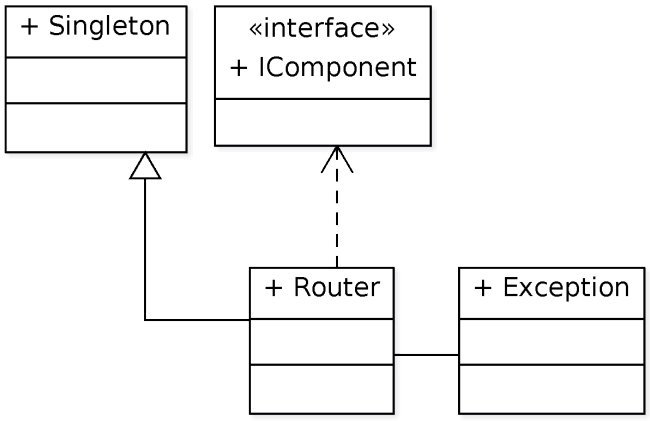


Figura 5.22: Diagrama UML del sistema Router

5.25 Session

El component *Session* s'encarrega de gestionar i mantindre la informació relativa a la sessió de PHP, que emmagatzema un conjunt de dades en memòria del servidor amb un identificador únic per a cada usuari.

Session permet mantindre informació necessària que s'ha de tindre en memòria entre *peticions HTTP* degut a que el protocol HTTP és un protocol sense estat i no pot recordar-se de dades que havia de mantindre per a futures peticions.

A banda, *Session* també pot emmagatzemar informació que necessite l'usuari i informació de components interns del marc de treball.

Session té un seguit de mètodes que ajuden a crear i destruir la sessió, a comprovar si hi ha emmagatzemada una dada en la sessió, a obtindre eixa informació i a substituir o desar eixa dada en la sessió.

Per finalitzar, *Session* implementa el patró de disseny *Singleton*, per la qual cosa, només hi ha una única instància de *Session* en tot el sistema a la que es pot accedir-ne globalment.

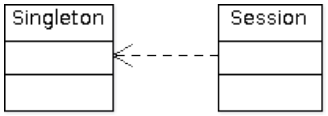


Figura 5.23: Diagrama UML del component Session

5.26 SOAP

Aquest projecte disposa d'una implementació completa per a proporcionar serveis web implementats mitjançant l'estàndard SOAP (*Simple Object Access Protocol*). Per una banda disposa dels components *Soap_Adapter* i *Soap_Server* que proporcionen les funcionalitats de servidor de serveis web SOAP mentre que per l'altra, la part del client la proporciona el component *Soap_Client*.

5.26.1 Part del client

El component que implementa part del client de SOAP, anomenat *Soap_Client* és un adaptador que utilitza la biblioteca de client SOAP nativa de PHP.

Aquest component és capaç de connectar-se a un servei web SOAP, obtindre una descripció d'ell - a través del corresponent document WSDL - , crear un proxy per a utilitzar el servei i exposar-lo a la API del marc de treball.

5.26.2 Part del servidor

La part del servidor de SOAP està implementada sobre la biblioteca *Pear::SOAP* degut als problemes que presenta l'API estàndard de SOAP de PHP i donat les facilitats que aquesta biblioteca proporciona per a generar el *document WSDL* automàticament a través de les classes exposades a SOAP.

El component *Soap_Server* adapta el servidor de SOAP de la citada biblioteca al marc de treball. Per fer que es comunique amb les funcionalitats del marc de treball desenvolupades en el sistema MVC i poder exposar els mètodes d'aquestes a serveis web SOAP, existeix el component *Soap_Adapter*, que s'encarrega d'analitzar cada mètode que es vol adaptar i proporcionar un representant - o proxy - per a que el servidor de SOAP es dirigeix-ca al mètode exposat a través de l'adaptador i així es puguen exposar a SOAP funcionalitats del marc de treball.

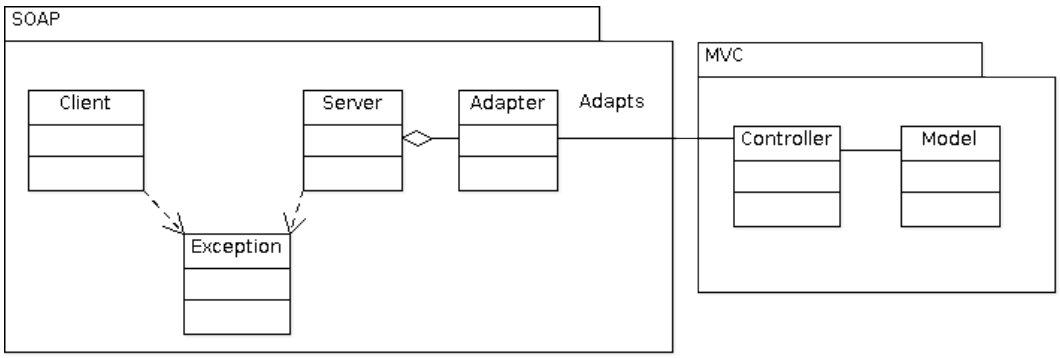


Figura 5.24: Diagrama UML del serveis web SOAP

5.27 Style

El component *Style* s'encarrega de gestionar els temes o estils CSS, carregar-los a petició de l'usuari o automàticament (incorporant les capçaleres a la vista corresponent) i proporcionar a l'usuari un llistat de temes per a que puga escollir el tema de la web

que més li convinga. Cada tema es defineix en un fitxer de configuració que indica per a cada mòdul i cada controlador de l'aplicació quins fitxers d'estils han de carregar-se. Aquesta càrrega és gestionada mitjançant la classe *Theme* que representa un tema - o agrupació d'estils - que conté tota la informació sobre un tema a més a més d'informació sobre els fitxers CSS corresponents al tema.

Style també implementa el patró de disseny *Singleton* fent-ho accessible des de qualsevol punt del marc de treball i el patró de disseny *Registry* amb el qual emmagatzema informació del temes d'una manera única i ben gestionada.

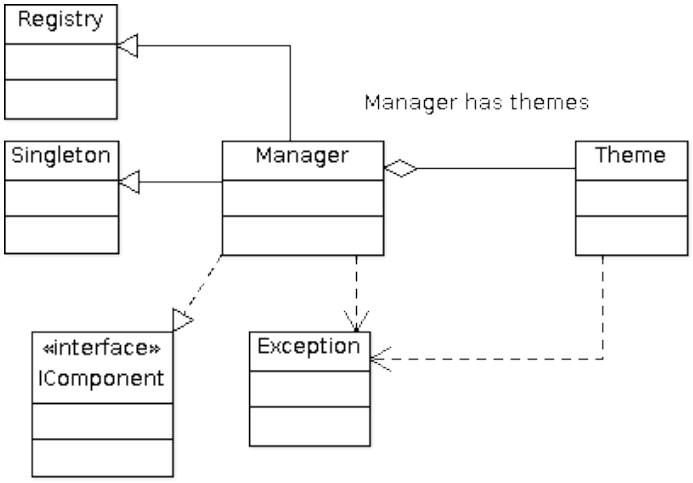


Figura 5.25: Diagrama UML del component *Style*

5.28 Widgets

El sistema de Widgets d'aquest projecte, permet utilitzar pedaços de codi escrits en llenguatge xhtml, javascript i css en les vistes corresponents a accions del sistema MVC. És molt útil per a webs que es basen en una aparença mestra i que necessiten components que es repeteixen una i una altra vegada en les diferents vistes. Exemples d'aplicació són els ≪breadcrumbs≫ o barra que indica al navegant en quin lloc es troba, els calendaris interactius de les barres de menú o qualsevol cosa que necessite ser inclosa en totes les pàgines web generades amb el marc de treball.

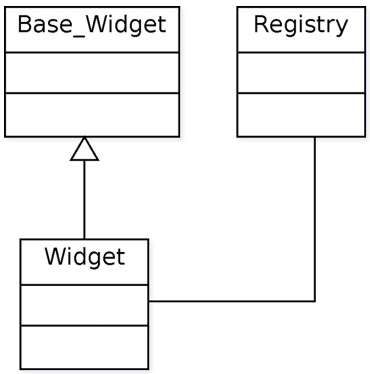


Figura 5.26: Diagrama UML del sistema de Widgets

Aquest component implementa el patró de disseny *Registry*, doncs l'utilitza per desar les opcions necessàries per l'execució de cada widget, i aquest per consultar-les.

Capítol 6

Patrons de disseny aplicats al projecte

6.1 Introducció

En aquest capítol es presenten els patrons de disseny aplicats a l'arquitectura i codi d'aquest projecte.

Un patró de disseny és una solució general a un problema comú i recurrent en el disseny de programari. Un patró de disseny no és un disseny acabat que es pot transformar directament en codi; és una descripció o plantilla per resoldre un problema que es pot utilitzar en moltes situacions diferents. Els patrons de disseny orientats a objectes normalment mostren relacions i interaccions entre classes o objectes, sense especificar les classes d'aplicació finals o objectes que hi estan implicats.

6.2 Singleton

El patró de disseny *Singleton* és un patró de disseny que es fa servir per a restringir la instanciació d'una classe a un objecte. Això és útil quan es necessita exactament un objecte per a coordinar accions a tot sistema. Molts components crítics del marc de treball implementen el patró *Singleton* fent que l'accés siga únic i global des de qualsevol part del sistema. En aquest projecte, el component *Singleton* manté un array d'instàncies d'objectes que implementen el patró de disseny Singleton identificant-se cada objecte amb el nom de la classe del que és instància. D'aquesta manera, estan emmagatzemades totes les instàncies dels components que ho implementen en un únic punt al que es té accés global des de tot el sistema.

6.3 Front Controller

El patró de disseny *Front Controller* és un patró de disseny que proporciona un únic punt d'entrada a l'aplicació web, ajuda a centralitzar la lògica de la petició web eliminant codi duplicat. Front Controller s'ajuda d'una sèrie de regles definides que proporcionen informació sobre la petició web i el codi o fitxers de codi que ha de carregar i executar. Té l'avantatge que, al ser l'únic punt d'entrada a l'aplicació web totes les

peticions passen per el mateix component evitant-se codi duplicat - el codi per gestionar les peticions web és únic - i a passar per diversos filtres (components *Filter* i *Filter Chain*) o comprovacions.

En aquest projecte el component que fa de Front Controller és el *Front Controller* que ajudat dels components *Router*, *Request* i dels *Filtres* és capaç d'obtindre la petició, analitzar el que es demana, cercar en el sistema d'enrutament els paràmetres del codi a executar, preparar les capçaleres HTTP de resposta a la petició web i executar un codi que resideix dins els mòduls de l'aplicació web.

6.4 Strategy

El patró de disseny Strategy (estratègia) permet desacoblar la implementació d'algorismes del client d'ells, permetent així escollir en temps d'execució el algorisme o estratègia a emprar a cada moment i per a cada necessitat.

El patró *Strategy* permet al *Front Controller* del sistema seleccionar en temps d'execució quin tipus d'objecte s'emprarà per despatxar la petició que ha arribat a ell. La forma de respondre a cada tipus de petició web està implementada en classes diferents que proveeixen la resposta d'una manera diferent i que a més a més, comparteixen alguns mètodes i propietats comuns amb una classe base.

6.5 Adapter/Wrapper

El patró de disseny *wrapper* o *adapter* és un patró de disseny que permet transformar la interfície d'una classe - l'adaptada - en una altra de manera que el client que fora incompatible amb la interfície de la classe adaptada, pogués utilitzar-la a través d'un objecte intermedi anomenat adaptador que té la interfície que espera el client i coneix la interfície de l'objecte adaptat permetent per delegació en l'objecte adaptat l'execució d'operacions.

En aquest projecte el patró de disseny *wrapper* s'utilitza per a exposar les accions del sistema MVC a serveis web. A partir d'una definició de les rutes d'un mòdul en els fitxers *routes.php* corresponents i mitjançant un objecte de la classe *FW_Soap_Adapter*,es pot generar un adaptador en temps d'execució que embolcalle la interfície de l'acció del controlador del mòdul, i l'expose al servidor SOAP. D'aquesta manera es pot generar el fitxer de descripció de serveis web WSDL automàticament en temps d'execució i que el servidor SOAP accepte peticions que després li seran re-dirigides al component *FW_Soap_Adapter* qui conté l'objecte del sistema MVC i és capaç d'encaminar la petició fins a l'acció corresponent, i tornar-li el resultat al servidor SOAP.

6.6 MVC: Model View Controller

El patró de disseny *Model View Controller* és una patró de disseny d'arquitectura que separa les dades d'una aplicació, la interfície d'usuari i la lògica de control en tres components diferents. La finalitat de MVC és desacoblar els models i les vistes, reduint la complexitat de l'arquitectura i facilitant les tasques de creació i manteniment de codi, donat que cada vista, cada controlador i cada model resideix en un fitxer diferent.

Model : El model accedeix a dades que poden estar situades en una base de dades o en fitxers. Obté o desa informació proporcionada o sol·licitada per el controlador.

Vista : La vista és la interfície d'usuari o les dades en formats XML, JSON o altres formats enviades a l'usuari en resposta a una petició a un controlador. El controlador obtindrà o crearà dades en el model, les processarà i les prepararà per a la vista o interfície d'usuari.

Controlador : El controlador és una classe que respon a events (en aquest projecte a peticions web), invoca al model per obtindre o desar dades que processarà segons les regles de negoci i se les passarà a la vista que les mostrarà a l'usuari.

6.6.1 Flux de funcionament del sistema MVC

1. L'usuari interactua amb l'aplicació web (exemple: pressiona un enllaç).
2. El navegador de l'usuari fa una petició web al punt d'entrada al marc de treball.
3. S'inicia el component *Front Controller* i invoca el *Router*
4. La petició web és processada en el *Router* fins trobar el codi a executar o bé no trobar-lo i retornar un error.
5. Es comprova la resposta del Router i els permisos que pot necessitar la petició.
6. Es carrega el controlador corresponent, s'instancia i es crida a l'acció corresponent.
7. El controlador obté/desa dades de/en el model, les processa i genera una vista.
8. L'usuari veu el resultat d'haver fet click en un enllaç.

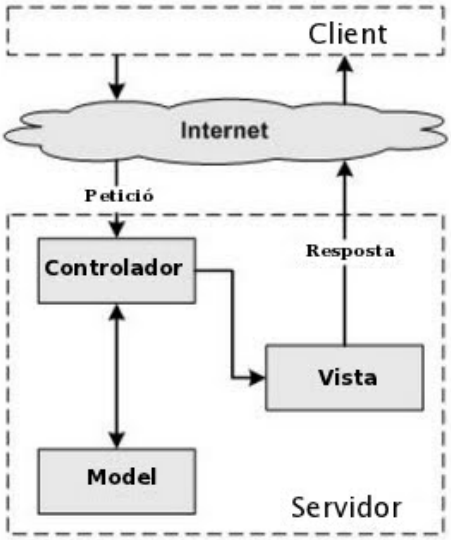


Figura 6.1: Patró de disseny MVC

6.7 HMVC: Hierarchical Model View Controller

El patró de disseny HMVC és similar al patró de disseny MVC comentat anteriorment, però HMVC es centra en la creació d'aplicacions web escal·lables en la que que les aplicacions es divideixen en conjunts de blocs MVC independents anomenats mòduls fent que l'aplicació web es puga distribuir en múltiples ubicacions permetent aplicacions web complexes, robustes i escal·lables.

En aquest projecte s'ha emprat una variació del patró de disseny HMVC junt amb el MVC permetent agrupar una sèrie de blocs MVC en una entitat anomenada *mòdul* que agrupa blocs MVC que tenen un sentit o pertanyen a un àrea de l'aplicació web. Exemple: El mòdul *Admin* integra els blocs MVC *usuaris*, *configuració d'aplicació web*, ...

Integrant tots els blocs MVC en un mòdul permetem que l'aplicació tinga unes funcionalitats clarament separables i diferenciables per àrees i que aquestes puguen formar part - o no - d'altres aplicacions web reutilitzant el codi gràcies a que cada mòdul i cada bloc MVC està aïllat de la resta de l'aplicació web i per tant és tant fàcil com copiar el mòdul complet a altra aplicació web i començar a funcionar. HMVC permet reutilitzar les diferents vistes i ¡l·layouts» per els components dels blocs MVC de cada mòdul. També es permet fer cridades a operacions en mòduls i controladors remots, encara que aquesta opció no s'aconsella donat l'acoblament entre mòduls que es produiria deixant de ser tant escal·lable l'aplicació web.

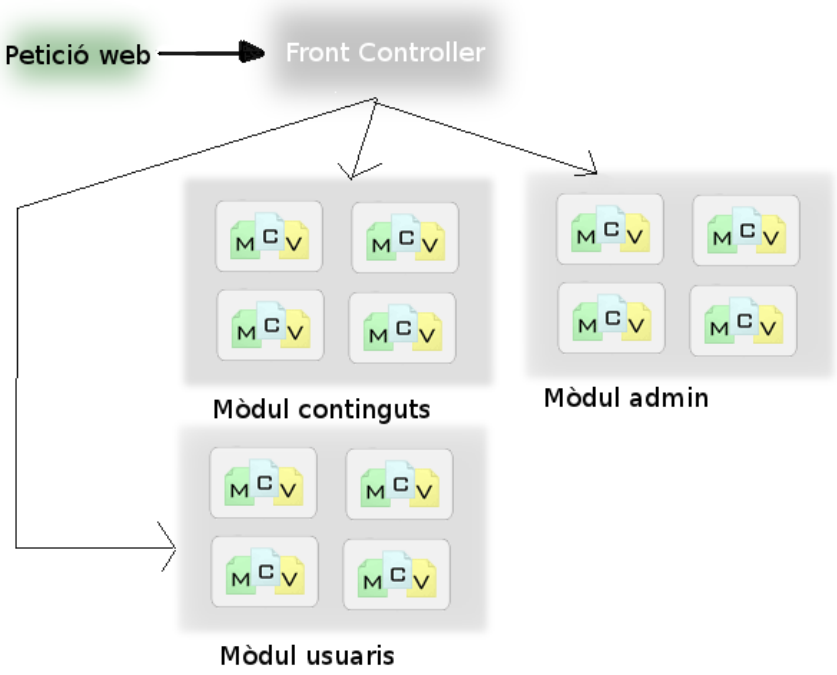


Figura 6.2: Patró de disseny HMVC

6.7.1 Avantatges de l'utilitzament del patró de disseny HMVC

Modularitat : Redueix les dependències entre els diferents components de l'aplicació.

Organització : Separar el treball en blocs de components MVC fa el treball més lleuger i estructurat.

Reutilització : Per la forma de modularitat de l'aplicació web fa més senzilla la reutilització de parts del codi font sense haver-les d'escriure de nou.

Extensibilitat : Fa l'aplicació més extensible sense sacrificar un manteniment senzill.

Distribució : L'aplicació funciona en base a mòduls que implementen el codi necessari per fer funcionar una part de l'aplicació web; copiant aquest mòdul - i configurant-lo - en altra aplicació web el mòdul funciona directament amb alguns petits ajustaments. Això fa que es puga tindre una aplicació web satèl·lit en altre sistema i que es comuniquen entre sí dos mòduls (utilitzant REST per exemple).

Mantenibilitat : Les funcionalitats al estar separades en mòduls i blocs MVC resulten més fàcils d'aïllar i de manipular/refactoritzar sense que afecte a altres parts de l'aplicació web.

6.8 Active Record

Active Record és un patró de disseny que facilita l'accés a dades en una base de dades. Active Record facilita el maneig i gestió d'objectes d'una base de dades relacional mitjançant la tècnica anomenada ORM (Object Relational Mapping), que associa una fila d'una taula d'una base de dades relacional en un objecte de manera que tinga els mateixos camps - propietats en l'objecte - i relacions amb altres taules.

Active Record proporciona al desenvolupador mètodes senzills que s'encarreguen d'escriure les consultes SQL necessàries per fer les operacions bàsiques conegudes com *CRUD* **C**reate , **R**ead, **U**pdate i **D**elete.

A més a més, dona suport a relacions entre taules mitjançant claus alienes que són transformades en relacions bidireccionals entre objectes, proporciona mètodes de validació dels tipus de dades (útils per a assegurar-se que els objectes continguen valors correctes i del tipus de dades especificat), callbacks (funcions que s'executen abans o després de cada una de les operacions), i mètodes personalitzades que pot escriure el desenvolupador.

Active Record ajuda al desenvolupador a no tindre que utilitzar SQL per obtindre les dades, a fer transparent i automàtic l'accés a dades d'una base de dades, a mantindre la persistència i integritat relacional entre les dades, i sobre tot, a evitar bona part dels errors de codi SQL que poden facilitar l'injecció de sentències SQL per trencar la base de dades.

El patró de disseny Active Record és un gran invent que està de moda en tots els marcs de treball actuals. Sense ell, i el patró de disseny MVC, les aplicacions web serien complicades de mantindre, de depurar, i sobre tot de desenvolupar.

6.9 Registry

El patró de disseny *Registry* és la solució a un problema que demana desar objectes o informació en un lloc del sistema per a després fer-ne us d'ells en una altra part. En altres paraules, és un indret on es poden desar tot tipus de valors en un contenidor únic de forma que cada valor és emmagatzemat en un *registre* en forma de **(clau,valor)**,

essent la clau la identificació de cada objecte. Un registre té mètodes per emmagatzemar informació, consultar si està emmagatzemada una informació, obtindre informació emmagatzemada i netejar l'informació emmagatzemada.

En aquest projecte s'ha emprat el patró *Registry* en forma d'una classe abstracta amb una col·lecció d'objectes. De forma que la classe client per a cada registre ha d'heretar de Registry i per herència podrà emprar aquest patró de disseny.

Aquest patró de disseny s'ha emprat en molts components com el registre de plugins el qual manté en memòria informació dels plugins disponibles en el marc de treball i sap obtindre informació de com instanciar-lo a l'hora d'utilitzar-lo o el contenidor de paràmetres *FW_Container_Parameter*. Aquest patró de disseny facilita el treball de guardar col·leccions de dades sense haver-se de preocupar de la forma en la que s'emmagatzemen.

6.10 Lazy Loading o «càrrega diferida»

Aquest patró dona solució al problema de la càrrega de biblioteques i components en memòria amb la càrrega diferida d'aquestos fins que no s'instancien o s'utilitzen. Això fa que els components no estiguen sempre carregats en memòria, que aquestos no es tinguen que carregar i inicialitzar-se sempre a cada petició i s'estalvia memòria i temps.

En aquest projecte hi ha certs components que són principals, estructurals i s'han de carregar a cada petició, la resta dels components o biblioteques es carreguen a demanda quan s'instancien. S'ha emprat l'ajuda de la biblioteca de PHP *SPL (Standard PHP Library)* amb els mètodes coneguts com *autoload* que són cridats per PHP al haver de carregar un component o biblioteca necessari per l'execució i a demanda. Hi ha diferents mètodes que s'encarreguen de carregar biblioteques i components del marc de treball, controladors del sistema MVC, models de l'usuari, ajudants (*helpers*) de vistes i classes d'usuari. Aquestos mètodes s'executen un darrere de l'altre fins que es troba el fitxer amb el codi font a carregar o es torna un error de fitxer no trobat.

A més a més, tots els components estan dissenyats de manera que no carreguen dependències o altres components fins que no es vagen a utilitzar expressament aquestos components.

Gràcies a la implementació d'aquest patró de disseny el marc de treball pot respondre ràpidament a peticions en temps de l'ordre dels 50-100 ms, ser eficient i estalviar memòria RAM.

6.11 Intercepting Filter

La majoria de les aplicacions tenen alguns requisits, com ara la seguretat i generació de missatges de depuració, que son per a totes les *peticions HTTP* que rep l'aplicació web. Per afegir aquesta funcionalitat per separat per a cada servei d'aplicació seria molt de temps, propens a errors i difícil de mantenir.

El patró *Intercepting Filter* dota a l'aplicació web dels recursos existents amb un filtre que intercepta cada petició HTTP i la va transformant fins formar una resposta. Un *filtre* pot processar, comprovar o redirigir les peticions d'aplicació. També pot postprocessar o reemplaçar el contingut de les respostes.

Els filtres es poden apilar un sobre l'altre per formar una cadena de filtres de tal manera que un filtre s'execute darrere d'un altre, fins que falle un d'aquestos filtres o es complete el procés amb èxit.

En aquest projecte s'empra aquest patró de disseny de forma conjunta amb el *Front Controller*, després que s'hagen obtingut dades correctes per part del *Router*. Es fa passar tota la petició HTTP per un seguit de filtres que comproven diferents coses i arriben a fallar o a generar un resultat. Resultat que si és satisfactori continuarà amb l'execució de l'acció en cadascuna de les especialitzacions del *Front Crontroller*.

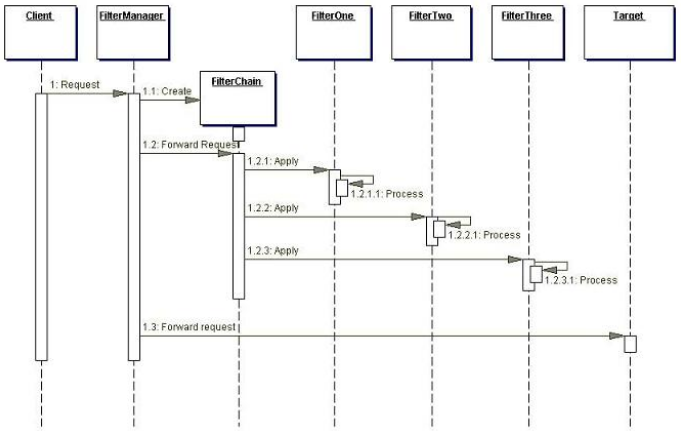


Figura 6.3: Patró de disseny *Intercepting Filter*

Capítol 7

Funcionament del sistema

7.1 Modes de funcionament

Aquest projecte té tres modes de funcionament principals

7.1.1 Aplicació web

Introducció

A continuació es detallaran tots els modes de funcionament per aplicacions web que disposa el marc de treball.

Aplicació web

Com a aplicació web, el marc de treball processa una petició que li demana que execute una acció que produeix codi html,javascript i css.

AJAX

En aquest mètode de funcionament, s'especifica al sistema que ha de tornar un document XML resultat de realitzar certa acció en el sistema. És útil per a aplicacions web 2.0 en les que es fa una petició AJAX XML al marc de treball i aquest ha de respondre en XML que serà processat en el client per actualitzar o generar algun component o el contingut d'una pàgina sense haver de recarregar aquesta pàgina.

Aquest mode de funcionament envia obligatòriament capçaleres http que indiquen que el contingut és un document XML. Per tant, si l'acció a executar no produeix un document XML, el navegador podria tindre problemes per a processar la resposta i no funcionaria correctament l'aplicació web.

JSON

En aquest mètode de funcionament, s'especifica al sistema que ha de tornar un document en format JSON resultat de realitzar certa acció en el sistema. Com amb el mètode de funcionament AJAX, aquest mètode també és útil per a actualitzar components d'una pàgina web i és un dels dos que més s'utilitza a nivell d'aplicacions web donada la facilitat de processament de JSON per part d'un navegador web donat que

JSON és part de JavaScript i tots els navegadors moderns saben interpretar-ho de manera única. Per tant és un bon format per obtindre dades en resposta a una petició al marc de treball; tenint com a principal avantatge el processament senzill i sense transformacions dins el navegador (XML necessita unes petites transformacions per poder obtindre el resultat).

Aquest mode de funcionament també envia capçaleres que http que indiquen que el contingut és un document JSON. Per tant, si l'acció no produeix un document JSON, el navegador podria tindre problemes per processar la resposta.

MIME

Amb aquest mode de funcionament es poden produir tot tipus de continguts - des de text pla a un document pdf passant per imatges o fitxers binaris - , cal especificar el tipus mime del contingut correctament i que l'acció a executar produeixca contingut d'eixe tipus mime. Utilitzant aquest mode de funcionament es pot fer que una petició AJAX (generada per una pàgina web en un navegador) torne text pla mitjançant el tipus mime *text/plain* podent tornar text pla sense estructurar al navegador (cosa que es desaconsella a menys que siga per un us molt senzill donat que no hi hauria estructura de dades que continga la informació i per tant podria ser més complicat de processar).

També es poden generar imatges (utilitzant la llibreria *GD* de PHP o similars) útil per generar *captchas* o gràfics estadístics o qualsevol imatge generada al vol en el marc de treball. En tot cas, s'ha d'especificar el tipus mime exacte de la imatge generada (*image/png*, *image/jpeg*, ...).

Emprant biblioteques externes en forma de plugins o biblioteques es podrien també generar documents PDF, en format OpenOffice.org, o formats de Microsoft Office e inclòs generar fitxers binaris/programes al vol (opció força complicada).

Plugins

Aquest darrer mode de funcionament dels d'aplicacions web permet que s'execute una funcionalitat situada dins un plugin i que retorne resultats al navegador.

7.1.2 Serveis WEB

Introducció

A continuació es detallaran tots els modes de funcionament que el marc de treball té per a serveis web.

SOAP

Aquest mode de funcionament permet que el marc de treball actue com a servidor de serveis web mitjançant el protocol SOAP i a través del transport HTTP. Amb el servidor SOAP es poden atendre peticions a serveis web implementats dins els marc de treball i proporcionar d'aquesta manera serveis web i interoperabilitat entre aplicacions, que escrites en qualsevol llenguatge de programació podrien obtindre i enviar dades des de/al marc de treball.

REST

Aquest mode de funcionament permet al marc de treball implementar serveis web amb arquitectura REST admetent peticions HTTP amb algun dels mètodes que especifica l'arquitectura REST (*GET*,*POST*,*PUT*,*DELETE*). Per a serveis web REST es poden implementar funcionalitats dins el marc de treball que obligatòriament han de tornar dades en forma d'array per a després transformar-les a formats JSON o XML. Amb un servei web REST es poden implementar projectes amb intercanvi d'informació entre aplicacions - especialment aplicacions en entorns mòbils - .

7.1.3 Tasques programades/sistema operatiu

Introducció

A continuació es detallaran els mètodes de funcionament aprofitables per a facilitar la programació de tasques en un sistema operatiu (generalment GNU-Linux/Unix).

CRON

Es poden programar tasques periòdiques per a executar-les amb el programador del sistema operatiu que executen accions dins el marc de treball (exemples: enviar newsletter automàticament, rebre missatges, generar inventaris/factures, ...). Per això, el marc de treball disposa d'accions de tipus CRON i dues maneres d'accedir a elles:

- Línia d'ordres: Es poden executar accions CRON mitjançant el punt d'entrada *cron.php* i arguments de la línia d'ordres que serveixen per indicar el nom de l'acció (únic per a tot el sistema) i els paràmetres que aquesta necessite per funcionar. Aquesta és indicada si es té accés al planificador de tasques del sistema operatiu en el que està el servidor.
- Petició HTTP: Es pot dirigir una petició HTTP al punt d'entrada *cron.php* i pasarli la URL amb els paràmetres necessaris. Això és útil si no es té accés directe al planificador de tasques del sistema operatiu del servidor, però l'allotjament web/hosting sí que permet planificar tasques a través de peticions web a scripts, o inclòs si volem fer una petició remota des d'un sistema remot.

Més informació de tasques programades en el capítol corresponent a tasques programades en la part de desenvolupament avançat.

Part III

El model de dades

Capítol 8

Introducció

8.1 Introducció

En aquesta part aprendrem una de les parts més importants de tota aplicació web: **el model de dades**. Aprendrem a emprar el patró de dades *Active Record* per fer més senzill, eficient i llegible el nostre model de dades i es mostrarà l'API de la base de dades utilitzada per accedir alternativament a la base de dades o per recuperar dades de consultes complicades.

Capítol 9

La base de dades

9.1 Introducció

En aquest capítol aprendrem a configurar i utilitzar la base de dades. A connectar-nos, recuperar dades, desar dades, obtindre informació sobre consultes i utilitzar l'API de la base de dades.

El component *FW_Database* proporciona una API completa per connectar a la base de dades i realitzar qualsevol operació necessària. [1]

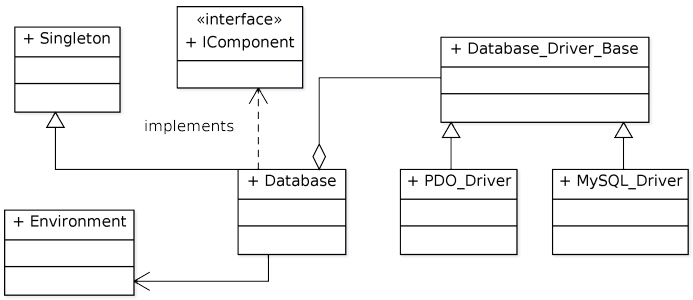


Figura 9.1: Diagrama UML del component Database

Com es mostra en el anterior diagrama UML, el component *Database* consta d'una classe central anomenada **Database** que junt a l'ajuda dels components *Environment* i *Config* és configurada. Per poder proveir suport a cadascun dels tipus de *SGBDs* comercials o dels nous sistemes *No-SQL* Database té una arquitectura flexible en la que cada ≪driver≫ (classe que dona suport a un motor de base de dades) implementa les seves funcionalitats per separat. No obstant, totes aquestes classes ≪driver≫ hereten funcionalitat i interfície sempre de la classe *FW_Database_Driver*, que proporciona uns mètodes abstractes que s'han de concretar en les clases ≪drivers≫.

[1] A excepció de la creació i execució de funcions personalitzades, de *triggers* (que hauran d'estar implementats per programari) i d'altres opcions específiques per a cada base de dades

Base de dades	Driver	Observacions
MySQL	mysql/mysqli	Implementació amb l'api nativa de PHP
SQLite	sqlite	Implementació amb l'api nativa de PHP
SQLite/SQLite3	Suport via PDO	Consultar funcionalitats en la documentació de PDO
PostgreSQL	postgresql	Implementació amb l'api nativa de PHP
Oracle	Suport via PDO	Consultar funcionalitats en la documentació de PDO
Microsoft SQL Server	Suport via PDO	Consultar funcionalitats en la documentació de PDO
Altres	Suport via PDO	Consultar funcionalitats en la documentació de PDO

Figura 9.2: Bases de dades suportades per el marc de treball

Nota: *PDO* és l'abreviatura de **P**hp **D**ata **O**bjects . Més informació sobre PDO en `http://www.php.net/manual/en/book.pdo.php` .
Actualment no estan programats els drivers per a suport de bases de dades de tipus *NoSQL*, que es programaran en futures versions i segons evolucionen les APIs de PHP per a aquestos tipus de bases de dades.

9.2 Configuració de la base de dades

Aquesta secció ens mostrarà com generar la configuració de la base de dades i configurar els drivers i la connexió a utilitzar.
La configuració de la base de dades és una configuració *global* i es fa en el fitxer `/framework/config/database.php`. Aquest fitxer té l'estructura de qualsevol fitxer de configuració estàndard del marc de treball, en el que es distingeix un array amb dos arrays com a valors de les claus *global* i *sections*.
És en el array amb la clau *sections* en el que crearem altres arrays que tinguen de clau el nom que vulguem donar-li a la connexió (ex: sistema, database, proves, ...) i dins d'aquest, una sèrie de paràmetres necessaris per establir la connexió. Aquest fitxer té el següent aspecte:

Listing 9.1: Configuració de les connexions de la base de dades en *database.php*

```
1  <?php
2  $config = array(
3      "sections" => array (
4          "default"   => array (
5            "host"     => "localhost",
6            "database" => "blog",
7            "prefix"   => "blog_",
8            "username" => "blog",
9            "password" => "blog",
10           "driver"   => "pdo",
11           "dsn"      => "mysql:host=localhost;dbname=blog"
12         )
13     ),
14     "global"  => array (
15         "activeRecord" => array (
16           "builder" => "SQL",
17           "builderOptions" => array (
18             "databaseConnection" => "default"
```

```
19                )
20             )
21          )
22    );
23    FW_Config::createConfig("database");
24    FW_Config::setConfig("database",$config);
25    ?>
```

Les opcions mostrades en la clau *global* s'explicaran en el capítol corresponent a la configuració del component *Active Record*.

Els paràmetres per a cada connexió a la base de dades són els següents:

Paràmetre	Tipus	Descripció
driver	cadena de caràcters	Driver de base de dades a utilitzar amb la connexió
prefix	cadena de caràcters	Prefix utilitzat en les taules de la base de dades
username	cadena de caràcters	Usuari de la base de dades
password	cadena de caràcters	Contrasenya de la base de dades
host	cadena de caràcters	*Host* del servidor de bases de dades
dsn	cadena de caràcters	Paràmetre utilitzat en la connexió mitjançant alguns drivers

Figura 9.3: Paràmetres de la configuració de connexions a bases de dades

També haurem de modificar el fitxer `/framework/config/environment.php` per indicar-li al component *Environment* la base de dades a utilitzar. [2] En cada entorn indicarem quines connexions de bases de dades que hem configurat en el fitxer `database.php` anteriorment volem utilitzar amb l'entorn.
Aquesta configuració es farà en forma d'array amb els noms de les connexions únicament.

Listing 9.2: Configuració de les connexions de la base de dades en cada entorn en el fitxer *environment.php*

```
1   <?php
2   $config = array(
3     "sections" => array (
4       "develop"    => array (
5             ...
6             "database" => array("default"),
7             ...
8        ),
9        "production"    => array (
10            ...
11            "database" => array("hosting"),
12            ...
13         ),
14    ),
15    "global"     => array (
16      "environmentInUse" => "develop"
17    )
```

[2] Podem utilitzar una o moltes bases de dades per a la nostra aplicació web, encara que només es recomana una per a cada entorn de desenvolupament per evitar inconsistències o fallades d'integritat amb les dades

```
18  );
19
20  FW_Config::createConfig("environment");
21  FW_Config::setConfig("environment",$config);
22  ?>
```

Amb tota aquesta configuració tindrem configurada i llesta la base de dades per a ser utilitzada.

9.3 API del component *Database*

9.3.1 Introducció

A continuació es pressenta la API per utilitzar la base de dades. Aquesta API permet fer les coses més elementals, afegint-li una gestió de les connexions configurades i un suport per obtindre dades de les taules de la base de dades.

9.3.2 Operacions sobre connexions

Amb aquesta operació podrem utilitzar una connexió a base de dades que hagem definit en el fitxer de configuració `database.php` i afegit a les connexions per a l'entorn a utilitzar en `environment.php`.

Listing 9.3: API de Database: Operacions sobre connexions

```
1   <?php
2   // utilitzar una connexio definida en database.php
3   void useConnection($name);
4
5   // obtindre el prefix de la connexio en us
6   string getPrefix(void);
7
8   // obtindre el nom de la configuracio de la connexio en us
9   string getConnectionInUse(void);
10
11  // connecta a la base de dades
12  bool connect(void);
13
14  // desconnecta de la base de dades
15  bool disconnect(void);
16  ?>
```

9.3.3 Operacions de consulta

A continuació s'exposa l'API per a totes les operacions de consulta bàsiques que han d'implementar tots els drivers de bases de dades.

Listing 9.4: API de Database: Operacions de consulta

```
1   <?php
2   // consultar la base de dades
3   mixed query($query);
```

```
4
5   // nombre de files afectades (operacions UPDATE, INSERT o
        ALTER)
6   int affectedRows(void);
7
8   // nombre de files retornades en el resultat d'una consulta
        SELECT
9   int numRows(void);
10
11  // obtindre les dades del resultat d'una consulta SELECT en
        forma d'array associatiu
12  array fetchArray(void);
13  array fetchAssoc(void);
14
15  // obtindre les dades del resultat d'una consulta SELECT en
        forma d'array
16  array fetchRow(void);
17
18  // obtindre les dades del resultat d'una consulta SELECT en
        forma d'objecte de PHP de la classe stdClass
19  array fetchObject(void);
20
21  // obte si existeix una taula en la base de dades
22  bool existsTable($name);
23
24  // inicia una transaccio
25  void begin(void);
26
27  // cancela una transaccio en marxa
28  void rollback(void);
29
30  // desa els canvis d'una transaccio en marxa
31  void commit(void);
32
33  // obte la clau primaria de la ultima fila inserida en una
        operacio INSERT
34  mixed getLastInsertedId(void);
35  ?>
```

9.3.4 Altres operacions

Per finalitzar, a continuació es presenten un parell d'operacions útils que ens poden donar informació sobre les columnes d'una taula o informació sobre el driver en us (dades de que proporciona generalment el motor de base de dades).

Listing 9.5: API de Database: Altres operacions

```
1   <?php
2   // obtindre informacio sobre el driver en us
3   mixed getInfo(void);
4
5   // obtindre informacio sobre les columnes d'una taula de la
        base de dades
```

```
6   array getTableFields($tableName);
7   ?>
```

9.4 Exemple complet

En aquest exemple ens connectarem a la base de dades, farem una operació de consulta (SELECT) i obtindrem un resultat el qual mostrarem:

Listing 9.6: Exemple de funcionament del component *Database*

```
1   <?php
2
3   // obtindre la instancia de la base de dades
4   $database = FW_Database::getInstance();
5
6   // executar una consulta contra la base de dades
7   $database->query("SELECT isbn,titol FROM proves_llibres
        WHERE autor LIKE 'A%'");
8
9   // si el resultat del query anterior te files ...
10  if ($database->numRows()) {
11    // mentre hi hagen files en el resultat ...
12    while ($result = $database->fetchAssoc()) {
13      print "ISBN={$result["isbn"]} / Titol={$result["titol"]}
            ";
14      print "<br/>";
15    }
16  }
17
18  ?>
```

Capítol 10

Active Record

10.1 Introducció

Active Record proporciona una manera neta, ràpida i senzilla d'obtindre les dades d'una base de dades en forma d'objectes mitjançant la *implementació d'una versió del patró de disseny ActiveRecord* original de Ruby On Rails - utilitzat amb molt d'èxit en Ruby On Rails - i descrit per *Martin Fowler* en *Patterns of Enterprise Application Architecture*. Abstrau al desenvolupador de gestionar directament SQL i redueix les possibilitats d'un atac mitjançant SQL-Injection.

A més a més, al transformar les dades de la base de dades relacionals en objectes, el desenvolupador es beneficia de tot un món d'avantatges que permet el disseny de sistemes orientat a objectes ja que cada objecte pot encapsular lògica i regles de negoci segons les necessitats del client i a la vegada tindre una forma senzilla i elegant de poder guardar eixos canvis en l'objecte i els objectes relacionats en una base de dades relacional sense molts esforços per al desenvolupador.

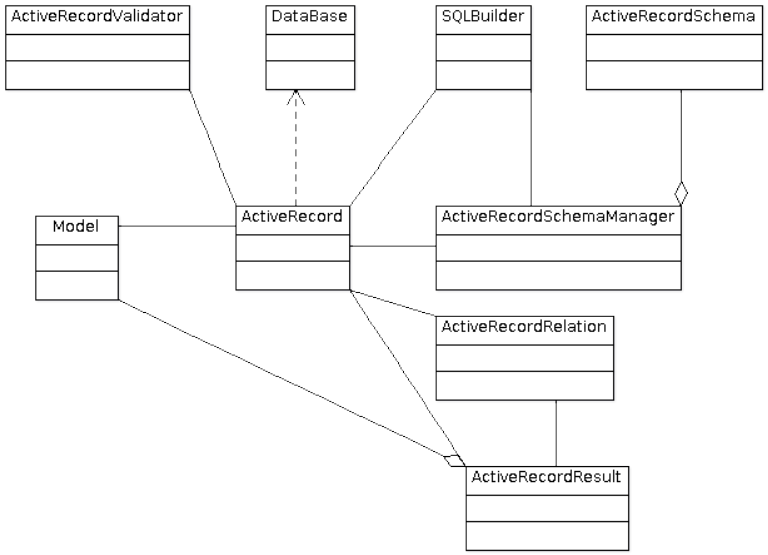


Figura 10.1: Diagrama UML del sistema ActiveRecord

10.2 El model

10.2.1 Introducció

Active Record basa el seu funcionament en objectes de tipus *Model* que representen una fila d'una taula de la base de dades relacional amb totes les seves columnes (propietats en el model) i claus alienes (relacions en el model).

Aquestos objectes són classes de PHP derivades de la classe *FW_ActiveRecord_Model* que proporciona tota la funcionalitat d'Active Record deixant-la llesta per operar amb només heretar la funcionalitat de la mencionada classe.

10.2.2 De base de dades relacional a model Active Record

Generar models d'Active Record és un procés realment senzill basat en una sèrie de convencions i sense haver-se de configurar.
Les *convencions* a seguir són les següents:

- Nom de la taula en minúscules, separant paraules amb guions baixos « _ » i sense cap espai.
  S'ha de fer així per maximitzar la compatibilitat amb tots els motors de bases de dades i tots els sistemes operatius.
  Exemples vàlids: « user_has_role » , « book_storage » , ...

- Nom del model (classe) idèntic al nom de la taula, el nom del fitxer ha de ser el nom de la taula però afegint-li « .class.php » i ha d'estar en el directori

`/app/lib/models`.
Exemples vàlids: « book_storage » → « book_storage.class.php »
« usuari » → « usuari.php »

- Totes les propietats (columnes en la taula del model relacional) del model seguiran les mateixes convencions que per els noms de les taules (mateixos motius) i tindran el modificador d'accés **protected** per a que Active Record, Model i altres classes puguen fer-ne us d'aquestes propietats.

- Si s'utilitza un prefix per a les taules de la base de dades, aquest prefix haurà d'estar inclòs en el nom de les taules, però no en el nom dels models. Active Record afegirà el prefix automàticament a les consultes SQL que genere per el model. Aquest prefix s'ha de configurar en la configuració de la base de dades corresponent (consulteu la secció del model de dades, component *Database*, configuració de la base de dades).

Una vegada presentades les convencions utilitzades ja podem construir models per utilitzar amb Active Record.
Presentem a continuació un exemple de com de fàcil és transformar una taula de la base de dades a un model d'Active Record:

La taula de la base de dades:

Listing 10.1: Exemple de taula de base de dades a modelar amb Active Record

```
1  CREATE TABLE post (
2    id INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT,
3    title VARCHAR(512) NOT NULL,
4    content TEXT NOT NULL,
5    created_at DATETIME NOT NULL,
6    author VARCHAR(255) NOT NULL,
7    permalink VARCHAR(255) NOT NULL UNIQUE,
8    status INTEGER(1) NOT NULL DEFAULT 0
9  );
```

Es convertiria en:

Listing 10.2: Model d'Active Record

```
1  <?php
2  class post extends FW_ActiveRecord_Model {
3
4    protected $id;
5    protected $title;
6    protected $content;
7    protected $created_at;
8    protected $author;
9    protected $permalink;
10   protected $status;
11
12 };
13 ?>
```

Aquestos models han d'estar obligatòriament en els següents directoris:

Models generals per a l'aplicació web :
Directori /app/lib/models .

Models generals per a aplicacions web internes del sistema :
Directori /framework/app/models .

Models a utilitzar amb un plugin :
Directori /app/lib/plugins/NOM_DEL_PLUGIN/models .

En les següents seccions es s'explicarà com fer relacions, definir restriccions, validacions, callbacks i funcionalitat d'usuari en els models creats amb Active Record.

10.2.3 Relacions

Una relació entre *Models* d'Active Record permet que dos models estiguen ≪relacionats≫ permetent emular el sistema de *claus alienes* del model relacional.
Així es dota al sistema Active Record de capacitat per a mantindre relacionats dos models que compartisquen dades o siguen composició d'altre model més gran.
Malgrat el sentit *unidireccional* de molts tipus de relacions entre taules del model relacional de bases de dades, Active Record proporciona un sentit *bidireccional*.
És a dir, que des d'un model es pot navegar cap a les classes relacionades i que des de les classes relacionades es pot tornar cap al model amb qui es relaciona.

Existeixen set tipus de relacions entre models que es detallaran a continuació, explicant-ne l'equivalència amb el model relacional i com definir-les en els nostres models.
Totes les relacions cal que estiguen definides en cada model. Per això s'utilitzen *arrays* amb el modificador *static* (per a no interferir en les propietats de cada model) i dins d'aquestos arrays es pot definir 1 ó moltes relacions, segons convinga.

Relació ≪has_one≫ (1:1)

Aquesta relació permet lligar un model a altre amb els mateixos valors d'una clau aliena que els relaciona. En aquest cas, cada instància del model ≪A≫ només podrà relacionar-se com a màxim amb una instància del model ≪B≫.
Aquestes relacions 1:1 no són gaire freqüents, donat que la majoria de models solen relacionar-se amb molts models. És més freqüent la relació 1:1 del tipus *belongs_to*.

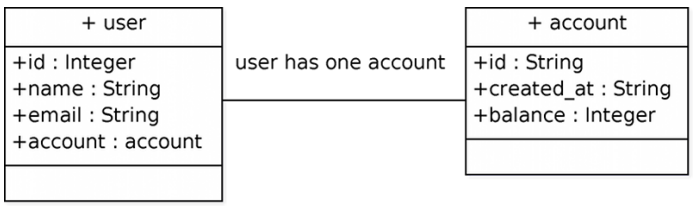


Figura 10.2: Active Record,Relació *has_one*

Listing 10.3: Active Record, Relacions: SQL ≪has_one≫

```sql
-- taula user
CREATE TABLE user (
  id INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT,
  name VARCHAR(128) NOT NULL,
  email VARCHAR(128) NOT NULL,
  account INTEGER(5) NOT NULL,
  FOREIGN KEY(account) REFERENCES account(id) ON DELETE
      CASCADE ON UPDATE CASCADE
);

-- taula account
CREATE TABLE account (
  id INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT,
  created_at DATATIME,
  balance DOUBLE
);
```

Listing 10.4: Active Record, Relacions: Model que implementa relació «has_one»

```php
<?php

// user.class.php
class user extends FW_ActiveRecord_Model {
  protected $id;
  protected $name;
  protected $email;
  protected $account;

  public static $has_one = array (
    array(
      "property" => "account",
      "table"    => "account",
      "srcColumn"=> "account",
      "dstColumn"=> "id",
      "update"   => "cascade",
      "delete"   => "cascade"
    )
  );

};


// account.class.php
class account extends FW_ActiveRecord_Model {
  protected $id;
  protected $created_at;
  protected $balance;
};

?>
```

Relació ≪belongs_to≫ (1:1)

Aquest tipus de relació és idèntic a l'anterior però amb la diferència que el model posseïdor de la clau aliena és el model ≪B≫ que es relaciona amb un model ≪A≫.
Aquest tipus de relacions es pot emprar per indicar relacions de propietat (objecte pertany a objecte), exemples:

- Un post *pertany* a un autor com a màxim.
- Una factura *és* d'un proveïdor com a màxim.
- Una comanda *pertany* a un client com a màxim.

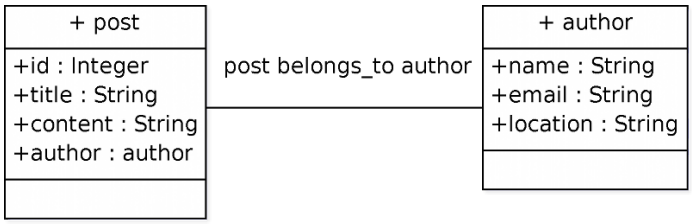


Figura 10.3: Active Record,Relació *belongs_to*

Listing 10.5: Active Record, Relacions: SQL ≪belongs_to≫

```
1  -- taula post
2  CREATE TABLE post (
3    id INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT,
4    title VARCHAR(255),
5    content TEXT,
6    author VARCHAR(100) NOT NULL,
7    FOREIGN KEY(author) REFERENCES author(name) ON UPDATE
         CASCADE ON DELETE CASCADE
8  );
9
10
11 -- taula author
12 CREATE TABLE author (
13   name VARCHAR(100) PRIMARY KEY NOT NULL,
14   email VARCHAR(128) NOT NULL,
15   location VARCHAR(255) NOT NULL
16 );
```

Listing 10.6: Active Record, Relacions: Model que implementa relació ≪belongs_to≫

```
1  <?php
2
3  // post.class.php
4  class post extends FW_ActiveRecord_Model {
5    protected $id;
6    protected $title;
7    protected $content;
```

```
8     protected $author;
9
10    public static $belongs_to = array (
11      array(
12        "property" => "author",
13        "table"    => "author",
14        "srcColumn"=> "author",
15        "dstColumn"=> "name",
16        "update"   => "cascade",
17        "delete"   => "cascade"
18      )
19    );
20
21  };
22
23
24  // author.class.php
25  class author extends FW_ActiveRecord_Model {
26    protected $name;
27    protected $email;
28    protected $location;
29  };
30
31  ?>
```

Relació ≪has_many≫ (1:N)

Aquest tipus de relació permet a una instància del model ≪A≫ relacionar-se amb moltes instàncies del model ≪B≫.
És un dels tipus de relacions més utilitzades donada la seva facilitat per a modelar-les.

- Un client *té* moltes comandes.
- Un post *té* molts comentaris.
- En un poble *viuen (té)* molts habitants.

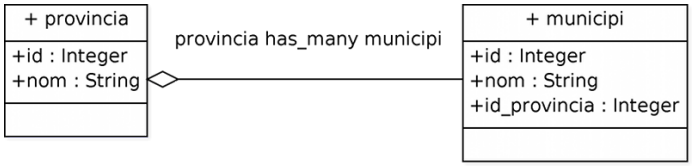


Figura 10.4: Active Record,Relació *has_many*

Listing 10.7: Active Record, Relacions: SQL ≪has_many≫

```
1   -- taula provincia
2   CREATE TABLE provincia (
3     id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
4     nom VARCHAR(50) NOT NULL UNIQUE
5   );
```

```
6
7   -- taula municipi
8   CREATE TABLE municipi (
9     id INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT,
10    nom VARCHAR(80) NOT NULL,
11    id_provincia INTEGER NOT NULL,
12    FOREIGN KEY(id_provincia) REFERENCES provincia(id) ON
          UPDATE CASCADE ON DELETE CASCADE
13  );
14
15  ?>
```

Listing 10.8: Active Record, Relacions: Model que implementa relació ≪has_many≫

```
1   <?php
2
3   // provincia.class.php
4   class provincia extends FW_ActiveRecord_Model {
5     protected $id;
6     protected $nom;
7
8     public static $has_many = array (
9       array(
10        "property" => "municipis",
11        "table"    => "municipi",
12        "srcColumn"=> "id",
13        "dstColumn"=> "id_provincia",
14        "update"   => "cascade",
15        "delete"   => "cascade"
16      )
17    );
18
19  };
20
21
22  // municipi.class.php
23  class municipi extends FW_ActiveRecord_Model {
24    protected $id;
25    protected $nom;
26    protected $id_provincia;
27  };
28
29  ?>
```

Relació ≪has_and_belongs_to_many≫ (N:M)

Aquest tipus de relació permet que una instància del model ≪A≫ estiga relacionada amb moltes instàncies del model ≪B≫ i que a la vegada, cada instància del model ≪B≫ estiga relacionada amb moltes instàncies del model ≪A≫.
Aquest tipus de relació necessita d'una taula intermèdia que relacione instàncies d'ambdós models i que només conté informació relativa a la clau aliena del model ≪A≫ i la clau aliena del model ≪B≫.

Exemples d'aquest tipus de relació serien:

- Una persona col·labora en molts grups de treball i cada grup de treball té molts col·laboradors.
- Un metge visita molts pacients i cada pacient és visitat per molts metges.

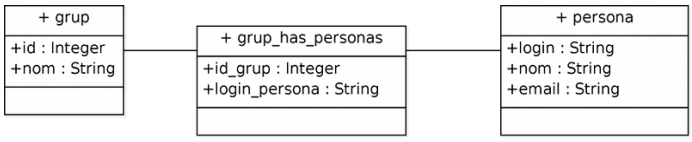


Figura 10.5: Active Record,Relació *has_and_belongs_to_many*

Listing 10.9: Active Record, Relacions: SQL «has_and_belongs_to_many»

```
1  -- taula grup
2  CREATE TABLE grup (
3    id INTEGER PRIMARY KEY NOT NULL AUTO_INCREMENT,
4    nom VARCHAR(255) NOT NULL
5  );
6
7  -- taula grup_has_personas
8  CREATE TABLE grup_has_personas (
9    id_grup INTEGER NOT NULL,
10   login_persona VARCHAR(50) NOT NULL,
11   PRIMARY KEY(id_grup,login_persona),
12   FOREIGN KEY(id_grup) REFERENCES grup(id) ON UPDATE
       RESTRICT ON DELETE RESTRICT,
13   FOREIGN KEY(login_persona) REFERENCES persona(login) ON
       UPDATE RESTRICT ON DELETE RESTRICT
14 );
15
16 -- taula persona
17 CREATE TABLE pesona (
18   login VARCHAR(50) NOT NULL PRIMARY KEY,
19   nom VARCHAR(100) NOT NULL,
20   email VARCHAR(100) NOT NULL
21 );
```

Listing 10.10: Active Record, Relacions: Model que implementa relació «has_and_belongs_to_many»

```
1  <?php
2
3  // grup.class.php
4  class grup extends FW_ActiveRecord_Model {
5    protected $id;
6    protected $nom;
7
8   public static $has_and_belongs_to_many = array (
9      array(
```

```
10          "property"  => "persones",
11          "srcTable"  => "grup",
12          "srcColumn" => "id",
13          "dstTable"  => "persona",
14          "dstColumn" => "login",
15          "throughTable" => "grup_has_personas",
16          "throughTableSrcColumn" => "id_grup",
17          "throughTableDstColumn" => "login_persona",
18          "update" => "restrict",
19          "delete" => "restrict"
20        )
21      );
22    };
23
24
25    // persona.class.php
26    class persona extends FW_ActiveRecord_Model {
27      protected $login;
28      protected $nom;
29      protected $email;
30    };
31
32    ?>
```

En aquest cas no cal representar la taula intermèdia amb una classe de PHP, basta amb estar en la base de dades ja que no recuperarem res d'informació d'ella i per tant no ha d'estar representada en el nostre model en PHP.

La taula ≪A≫ es correspon amb el model ≪A≫, la ≪B≫ amb el model ≪B≫ i la taula ≪T≫ és la taula intermèdia que permet representar les relacions N:M entre dos models, que està formada per les columnes de procedència de A i destí de B sent ambdues columnes la clau primària d'aquesta taula i a la vegada clau alienes a les respectives taules A i B.

Relació ≪has_one_through≫ (1:1)

Aquest tipus de relació és idèntic a la relació 1:1, només que empra un tercer model per relacionar una instància del model de destí. És a dir, que una instància de ≪A≫ es relaciona com a màxim amb una instància del model ≪B≫ a través de com a màxim una instància d'un model ≪C≫.
Un exemple d'aquest tipus de relació seria el següent: ≪Un usuari *està subscrit* a una revista≫ recuperant informació sobre la subscripció.
Aquest tipus de relació és útil quan volem obtindre dades d'un model a través del qual es fa la relació.

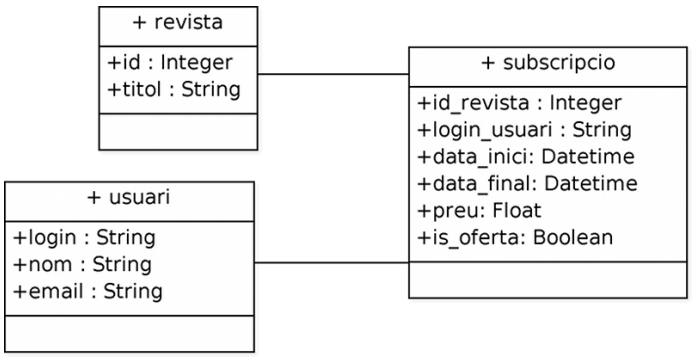


Figura 10.6: Active Record,Relació *has_one_through*

Listing 10.11: Active Record, Relacions: SQL «has_one_through»

```
1  -- taula usuari
2  CREATE TABLE usuari (
3    login VARCHAR(50) PRIMARY KEY NOT NULL,
4    nom VARCHAR(100) NOT NULL,
5    email VARCHAR(100) NOT NULL
6  );
7
8  -- taula revista
9  CREATE TABLE revista (
10   id INTEGER PRIMARY KEY NOT NULL,
11   titol VARCHAR(100) NOT NULL UNIQUE
12 );
13
14 -- taula suscripcio
15 CREATE TABLE suscripcio (
16   id_revista INTEGER NOT NULL,
17   login_usuari VARCHAR(50) NOT NULL,
18   data_inici DATETIME NOT NULL,
19   data_final DATETIME NOT NULL,
20   preu DOUBLE NOT NULL DEFAULT 0.0,
21   is_oferta INTEGER(1) NOT NULL,
22   FOREIGN KEY(id_revista) REFERENCES revista(id) ON UPDATE
         RESTRICT ON DELETE RESTRICT,
23   FOREIGN KEY(login_usuari) REFERENCES usuari(login) ON
         UPDATE RESTRICT ON DELETE RESTRICT
24 );
```

Listing 10.12: Active Record, Relacions: Model que implementa relació «has_one_through»

```
1  <?php
2
3  // usuari.class.php
4  class usuari extends FW_ActiveRecord_Model {
5    protected $login;
6    protected $nom;
```

```
7     protected $email;
8
9     public static $has_one_through = array (
10      array(
11        "property"  => "revista",
12        "srcTable"  => "usuari",
13        "srcColumn" => "login",
14        "dstTable"  => "revista",
15        "dstColumn" => "id",
16        "throughTable" => "subscripcio",
17        "throughTableSrcColumn" => "login_usuari",
18        "throughTableDstColumn" => "id_revista",
19        "update" => "restrict",
20        "delete" => "restrict"
21      )
22    );
23
24  };
25
26  // revista.class.php
27  class revista extends FW_ActiveRecord_Model {
28    protected $id;
29    protected $titol;
30  };
31
32
33  // suscripcio.class.php
34  class suscripcio extends FW_ActiveRecord_Model {
35    protected $id_revista;
36    protected $login_usuari;
37    protected $data_inici;
38    protected $data_final;
39    protected $preu;
40    protected $is_oferta;
41  };
42
43  ?>
```

Ací hem representat la taula intermèdia en el model de dades en PHP perquè anem a utilitzar-la obtenint-ne informació d'ella.

Relació ≪has_many_through≫ (1:N)

Aquest tipus de relació és idèntic a la relació 1:N, només que empra un tercer model per relacionar una instància del model destí. És a dir, que una instància de ≪A≫ es relaciona amb moltes instàncies de ≪B≫ a través de com a màxim una instància d'un model ≪C≫.
Un exemple d'aquest tipus de relació seria el següent: ≪En una província *viuen* molts usuaris ≫. En aquest cas també volem saber la informació del municipi en el que viu.
Aquest tipus de relació és útil quan volem obtindre dades d'un model a través del qual es fa la relació.

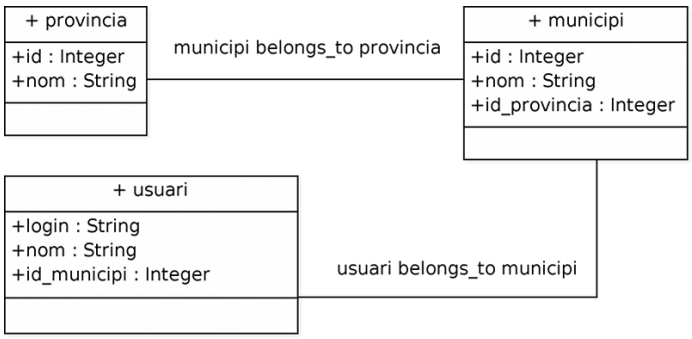


Figura 10.7: Active Record,Relació *has_many_through*

Listing 10.13: Active Record, Relacions: SQL «has_many_through»

```
1  -- taula provincia
2  CREATE TABLE provincia (
3    id VARCHAR(50) INTEGER NOT NULL PRIMARY KEY,
4    nom VARCHAR(100) NOT NULL UNIQUE
5  );
6
7  -- taula usuari
8  CREATE TABLE usuari (
9    login VARCHAR(50) NOT NULL PRIMARY KEY,
10   nom VARCHAR(100) NOT NULL,
11   id_municipi INTEGER NOT NULL,
12   FOREIGN KEY(id_municipi) REFERENCES municipi(id) ON DELETE
         RESTRICT ON UPDATE RESTRICT
13 );
14
15 -- taula municipi
16 CREATE TABLE municipi (
17   id INTEGER NOT NULL PRIMARY KEY,
18   nom VARCHAR(100) NOT NULL,
19   id_provincia INTEGER NOT NULL,
20   FOREIGN KEY(id_provincia) REFERENCES provincia(provincia)
        ON UPDATE RESTRICT ON DELETE RESTRICT
21 );
```

Listing 10.14: Active Record, Relacions: Model que implementa relació «has_many_through»

```
1  <?php
2  // provincia.class.php
3  class provincia extends FW_ActiveRecord_Model {
4    protected $id;
5    protected $nom;
6
7    public static $has_many_through = array (
8      array(
9        "property"  => "usuaris",
10       "srcTable"  => "provincia",
```

```
        "srcColumn" => "id",
        "dstTable"  => "usuari",
        "dstColumn" => "id_municipi",
        "throughTable" => "municipi",
        "throughTableSrcColumn" => "id_provincia",
        "throughTableDstColumn" => "id",
        "update" => "restrict",
        "delete" => "restrict"
      )
    );
};

// usuari.class.php
class usuari extends FW_ActiveRecord_Model {
  protected $login;
  protected $nom;
  protected $id_municipi;
};


// municipi.class.php
class municipi extends FW_ActiveRecord_Model {
  protected $id;
  protected $nom;
  protected $id_provincia;
};
?>
```

Ací hem representat la taula intermèdia en el model de dades en PHP perquè anem a utilitzar-la obtenint-ne informació d'ella.

Relació ≪has_many_by_sql≫ (1:N)

Aquest tipus de relació està pensat per a relacions amb condicions complexes més enllà de les claus alienes i que s'escapen a la representació possible amb els altres tipus de relacions existents.
Permet representar una relació 1:N restringint amb algun criteri les instàncies que es poden relacionar amb ella.
Aquesta relació és útil quan volem imitar al concepte *vista* dels *SGBD*, o quan volem tindre permanentment relacionades unes dades que considerem interessants per el nostre negoci.
Exemples d'aquest tipus de relació serien:

- Relació per obtindre tots els municipis d'una província que comencen per 'Al' i tinguen un número d'habitats major a 5.000 persones.
- Relació per obtindre aquells socis d'una associació esportiva que tinguen sexe home i més de 18 anys.
- Relació per obtindre els articles de preu major a 10.000 euros que ens proveu el proveïdor ≪A≫ en exclusiva.

Totes aquestes relacions es composen d'una consulta SQL que genera la relació 1:N amb restriccions en la part del *WHERE*. Aquestes restriccions poden ser les que s'adapten a les nostres regles de negoci.

Listing 10.15: Active Record, Relacions: Model que implementa relació «has_many_by_sql»

```php
<?php

// provincia.class.php
class provincia extends FW_ActiveRecord_Model {
  protected $id;
  protected $nom;

  public static $has_many = array (
    array(
      "property" => "municipis",
      "table"    => "municipi",
      "srcColumn"=> "id",
      "dstColumn"=> "id_provincia",
      "update"   => "cascade",
      "delete"   => "cascade"
    )
  );

  public static $has_many_by_sql = array (
    array(
      "property" => "municipis_sant",
      "table"    => "municipi",
      "srcColumn"=> "id",
      "dstColumn"=> "id_provincia",
      "update"   => "cascade",
      "delete"   => "cascade",
      "conditions" => array (
        array (
          "name"     => "UPPER(nom)",
          "operator" => "LIKE",
          "value"    => "SANT%"
        )
    )
  );

};


// municipi.class.php
class municipi extends FW_ActiveRecord_Model {
  protected $id;
  protected $nom;
  protected $id_provincia;
};

?>
```

Aquest tipus de relació genera les mateixes consultes que amb les relacions del tipus *has_many*. A aquesta consulta se li afegeix una clàusula *WHERE* que serà la clàusula generada a partir de les condicions amb les que es faça la relació.

relacions reflexives (1:N)

Les relacions reflexives (1:N) són molt útils per representar jerarquies de dades. Exemples:

- Jerarquia de categories.
- Jerarquia de cap-treballadors.
- Jerarquia de pàgines (pàgines relacionades).
- ....

Active Record proveu al desenvolupador d'una funcionalitat per representar aquestos tipus de relacions (només les 1:N) d'una forma molt senzilla amb només configurar un array. Proveu els mètodes útils i necessaris per obtindre el pare de l'objecte actual, obtindre els objectes fills o obtindre els objectes germans.
Active Record aprofita la forma d'aquestes de relacions que generen arbres N-aris en els que el node arrel (en aquest cas una instància d'un model) i els nodes fills (també instàncies de models). D'aquesta forma es pot navegar per aquestos arbres de dades i utilitzar les dades com un arbre N-ari.

Listing 10.16: Active Record, Relacions: Relacions reflexives mitjançant «acts_as_tree»

```
<?php
class category extends FW_ActiveRecord_Tree {

    public $id;
    public $id_parent;
    public $title;
    public $description;
    public $author;
    public $image;
    public $created_at;
    public $password;
    public $posts;


    /**
     * fer que el model actue com un arbre
     * utilitzant relacions 1:n reflexives
     */

    public static $acts_as_tree = array (
        // columna que actua d'identificador del node
        "idColumn"     => "id",
        // columna que actua de id del node pare (clau
            aliena)
        "parentColumn" => "id_parent",
        // ordre dels nodes germans
        "siblingOrder" => array (
            array (
                "column" => "title",
                "type"   => "ASC"
```

```
30                  )
31              )
32          );
33
34          // altres relacions ...
35          public static $has_and_belongs_to_many = array (
36            array(
37              "property"  => "posts",
38              "srcTable"  => "category",
39              "srcColumn" => "id",
40              "dstTable"  => "post",
41              "dstColumn" => "id",
42              "throughTable" => "post_has_category",
43              "throughTableSrcColumn" => "id_category",
44              "throughTableDstColumn" => "id_post",
45              "update" => "restrict",
46              "delete" => "restrict"
47            )
48          );
49
50          public static $belongs_to = array (
51              array(
52                  "property" => "author",
53                  "table"    => "user",
54                  "srcColumn"=> "author",
55                  "dstColumn"=> "username",
56                  "update"   => "restrict",
57                  "delete"   => "restrict"
58              )
59          );
60
61      };
62      ?>
```

En l'exemple es pot veure una classe del model de dades (heretant de la classe *FW_ActiveRecord_Tree* que representa una categoria (relació reflexiva $categoria \rightarrow categoria_{(}pare)$ 1:N) configurant-la en el array *acts_as_tree*, amb aquesta definició s'aconsegueix que el model actue com un arbre definint sobre ell la relació $id\_parent(enelfill) \rightarrow id(enelpare)$.
En l'API d'Active Record Tree detallada més endavant es mostra com utilitzar els mètodes que proporciona per navegar a través de l'arbre de nodes (de tipus categoria en aquest exemple).

10.2.4 Validacions

La validació de dades és un procés per el qual el sistema Active Record comprova cada propietat del model abans de cada operació d'actualització o inserció, fent que només s'inserisquen dades correctes i que respecten els tipus de dades i llargàries de les columnes de les taules de la base de dades, així com la integritat de les dades .
Les comprovacions realitzades per defecte inclouen els següents tipus de comprovacions:

- Comprovacions de tipus de dades.

- Comprovacions de llargària de les dades.
- Comprovacions de valors nuls.
- Comprovacions de clau primària.
- Comprovacions de clau alienes.

Generalment les validacions per defecte són suficients per a evitar introduir dades incorrectes en la base de dades, malgrat tot això, existeix un altre tipus de validació en el que segons les regles de negoci del propi usuari es poden fer validacions de les propietats.
Aquestes validacions personalitzades, s'han d'implementar en forma de mètodes amb el modificador d'accés *protected* i que tinguen un nom que comence per *validate* .
Aquestos mètodes seran executats en primer lloc abans de començar amb la validació de propietats per defecte. Han de retornar els valors **true** (si és correcta la validació) o **false** (si la validació és incorrecta). Una vegada un mètode o una validació d'una propietat retorne el valor **false**, es detindrà el procés de validació i es cancel·larà qualsevol operació d'inserció o d'actualització en marxa.

En el següent exemple es pot observar un exemple de validació personalitzada que comprova la restricció d'unicitat per a la propietat *permalink*:

Listing 10.17: Active Record: Validació personalitzada

```
<?php
class post extends FW_ActiveRecord_Model {
  protected $id;
  protected $title;
  protected $content;
  protected $created_at;
  protected $author;
  protected $permalink;
  protected $status;

  // validacio personalitzada per una restricccio d'unicitat
  protected function validatePermalink() {
    $conditions = array (
      array (
        "name"     => "permalink",
        "operator" => "=",
        "value"    => $this->_permalink
      )
    );
    $select = post::find($conditions);
    return (!$select->hasResult());
  }

};
?>
```

10.2.5 Callbacks

Un *callback* és una funció o mètode que s'executa just abans, després o en meitat d'una operació.
Per ajudar a millorar la integritat relacional de les dades contingudes en els models d'Active Record, el programador té al seu abast un gran nombre d'aquestos callbacks que pot implementar en el seu model.
La funció que té cada callback està explicada en el següent codi. Tots els callbacks han d'implementar-se amb el modificador d'accés **protected** de forma que *FW_ActiveRecord* puga executar-los quan siga convenient. Hauran, també de tornar un valor *booleà* **true** o **false** segons la lògica de negoci que implementen. Un valor de false de retorn d'un callback signfica - al igual que en el cas de la validació - la interrupció de l'acció en curs.

Listing 10.18: Active Record: Callbacks

```
<?php
// despres de fer l'operacio find i construir el model
bool afterFind(void);

// abans de fer una consulta d'existencia de dades
bool beforeExists(void);

// despres de fer una consulta d'existencia de dades
bool afterExists(void);

// abans d'esborrar les dades d'un model
bool beforeDelete(void);

// despres d'esborrar les dades d'un model
bool afterDelete(void);

// abans d'actualitzar les dades d'un model
bool beforeUpdate(void);

// despres d'actualitzar les dades d'un model
bool afterUpdate(void);

// abans d'inserir les dades d'un model
bool beforeInsert(void);

// despres d'inserir les dades d'un model
bool afterInsert(void);

// abans d'executar l'operacio save
bool beforeSave(void);

// despres d'executar l'operacio save
bool afterSave(void);
?>
```

Aquestos callbacks serveixen per fer moltes funcions, en concret:

- Codificació/descodificació de dades (utf8 a iso-8859-1).

- Conversió d'unitats.
- Cridar a altres mètodes del model que facen tasques de manteniment ...

10.2.6 Funcionalitat d'usuari

Dins el model és possible generar funcionalitat d'usuari que el propi model de dades requereix-ca.
Es poden crear qualsevol tipus de mètodes on s'implemente la lògica de negoci necessària per a cada model.

10.2.7 Esquemes

Part del funcionament del component *Active Record* es bassa en conèixer de quines propietats està compost cada model, de quines propietats formen part de la clau primària, quins tipus de dades té cada propietat, ...
Aquest coneixement es genera interrogant al motor de base de dades sobre una fila d'una taula concreta, fent introspecció en el model de dades i veient quines propietats i relacions té. Tota aquesta informació sobre un model és desada en forma d'esquema utilitzant un component dins d'Active Record anomenat *ActiveRecord_Metadata_Schema* .
Degut al gran nombre d'esquemes (1 per cada model existent en el sistema), existeix un component encarregat de coordinar la seva generació i obtindre aquell esquema necessari a cada moment. Aquest component s'anomena *ActiveRecord_Metadata_Manager*. S'encarrega de generar tots els esquemes i desar-los en un fitxer anomenat *serialized_activerecord_manager.ser* situat dins el directori `/framework/cache/framework/schemas` .
Si es genera un nou model, és necessari esborrar aquest fitxer i deixar que el sistema el genere de nou (generarà el fitxer d'esquemes amb els nous models trobats).
Aquest fitxer és necessari donat que no es pot estar constantment demanant informació a la base de dades, per qüestions d'eficiència i de temps.

10.3 API d'Active Record

En aquesta secció es presenta l'API del component d'Active Record.

Primerament veurem els tipus de dades resultats de moltes operacions, després les operacions de cerca, operacions amb funcions de columna (count,avg,sum,min,max) i operacions de desat de dades i altres operacions importants en el cicle de vida del model de dades d'una aplicació web.

10.3.1 Tipus de dades de l'API

Model i Tree

Els objectes *FW_ActiveRecord_Model* i *FW_ActiveRecord_Tree* (objectes base d'Active Record que representen les dades) són els objectes dels que ha de derivar tot model d'Active Record.

Proporcionen una serie de funcionalitats interessants per convertir les dades contingudes en un model en formats estàndards o generar un model a partir de les dades en aquestos formats.

Listing 10.19: API de *FW_ActiveRecord_Model*

```php
<?php
/* operacions de transformacio entre formats estandards de
    les dades */

// obte un array amb totes les dades del resultat
array toArray(array $properties=array(),array
    $transformations=array());

// obte un document XML amb totes les dades del resultat
string toXML($root="",array $properties=array(),array
    $transformations=array(),array $cdatas=array());

// obte un document JSON amb totes les dades del resultat
string toJSON(array $properties=array(),array
    $transformations=array(),$headers=true,$jsonFlags=null);

// obte un document CSV amb totes les dades del resultat
string toCSV (array $properties=array(),array
    $transformations=array(),$headers=true,$delimiter=",");

// genera les dades d'un model a partir de dades en format
    JSON
FW_ActiveRecord_Model static fromJSON($json="",$relations=
    false);

// genera les dades d'un model a partir de dades en format
    XML
FW_ActiveRecord_Model static fromXML($xml="",$relations=true
    );


// genera les dades d'un model a partir de dades en forma d'
    array
FW_ActiveRecord_Model static fromArray(array $data,
    $relations=false);


/* altres operacions importants */

// obtindre les propietats que te el model
array getModelProperties(void);

// obte una representacio serialitzada del model i les seves
     dades
string serialize(void);
?>
```

Listing 10.20: API de *FW_ActiveRecord_Tree*

```php
<?php
// obte si aquest node es o no l'arrel de l'arbre
bool isRoot(void);

// obte si aquest node te un node pare o per el contrari es
    un node pare
bool hasParent(void);

// obte si aquest node te nodes germans (altres fills del
    seu node pare)
bool hasSiblings(void);

// obte si aquest node te nodes fills
bool hasChildren(void);

// obte el node pare
FW_ActiveRecord_Tree getParent(void);

// obte els nodes fills d'aquest node
FW_ActiveRecord_Result getChildren(void);

// obte els nodes germans d'aquest node
FW_ActiveRecord_Result getSibling(void);

// obte l'arrel de l'arbre que forma part aquest node
FW_ActiveRecord_Tree root(void);
?>
```

Resultats (*FW_ActiveRecord_Result*)

El tipus de dades per expressar un resultat en les operacions de cerca s'anomena *FW_ActiveRecord_Result*. Conté un array de dades que es pot recórrer amb bucles for,foreach i d'altres gràcies a que implementa vàries interfícies *d'iteradors*.
En el següent codi es mostren les operacions més importants sobre aquest tipus de dades:

Listing 10.21: API de *FW_ActiveRecord_Result*

```php
<?php
/* operacions sobre el resultat */

// obte si hi ha o no elements en el resultat
bool hasResult(void);

// obte el nombre d'elements del resultat
int count(void);
int numResults(void);

// obte el tipus de resultat (nom del model) que emmagatzema
     aquest resultat
string getResultType(void);


/* operacions d'obtencio d'elements */
```

```
16
17  // obte el primer element del resultat
18  FW_ActiveRecord_Model first(void);
19
20  // obte el darrer element del resultat
21  FW_ActiveRecord_Model last(void);
22
23  // esborra tots els elements del resultat
24  void clear(void);
25
26  // obte tots els elements del resultat
27  array getObjects(void);
28
29
30  /* operacions de transformacio en formats estandards de les
        dades del resultat */
31
32  // obte un array amb totes les dades del resultat
33  array toArray(array $properties=array(),array
        $transformations=array());
34
35  // obte un document XML amb totes les dades del resultat
36  string toXML($root,$elementRoot="",array $properties=array()
        ,array $transformations=array(),array $cdatas=array());
37
38  // obte un document JSON amb totes les dades del resultat
39  string toJSON(array $properties=array(),array
        $transformations=array(),$headers=true,$jsonFlags=null);
40
41  // obte un document CSV amb totes les dades del resultat
42  string toCSV (array $properties=array(),array
        $transformations=array(),$delimiter=",");
43
44  ?>
```

Relacions (*FW_ActiveRecord_Relation*)

Dins d'una instància d'un model ens podem trobar amb propietats que es refereixen a objectes relacionats amb l'objecte que estem tractant. Aquestes propietats tindran un valor que serà de tipus *FW_ActiveRecord_Relation*, que representa una relació amb tots els objectes relacionats i proporciona certes operacions útils per mantindre la integritat de les dades.
Donat que aquest tipus de dades està definit com herència del tipus de dades *FW_ActiveRecord_Result* , també té accés a totes les funcionalitats que té aquest tipus de dades i que ajuden a recórrer la col·lecció de dades relacionada.
FW_ActiveRecord_Relation té una operació molt útil anomenada *find* que permet fer cerques dins els objectes relacionats.

En el següent codi es mostren les operacions més importants sobre aquest tipus de dades:

Listing 10.22: API de *FW_ActiveRecord_Relation*

```
1   <?php
```

```
    // cerca dins el resultat dels models relacionats
    FW_ActiveRecord_Result find(array $conditions=array(),
        array $orders=array(),$limit=0,$offset=0);

    // afegeix un objecte de tipus model a la relacio
    void add(FW_ActiveRecord_Model $object);

    // actualitza totes les dades relaciones amb la relacio
    mixed update(void);

    // esborra totes les dades relaciones amb la relacio
    mixed delete(void);

    // obte el valor de la propietat que representa el valor
        de la clau aliena sobre la que esta la relacio
    mixed getOldValue(void);
  ?>
```

Condicions per fer una cerca

Per indicar-li a moltes de les operacions les condicions amb les que volem cercar les dades, definirem un array semblant al següent codi:

Listing 10.23: Definir condicions per cercar dades amb Active Record

```
  <?php
    $conditions = array (
      // aci aniran les condicions de cerca
    );
    // condicio simple
    array (
      "name"     => "nom_de_la_propietat",
      "operator" => "operador",
      "value"    => "valor_per_fer_la_cerca"
    ),
    // condicions complexes
    array (
      "name"     => "nom_de_la_propietat",
      "operator" => "IN",
      "value"    => " 'valor1','valor2','valor3' "
    ),
    // condicions amb funcions de dates
    array (
      "name"     => "YEAR(data)",
      "operator" => ">",
      "value"    => "1999"
    ),
    // condicions amb cerca de text
    array (
      "name"     => "UPPER(columna)",
      "operator" => "LIKE",
      "value"    => "A%B%C"
    ),

```

```
30
31      // operador simple (AND,OR, ...)
32      array ( "condition" => "AND" ),
33
34
35      // combinacions d'ambdos
36
37      /**
38      * Posts que son del any 2010
39      * i tenen el titol LIKE 'A%'
40      * i foren escrits per el usuari
41      * 'andreums'
42      */
43      $conditions = array (
44        array (
45          "name"     => "YEAR(created_at)",
46          "operator" => "=",
47          "value"    => "2010"
48        ),
49        array ("condition" => "AND"),
50        array (
51          "name"     => "title",
52          "operator" => "LIKE",
53          "value"    => "A%"
54        ),
55        array ("condition" => "AND"),
56        array (
57          "name"     => "author",
58          "operator" => "=",
59          "value"    => "andreums"
60        )
61      );
62
63    ?>
```

Com es veu en el codi anterior, és senzill definir unes condicions per cercar dades. Aquestes condicions es traduiran després a SQL i junt a l'ajuda que proporciona Active Record, s'obtindrà el query SQL que serà passat al component *Database* per a la seva execució i retornarà un resultat que Active Record analitzarà.

Condicions per fer ordenació de dades

Per indicar-li a moltes de les operacions la nostra preferència a l'hora d'ordenar les dades, definirem un array semblant al següent codi:

Listing 10.24: Definir condicions per l'ordenació de dades d'Active Record

```
1   <?php
2     $order = array (
3       array (
4         "column" => "created_at",
5         "type"   => "ASC"
6       ),
7       array (
```

```
        "column" => "value",
        "type"   => "DESC"
      )
    );
?>
```

10.3.2 Operacions de cerca de dades (find)

Aquest grup d'operacions és el més important de tot Active Record. A partir d'unes condicions obté un objecte de tipus *FW_ActiveRecord_Result* amb les dades que resulten d'aplicar les restriccions de les condicions a la taula de la base de dades que és el model, a banda de cercar automàticament totes les relacions i obtindre els objectes relacionats amb cada objecte trobat.

Listing 10.25: API Active Record: Operacions de cerca

```
<?php
// operacio de cerca principal
FW_ActiveRecord_Result static find($conditions=array(),
    $orders=array(),$groups=array(),$havings=array(),$limit=
    false,$offset=false);

// operacio de cerca per valor d'una columna
FW_ActiveRecord_Result static findBynom_de_la_columna('valor
    ');

// operacio de cerca paginada (parameters offset i limit)
FW_ActiveRecord_Result static findPaginated($conditions=
    array(),$orders=array(),$limit=false,$offset=false);
?>
```

10.3.3 Operacions amb funcions de columna

Aquest grup d'operacions són les funcions de columna **AVG**,**MIN**,**MAX**,**SUM** i **COUNT** les quals es poden aplicar a una propietat/columna i obtindre el resultat desitjat aplicant-li unes condicions de filtrat.

Listing 10.26: API Active Record: Operacions amb funcions de columna

```
<?php
// fa la funcio de columna SUM amb una columna
mixed static sum($column,$conditions=array(),$orders=array()
    );

// fa la funcio de columna MIN amb una columna
mixed static min($column,$conditions=array(),$orders=array()
    );

// fa la funcio de columna MAX amb una columna
mixed static max($column,$conditions=array(),$orders=array()
    );

// fa la funcio de columna AVG amb una columna
```

```
12  mixed static avg($column,$conditions=array(),$orders=array()
        );
13
14  // fa la funcio de columna COUNT amb una columna
15  mixed static count($column,$conditions=array(),$orders=array
        ());
16  ?>
```

10.3.4 Desat, actualització i inserció, esborrament i existència de dades

La única forma amb la que podem desar les dades (actualitzar o inserir) és l'operació *save*, aquesta operació fa tot per nosaltres i determina què s'ha de fer amb les dades en temps real.
Esborrarem dades mitjançant les operacions *destroy* o el seu alias *delete*.
Provarem que unes dades no existeixen en la base de dades amb l'operació *exists*.
Per últim, comprovarem si les dades són vàlides (veure apartat dedicat a la validació d'Active Record) amb l'operació *validate*. Totes aquestes operacions retornen **true** o **false** segons siga l'èxit o fracàs de l'operació.

Listing 10.27: API Active Record: Operacions de desat, actualització i inserció, esborrament i existència de dades

```
1   <?php
2   // desa (insereix o actualitza) el valor del model en la
        base de dades
3   bool save(void);
4
5   // esborra les dades de la base de dades
6   bool destroy(void);
7   bool delete(void);
8
9   // comprova si existeix una fila amb les dades del model en
        la base de dades
10  bool exists(void);
11
12  // determina si es valid o no el model (veure apartat
        dedicat a la validacio)
13  bool validate(void);
14  ?>
```

10.3.5 Exemples

Per terminar aquest capítol, es pressenten uns exemples de com utilitzar l'API d'Active Record per recuperar i crear dades:

Listing 10.28: Exemples d'us de l'API de *FW_ActiveRecord_Result*

```
1   <?php
2
3   // obtindre totes les categories (find sense parametres)
4   $categories = category::find();
```

```
5
6   // si hi ha resultat en les categories
7   if ($categories->hasResult()) {
8     foreach ($categories as $category) {
9       print $category->title."<br/>";
10    }
11  }
12
13  // obtindre la provincia de Valencia
14  $provincia  = provincia::findByprovincia("Valencia");
15  // si hi ha resultat
16  if ($provincia->hasResult()) {
17      // obtindre el primer resultat
18      $provincia = $provincia->first();
19      print "<h2>Provincia de Valencia</h2>";
20      print "<hr/>";
21      print "<h4>Poblacions</h4>";
22      print "<p>Te {$provincia->poblaciones->count()}
            poblacions</p>";
23      print "<hr/>";
24      // mostrar el nom de cada poblacio
25      foreach ($provincia->poblaciones as $poblacion) {
26          print "<p>{$poblacion->poblacion}</p>";
27      }
28  }
29
30  // obtindre les poblacions de la provincia de Valencia que
        tenem nom comencat per Beni%
31  $condicions = array (
32      array (
33          "name"     => "poblacion",
34          "operator" => "LIKE",
35          "value"    => "Beni%"
36      ),
37      array (
38          "name"     => "idprovincia",
39          "operator" => "=",
40          "value"    => "36"
41      )
42  );
43  $poblacions = poblacion::find($condicions);
44
45
46  // crear un llibre nou
47  $book         = new book();
48  $book->isbn   = 001;
49  $book->title  = "Llibre de proves";
50  $book->author = "Fulanito de Tal";
51
52  // desar-lo en la base de dades
53  if ($book->save()) {
54      print "<p>Llibre desat amb exit</p>";
55  }
56
```

```
57  // modificar informacio d'un model i despres desarlo en la
        base de dades
58  $book = book::findByisbn(001);
59  if ($book->hasResult()) {
60      $book = $book->first();
61      $book->title = "Proves";
62
63      if ($book->save()) {
64          print "<p>Llibre actualitzat amb exit</p>";
65      }
66  }
67  ?>
```

Part IV

Desenvolupament d'aplicacions web

Capítol 11

Estructura de les aplicacions web

11.1 Introducció

En aquest capítol presentarem l'estructura de l'aplicació web que anem a desenvolupar. En primer lloc cal dir que l'aplicació és totalment modular i que aprofitant el patró de disseny (HMVC *Hierarchical-Model-View-Controller*) que promou la modularització d'una aplicació estructurada mitjançant el patró de disseny MVC (*Model View Controller*) en múltiples mòduls MVC aïllant així el codi en funcionalitats comunes i evitant (amb la comunicació entre mòduls) la repetició de codi font.

11.2 Directoris

En les imatges següents s'aprecien els directoris que conté una aplicació web

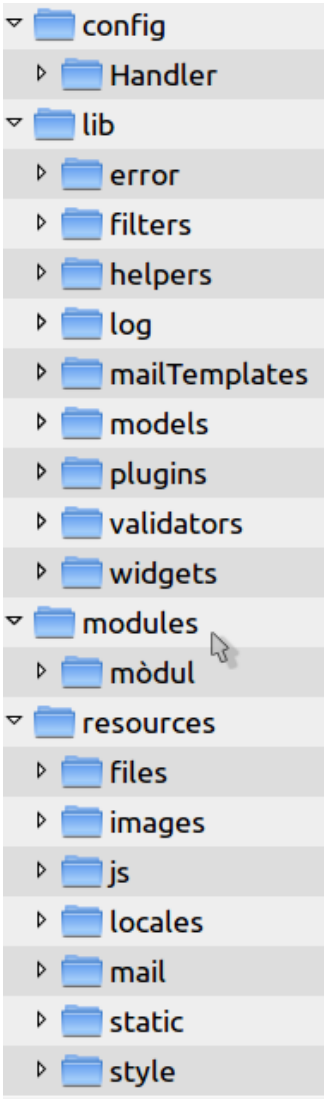


Figura 11.1: Directoris d'una aplicació web (dins el directori */app*)

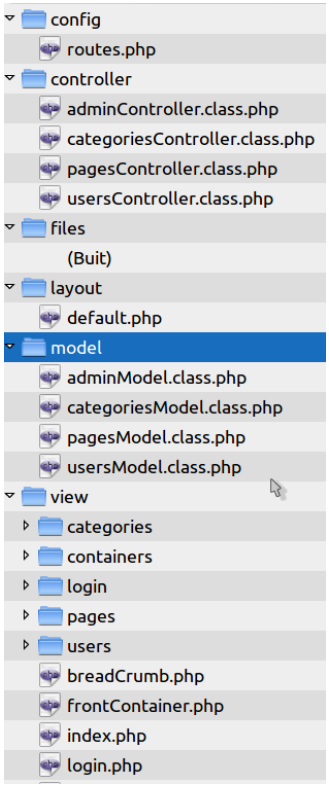


Figura 11.2: Directoris d'un mòdul (dins el directori */app/modules*)

11.3 Estructura d'un mòdul

Cada mòdul del desenvolupament es crearà dins un directori en el directori */app/modules* del marc de treball i amb la següent estructura:

/app/modules : Directori pare on resideixen tots els mòduls de la nostra aplicació web.

/app/modules/mòdul : Directori que contindrà el nostre mòdul.

/app/modules/mòdul/config : Directori que contindrà la configuració del nostre mòdul.

/app/modules/mòdul/config/routes.php : Fitxer que contindrà informació d'enrutament del nostre mòdul.

/app/modules/mòdul/controller : Directori que contindrà els controladors del nostre mòdul.

/app/modules/mòdul/model : Directori que contindrà els models del nostre mòdul.

/app/modules/mòdul/view : Directori que contindrà les diferents vistes del nostre mòdul.

/app/modules/mòdul/view/error : Directori que contindrà les diferents vistes d'errors del nostre mòdul.

/app/modules/mòdul/layout : Directori que contindrà els diferents layouts del nostre mòdul.

11.4 Estructura MVC

11.4.1 Introducció

En aquesta secció s'introduirà la terminologia bàsica del patró de disseny MVC així com es presentarà el codi necessari per a crear un mòdul d'una aplicació web amb el marc de treball del projecte.

11.4.2 El controlador

Un controlador és una classe conté la part de lògica de negoci d'una aplicació mantenint-la separada de l'obtenció de dades i de la interfície d'usuari. Aquesta classe s'encarrega de gestionar l'execució d'una determinada acció (mètode d'un controlador), demanant dades al model, processant-los després i finalment mostrant a l'usuari el resultat de l'acció en una vista (interfície gràfica).

Donat el principi de separació de capes que introdueix el patró de disseny MVC, el controlador ha de limitar-se a treballar amb la lògica de negoci. És a dir, amb dades que provenen d'una petició (que és qui activa l'execució d'una acció d'un controlador) o del model (base de dades/fonts de dades).

El controlador (classe *FW_mvc_BaseController*)

Cada controlador de cada mòdul d'una aplicació ha de residir en el directori *controller* dins cada mòdul i ha de tindre la següent estructura de codi PHP:

Listing 11.1: Controlador bàsic

```
<?php
class elMeuControladorController extends
    FW_mvc_BaseController {

  public function action1($param1,$param2) {
    ...
  }

  public function actionN($param1,$param2,$param3) {
    ...
  }
};
?>
```

El nom de la classe ha de ser exactament de l'esquema {nomDelControlador}Controller i heretar de la classe abstracta i bàsica *FW_mvc_BaseController* que és qui proporciona la funcionalitat bàsica d'un controlador ha d'estar en un fitxer anomenat *nomDelControladorController.class.php* en el directori controller del mòdul.

Un mòdul pot tindre un o més controladors que **obligatòriament han d'estar relacionats amb un model** situat en el directori *model* de cada mòdul. Cada controlador pot tindre una o més accions que seran mètodes de la classe del controlador **sense paràmetres** Les accions d'un controlador tenen el següent aspecte:

Listing 11.2: Acció d'un controlador

```php
<?php
// accions que mostren pagines
public function mostrarFormulariContacte() {
        $this->renderView("formularis".DS."contacte");
        return;
}
// accions amb parametres
public function mostrarPagina() {
        $id = $this->escape($this->_request->getParameter("
            id"));
        if ($this->filter->isNumeric($id)) {
                $pagina = $this->_model->getPaginaById($id);
                if ($pagina!==null) {
                        $this->set("pagina",$pagina);
                        $this->renderView("pagines".DS."
                            pagina");
                        return;
                }
                else {
                        $this->renderErrorPage("pagines".DS.
                            "paginaNoTrobada");
                        return;
                }
        }
        else {
                $this->renderErrorPage("pagines".DS."
                    paginaNoTrobada");
                return;
        }
}
?>
```

API del controlador

L'API del controlador proveu al desenvolupador de les següents operacions:

Listing 11.3: API del component *FW_mvc_BaseController*

```php
<?php
// obte l'objecte FW_Request
FW_Request request(void);

// obte el usuari loggejat en el sistema
FW_Authentication_User user(void);

// obte l'objecte FW_Session
FW_Session session(void);

// obte el model
FW_mvc_BaseModel model(void);

// obte l'objecte FW_Context
```

```
15  FW_Context context(void);
16
17
18  // redirecciona a una pagina d'error 403
19  void forbidden(void);
20
21  // redirecciona a una pagina d'error 404
22  void notFound(void);
23
24  // parseja i escapa una variable de text
25  string sanitize($variable);
26  string escape($data,$stripTags);
27
28  // convert una variable de text per mostrar-la per pantalla
29  string display($data);
30
31  // renderitza un layout
32  void renderLayout($layout);
33
34  // configura un slot del layout
35  void setSlot($name,$view,array $variables=array());
36
37  // obte un slot del layout
38  string getSlot($name);
39
40  // obte si esta disponible un slot (si te contingut)
41  bool hasSlot($name);
42
43  ?>
```

11.4.3 El model

Un model és una classe que conté l'accés a dades ja siguen d'una base de dades, d'un fitxer o d'un servei web. Els mètodes del model seran invocats per el controlador amb la finalitat d'obtindre dades o de desar dades en una font de dades. El model ha de limitar-se a l'accés a les fonts de dades, mentre que tot el treball de processament de dades s'ha de fer en el controlador.

El model (classe *FW_mvc_BaseModel*)

Un model s'ha de correspondre necessàriament amb un controlador del mateix nom, per tant, cada classe de model s'ha d'anomenar de la manera {nomDelControlador}Model i heretar de la classe abstracta i bàsica *FW_mvc_BaseModel* que és qui proporciona la funcionalitat bàsica d'un model ha d'estar en un fitxer anomenat *nomDelControlador-Model.class.php* en el directori model del mòdul. Per tant, cada controlador d'un mòdul té associat un únic model del que obtindre dades.
Un model té el següent aspecte:

Listing 11.4: Model bàsic

```
1  <?php
2  class elMeuControladorModel extends FW_mvc_BaseModel {
3
```

```
4     public function getAllPages() {
5        $pages = page::find();
6        if ($pages->hasResult()) {
7          return $pages;
8        }
9     }
10
11    public function getPageById($id) {
12       $page = page::findById($id);
13       if ($page->hasResult()) {
14         return $page->first();
15       }
16    }
17
18  };
19  ?>
```

En un model pot haver-ne cero o més mètodes que poden - o no - tindre paràmetres; segons convinga per a cada situació. Generalment un model obtindrà dades de la base de dades (directament o a través d'*Active Record*), d'un fitxer XML o d'un servei web.

API del model

El model té una API senzilla en la que la única funcionalitat és obtindre accés a la base de dades.

Listing 11.5: API del component *FW_mvc_BaseModel*

```
1  <?php
2  // obtindre la instancia de la base de dades
3  FW_Database database(void);
4  ?>
```

11.4.4 La vista

Una vista és un fragment de codi HTML amb una lògica de control mínima de PHP que mostra resultats de les accions realitzades en una acció del controlador. Donat que les dades ja han estat processades en el controlador, la vista s'encarregarà únicament de mostrar eixes dades. Utilitzant per aquesta tasca únicament les estructures *if, elsif, else* i els bucles *for, foreach, do...while i while* amb la única finalitat de recòrrer les col·leccions de dades estructurades i poder accedir a les seves propietats amb tal de mostrar-les.

La vista és una espècie de plantilla de codi HTML re-utilitzable on es mostra un resultat d'una acció que pot tindre dades diferent per a cada usuari.

Vistes i Vistes parcials

Una vista pot incloure una vista que mostre un codi d'un disseny d'una taula HTML per mostrar una col·lecció de dades, però per modularitat, si volem mantindre controlat el codi on es mostra cada fila de la tabla; podem mantindre aquest aïllat en una altra vista i incloure'l quan convinga. Aquesta vista 'especial' s'anomena vista parcial essent una ajuda per a la re-utilització de codi i el manteniment d'eixe codi. Malgrat tot, una vista

Estructura d'una vista en PHP

Listing 11.6: Vista simple

```
<h3>Benvingut!</h3>
<p>Estimat <?php print $user->getDisplayName(); ?> sigues
    benvingut a casa</p>
```

Listing 11.7: Vista

```
<h4>Llistat de pagines</h4>

<table>
   <thead>
        <tr>
         <th>ID</th>
         <th>Titol</th>
         <th>Autor</th>
         <th>Data</th>
        </tr>
   </thead>
   <tbody>
  <?php
    // si hi ha pagines
    if (count($pagines)>0) {
    foreach ($pagines as $pagina) {
      $this->renderPartialView("pagines/taula/fila",array("
          pagina"=>$pagina));
    }
    }
  ?>
   </tbody>
</table>
```

Listing 11.8: Vista parcial

```
<tr>
 <td> <?php print $pagina->id; ?> </td>
 <td> <?php print $this->display($pagina->getTitle()); ?> </
     td>
 <td> <?php print $pagina->author->getFullName(); ?> </td>
 <td> <?php print timeHelper::displayFullData($pagina->
     getDate()); ?> </td>
</tr>
```

11.4.5 Layouts web

Un *Layout* és una vista que conforma l'estructura d'una pàgina web que generalment és estàtica i en la que només canvia una part del disseny a cada petició que fa l'usuari. Utilitzarem el framework de CSS *Blueprint* - que es pot obtindre des del lloc web `http://www.blueprintcss.org/` - ja que facilita molt la creació de *Layouts* web amb CSS i XHTML.

Creació de Layouts

Un *Layout* d'una aplicació web és un fitxer de codi en PHP que conté l'esquelet d'una pàgina web (de les seccions que no varien) junt a codi per recuperar i definir les seccions que varien.
Aquestes seccions són els anomenats *Slots*. Un slot defineix una secció o zona en la que hi anirà contingut variable (eixides de vistes d'accions).
El que fan les accions del controlador és definir què va en cada slot (vista i variables) i retornar l'ordre de renderitzar un layout concret. Després el component *Layout* s'encarrega de fer la resta, és a dir, d'aconseguir les dades que van en el slot i de col·locar-les en el seu indret corresponent.

11.4.6 Creació de Layouts

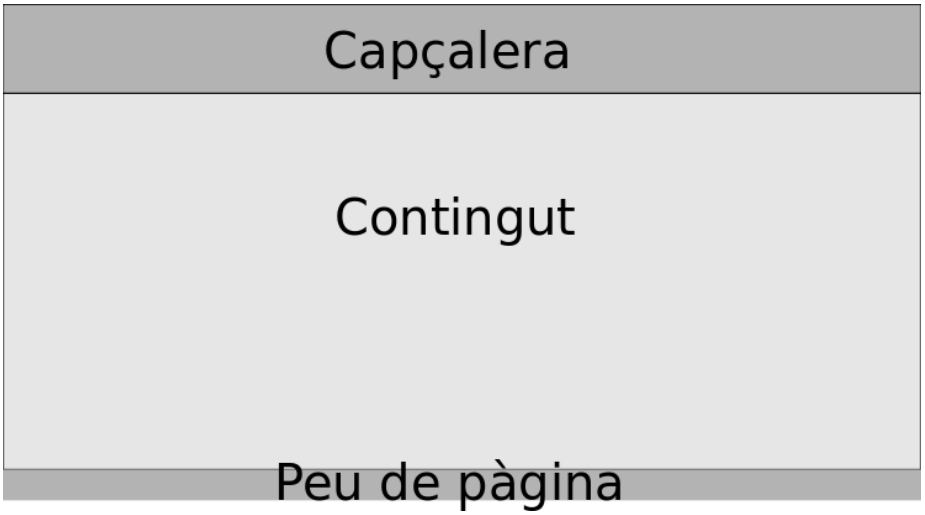


Figura 11.3: Layout bàsic

Generarem aquest layout amb el següent codi font:

Listing 11.9: Layout bàsic

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta http-equiv="Content-Type" content="text/html;
           charset=utf-8" />
5      <title>Demo de layout</title>
6      <?php
7          // mostrar els estils CSS per aquest modul
8          print FW_Style_Manager::getInstance()->displayStyle(
               FW_Context::getInstance()->router->route);
9
10         // mostrar els estils CSS per aquesta accio
11         $styles = FW_Context::getInstance()->router->styles;
12         if (count($styles)>0) {
```

```
13              foreach ($styles as $style) {
14                $file  = $style["file"];
15                $media = $style["media"];
16                print style_tag($file,$media).'\n';
17              }
18            }
19        ?>
20      </head>
21
22      <body>
23        <div id="wrapper">
24
25          <div id="container" class="container">
26
27            <div id="header" class="container">
28              <h1>Demo de layout</h1>
29            </div>
30
31            <div id="content" class="container">
32              <?php
33                // si existeix el slot a mostrar ...
34                if ($this->hasSlot("content")) {
35                  // imprimir els seus contingut
36                  print $this->getSlot("content");
37                }
38              ?>
39            </div>
40
41            <hr class="space" />
42
43          </div>
44        </div>
45
46        <div id="footer" class="container">
47          <p>Demo layout</p>
48        </div>
49        <?php
50        // mostrar els scripts per aquesta accio
51        $scripts = FW_Context::getInstance()->router->scripts;
52        if (count($scripts)>0) {
53            foreach ($scripts as $script) {
54                print html::script_tag($script);
55            }
56        }
57        ?>
58      </body>
59    </html>
```

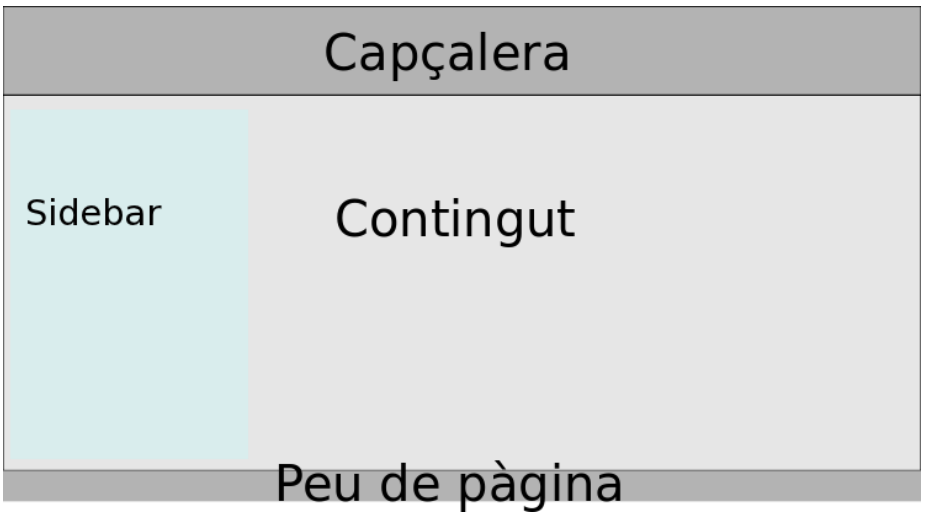


Figura 11.4: Layout complex

Generarem aquest layout amb el següent codi font:

Listing 11.10: Layout complex

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
        charset=utf-8" />
    <title>Demo de layout</title>
    <?php
        // mostrar els estils CSS per aquest modul
        print FW_Style_Manager::getInstance()->displayStyle(
            FW_Context::getInstance()->router->route);

        // mostrar els estils CSS per aquesta accio
        $styles = FW_Context::getInstance()->router->styles;
        if (count($styles)>0) {
          foreach ($styles as $style) {
            $file  = $style["file"];
            $media = $style["media"];
            print style_tag($file,$media).'\n';
          }
        }
    ?>
  </head>

  <body>
    <div id="wrapper">

      <div id="container" class="container">

        <div id="header" class="container">
          <h1>Demo de layout</h1>
```

```
29          </div>
30
31          <aside id="sidebar" class="container span-5 column">
32              <?php
33              if ($this->hasSlot("sidebar")) {
34                  print $this->getSlot("sidebar");
35              }
36              else {
37                  $this->renderView("defaultSidebar");
38              }
39              ?>
40          </aside>
41
42          <div id="content" class="container span-18 column
                last">
43            <?php
44              // si existeix el slot a mostrar ...
45              if ($this->hasSlot("content")) {
46                // imprimir els seus contingut
47                print $this->getSlot("content");
48              }
49            ?>
50          </div>
51
52          <hr class="space" />
53
54        </div>
55      </div>
56
57      <div id="footer" class="container">
58        <p>Demo layout</p>
59      </div>
60      <?php
61      // mostrar els scripts per aquesta accio
62      $scripts = FW_Context::getInstance()->router->scripts;
63      if (count($scripts)>0) {
64          foreach ($scripts as $script) {
65              print html::script_tag($script);
66          }
67      }
68      ?>
69    </body>
70  </html>
```

Els layouts han d'estar definits en el directori `layout` del mòdul a utilitzar.

Capítol 12

El sistema d'enrutat

12.1 Introducció

El sistema d'enrutat és el component del sistema que s'encarrega d'obtindre a partir de la URL i dades de la petició HTTP els paràmetres que indiquen al sistema què ha d'executar.
Aquest sistema basa el procés en una sèrie de dades - carregades en arrays de PHP - que determinen el què s'ha d'executar, el tipus d'acció que és, si cal autenticació per executar eixa acció, si l'acció és cacheable (es pot desar el seu resultat en cache) i els paràmetres que necessita cada acció per poder-se executar.
A partir d'aquestes dades, el component *Front Controller* amb l'ajuda de les seves especialitzacions (una per cada tipus d'acció) i el component *Filter* es pot generar una cridada a l'acció corresponent, sent la resposta d'aquesta acció la que es mostra a l'usuari o s'envia al servei web.

12.2 Estructura d'una ruta

L'estructura bàsica d'una ruta és la següent:

Listing 12.1: Ruta bàsica

```
<?php
FW_Router::Connect(
  array (
    'url'    => '/url/a utilitzar en /accio',
    'type'   => 'tipus d\'accio',
    'cache' => 'indica si es cacheable',
    'parameters' => array (
      'param1' => array (
        'name'    => 'param1',
        'type'    => 'tipus_de_dades',
        'format' => 'format_de_les_dades',
      )
    ),
    'authentication' => 'indica si cal autenticacio',
    'module'      => 'modul',
    'controller' => 'controlador',
```

```
17        'action'      => 'accio',
18        'internal'    => 'si_es_intern_o_extern',
19        'pattern'     => '#^/url/a utilitzar en /accio[/]*$#',
20        'parameterOrder' =>
21            array (
22              0 => 'param1',
23            ),
24        )
25    );
26  ?>
```

12.3 Tipus de d'accions

Existeixen deu tipus d'accions que es tracten de manera diferent en el component *Front Controller*. Cada ruta ha de tindre en el seu paràmetre *type* el valor del tipus que és. En aquesta secció es mostren tots aquestos tipus d'accions i es mostren els seus paràmetres.

12.3.1 Accions d'aplicació

A continuació es pressenten els tipus d'accions **application**, **ajax**, **json** i **mime**. Totes tenen els mateixos paràmetres:

module : El mòdul on resideix la funcionalitat a executar.

controller : El controlador on resideix la funcionalitat a executar.

action : L'acció a executar.

internal : Si està ofert per el framework valor a **true**, en cas contrari valor a **false**.

mime : Només en accions de tipus MIME, indica el tipus *mime* que retorna l'acció.

Les accions de tipus *application* disposen a més a més dels següents paràmetres:

scripts : Array de fitxers de Java Script que seran carregats a demanda per a cada acció, reduint la necessitat d'incloure totes les funcionalitats del client per a totes les accions.

styles : Array de fitxers d'estil CSS que seran carregats a demanda per a cada acció, així pot ser personalitzat el tema CSS exclusiu per una acció.

Title : Títol que se li pot donar a una pàgina.

Listing 12.2: Càrrega d'scripts i estils a demanda

```php
<?php
// en la definicio d'una ruta

// carrega d'scripts javascript a demanda
'scripts' =>  array (
  0 => 'js/jquery/plugins/form/jquery.form.js',
  1 => 'js/jquery/plugins/validation/jquery.validate.min.js',
  2 => 'js/jquery/plugins/validation/additional-methods.js',
  3 => 'js/application/index/main.js',
  4 => 'js/application/user/password.js',
)

// carrega d'estils CSS a demanda
'styles' => array (
  0 =>  array (
    'file'      => 'style/default/application/tables.css',
    'media'     => array('screen','print'),
    'alternate' => false
  ),
),

?>
```

12.3.2 Serveis web

soap : Implementa accions que s'exposen a serveis web en SOAP mitjançant el sistema de SOAP.
Necessita els següents paràmetres:

module : El mòdul on resideix la funcionalitat a executar.

controller : El controlador on resideix la funcionalitat a executar.

action : L'acció a executar.

internal : Si està ofert per el framework valor a **true**, en cas contrari valor a **false**.

description : Descripció de l'operació del servei web.

namespace : Espai de noms del servei web.

service : Nom del servei web.

rest : Implementa accions referents a serveis web de tipus REST.
Necessita els següents paràmetres:

module : El mòdul on resideix la funcionalitat a executar.

controller : El controlador on resideix la funcionalitat a executar.

action : L'acció a executar.

internal : Si està ofert per el framework valor a **true**, en cas contrari valor a **false**.

method : Mètode HTTP utilitzat per accedir-ne a aquesta operació del servei web.

mime : Tipus de dades *mime* (si no retornara dades en formats *JSON* o *XML*.

12.3.3 Plugins

Els plugins també poden implementar funcionalitats accessibles des d'una URL (ex: pàgina de configuració del plugin, estadístiques, ...).
Necessita els següents paràmetres:

plugin : El plugin a executar.

action : L'acció del plugin a executar.

mime : El tipus de dades *mime* que retorna aquesta acció.

12.3.4 Altres tipus d'accions

cron : Implementa una operació preparada per a ser executada per el planificador de tasques del sistema operatiu. Necessita els següents paràmetres:

module : El mòdul on resideix la funcionalitat a executar.

controller : El controlador on resideix la funcionalitat a executar.

action : L'acció a executar.

internal : Si està ofert per el framework valor a **true**, en cas contrari valor a **false**.

static : Aquesta acció s'encarrega de mostrar pàgines web html estàtiques.
És útil per a utilitzar pàgines 'antigues' estàtiques html mentre es fa una migració a una aplicació web. Necessita el següent paràmetre:

file : Fitxer a mostrar.

redirect : Aquesta acció només fa una re-direcció a la URL especificada en el paràmetre.
És útil per a fer migracions o proporcionar URLs *SEO*.
Necessita el següent paràmetre:

redirect : URL a la que serà redirigit l'usuari.

12.4 Paràmetres

Les rutes dels tipus *application,ajax,json,mime,rest,soap,cron i plugin* suporten paràmetres per mitjà de la URL.
Un paràmetre per la URL és afegir-li un fragment de text a la URL que determine el valor del paràmetre.
Aquestos paràmetres poden anar en qualsevol punt de la URL, encara que generalment es recomana que vagen al final de tota la URL, ja que poden provocar col·lisions amb altres URLs d'altres accions i fer que no s'execute l'acció correcta.
Aquestos paràmetres es traslladen (una vegada feta la comprovació en el component *Router*) a l'acció per a que siga executada amb els paràmetres corresponents.
Els paràmetres poden ser de qualsevol dels tipus bàsics (string, integer, float, double)

o ser de qualsevol tipus amb un format especial especificat mitjançant **expressions regulars**, que en aquest cas correspon al desenvolupador crear i comprovar que funciona en tots els valors possibles del paràmetre.

La descripció dels paràmetres en una ruta es fa mitjançant dos arrays (*parameters* i *parameterOrder*).

El primer array conté estructures de dades (arrays) que defineixen a cada paràmetre:

Listing 12.3: Definició de paràmetres en una ruta

```php
<?php
  /**
   * En la definicio d'una ruta
   */
   "parameters" => array (
      "nom_del_parameter" => array (
        "name"   => "nom_del_parameter",
        "type"   => "tipus_del_parameter",
        "format" => "format_del_parameter"
      )
   )
   ...

  /**
   * Exemple:
   */
  "parameters" => array (
      "categoria" => array (
        "name"   => "categoria",
        "type"   => "string",
        "format" => false // no necesita cap format especial
      )
      "subcategoria" => array (
        "name"   => "subcategoria",
        "type"   => "string",
        "format" => false // no necesita cap format especial
      ),
      "id" => array (
        "name"   => "id",
        "type"   => "string",
        "format" => "[\d+]{1,2}"
      )
   )

?>
```

El segon array és necessari, ja que el component *Router* ha de saber en quin ordre estan descrits els paràmetres, per això definim els segon array com segueix:

Listing 12.4: Ordre de paràmetres en definició d'una ruta

```php
<?php
  /**
```

```
3       * En la definicio d'una ruta
4       */
5      "parameterOrder" => array (
6        0 => "categoria",
7        1 => "subcategoria",
8        2 => "id"
9      )
10   ?>
```

12.5 Autenticació

Una ruta pot necessitar autenticació d'usuari per accedir a una zona privada.
Per indicar-li a una ruta que necessita autenticació podem programar-lo com en el següent codi font:

Listing 12.5: Definició d'autenticació en una ruta

```
1    <?php
2      /**
3       * En la definicio d'una ruta
4       */
5
6      // autenticacio desactivada
7      "authentication" => false,
8
9      // autenticacio amb qualsevol rol
10     "authentication" => array(),
11
12     // autenticacio d'usuaris amb rol...
13     "authentication" => array(
14       "roles" =>  "rol1","rol2","rol3"
15     ),
16   ?>
```

12.6 Caché

Si volem que el resultat d'una de les nostres pàgines servides per el framework puga ser desada en la caché del client durant un termini de temps per evitar que es sobrecarregue el sistema i es facen peticions de més a recursos que no canvien, hem d'enviar-li al navegador les capçaleres necessàries i especificades en l'estàndard HTTP que indiquen al client que eixa resposta és cacheable.
Es poden cachear tots els tipus d'accions a excepció de les de tipus cron i redirect. Fer les accions cacheables és molt senzill i requereix només d'un paràmetre:

Listing 12.6: Definició d'una ruta com a cacheable

```
1    <?php
2      /**
3       * En la definicio d'una ruta
4       */
5
```

```
6     // cache desactivada
7     "cache" => false
8
9     // cachear, validessa 60 segons
10    "cache" => 60
11  ?>
```

12.7 Caché de rutes

Amb la finalitat d'accelerar el procés d'enrutament, el marc de treball disposa d'una caché de rutes que són una sèrie de fitxers on es desa la informació de totes les rutes d'un mateix tipus. Aquestos fitxers estan situats en el directori `/framework/cache/framework/router` i tenen extensió *.ser* .
Si es desenvolupa una acció nova i es 'connecta' mitjançant algun fitxer de rutes, s'haurà de regenerar aquesta caché de rutes. Per fer-ho, només cal que s'esborren tots el fitxers d'eixe directori i es faça una petició a qualsevol URL de la nostra aplicació web, així es regenerarà aquesta caché.

Capítol 13

Autenticació

13.1 Introducció

El component *Authentication* serveix per proveir al marc de treball de serveis d'autenticació d'usuaris.
Els usuaris poden ser autenticats per tres mètodes:

- Base de dades.
- Fitxers .htaccess .
- Servidor corporatiu IMAP.

13.2 Configuració de l'autenticació

L'autenticació ha de configurar-se en el fitxer de configuració *authentication.xml* situat en el directori `/framework/config` i amb un contigut semblant al següent codi font:

Listing 13.1: Configuració del component *Authentication*

```
1  <?php
2  $config = array(
3      "sections" => array (
4              "database" => array (
5                  // largaries min i max per usuari i
                       contrasenya
6                  "lengths" => array (
7                      "min" => 6,
8                      "max" => 50
9                  ),
10                 // font de dades
11                 "datasource" => array (
12                     "type"     => "database",
13                     "table"    => "user",
14                     "username" => "username",
15                     "password" => "password",
16                     "status"   => "status",
```

```
                        "role"      => "role",
                        "crypt"     => "sha1"
                    ),
                    // definicio de rols
                    "roles" => array (
                        "multiple" => true,
                        "table"    => "role",
                        "join"     => "user_has_roles",
                        "user"     => "username",
                        "role"     => "role"
                    ),
                    // codis d'error
                    "codes" => array (
                        "success"   => 200,
                        "forbidden" => 403,
                        "blocked"   => 402,
                        "error"     => 500
                    ),
                    // definicio de les dades d'usuari
                    "usersource" => array (
                        "table"    => "user",
                        "username" => "username",
                        "columns"  => array (
                            "username",
                            "email",
                            "name",
                            "language",
                            "theme",
                            "display_name",
                            "status"
                        )
                    )
                ),
                "file" => array (
                    "lengths" => array (
                        "min" => 6,
                        "max" => 50
                    ),
                    "datasource" => array (
                        "type"     => "htpasswd",
                        "filename" => "framework/.htpasswd",
                        "crypt"    => "sha"
                    ),
                    "roles" => array (
                        "multiple" => true,
                        "table"    => "role",
                        "join"     => "user_has_roles",
                        "user"     => "username",
                        "role"     => "role"
                    ),
                    "codes" => array (
                        "success"   => 200,
                        "forbidden" => 403,
                        "blocked"   => 402,
```

```
71                          "error"     => 500
72                      ),
73                      "usersource" => array (
74                          "table"    => "user",
75                          "username" => "username",
76                          "columns"  => array (
77                              "username",
78                              "email",
79                              "name",
80                              "language",
81                              "theme",
82                              "display_name",
83                              "status"
84                          )
85                      )
86                  ),
87                  "imap" => array (
88                      "lengths" => array (
89                          "min" => 6,
90                          "max" => 50
91                      ),
92                      "datasource" => array (
93                          "type"     => "imap",
94                          "host"     => "imap.gmail.com",
95                          "port"     => "993"
96                      ),
97                      "roles" => array (
98                          "multiple" => true,
99                          "table"    => "role",
100                         "join"     => "user_has_roles",
101                         "user"     => "username",
102                         "role"     => "role"
103                     ),
104                     "codes" => array (
105                         "success"   => 200,
106                         "forbidden" => 403,
107                         "blocked"   => 402,
108                         "error"     => 500
109                     ),
110                     "usersource" => array (
111                         "table"    => "user",
112                         "username" => "username",
113                         "columns"  => array (
114                             "username",
115                             "email",
116                             "name",
117                             "language",
118                             "theme",
119                             "display_name",
120                             "status"
121                         )
122                     )
123                 )
124             ),
```

```
125             "global" => array (
126                 "default" => "database"
127             )
128     );
129
130     FW_Config::createConfig("authentication");
131     FW_Config::setConfig("authentication",$config);
132     ?>
```

En aquest codi font es poden distingir tres seccions que corresponen amb els tres mètodes d'autenticació descrits abans.
Generalment utilitzarem només autenticació via bases de dades (a menys que hi haja algun sistema d'autenticació antic o corporatiu).

13.3 Creació de la taula d'usuaris

Amb el següent codi SQL podrem crear la taula d'usuaris en la nostra base de dades.

Listing 13.2: Ordres SQL per crear les taules de dades on es desarà la informació de l'usuari

```
1   CREATE TABLE user (
2     username VARCHAR(50) NOT NULL PRIMARY KEY,
3     password VARCHAR(50) NOT NULL,
4     email VARCHAR(50) NOT NULL,
5     name VARCHAR(50) NOT NULL,
6     activation_key VARCHAR(50) NOT NULL,
7     date_register DATETIME,
8     language VARCHAR(10) NOT NULL,
9     theme VARCHAR(50) NOT NULL DEFAULT 'default',
10    display_name VARCHAR(128) NOT NULL,
11    status INTEGER(1) NOT NULL DEFAULT 0
12  );
13
14  CREATE TABLE user_has_roles (
15    username VARCHAR(50) NOT NULL,
16    role VARCHAR(20) NOT NULL,
17    PRIMARY KEY(username,role)
18  );
19
20  CREATE TABLE role (
21    role VARCHAR(20) NOT NULL PRIMARY KEY,
22    enabled INTEGER(1) NOT NULL DEFAULT 1,
23    description TEXT
24  );
```

Nota: Les contrasenyes són encriptades mitjançant els algorismes **SHA1** o **MD5**, consulteu el fitxer de configuració del component Authentication per saber quin algorisme emprar al desar la contrasenya.

13.4 Autenticar un usuari

Per autenticar un usuari utilitzarem el següent snippet de codi:

Listing 13.3: Component *Authentication*: Snippet per autenticar usuaris

```php
<?php
  /* en un metode d'un controlador */

  // creem unes noves credencials
  $credentials = new FW_Authentication_Credentials("usuari",
      "contrasenya");

  // creem un contenedor de parametres i li assignem valors
  $parameters  = new FW_Container_Parameter();
  $parameters->credentials = $credentials;
  $parameters->type        = "database";
  $parameters->rules       = "database";

  // creem la instancia del component FW_Authentication
      configurant-la amb les dades del contenedor de
      parametres
  $auth        = new FW_Authentication($parameters);

  // cridem a la operacio login (200 es el codi d'error per
      indicar que el proces va terminar satisfactoriament)
  if ($auth->login()===200) {
    // autenticat correctament

    // obtenim el usuari loggejat en el sistema
    $user = $auth->getUser();
  }
  else {
    // error en usuari o contrasenya o ambdos
  }
?>
```

Capítol 14

Els entorns

14.1 Introducció

Un entorn és una sèrie de regles que ha de seguir el marc de treball a l'hora de l'execució de les accions que composen la nostra aplicació web. Un entorn defineix quines bases de dades utilitzar, quin sistema de caché utilitzar, com ha de ser el loguejat de les dades de depuració, ...

14.2 Tipus d'entorns

Generalment es defineixen tres entorns:

development : És un entorn que té actiu totes les dades de depuració i loguejat, que no utilitza caché i que té una base de dades amb dades per fer proves.
Aquest entorn s'utilitza exclusivament per programar les funcionalitats de les nostres aplicacions web.

testing : Semblant a l'entorn de desenvolupament, però que aplica a més a més la caché i una base de dades en les que existeixen dades de prova.
Aquest entorn és utilitzat prèviament a llançar una aplicació web i mentre aquesta està en proves.

production : Aquest entorn té inactives totes les funcionalitats de loguejat i depuaració (a excepció de les pàgines sense permís per executar-se (403) i les no trobades (404)). La base de dades és la definitiva, la que conté les dades que són utilitzades a cada petició i que són reals. La caché està activa al màxim per ajudar a reduir el nombre de peticions web.
Aquest entorn s'utilitza quan la web està en producció, és a dir, en funcionament.

14.3 Configuració dels entorns

El component *FW_Environment* s'encarrega de coordinar els diferents entorns i d'obtindre la informació corresponent a cada necessitat del marc de treball consulta les dades de la configuració del fitxer *environment.php* .

Listing 14.1: Configuració del component *FW_Environment*

```php
<?php
$config = array(
    "sections" => array (
        "develop"   => array (
             "log"      => array("all"),
             "debug"    => array("all"),
             "error"    => array("all"),
             "database" => array("default"),
             "cache"    => array()
        ),
        "testing"   => array (
             "log"      => array("all"),
             "debug"    => array("all"),
             "error"    => array("all"),
             "database" => array("default"),
             "cache"    => array("default")
        ),
        "production"   => array (
             "log"      => array("error"),
             "debug"    => array(),
             "error"    => array("fatal"),
             "database" => array("default"),
             "cache"    => array("default")
        ),
    ),
    "global"    => array (
        "environmentInUse" => "develop"
    )
);
FW_Config::createConfig("environment");
FW_Config::setConfig("environment",$config);
?>
```

Cada entrada en «sections» es refereix a un entorn.

Capítol 15

Programació d'accions

15.1 Introducció

En aquest capítol veurem com es fa per programar accions de diversos tipus i generar un resultat.
Donada la llargària d'aquesta memòria, només es presentarà la construcció d'unes poques accions, a partir d'aquestes accions i dels capítols corresponents a serveis web i el capítol de tasques programades, el lector estarà familiaritzat amb el marc de treball i podrà programar qualsevol tipus d'aplicació web per complicada que siga.
A causa - també - de l'espai disponible, no es generarà cap codi Java Script (el lector pot emprar el framework que més conega per els seus desenvolupaments; basta amb carregar els fitxers bàsics amb el layout i els fitxers per a cada funcionalitat a demanda i definits en les definicions de cada ruta), però sí els estils CSS ja que òbviament són necessaris per mostrar el resultat en el navegador.

15.2 Estructura de la demostració

Per aquesta demostració crearem un mòdul anomenat **index** amb un controlador i el seu corresponent model, sobre el que treballarem i utilitzarem com a suport en aquesta demostració.

Després de crear el mòdul definirem el model dades que emprarem. Per a això definirem els models i mostrarem els codis SQL per crear les taules referides als següents models:

post : Les entrades que mostrarem.

comment : Comentaris de les entrades.

user : Usuari del sistema.

role : Rol d'un usuari del sistema.

Listing 15.1: Model de dades per a la demostració

```
1  <?php
2
3  // /app/lib/models/comment.class.php
```

```
4   class comment extends FW_ActiveRecord_Model {
5
6     protected $id;
7     protected $id_post;
8     protected $author;
9     protected $created_at;
10    protected $content;
11
12
13    public static $belongs_to = array (
14      array(
15      "property" => "post",
16      "table"    => "post",
17      "srcColumn"=> "id_post",
18      "dstColumn"=> "id",
19      "update"   => "cascade",
20      "delete"   => "cascade"
21      )
22    );
23
24    public static $has_one = array (
25      array(
26      "property" => "author",
27      "table"    => "user",
28      "srcColumn"=> "author",
29      "dstColumn"=> "username",
30      "update"   => "restrict",
31      "delete"   => "restrict"
32      )
33    );
34
35    public function getDate() {
36      return timeHelper::getFullHumanDate($this->created_at);
37    }
38  };
39
40
41  // /app/lib/models/post.class.php
42  class post extends FW_ActiveRecord_Model {
43
44      protected $id;
45      protected $title;
46      protected $excerpt;
47      protected $content;
48      protected $created_at;
49      protected $author;
50      protected $is_commentable;
51      protected $permalink;
52      protected $status;
53
54      public function isCommentable() {
55          if ($this->is_commentable==="1") {
56              return true;
57          }
```

```
58            return false;
59        }
60
61        public function getAuthor() {
62            return $this->author->first()->getDisplayName();
63        }
64
65        public function getDate() {
66            return timeHelper::getFullHumanDate($this->
                  created_at);
67        }
68
69        public function getImage() {
70            $image   = "";
71            $pattern = '/<\s*img [^\>]*src\s*=\s
                  *[\""\']?([^\""\'\s>]*)/i';
72            if (preg_match($pattern,(string) $this->content,
                  $matches)) {
73                $image = $matches[1];
74            }
75            if (empty($image)) {
76                $image = BASE_URL."/images/noimage.png";
77            }
78            return $image;
79        }
80
81        public static $has_one = array (
82            array(
83                "property" => "author",
84                "table"    => "user",
85                "srcColumn"=> "author",
86                "dstColumn"=> "username",
87                "update"   => "restrict",
88                "delete"   => "restrict"
89            )
90        );
91
92        public static $has_many = array (
93            array(
94                "property" => "comments",
95                "table"    => "comment",
96                "srcColumn"=> "id",
97                "dstColumn"=> "id_post",
98                "update"   => "cascade",
99                "delete"   => "cascade"
100           )
101       );
102
103   };
104
105
106   // /app/lib/models/role.class.php
107   class role extends FW_ActiveRecord_Model {
108
```

```
109     protected $role;
110     protected $enabled;
111   };
112
113   // /app/lib/models/user.class.php
114   class user extends FW_ActiveRecord_Model {
115
116
117       protected $username;
118       protected $password;
119       protected $email;
120       protected $role;
121       protected $name;
122       protected $activation_key;
123       protected $date_register;
124       protected $language;
125       protected $theme;
126       protected $display_name;
127       protected $status;
128       protected $url;
129       protected $bio;
130       protected $image;
131       protected $notifications;
132
133       public static $has_and_belongs_to_many = array (
134         array(
135                 "property"  => "role",
136                 "srcTable"  => "user",
137                 "srcColumn" => "username",
138                 "dstTable"  => "role",
139                 "dstColumn" => "role",
140                 "throughTable" => "user_has_roles",
141                 "throughTableSrcColumn" => "username",
142                 "throughTableDstColumn" => "role",
143                 "update" => "restrict",
144                 "delete" => "restrict"
145             )
146       );
147
148       public function getDisplayName() {
149           return $this->display_name;
150       }
151
152       public function setPreferences($preferences) {
153
154           if (isset($preferences["name"]) && strlen(
                $preferences["name"])>0) {
155               $this->name = $this->_filter->sanitizeString(
                    $preferences["name"],true);
156           }
157
158           if (isset($preferences["email"]) && strlen(
                $preferences["email"])>0) {
159               $this->email = $this->_filter->sanitizeString(
```

```
                    $preferences["email"]);
160         }
161
162         if (isset($preferences["language"]) && strlen(
                $preferences["language"])>0) {
163             $this->preferredLanguage = $this->_filter->
                    sanitizeString($preferences["language"]);
164         }
165
166         if (isset($preferences["notifications"]) && strlen(
                $preferences["notifications"])>0) {
167             if ( ($preferences["notifications"]=="on") || (
                    $preferences["notifications"]=="off") ) {
168                 $this->notifications = $preferences["
                        notifications"];
169             }
170         }
171
172         if ($this->validateData()) {
173             $this->save();
174             return true;
175         }
176         else {
177             return false;
178         }
179     }
180
181     public function getName() {
182         return utf8_decode($this->name);
183     }
184
185
186     public function getRole() {
187
188         $roles = array();
189         if ($this->role instanceof FW_ActiveRecord_Relation)
                {
190             foreach ($this->role as $role) {
191                 $roles []= $role->role;
192             }
193             return implode(' ',$roles);
194         }
195
196         return $this->role;
197     }
198
199 };
200
201
202
203 ?>
```

15.3 Definició del layout

Definirem el layout estàndard per a les demostracions com a un layout que conté dos columnes (menú a l'esquerra, contingut a la dreta), capçalera i peu de pàgina. Aquest layout té el següent aspecte:

Listing 15.2: Layout per a les demostracions

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
        charset=utf-8" />
    <title>Blog</title>
    <?php
      // mostrar els estils CSS per aquest modul
      print FW_Style_Manager::getInstance()->displayStyle(
          FW_Context::getInstance()->router->route);

      // mostrar els estils CSS per aquesta accio
      $styles = FW_Context::getInstance()->router->styles;
      if (count($styles)>0) {
        foreach ($styles as $style) {
          $file  = $style["file"];
          $media = $style["media"];
          print style_tag($file,$media).'\n';
        }
      }
    ?>
  </head>

  <body>
    <div id="wrapper">

      <div id="container" class="container">

        <!-- Capsalera -->
        <div id="header" class="container">
          <h1>Blog</h1>
        </div>

        <!-- Menu lateral -->
        <div id="sidebar" class="container span-5 column">
          <?php
            if ($this->hasSlot("sidebar")) {
              print $this->getSlot("sidebar");
            }
            else {
              $this->renderView("sidebar/default");
            }
          ?>
        </div>

         <!-- Zona de contingut -->
```

```
45          <div id="content" class="container span-16 column
                last">
46            <?php
47            // si existeix el slot a mostrar ...
48            if ($this->hasSlot("content")) {
49                // imprimir els seus contingut
50                print $this->getSlot("content");
51            }
52            ?>
53          </div>
54
55          <hr class="space" />
56        </div>
57      </div>
58
59      <!-- Peu de pagina -->
60      <div id="footer" class="container">
61        <p>Blog</p>
62      </div>
63
64
65      <?php
66      // mostrar els scripts per aquesta accio
67      $scripts = FW_Context::getInstance()->router->scripts;
68      if (count($scripts)>0) {
69          foreach ($scripts as $script) {
70              print html::script_tag($script);
71          }
72      }
73      ?>
74      </body>
75    </html>
```

Aquest layout el desarem en el directori *layout* de cada mòdul.

15.4 Programació de les accions «veure últims 5 entrades» i «veure entrada»

Aquestes dos accions s'encarreguen de mostrar entrades.

15.4.1 Acció «veure últims 5 entrades»

Aquesta acció mostrarà els darrers 5 entrades inserits en el sistema:

Listing 15.3: Codi font per a l'acció index (veure últimes 5 entrades)

```
1   <?php
2   // accio index (indexContorller.class.php)
3
4   public function index() {
5       // obtindre els 5 posts mes recents
6       $posts = $this->model()->getLatestsPosts();
7       // si hi ha posts recents
```

```
8       if ($posts->hasResult()) {
9           $this->setSlot("content","posts/posts",array("posts"
                =>$posts));
10      }
11      else {
12          $this->setSlot("content","posts/noPosts");
13      }
14      // renderitzar el layout per defecte
15      $this->renderLayout("default");
16  }
17
18
19  // indexModel.class.php
20  public function getLatestsPosts() {
21    // condicions: el post estiga publicat
22    $conditions = array (
23        array (
24            "name"     => "status",
25            "operator" => "=",
26            "value"    => "1"
27        )
28    );
29    // ordre de creacio per data descendent
30    $orders = array (
31        array (
32            "column" => "created_at",
33            "type"   => "DESC"
34        )
35    );
36
37    // cercar les entrades
38    $posts = post::find($conditions,$orders,array(),array(),5)
          ;
39    return $posts;
40  }
41  ?>
```

15.4.2 Acció «veure entrada»

Aquesta acció mostrarà una entrada basant-se en el seu atribut *permalink* que és únic:

Listing 15.4: Codi font per a l'acció viewPost (veure entrada)

```
1   <?php
2
3   // accio viewPost (indexController.class.php)
4   public function viewPost($permalink) {
5       // decodificar la url del permalink
6       $permalink = urldecode($permalink);
7
8       // cercar l'entrada amb el permalink
9       $post      = $this->model()->getPostByPermalink(
            $permalink);
10
```

```
11        // si existeix eixa entrada, mostrar-la
12        if ($post!==null) {
13            $this->setSlot("content","posts/post",array("post"=>
                  $post));
14        }
15        else {
16            $this->setSlot("content","posts/notFound");
17        }
18        // renderitzar el layout per defecte
19        $this->renderLayout("default");
20    }
21
22    // indexModel.class.php
23    public function getPostByPermalink($permalink) {
24        // condicions de cerca (permalink i estat->publicat)
25        $conditions = array (
26            array (
27                "name"      => "permalink",
28                "operator" => "=",
29                "value"     => $permalink
30            ),
31            array (
32                "condition" => "AND"
33            ),
34            array (
35                "name"      => "status",
36                "operator" => "=",
37                "value"     => "1"
38            )
39        );
40
41        // cercar l'entrada
42        $post = post::find($conditions);
43        // si hi ha resultats ...
44        if ($post->hasResult()) {
45            // retornar el primer resultat
46            return $post->first();
47        }
48    }
49
50    ?>
```

15.4.3 Vistes

A continuació es detallen les vistes per l'acció index:

Listing 15.5: Vistes per l'acció index

```
1   <?php
2
3   // vista posts/posts.php
4     <div class="span-16 last">
5       <h2>Benvingut al blog</h2>
6       <hr />
```

```
7         <p>Lorem ipsum dolor sit amet...</p>
8       </div>
9
10      <div class="span-16 last">
11        <h3>Entrades</h3>
12        <hr />
13      </div>
14
15      <?php
16      // mostrar per a cada entrada un breu contingut
17      foreach ($posts as $post) {
18          $this->renderView("posts/excerpt",array("post"=>$post)
                );
19      }
20      ?>
21
22
23
24    // vista posts/excerpt.php
25    <div class="span-16 container last">
26      <div class="span-2 column">
27              <img src="<?php print $post->getImage(); ?>" alt="
                    <?php print $this->display($post->title); ?>"
                    width="75" />
28      </div>
29      <div class="span-14 column last">
30              <h4><?php print html::link_to_internal("index","
                    index","viewPost",$this->display($post->title),
                    array("permalink"=>$post->permalink));?></h4>
31              <p class="alt"><?php print _("Per ").html::
                    link_to_internal("user","user","
                    displayUserProfile", $this->display($post->
                    author->first()->display_name),array("username"
                    =>$post->author->first()->username))._(" el ").
                    $post->getDate(); ?></p>
32      </div>
33
34      <div class="span-16 container last">
35              <?php print $this->display($post->excerpt); ?>
36      </div>
37
38      <div class="span-12 column">  </div>
39      <div class="span-4 last">
40              <p>(<?php print count($post->comments); ?>)
                    comentaris | <?php print html::link_to_internal
                    ("index","index","viewPost","Llegir mes",array(
                    "permalink"=>$post->permalink));?></p>
41
42      </div>
43    </div>
44
45
46    // vista noPosts.php
47    <div class="span-16 last">
```

```
48      <h3>Error</h3>
49      <hr />
50      <p>Malauradament no disposem d'entrades per mostrar. Torne
            en uns dies a visitar-nos!</p>
51  </div>
52
53  ?>
```

I també les de l'acció viewPost:

Listing 15.6: Vistes per l'acció viewPost

```
1   <?php
2   // vista posts/post.php
3   <div class="span-16 container last">
4     <div class="span-2 column">
5             <img src="<?php print $post->getImage(); ?>" alt="
                  <?php print $this->display($post->title); ?>"
                  width="75" />
6     </div>
7     <div class="span-14 column last">
8             <h2><?php print html::link_to_internal("index","
                  index","viewPost",$this->display($post->title),
                  array("permalink"=>$post->permalink));?></h2>
9             <p class="alt"><?php print _("Per ").html::
                  link_to_internal("user","user","
                  displayUserProfile", $this->display($post->
                  author->first()->display_name),array("username"
                  =>$post->author->first()->username)._(" el ").
                  $post->getDate(); ?></p>
10    </div>
11
12    <div class="span-16 container last">
13            <?php print $this->display($post->content); ?>
14    </div>
15  </div>
16
17
18  <?php
19      // si es commentable ...
20      if ($post->is_commentable || count($post->comments)>0) {
21          $this->renderView("posts/comments/comments",array("
                comments"=>$post->comments));
22      }
23      if ($post->is_commentable) {
24          if ($this->user()!==false) {
25              $this->renderView("posts/comments/commentForm",
                    array("id_post"=>$post->id));
26          }
27      }
28  ?>
29
30
31
32
```

```
33  // vista posts/notFound.php
34  <div class="span-16 last">
35    <h3>Error</h3>
36    <hr />
37    <p>Lentrada que ha sol.licitat no existeix o no esta
          disponible</p>
38  </div>
39
40
41
42  // vista posts/comments/comments.php
43  <hr />
44  <hr class="space" />
45  <div class="span-16 container last">
46  <h2>Comentaris</h2>
47  <hr />
48  <?php
49  if (count($comments)===0)  { ?>
50      <div class="span-16 last">
51          <p>Aquesta entrada no te comentaris</p>
52      </div>
53  <?php
54  }
55  else {
56      foreach ($comments as $comment) {
57          $this->renderView("posts/comments/comment",array("
                comment"=>$comment));
58          ?>
59          <hr class="space" />
60          <?php
61      }
62  }
63  ?>
64  <hr class="space" />
65  </div>
66
67
68  // vista posts/comments/comment.php
69  <div class="span-12 prepend-2 last">
70    <p class="alt"><?php print _("Per ").html::
          link_to_internal("user","user","displayUserProfile",
          $this->display($comment->author->first()->display_name)
          ,array("username"=>$post->author->first()->username))._
          (" el ").$comment->getDate(); ?></p>
71    <br/>
72    <?php print $this->display($comment->content); ?>
73  </div>
74
75  ?>
```

15.4.4 Captures de pantalla

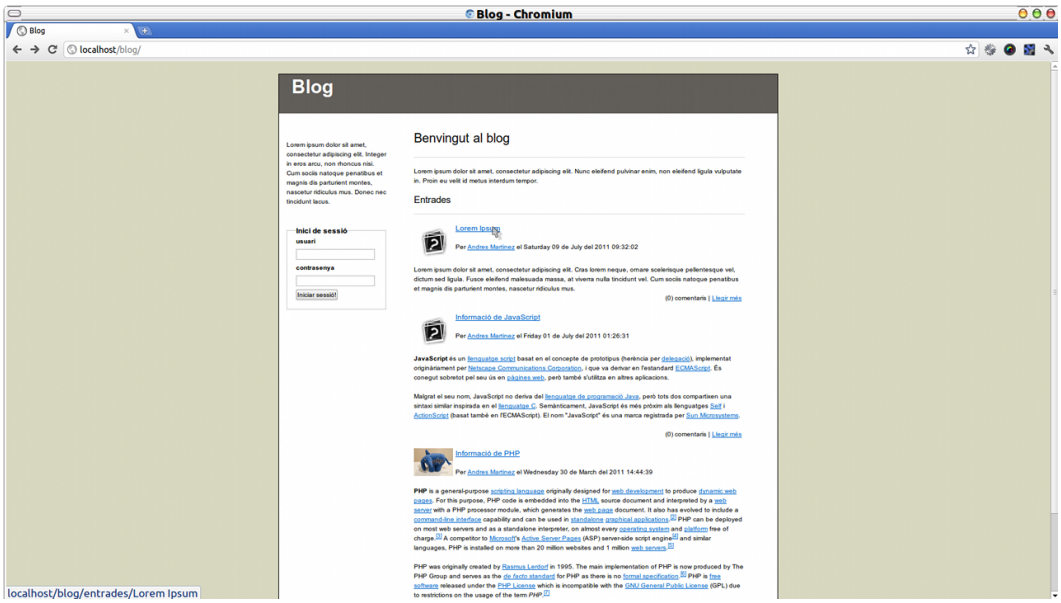


Figura 15.1: Acció index

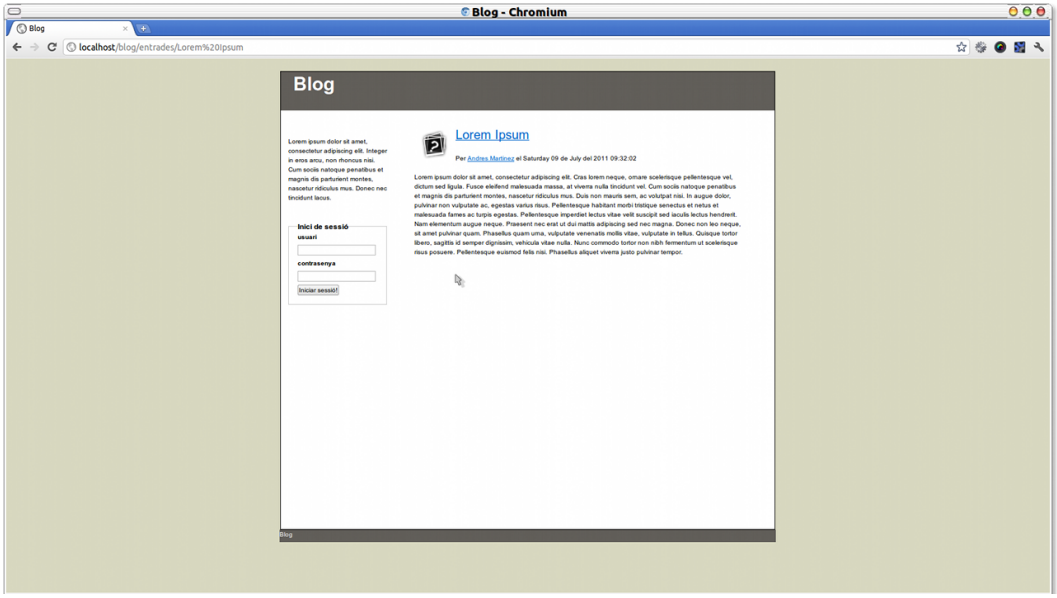


Figura 15.2: Acció viewPost

Figura 15.3: Acció viewPost (zona comentaris)

15.4.5 Rutes

Per connectar les accions al sistema utilitzarem aquestes dos rutes:

Listing 15.7: Rutes per a les accions index i viesPost

```
1  <?php
2  // rutes per a les accions index i viewPost
3
4  FW_Router::Connect(
5  array (
6    'url' => '/',
7    'type' => 'app',
8    'cache' => false,
9    'authentication' => false,
10   'module' => 'index',
11   'controller' => 'index',
12   'action' => 'index',
13   'internal' => false,
14   'styles' =>
15   array (
16   ),
17   'scripts' =>
18   array (
19     0 => 'js/application/index/main.js',
20   ),
21   'title' => 'Blog',
22   'pattern' => '#^/[/]*$#',
23   'parameterOrder' =>
24   array (
```

```
25      ),
26    )
27    );
28
29    FW_Router::Connect(
30    array (
31      'url' => '/entrades/:permalink',
32      'type' => 'app',
33      'cache' => false,
34      'parameters' =>
35      array (
36        'permalink' =>
37        array (
38          'name' => 'permalink',
39          'type' => 'string',
40          'format' => false,
41        ),
42      ),
43      'authentication' => false,
44      'module' => 'index',
45      'controller' => 'index',
46      'action' => 'viewPost',
47      'internal' => false,
48      'styles' =>
49      array (
50      ),
51      'scripts' =>
52      array (
53      ),
54      'pattern' => '#^/entrades/(?:([^/]+))[/]*$#',
55      'parameterOrder' =>
56      array (
57        0 => 'permalink',
58      ),
59    )
60    );
61    ?>
```

15.5 Programació de les accions «iniciar sessió» i «tancar sessió»

Aquestes dos accions són accions que en general tenen totes les aplicacions web pensades per fer-se servir per varis usuaris.

15.5.1 Acció iniciar sessió (login)

Aquesta acció permetrà a l'usuari iniciar sessió en el sistema i per tant accedir a les zones a les que es necessita autenticació o a les zones restringides a usuaris autenticats en el sistema.
L'acció la programarem en el controlador *indexController* del mòdul index. Espera dos paràmetres enviats via POST per un formulari, una vegada obtinguts aquestos

paràmetres es «netegen» i es passen al component *Authentication* per a que intente autenticar l'usuari.
Si l'autenticació és satisfactòria, la pàgina es recarregarà i donarà la benvinguda a l'usuari, al que se li permetrà entrar en zones restringides.

Listing 15.8: Codi font per a l'acció login (iniciar sessió)

```php
<?php
// accio login (indexController.php)
public function login() {

  // obtindre el parametre username via POST
  $username = $this->request()->post("username");
  // netegem el valor del parametre
  $username = $this->sanitize($username);

  // obtindre el parametre password via POST
  $password = $this->request()->post("password");
  // netegem el valor del parametre
  $password = $this->sanitize($password);

  // creem unes noves credencials
  $credentials = new FW_Authentication_Credentials($username
      ,$password);

  // creem un contenedor de parametres i li assignem valors
  $parameters  = new FW_Container_Parameter();
  $parameters->credentials = $credentials;
  $parameters->type        = "database";
  $parameters->rules       = "database";

  // creem la instancia del component FW_Authentication
      configurant-la amb les dades del contenedor de
      parametres
  $auth        = new FW_Authentication($parameters);

  // cridem a la operacio login

  //  Si login retorna codi (200 es el codi d'error per
      indicar que el proces va terminar satisfactoriament)
  if ($auth->login()===200) {
      $url = BASE_URL;
      header("Location: {$url}");
  }
  // en cas contrari el usuari o la contrasenya, o ambdos
      son incorrectes
  else {
      $url = BASE_URL;
      header("Location: {$url}");
  }
}
?>
```

15.5.2 Tancar sessió (logout)

Aquesta acció permet a l'usuari tancar la sessió en el sistema. Una vegada tancada la sessió en el sistema, l'usuari haurà de tornar a fer login si vol accedir a les zones restringides.

Listing 15.9: Codi font per a l'acció logout (tancar sessió)

```php
<?php
// accio logout (indexController.php)
public function logout() {
  // indicar al component Authentication que destrueix-ca la
       sessio
  FW_Authentication::getInstance()->logout();

  // redirigir a la pagina principal
  $url = BASE_URL;
  header("Location: {$url}");
}

?>
```

15.5.3 Rutes

Per connectar les accions al sistema utilitzarem aquestes dos rutes:

Listing 15.10: Rutes per a les accions login (iniciar sessió) i logout (tancar sessió)

```php
<?php
// rutes per a les accions login i logout

FW_Router::Connect(
array (
  'url' => '/login',
  'type' => 'app',
  'cache' => false,
  'parameters' => array(),
  'authentication' => false,
  'module' => 'index',
  'controller' => 'index',
  'action' => 'login',
  'internal' => false,
  'styles' =>
  array (
  ),
  'scripts' =>
  array (
  ),
  'pattern' => '#^/login[/]*$#',
  'parameterOrder' => array()
)
);

FW_Router::Connect(
```

```
27   array (
28     'url' => '/logout',
29     'type' => 'app',
30     'cache' => false,
31     'parameters' => array(),
32     'authentication' => false,
33     'module' => 'index',
34     'controller' => 'index',
35     'action' => 'logout',
36     'internal' => false,
37     'styles' =>
38     array (
39     ),
40     'scripts' =>
41     array (
42     ),
43     'pattern' => '#^/logout[/]*$#',
44     'parameterOrder' => array()
45   )
46   );
47
48   ?>
```

15.5.4 Vistes per aquestes accions

La següent vista desencadena l'acció login:

Listing 15.11: Vista per l'acció login (iniciar sessió)

```
1   <div class="span-5">
2           <?php
3               print form::formTag("login",html::
                    link_for_internal("index","index","login",
                    array(),false),"post");
4               print form::openFieldsetTag();
5               print form::legendTag("Inici de sessio");
6       ?>
7           <div class="span-4 last">
8                   <?php print form::labelTag("username","
                        usuari");?>
9           </div>
10          <div class="span-4 last">
11                  <?php print form::textInput("username","","
                        span-4"); ?>
12          </div>
13
14          <div class="span-4 last">
15                  <?php print form::labelTag("password","
                        contrasenya");?>
16          </div>
17          <div class="span-4 last">
18                  <?php print form::passwordInput("password","
                        ","span-4"); ?>
19          </div>
```

```
20
21          <?php
22              print form::submitInput("submit","Iniciar sessio!");
23              print form::closeFieldsetTag();
24              print form::closeFormTag();
25              ?>
26      </div>
```

La següent vista desencadena l'acció logout:

Listing 15.12: Vista per l'acció logout (tancar sessió)

```
1   <p>Benvingut al sistema!
2   <br/>
3   <strong><?php print $this->display($this->user()->
        display_name); ?></strong>
4   <br/>
5   <?php print html::link_to_internal("index","index","logout",
        "Tancar sessio"); ?>
6   </p>
```

15.5.5 Captures de pantalla

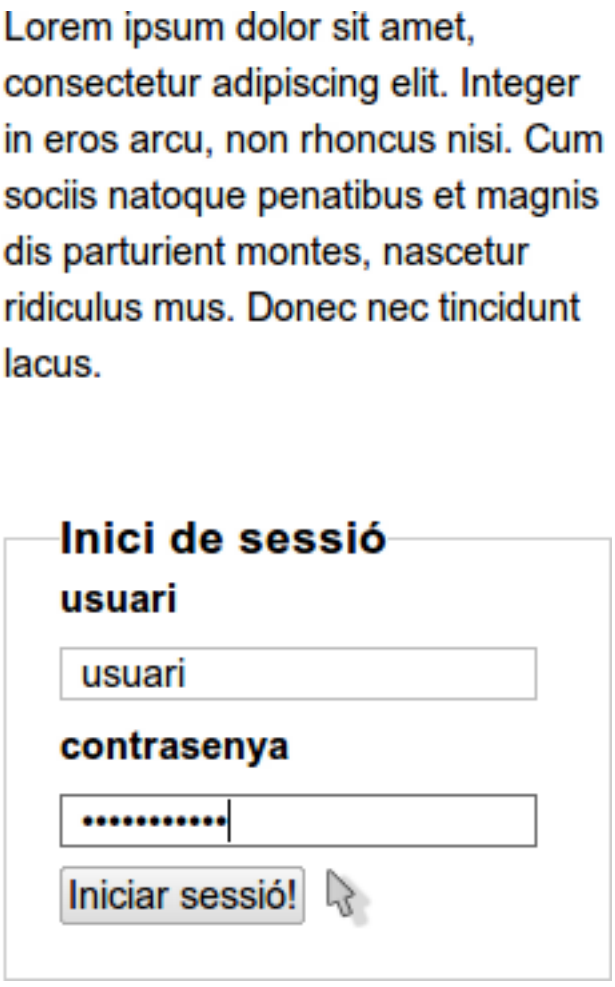


Figura 15.4: Acció login

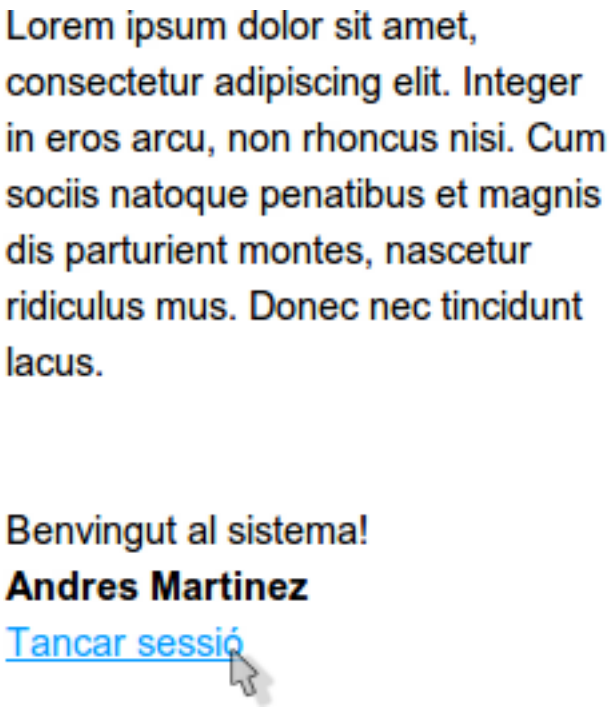


Figura 15.5: Acció logout

Part V

Desenvolupament de plugins i helpers

Capítol 16

Introducció

16.1 Introducció

En aquesta part es presente una sèrie de components que ajuden al marc de treball en el desenvolupament. Primerament es presenten els *Plugins*, seguidament els *Widgets* i per finalitzar es presenten els *helpers*.

16.2 Creació del mòdul a utilitzar en les desmostracions

Per al desenvolupament d'aquest capítol crearem un nou mòdul anomenat **plugin** i el situarem sota el directori `app`. En el mòdul acabat de crear, crearem un controlador anomenat *pluginController* amb el seu corresponent model.

A continuació s'exposa el codi font de cadascun d'aquestos controladors i els seus corresponents models:

Listing 16.1: Codi font per crear els controladors i models per a les demostracions

```
1  <?php
2    // pluginController.class.php
3    class pluginController extends FW_mvc_BaseController {
4
5    };
6
7    // pluginModel.class.php
8    class pluginModel extends FW_mvc_BaseModel {
9
10   };
11 ?>
```

16.2.1 Creació del ≪layout≫ per a les demostracions

A continuació crearem el ≪layout≫ a emprar en les demostracions. Crearem un fitxer de text amb el següent codi font que desarem com a `/app/modules/plugin/demo/default.php`

Listing 16.2: Layout utilitzat per a les demostracions

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
        charset=utf-8" />
    <title>Demo de plugins</title>

    <?php
        $params  = FW_Context::getInstance()->router->route;
        $manager = FW_Style_Manager::getInstance();
        print $manager->displayStyle($params);

        $styles = FW_Context::getInstance()->router->styles;
        if (count($styles)>0) {
            foreach ($styles as $style) {
                print style_tag("{$style["file"]}",$style["
                    media"]).'\n';
            }
        }
    ?>
  </head>

  <body>
    <div id="wrapper">
      <div id="container" class="container">

        <div id="header" class="container">
          <h1>Plugins</h1>
        </div>

        <div id="content" class="container">
          <?php
            if ($this->hasSlot("content")) {
              print $this->getSlot("content");
            }
          ?>
        </div>

        <hr class="space" />
      </div>
    </div>

    <div id="footer" class="container">
      <p>Demo de plugins</p>
    </div>
  </body>
</html>
```

16.2.2 Creació dels estils CSS per a les demostracions

Per últim, crearem els estils *CSS* necessaris per a la demostració. Aquest codi el desarem en el fitxer `/style/default/application/plugin/style.css`

Listing 16.3: CSS per a les demostracions

```
* {
  margin: 0;
  padding: 0;
}

html, body, #wrapper {
  height: 95%;
}

body {
  background: none repeat scroll 0 0 #D8D8C0;
  color: #000000;
}

#wrapper {
  height: auto;
  min-height: 95%;
  margin: 0 auto;
  width: 950px;
  margin-top: 25px;
  margin-bottom: 25px;
  background: none repeat scroll 0 0 #FFFFFF;
  border: 1px solid #000000;
}

#content {
  padding: 20px;
}
#header {
  background: #9976c0;
  height: auto;
  color: #FFFFFF;
}

#header h1 {
  color: #FFFFFF;
  font-size: 35px;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: bolder;
}

#header a {
  color: #FFFFFF;
}

#header a:hover {
  color: #000000;
  background: #FFFFFF;
```

```
49  }
50
51  #footer {
52    background: #9976c0;
53    color: #FFFFFF;
54    position: relative;
55    margin-top: -25px;
56    height: 25px;
57    clear: both;
58    width: 955px;
59  }
60
61  #content {}
```

També modificarem el fitxer que carrega els estils amb el component *Style Manager*, en concret hem de modificar el fitxer */style/default/style.css* i afegir-li en la clau *modules* → *external* un array que descriurà els estils a carregar per a les accions del mòdul *plugin*

Listing 16.4: Definicions d'estils per el mòdul plugin

```
1  <?php
2
3  // definicio d'estils per altres moduls externs
4  "webservices"  => array (
5    "default"=> array (
6      0 => array("file"=>"reset.css","media"=>array("screen"),
           "alternate"=>false),
7      1 => array("file"=>"blueprint/screen.css","media"=>array
           ("screen"),"alternate"=>false),
8      2 => array("file"=>"application/plugin/style.css","media
           "=>array("screen"),"alternate"=>false)
9    )
10 ),
11 // altres definicions d'estils per a altres moduls externs
12 ?>
```

Capítol 17

Plugins

17.1 Introducció

En aquest capítol aprendrem a crear i utilitzar plugins. Crearem un plugin senzill de demostració que mostre tota la funcionalitat bàsica dels plugins i un plugin que genere mapes de Google Maps (generarà el codi font en Java Script del mapa).

Un plugin permet al desenvolupador estendre les funcionalitats d'una aplicació web ja siga mitjançant biblioteques externes, codis de tercers o funcionalitats pròpies. Aquestes noves funcionalitats, agrupades en forma de *Plugin* permeten al desenvolupador un accés ràpid i senzill a funcionalitats que no té de sèrie el marc de treball i que poden ser-li útils al desenvolupador en les seves aplicacions web. Exemples:

- Plugin creador de mapes de *Google Maps* per mostrar mapa junt a fitxa de clients.
- Plugin que envie i reba missatges SMS a través de serveis web.
- Plugin que mostre un feed RSS.
- Plugin que consumeix-ca l'API web de *Flickr* i obtinga fotografies.
- Plugin per a fer que l'aplicació web envie *twits* a *Twitter*.
- Plugin que permeta a l'aplicació web controlar qualsevol microcontrolador connectat al port sèrie.
- ...

Creant plugins evitarem tindre que repetir codi de funcionalitats, reutilitzarem codi font, crearem una API senzilla per a consumir o genera un servei web que té una API complicada o amagarem darrere d'un plugin un sistema per connectar un microcontrolador a la xarxa i que es puga utilitzar en les nostres aplicacions web.

El sistema de plugins del marc de treball es composa d'una classe anomenada *Plugin_Registry*, que carrega, gestiona i proporciona les funcionalitats necessàries per obtindre el Plugin que es necessite a cada moment. Plugin_Registry conté un registre de Plugins disponibles en el sistema i que poden utilitzar-se per realitzar qualsevol acció i obtindre un resultat a l'acció executada.

Cada Plugin escrit per aquest marc de treball ha d'estendre de la clase *Plugin_Base*, que proporciona funcionalitats de càrrega de fitxers o biblioteques, de càrrega dinàmica

de models de dades (els plugins *també poden utilitzar el sistema ActiveRecord*), de gestió de propietats/opcions de cada plugin dins la base de dades i de càrrega i processament d'informació del fitxer de descripció de cada plugin `options.php` i `plugin.php`. A més a més, proporciona mètodes que inicialitzen i configuren el plugin segons les opcions a utilitzar.

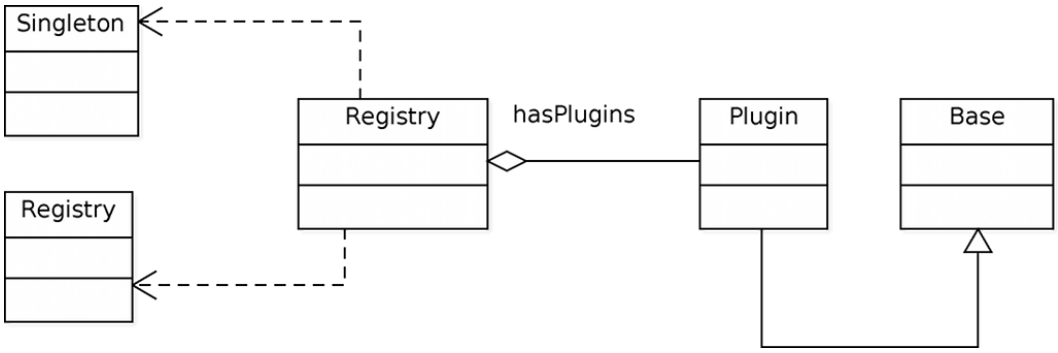


Figura 17.1: Components del sistema de Plugins

17.2 API de Plugins

A continuació es presenten les API del registre de plugins i del plugin base.

17.2.1 API de *Plugin_Registry*

L'API del component *Plugin_Registry* proveu al desenvolupador de cinc operacions bàsiques:

Obtindre un plugin :

Listing 17.1: API de Plugin_Registry: Obtindre un plugin

```
<?php
// obtindre instancia de FW_Plugin_Registry
FW_Plugin_Registry getInstance(void);

// obtindre un plugin
mixed getPlugin($name);

// obtenim la instancia del registre de plugins
$registry = FW_Plugin_Registry::getInstance();

// obtenim el plugin dessitjat
$plugin   = $registry->getPlugin("plugin");
?>
```

Instal.lar un plugin : Per instal·lar un plugin cal que el plugin tinga el mètode *install* definit i amb les instruccions necessàries per instal·lar (crear taules de la base de dades, opcions, ...) el plugin.

Listing 17.2: API de Plugin_Registry: Instal·lar un plugin

```
<?php

// obtindre instancia de FW_Plugin_Registry
FW_Plugin_Registry getInstance(void);

// instalar un plugin
mixed install($name,array $arguments=array());


// obtenim la instancia del registre de plugins
$registry = FW_Plugin_Registry::getInstance();

// instalem el plugin
$result   = $registry->install("plugin",array("a"=>"0","
    b"=>"1"));
?>
```

Desinstal.lar un plugin : Per desinstal·lar un plugin cal que el plugin tinga el mètode *uninstall* definit i amb les ordres necessàries per desinstal·lar (esborrar taules de la base de dades, opcions, fitxers, ...) el plugin.

Listing 17.3: API de Plugin_Registry: Desinstal·lar un plugin

```
<?php
// obtindre instancia de FW_Plugin_Registry
FW_Plugin_Registry getInstance(void);

// desintalar un plugin
mixed uninstall($name,array $arguments=array());

// obtenim la instancia del registre de plugins
$registry = FW_Plugin_Registry::getInstance();

// desinstalem el plugin
$result   = $registry->uninstall("plugin",array());
?>
```

Comprovar si existeix (i està carregat) un plugin :

Listing 17.4: API de Plugin_Registry: Comprovar si existeix (i està carregat) un plugin

```
<?php
// obtindre instancia de FW_Plugin_Registry
FW_Plugin_Registry getInstance(void);

// veure si existeix un plugin
bool hasPlugin($name);


// obtenim la instancia del registre de plugins
```

```
10  $registry = FW_Plugin_Registry::getInstance();
11
12  // veiem si existeix un plugin
13  if (!$registry->hasPlugin("plugin")) {
14    // error
15  }
16
17  ?>
```

Obtindre tots els plugins carregats en el sistema :

Listing 17.5: API de Plugin_Registry: Obtindre tots els plugins carregats en el sistema

```
1   <?php
2   // obtindre instancia de FW_Plugin_Registry
3   FW_Plugin_Registry getInstance(void);
4
5   // obtindre tots els plugins carregats en el sistema
6   array getPlugins(void);
7
8   // obtenim la instancia del registre de plugins
9   $registry = FW_Plugin_Registry::getInstance();
10
11  // obtenim els plugins
12  $plugins  = $registry->getPlugins();
13  ?>
```

17.2.2 API del *Plugin* base

L'API del plugin base podem dividir-la en quatre grups d'operacions:

Opcions/configuració utilitzant base de dades

Amb un plugin és possible emmagatzemar dades de configuració o valors d'opcions en la base de dades. Per a això disposem dels següents mètodes que ens faciliten la tasca:

Listing 17.6: API de Plugin: Opcions/configuració utilitzant la base de dades

```
1   <?php
2       // desar una opcio en base de dades
3       void setDBOption($name,$value);
4
5       // recuperar una opcio de base de dades
6       mixed getDBOption($name);
7
8       // en un metode d'un plugin
9       ...
10      $value  = $this->getDBOption("value");
11      $value += $increment;
12      $this->setDBOption("value",$value);
13      ...
14  ?>
```

La taula de la base de dades en la que es desen aquestes dades es pot crear mitjançant el següent codi SQL:

Listing 17.7: API de Plugin: Creació de la taula *blog_options* en la base de dades

```
CREATE TABLE plugin_option (
  plugin VARCHAR(50) NOT NULL,
  name   VARCHAR(50) NOT NULL,
  value  VARCHAR(512) NOT NULL,
  PRIMARY KEY(plugin,name,value)
);
```

Opcions utilitzant fitxer d'opcions options.php

En cada plugin, en el directori */lib* es pot desar un fitxer de text anomenat *options.php* que continga totes les opcions a utilitzar en el plugin. Cada opció pot tindre un nom (o clau) de la forma següent: *opcions.foo.bar* amb independència de la llargària del nom, tot ha d'estar separat per el caràcter '.' (punt). A continuació es mostra un fitxer de configuració utilitzat per un plugin:

Listing 17.8: API de Plugin: Exemple de fitxer *options.php*

```
<?php
$options = array (
  "center" => "Universitat Politecnica de Valencia, Valencia
      ",
  "home"   => "DSIC, Universitat Politecnica de Valencia,
      Valencia",
  "geocoder" => array (
    "format" => "json",
    "cache"  => "true"
  )
);
?>
```

És necessari definir el array *$options*, si no es fa així, el plugin no carregarà les opcions. A continuació es mostren els mètodes que ajuden a carregar, obtindre, modificar i afegir les opcions d'aquest fitxer:

Listing 17.9: API de Plugin: API per utilitzar el fitxer *options.php*

```
<?php
    // recuperar una opcio del fitxer de configuracio
       options.php
    mixed getOption($name);

    // modificar el valor d'una opcio
    void setOption($name,$value);

    // veure si existeix una opcio en fitxer options.php
    bool existsOption($key);

    // obte totes les opcions del fitxer options.php
    array getOptions(void);
```

```
13
14      // afegeix una opcio a un fitxer de configuracio
15      void addOption($key,$value);
16
17      // desa les opcions en el fitxer options.php
18      bool saveOptions(void);
19
20      // recarrega les opcions del fitxer options.php
21      void reloadOptions(void);
22
23  ?>
```

Instal.lar/desinstal.lar un plugin

Es pot instal·lar o desintal·lar un plugin (operacions de creació, modificació o esborrat en la base de dades, creació o modificació de fitxers, ...). En aquesta versió del projecte no és molt útil aquesta funcionalitat, però s'ha programat per a futures versions que permetran fer paquets amb els plugins i hi existirà un gestor de plugins que instal·larà o desinstal·larà un plugin. Malgrat la futuritat d'aquestes funcionalitats, el plugin ha d'implementar aquestos mètodes (encara que estiguen buits).

Listing 17.10: API de Plugin: Instal·lació/desinstal·lació d'un plugin

```
1  <?php
2  // instal.la un plugin
3  mixed install(array $arguments=array());
4
5  // desintala un plugin
6  mixed uninstall(array $arguments=array());
7  ?>
```

Renderitzar vistes

Un plugin pot tindre rutes que accedeixen a una funcionalitat del plugin i aquesta funcionalitat pot mostrar vistes o errors, que poden ser útils per al funcionament habitual del plugin. Exemple:

- Permetre una configuració gràfica i visual d'un plugin.
- Mostrar resultats d'accions del plugin.
- ...

Listing 17.11: API de Plugin: Renderitzar vistes en un plugin

```
1  <?php
2  // desa una variable per a les vistes
3  void set($name,$value);
4
5  // renderitza una vista
6  void renderView($view);
7
8  // renderitza un error
```

```
9   void renderError($view);
10  ?>
```

Les vistes han d'estar en el directori *view* del plugin i els errors en el directori *error*

17.3 Plugin bàsic

Per terminar, es mostra l'esquelet d'un plugin bàsic amb els mètodes mínims que ha de tindre per poder funcionar:

Listing 17.12: Esquelet d'un plugin

```
1   <?php
2   // /app/lib/plugins/PluginMinim/PluginMinim.class.php
3   class PluginMinim extends FW_Plugin_Base {
4
5     protected function _configure(array $parameters=null) {
6       // aci configurarem el nostre plugin
7     }
8
9     protected function _initialize(array $parameters=null) {
10      // aci inicialitzarem el nostre plugin
11    }
12
13    protected function install(array $arguments=array()) {
14      // aci anira el codi d'instal.lacio del nostre plugin
15    }
16
17    protected function uninstall(array $arguments=array()) {
18      // aci anira el codi de desinstal.lacio del nostre
            plugin
19    }
20
21  };
22  ?>
```

I una ruta per accedir a una funcionalitat d'un plugin:

Listing 17.13: Ruta bàsica per accedir una funcionalitat d'un plugin

```
1   <?php
2   FW_Router::Connect(
3   array (
4     'url'    => '/plugins/saludar/:name',
5     'type'   => 'plugin',
6     'cache'  => false,
7     'plugin' => 'demo',
8     'action' => 'saludar',
9     'internal' => true,
10    'mime'    => 'text/html',
11    'parameters' =>
12    array (
13      'name' =>
14      array (
```

```
15          'name'    => 'name',
16          'type'    => 'string',
17          'format' => false,
18        ),
19      ),
20      'authentication' => false,
21      'pattern' => '#^/plugins/saludar/(?:([^/]+))[/]*$#',
22      'parameterOrder' =>
23        array ( 0 => 'name' ),
24      )
25    );
26
27    ?>
```

17.4 Creació d'un plugin d'exemple: Generador de Google Maps

En aquesta secció crearem un plugin d'exemple; aquest plugin ens servirà per generar mapes de *Google Maps*.

Aquest plugin generarà el codi font en JavaScript necessari per generar un mapa de Google Maps sobre el que es podrà posar marcadors i utilitzar els serveis de geolocalització proveïts per Google. Primer crearem el codi de la classe base del plugin, després li afegirem els components (*marker*,*latLng*,*maps* i *geocoder*).

17.4.1 Programació del Plugin

Programarem un plugin que cree mapes amb una funcionalitat bàsica (crear mapa, centrar mapa i afegir marcadors al mapa).

Listing 17.14: Codi font del plugin per generar mapes de Google Maps

```
1   <?php
2   class maps extends FW_Plugin_Base {
3
4       /**
5        * The map
6        *
7        * @var map
8        */
9       private $_map;
10
11
12      /**
13       * Not used here
14       * @param array $parameters
15       */
16      protected function _initialize(array $parameters=null)
            {}
17
18      /**
19       * Not used here
```

```
20          * @param array $parameters
21          */
22         protected function _configure(array $parameters=null) {}
23
24         /**
25          * Creates a new map
26          *
27          * @param string $name The name of the map
28          * @param string $zoom The zoom of the map
29          * @param string $type The type of the map
30          *
31          * @return void
32          */
33         public function createMap($name="map",$zoom=10,$type="
               road") {
34             $map         = new map($name,$zoom,$type);
35             $this->_map = $map;
36         }
37
38         /**
39          * Displays the map
40          *
41          * @return string
42          */
43         public function display() {
44             if ($this->_map!==null) {
45                 return $this->_map->display();
46             }
47         }
48
49         /**
50          * Sets the center of the map by coordinates
51          *
52          * @param double $latitude The latitude
53          * @param double $longitude The longitude
54          *
55          * @return void
56          */
57         public function setCenter($latitude,$longitude) {
58             if ($this->_map!==null) {
59                 $this->_map->setCenter($latitude,$longitude);
60             }
61         }
62
63         /**
64          * Sets the center of the map by an address
65          *
66          * @param string $address The address
67          *
68          * @return bool
69          */
70         public function setCenterByAddress($address) {
71             if ($this->_map!==null) {
72                 $this->_map->setCenterByAddress($address);
```

```
73              }
74          }
75
76          /**
77           * Creates a Marker and adds to the map
78           *
79           * @param mixed $position The position of the marker
80           * @param string $title The title of the marker
81           * @param string $text The text to the infowindow of the
                marker
82           * @param string $icon An url to an icon for the marker
83           */
84          public function addMarker($position,$title,$text="",
                $icon="") {
85              $marker = new marker($position,$title,$text,$icon);
86              if ($this->_map!==null) {
87                  $this->_map->addMarker($marker);
88              }
89          }
90      };
91      ?>
```

17.4.2 Programació del *Geocoder*

Un *geocoder* és un servei web de localització geogràfica que permet obtindre informació geogràfica sobre una adreça o unes coordenades. Un geocoder té dos operacions bàsiques:

- Geocoding: A partir d'una adreça obtindre les dades i coordenades de la mateixa.
- Reverse geocoding: A partir d'unes coordenades obtindre l'adreça i les dades de la mateixa.

Per al nostre plugin utilitzarem la *Geocoder API de Google Maps* [1] que ens proveu de les dos operacions mencionades abans. Utilitzarem aquesta API demanant resultats en format JSON i *emmagatzemant els resultats en caché* per tal de no generar més peticions de dades en la xarxa. Aquest servei web ens servirà per poder centrar el mapa en una adreça (o punt) i crear marcadors sobre el mapa.

Listing 17.15: Codi font per utilitzar el servei web de *geolocalització* de Google Maps

```
1   <?php
2   class geocoder {
3
4       /**
5        * The complete geocoder URL
6        *
7        * @var string
8        */
9       private $_url;
10
```

[1] Més informació en l'adreça web `http://code.google.com/intl/ca-ES/apis/maps/documentation/geocoding/`

```
11      /**
12       * Makes a query to the Geocoder webservice
13       *
14       * @param string $url The url for the geocoder
15       *
16       * @return mixed
17       */
18      private function _httpConnection($url) {
19          $parameters = new FW_Container_Parameter();
20          $parameters->endpoint = $url;
21          $parameters->method   = "GET";
22          $parameters->type     = "";
23
24          $client   = new FW_Rest_Client($parameters);
25          $client->exec();
26          $response = $client->getResponse();
27          return $response->getBody();
28      }
29
30      /**
31       * Builds the URL for the geocoder service
32       *
33       * @return string
34       */
35      private function _builGeocoderURL() {
36          $url        = "http://maps.google.com/maps/api/
                geocode/json?sensor=true&language=LANGUAGE";
37          $url        = str_replace("LANGUAGE",FW_Locale::
                getInstance()->getLocale(),$url);
38          $this->_url = $url;
39          return $url;
40      }
41
42      /**
43       * Gets the coordinates from an address
44       *
45       * @param string $address The address
46       *
47       * @return mixed
48       */
49      public function getCoordinates($address) {
50          $address     = urlencode($address);
51          $url         = $this->_builGeocoderURL();
52          $url        .= "&address={$address}";
53
54          $results      = $this->_checkResultInCache($url);
55          if ($results===null) {
56              $data        = $this->_httpConnection($url);
57              $results     = $this->_process($data);
58              $this->_setResultInCache($url,$results);
59          }
60          return $this->_getResult($results);
61      }
62
```

```
63          /**
64           * Gets the addres that belongs to the coordinates
65           *
66           * @param double $lat The latitude
67           * @param double $lng The longitude
68           *
69           * @return mixed
70           */
71          public function getAddress($lat,$lng) {
72              $address     = "";
73              $url         = $this->_builGeocoderURL();
74              $url        .= "&latlng={$lat},{$lng}";
75              $results      = $this->_checkResultInCache($url);
76              if ($results===null) {
77                  $data        = $this->_httpConnection($url);
78                  $results     = $this->_process($data);
79                  $this->_setResultInCache($url,$results);
80              }
81              return $this->_getResult($results);
82          }
83
84          /**
85           * Processes the geocoder result
86           *
87           * @param mixed $data The result of the geocoder
88           *
89           * @return array
90           */
91          private function _process($data) {
92              $results    = array();
93              $json       = json_decode($data);
94              if ($json!=null) {
95                  if ($json->status=="OK") {
96                      foreach ($json->results as $result) {
97                          $results []= $this->_processResult(
                                $result);
98                      }
99                  }
100             }
101             return $results;
102         }
103
104         /**
105          * Proccesess every result of a geocoder request
106          *
107          * @param array $result The result of a geocoder request
108          *
109          * @return array
110          */
111         private function _processResult($result) {
112             $data                       = array();
113             $data["types"]              = $result->types;
114             $data["address_components"] = array();
```

```
115             $data["formated_address"]    = $result->
                    formatted_address;
116
117             if (isset($result->address_components)) {
118                 foreach ($result->address_components as
                        $component) {
119                     $data["address_components"] []= $this->
                            _processAddressComponent($component);
120                 }
121             }
122             if (isset($result->geometry)) {
123                 $geometry = $this->_processGeometry($result->
                        geometry);
124             }
125             $data ["geometry"] = $geometry;
126             return $data;
127         }
128
129         /**
130          * Processes the address of a geocoder result
131          *
132          * @param array $component
133          *
134          * @return array
135          */
136         private function _processAddressComponent($component) {
137             $data = array (
138                 "long_name"  => $component->long_name,
139                 "short_name" => $component->short_name,
140                 "types"              => $component->types
141             );
142             return $data;
143         }
144
145         /**
146          * Processes the geometry of a geocoder result
147          *
148          * @param array $geometry
149          *
150          * @return array
151          */
152         private function _processGeometry($geometry) {
153             $data = array (
154                 "location"  => new latLng($geometry->location->
                        lat,$geometry->location->lng),
155                 "viewport"  => array (
156                     "southwest" => new latLng($geometry->
                            viewport->southwest->lat,$geometry->
                            viewport->southwest->lng),
157                     "northeast" => new latLng($geometry->
                            viewport->northeast->lat,$geometry->
                            viewport->northeast->lng)
158                  ),
159                 "location_type" => $geometry->location_type
```

```
160             );
161             return $data;
162         }
163 
164 
165         /**
166          * Gets the addresses from a geocoder result
167          *
168          * @param array $results The results
169          *
170          * @return array
171          */
172         private function _getAddressFromResults($results) {
173             $address = array();
174             if (count($results)>0) {
175                 foreach ($results as $result) {
176                     $address []= $result["formated_address"];
177                 }
178             }
179             return $address;
180         }
181 
182         /**
183          * Gets coordinates from a geocoder result
184          *
185          * @param array $results The results
186          *
187          * @return array
188          */
189         private function _getCoordinatesFromResults($results) {
190             $coordinates = array();
191             if (count($results)>0) {
192                 foreach ($results as $result) {
193                     $coordinates []= $result["geometry"]["
                            location"];
194                 }
195             }
196             return $coordinates;
197         }
198 
199         /**
200          * Gets the result of a geocoder request
201          *
202          * @param array $results
203          *
204          * @return array
205          */
206         private function _getResult($results) {
207              $addresses   = $this->_getAddressFromResults(
                     $results);
208              $coordinates = $this->_getCoordinatesFromResults(
                     $results);
209 
210              $return      = array();
```

```
211
212             if ( (count($addresses)===count($coordinates)) && (
                    count($addresses)>0)) {
213                 $elements = count($addresses);
214                 for ($i=0;$i<$elements;$i++) {
215                     $return []= array (
216                         "address"     => $addresses[$i],
217                         "coordinates" => $coordinates[$i]
218                     );
219                 }
220             }
221             return $return;
222         }
223
224
225         /**
226          * Checks if the result is in cache
227          *
228          * @param string $url The url
229          *
230          * @return mixed
231          */
232         private function _checkResultInCache($url) {
233             $id    = md5($url);
234             $cache = new FW_Cache();
235             $data  = $cache->get($id,"mapsPluginCache");
236             if ($data!==null) {
237                 $contents = $data->getContents();
238                 $results  = unserialize($contents);
239                 return $results;
240             }
241         }
242
243         /**
244          * Stores the result in cache for a month
245          *
246          * @param string $url The url
247          * @param mixed $result The result of the geocoder
248          *
249          * @return bool
250          */
251         private function _setResultInCache($url,$result) {
252             $result = serialize($result);
253             $id     = md5($url);
254             $cache  = new FW_Cache();
255             return  $cache->save($id,"mapsPluginCache",$result
                        ,2592000);
256         }
257     };
258     ?>
```

17.4.3 Programació de la classe *LatLng*

Aquesta classe és un *wrapper* sobre la classe *gLatLng* de la API de Google Maps que serveix per representar coordenades en un mapa. Conté els mètodes necessaris per crear, establir i obtindre la longitud i la latitud d'un punt en el mapa.

Listing 17.16: Codi de la classe latLng

```php
<?php
class latLng {

    /**
     * The latitude
     *
     * @var double
     */
    private $_lat;

    /**
     * The longitude
     *
     * @var double
     */
    private $_lng;

    /**
     * Constructor of latLng
     *
     * @param double $lat The latitude
     * @param double $lng The longitude
     *
     * @return void
     */
    public function __construct($lat,$lng) {
        $this->_lat = $lat;
        $this->_lng = $lng;
    }

    /**
     * Gets the latitude
     *
     * @return double
     */
    public function lat() {
        return $this->_lat;
    }

    /**
     * Gets the longitude
     *
     * @return double
     */
    public function lng() {
        return $this->_lng;
    }
```

```
48
49              /**
50           * Sets the latitude
51           *
52           * @param double $lat The latitude
53           *
54           * @return void
55           */
56          public function setLatitude($lat) {
57              $this->_lat = $lat;
58          }
59
60          /**
61           * Sets the longitude
62           *
63           * @param double $lng The longitude
64           *
65           * @return void
66           */
67          public function setLongitude($lng) {
68              $this->_lat = $lat;
69          }
70
71          /**
72           * Gets the Javascript code for Google Maps V3
73           *
74           * @return string
75           */
76          public function getJavaScript() {
77              $lat  = str_replace(",",".",$this->_lat);
78              $lng  = str_replace(",",".",$this->_lng);
79              $code = "new google.maps.LatLng({$lat},{$lng})";
80              return $code;
81          }
82
83          /**
84           * Encodes the lat and lng for a value to be used in an
                 URL
85           *
86           * @return string
87           */
88          public function toUrlValue() {
89              return round($this->_lat,6).",".round($this->lng,6);
90          }
91
92      };
93
94      ?>
```

17.4.4 Programació de la classe *Marker*

Aquesta classe representa un marcador d'un mapa de Google Maps. És un *wrapper* sobre la classe *gMarker*. Permet inserir un marcador en un mapa i personalitzar-lo amb

les opcions de títol, icona i text que apareixerà sobre el mapa al fer click en el marcador.

Listing 17.17: Codi de la classe marker

```
<?php
class marker {

    /**
     * The position of the marker
     *
     * @var mixed
     */
    private $_position;

    /**
     * The title of the marker
     *
     * @var string
     */
    private $_title;

    /**
     * The text of the InfoWindow of the marker
     *
     * @var string
     */
    private $_text;


    /**
     * The icon of the marker
     *
     * @var string
     */
    private $_icon;


    /**
     * Creates a Marker
     *
     * @param mixed $position The position of the marker
     * @param string $title The title of the marker
     * @param string $text The text to the infowindow of the
         marker
     * @param string $icon An url to an icon for the marker
     */
    public function __construct($position,$title,$text="",
        $icon="") {
        $this->_position       = $position;
        $this->_title          = $title;
        $this->_text = $text;
        $this->_icon           = $icon;

        if (!$position instanceof latLng) {
```

```
49                  $geocoder = new geocoder();
50                  $result   = $geocoder->getCoordinates($position)
                        ;
51                  if (!empty($result)) {
52                      $this->_position = $result[0]["coordinates"
                            ];
53                  }
54              }
55          }
56
57
58          /**
59           * Generates the javascript code for the marker
60           *
61           * @param string $mapName The name of the map
62           *
63           * @return string
64           */
65          public function generateJavaScript($mapName) {
66              $code  = "new google.maps.Marker ({\n";
67              $code .= "\t position: ".$this->_position->
                    getJavaScript().",\n";
68              $code .= "\t map: {$mapName},\n";
69              if (!empty($this->_icon)) {
70                  $code .= "\t icon: \"{$this->_icon}\",\n";
71              }
72              $code .= "\t title: \"{$this->_title}\" \n";
73              $code .= "});\n\n";
74              return $code;
75          }
76
77          /**
78           * Generates the javascript code for the infowindow of
                 the marker
79           *
80           * @param string $mapName The name of the map
81           * @param string $markerName The name of the marker
82           *
83           * @return string
84           */
85          public function generateInfoWindowJavaScript($mapName,
                $markerName) {
86              $code  = "var {$markerName}_infoWindow = new google.
                    maps.InfoWindow ({\n";
87              $code .= "\t content: \"{$this->_text}\" \n});\n";
88
89              $code .= "google.maps.event.addListener({$markerName
                    },'click',function() {\n";
90              $code .= "\t {$markerName}_infoWindow.open({$mapName
                    },{$markerName}); \n});\n";
91              return $code;
92          }
93
94
```

```
95          /**
96           * Checks if this marker has an infowindow
97           *
98           * @return boolean
99           */
100         public function hasInfoWindow() {
101             return (!empty($this->_text));
102         }
103
104     };
105     ?>
```

17.4.5 Programació de la classe *Map*

La classe *Map* ens permetrà representar un mapa que generarà el codi font en Java Script necessari per a mostrar-lo en una vista.

Listing 17.18: Codi de la classe map

```
1   <?php
2   class map {
3       /**
4        * The name of the map
5        *
6        * @var string
7        */
8       private $_name;
9
10      /**
11       * The type of the map
12       *
13       * @var string
14       */
15      private $_type;
16
17      /**
18       * The zoom of the map
19       *
20       * @var int
21       */
22      private $_zoom;
23
24      /**
25       * The center of the map
26       *
27       * @var latLng
28       */
29      private $_center;
30
31      /**
32       * An array of markers
33       *
34       * @var array
35       */
```

```
36      private $_markers;
37
38      /**
39       * Creates a new map
40       *
41       * @param string $name The name of the map
42       * @param int $zoom The zoom of the map
43       * @param string $type The type of the map
44       */
45      public function __construct($name="map",$zoom=10,$type="
            road") {
46
47          if (!empty($name)) {
48              $this->setName($name);
49          }
50          if (is_int($zoom)) {
51              $this->setZoom($zoom);
52          }
53
54          if (!empty($type)) {
55              $this->setType($type);
56          }
57
58          $this->_markers = array();
59          $this->_center  = new latLng(0,0);
60      }
61
62
63      /**
64       * Sets the name of the map
65       *
66       * @param string $name The name of the map
67       *
68       * @return void
69       */
70      public function setName($name) {
71          $this->_name = $name;
72      }
73
74      /**
75       * Sets the Zoom of the map
76       *
77       * @param int $zoom The zoom of the map (values between
               1 and 18)
78       *
79       * @return void
80       */
81      public function setZoom($zoom) {
82          if (is_int($zoom)) {
83              if ($zoom>0 || $zoom<19) {
84                  $this->_zoom = $zoom;
85              }
86          }
87      }
```

```
88
89          /**
90           * Sets the type of the map
91           *
92           * @param string $type The type of the map
93           */
94          public function setType($type) {
95              $this->_type = $type;
96          }
97
98          /**
99           * Gets the JavaScript map type
100          *
101          * @return string
102          */
103         private function _getMapTypeId() {
104             $mapType = $this->_type;
105             switch ($mapType) {
106                 case "hybrid":
107                     return "google.maps.MapTypeId.HYBRID";
108                 break;
109
110                 case "satellite":
111                     return "google.maps.MapTypeId.SATELLITE";
112                 break;
113
114                 case "terrain":
115                     return "google.maps.MapTypeId.TERRAIN";
116                 break;
117
118                 default:
119                 case "road":
120                     return "google.maps.MapTypeId.ROADMAP";
121                 break;
122             };
123         }
124
125
126         /**
127          * Sets the center of the map by coordinates
128          *
129          * @param double $latitude The latitude
130          * @param double $longitude The longitude
131          *
132          * @return void
133          */
134         public function setCenter($latitude,$longitude) {
135             $this->center_lat = $latitude;
136             $this->center_lon = $longitude;
137         }
138
139         /**
140          * Sets the center of the map by an address
141          *
```

```
142         * @param string $address The address
143         * @return bool
144         */
145        public function setCenterByAddress($address) {
146            $geocoder = new geocoder();
147            $results  = $geocoder->getCoordinates($address);
148            if (count($results)>0) {
149                $coordinates      = $results[0]["coordinates"];
150                $this->center_lat = $coordinates->lat();
151                $this->center_lon = $coordinates->lng();
152                $this->_center    = new latLng($this->center_lat
                       ,$this->center_lon);
153                return true;
154            }
155            return false;
156        }
157
158        /**
159         * Adds a marker to the map
160         *
161         * @param marker $marker The marker to add
162         *
163         * @return void
164         */
165        public function addMarker(marker $marker) {
166            $this->_markers []= $marker;
167        }
168
169        /**
170         * Displays the map
171         *
172         * @return string
173         */
174        public function display() {
175            $code = "";
176            $code .= $this->_getJavaScriptIncludes();
177            $code .= "<script type=\"text/javascript\">\n";
178            $code .= $this->_getMapJavaScript();
179            $code .= "</script>";
180            return $code;
181        }
182
183
184        /**
185         * Returns the Include of JavaScript to the Google Maps
                V3 API
186         * @return string
187         */
188        private function _getJavaScriptIncludes() {
189            $code  = "<script type=\"text/javascript\" src=\"
                   http://maps.google.com/maps/api/js?sensor=true\">
                    </script>\n";
190            return $code;
191        }
```

```
192
193         /**
194          * Gets the map JavaScript code
195          *
196          * @return string
197          */
198         private function _getMapJavaScript() {
199             $name = $this->_name;
200             $code  =  "var {$name}_options = {\n \t zoom: {$this
                ->_zoom},\n \t center: ".$this->_center->
                getJavaScript().",\n \t mapTypeId: ".$this->
                _getMapTypeId($name)." \n};\n";
201             $code .=  "var {$name} = new google.maps.Map(
                document.getElementById(\"{$name}\"),{$name}
                _options);\n\n";
202             $code .=  $this->_getMarkersJavaScript($name);
203             return $code;
204         }
205
206
207         /**
208          * Gets the JavaScript code of the markers
209          *
210          * @return string
211          */
212         private function _getMarkersJavaScript() {
213             $code    = "";
214             $markers = $this->_markers;
215             for ($i=0;$i<count($markers);$i++) {
216                 $name  = "{$this->_name}_marker{$i}";
217                 $code .= "var {$name} = ".$markers[$i]->
                    generateJavaScript($this->_name)."\n";
218                 $code .=  $markers[$i]->
                    generateInfoWindowJavaScript($this->_name,
                    $name)."\n";
219
220             }
221             return $code;
222         }
223     };
224     ?>
```

17.4.6 Codi de proves

Després d'haver programat el plugin sencer, ha arribat l'hora de provar-lo. Per això crearem una acció en el controlador *pluginController* que hem creat en la introducció a aquesta part de la memòria. Aquesta acció obtindrà una instància del plugin **maps** a través de *Plugin_Registry* , crearà un mapa, el centrarà al centre de València i afegirà uns marcadors a vàries universitats de València.

El resultat que obtindrem serà paregut a la següent captura de pantalla.

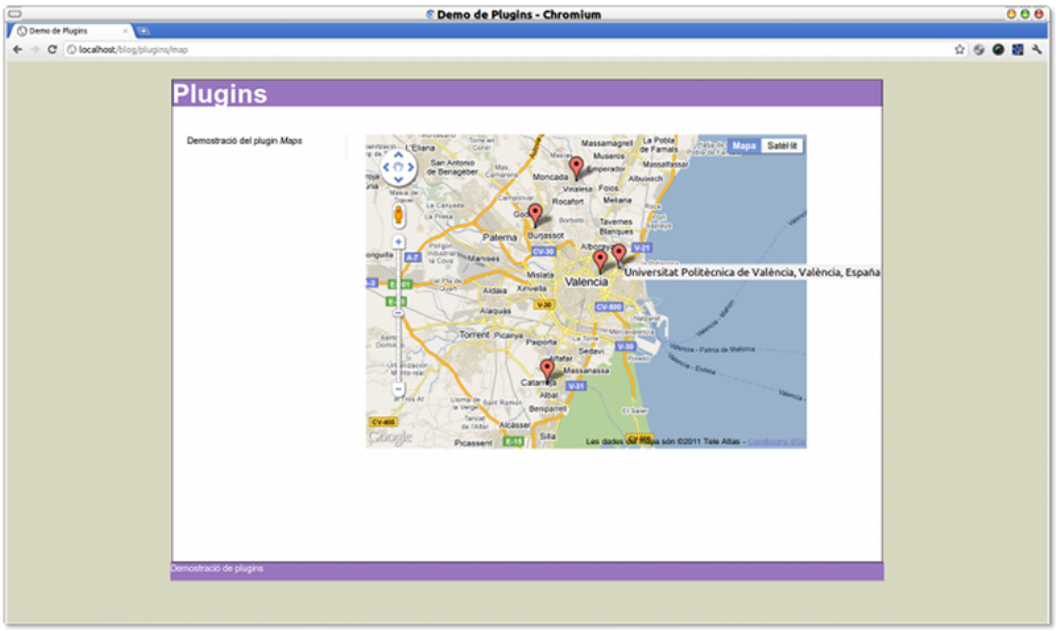


Figura 17.2: Captura de pantalla de demostració del plugin *map*

Listing 17.19: Codi font de l'acció *demoMaps*

```
<?php

// en pluginController.class.php
public function demoMaps() {
  // obtenim una instancia del registre de plugins
  $preg = FW_Plugin_Registry::getInstance();
  // obtenim el plugin maps
  $map  = $preg->getPlugin("maps");

  // si hem obtingut el plugin maps
  if ($map!==null) {
    // creem un mapa nou
    $map->createMap("map",11,"road");

    // cemtrem el mapa en el centre de Valencia
    $map->setCenterByAddress("Carrer de Xativa, Valencia,
        Espanya");

    // adreces a afegir marcadors al mapa
    $markers = array (
      "Universitat Politecnica de Valencia, Valencia,
          Espanya",
      "Universitat de Valencia, Valencia, Espanya",
      "Universitat de Valencia, Burjassot, Espanya",
      "CEU-San Pablo, Alfara del Patriarca, Espanya",
      "Florida Universitaria, Catarroja, Espanya"
    );
```

```
26        foreach ($markers as $aux) {
27          // afegim el marcador al mapa
28          $map->addMarker($aux,$aux,$aux);
29        }
30        // donem valor al slot content
31        $this->setSlot("content","maps/map",array("map"=>$map));
32      }
33      // renderitzem el layout
34      $this->renderLayout("default");
35    }
36    ?>
```

Creació de la vista map

A continuació crearem la vista *map*.

Listing 17.20: Codi font de la vista *map*

```
1   <div class="span-5 column colborder">
2     <p>Demostracio del plugin <em>Maps</em> </p>
3   </div>
4   <div class="span-15 column last" id="map" style="height: 420
        px;">
5     <!-- Mostrem el mapa -->
6     <?php print $map->display(); ?>
7   </div>
```

Creació de les rutes

Per terminar, crearem les rutes necessàries per mostrar el mapa.

Listing 17.21: Rutes per a la demostració

```
1   <?php
2   FW_Router::Connect(
3   array (
4     'url' => '/plugins/map',
5     'type' => 'app',
6     'cache' => false,
7     'parameters' => array (),
8     'authentication' => false,
9     'module' => 'plugin',
10    'controller' => 'plugin',
11    'action' => 'demoMaps',
12    'internal' => false,
13    'pattern' => '#^/plugins/map[/]*$#',
14    'parameterOrder' => array ()
15    )
16  );
17  ?>
```

Capítol 18

Helpers

18.1 Introducció

Un *Helper* és una classe de component que ajuda a generar el codi font en les vistes d'una aplicació web. Generalment són classes amb poc de codi i mètodes estàtics que generen codi html o Java Script. Aquestes classes ens poden ajudar a mostrar elements html que és complex crear-los, per exemple un desplegable de selecció d'un formulari, generació del codi font html d'un formulari complet, ... També poden ser classes útils que per exemple ajuden a convertir entre formats de unitats (temperatura, distància, ...) o qualsevol cosa que es puga requerir en una vista. Els *helpers* han d'estar implementats en PHP i han d'estar en el directori `/app/lib/helpers`.

18.2 Creació de helpers

Un helper és una classe de PHP que conté codi per general (habitualment codi font en html). A l'hora de dissenyar un helper, el millor és que els mètodes de la classe siguen estàtics per a poder fer cridades senzilles com
print html::link_to("http://www.google.es","Google"); i que produeix-ca un enllaç html.

Listing 18.1: Helper de demostració que ajuda a crear formularis html

```
1  <?php
2  class form {
3
4      /**
5       * Generates a form tag
6       *
7       * @param string $id The id for the form
8       * @param string $action The action of the form
9       * @param string $method The method of the form (post/
            get)
10      * @param string $enctype The encoding of the form
11      * @param string $class The class name of the form
12      * @param string $style The style of the form
13      *
14      * @return string
```

```
15          */
16         public static function formTag($id,$action,$method="POST
               ",$enctype="",$class="",$style="") {
17             $code   = "<form id=\"{$id}\" action=\"{$action}\"";
18             $method = strtolower($method);
19 
20             if (!in_array($method,array("get","post"))) {
21                 throw new FW_Exception("Form method can't be {
                       $method}");
22             }
23             else {
24                 $code .= " method=\"{$method}\"";
25             }
26 
27             if (!empty($enctype)) {
28                 $code .= " enctype=\"{$enctype}\" ";
29             }
30 
31             if (!empty($class)) {
32                 $code .= " class=\"{$class}\" ";
33             }
34 
35             if (!empty($style)) {
36                 $code .= " style=\"{$style}\" ";
37             }
38 
39             $code .= " >";
40             return $code;
41         }
42 
43         /**
44          * Generates a fieldset tag
45          *
46          * @return string
47          */
48         public static function openFieldsetTag() {
49             return "<fieldset>";
50         }
51 
52         /**
53          * Generates a fieldset closing tag
54          *
55          * @return string
56          */
57         public static function closeFieldsetTag() {
58             return "</fieldset>";
59         }
60 
61         /**
62          * Generates a form closing tag
63          *
64          * @return string
65          */
66         public static function closeFormTag() {
```

```
67              return "</form>";
68          }
69
70          /**
71           * Generates a textarea tag
72           *
73           * @param string $id The id of the textarea
74           * @param string $value The value of the textarea
75           * @param string $rows The number of rows of the
                 textarea
76           * @param string $cols The number of columns of the
                 textarea
77           * @param string $class The class name of the textarea
78           * @param string $style The style of the textarea
79           *
80           * @return string
81           */
82          public static function textArea($id,$value="",$rows=0,
                $cols=0,$class="",$style="") {
83              $code = "<textarea id=\"{$id}\" name=\"{$id}\" ";
84              if ($rows!==0) {
85                  $code .= " rows=\"{$rows}\" ";
86              }
87              if ($cols!==0) {
88                  $code .= " cols=\"{$cols}\" ";
89              }
90
91              if (!empty($class)) {
92                  $code .= " class=\"{$class}\" ";
93              }
94
95              if (!empty($style)) {
96                  $code .= " style=\"{$style}\" ";
97              }
98
99              $code .= ">";
100             if (!empty($value)) {
101                 $code .= $value;
102             }
103             $code .= "</textarea>";
104             return $code;
105         }
106         ...
107         segueix en /framework/lib/Helpers/form.class.php
108         ...
109     };
110     ?>
```

Part VI

Desenvolupament de serveis web

Capítol 19

Introducció

19.1 Introducció

En aquest capítol s'introduirà el desenvolupament de serveis web amb el marc de treball. S'explicarà tant com consumir serveis web SOAP/REST com generar una sèrie de serveis web amb el marc de treball.

Un servei web (també conegut com Web Service en anglès) és una col·lecció de protocols i estàndards que serveix per intercanviar dades entre aplicacions. Diferents aplicacions de programari desenvolupades en llenguatges de programació diferents i executades sobre qualsevol plataforma poden utilitzar els serveis web per l'intercanvi de dades en una xarxa com Internet. Aquesta gran interoperatibilitat s'aconsegueix gràcies a l'adopció d'estàndards oberts. Les organitzacions *OASIS* i *W3C* són les responsables de l'arquitectura i reglamentació dels serveis web. Per garantir la interoperatibilitat entre les diferents implementacions existeix un organisme, el *WS-I*, que és l'encarregat d'especificar de forma exhaustiva tots els aspectes d'aquests estàndards.

19.2 Creació del mòdul a utilitzar en les desmostracions

Per al desenvolupament d'aquest capítol crearem un nou mòdul anomenat **webservices** i el situarem sota el directori `app`. En el mòdul acabat de crear, crearem tres controladors (*restController*, *newsPortalController* i *soapController*) amb els seus corresponents models. Aquestos controladors seran els que farem servir per a desenvolupar serveis web REST i SOAP.

A continuació s'exposa el codi font de cadascun d'aquestos controladors i els seus corresponents models:

Listing 19.1: Codi font per crear els controladors i models per a les demostracions

```
1  <?php
2
3  // /app/modules/webservices/controller/soapController.class.
       php
4  class soapController extends FW_mvc_BaseController {
5
```

```
};

// /app/modules/webservices/model/soapModel.class.php
class soapModel extends FW_mvc_BaseModel {

};

// /app/modules/webservices/controller/restController.class.
    php
class restController extends FW_mvc_BaseController {

};

// /app/modules/webservices/model/restModel.class.php
class restModel extends FW_mvc_BaseModel {

};

// /app/modules/webservices/controller/newsPortalController.
    class.php
class newsPortalController extends FW_mvc_BaseController {

};

// /app/modules/webservices/model/newsPortalModel.class.php
class newsPortalModel extends FW_mvc_BaseModel {

};

?>
```

19.2.1 Creació del «layout» per a les demostracions

A continuació crearem el «layout» a emprar en les demostracions. Crearem un fitxer de text amb el següent codi font que desarem com a `/app/modules/webservices/layout/default.php`

Listing 19.2: Layout utilitzat per a les demostracions

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html;
        charset=utf-8" />
    <title>Demo de serveis web</title>

    <?php
        $params  = FW_Context::getInstance()->router->route;
        $manager = FW_Style_Manager::getInstance();
        print $manager->displayStyle($params);

        $styles = FW_Context::getInstance()->router->styles;
        if (count($styles)>0) {
```

```
14                foreach ($styles as $style) {
15                    print style_tag("{$style["file"]}",$style["
                          media"]).'\n';
16                }
17            }
18        ?>
19      </head>
20
21      <body>
22        <div id="wrapper">
23          <div id="container" class="container">
24
25            <div id="header" class="container">
26              <h1>Serveis webs</h1>
27            </div>
28
29            <div id="content" class="container">
30              <?php
31                if ($this->hasSlot("content")) {
32                  print $this->getSlot("content");
33                }
34              ?>
35            </div>
36
37            <hr class="space" />
38          </div>
39        </div>
40
41        <div id="footer" class="container">
42          <p>Demo de serveis web</p>
43        </div>
44      </body>
45    </html>
```

19.2.2 Creació dels estils CSS per a les demostracions

Per últim, crearem els estils *CSS* necessaris per a la demostració. Aquest codi el desarem en el fitxer `/style/default/application/webservices/style.css`

Listing 19.3: CSS per a les demostracions

```
1   * {
2     margin: 0;
3     padding: 0;
4   }
5
6   html, body, #wrapper {
7     height: 95%;
8   }
9
10  body {
11    background: none repeat scroll 0 0 #D8D8C0;
12    color: #000000;
13  }
```

```
14
15  #wrapper {
16    height: auto;
17    min-height: 95%;
18    margin: 0 auto;
19    width: 950px;
20    margin-top: 25px;
21    margin-bottom: 25px;
22    background: none repeat scroll 0 0 #FFFFFF;
23    border: 1px solid #000000;
24  }
25
26  #content {
27    padding: 20px;
28  }
29
30  #header {
31    background: #615E59;
32    height: auto;
33    color: #FFFFFF;
34  }
35
36  #header h1 {
37    color: #FFFFFF;
38    font-size: 35px;
39    font-family: Arial, Helvetica, sans-serif;
40    font-weight: bolder;
41  }
42
43  #header a {
44    color: #FFFFFF;
45  }
46
47  #header a:hover {
48    color: #000000;
49    background: #FFFFFF;
50  }
51
52  #footer {
53    background: #615E59;
54    color: #FFFFFF;
55    position: relative;
56    margin-top: -25px;
57    height: 25px;
58    clear: both;
59    width: 955px;
60  }
61
62  #content {
63    background: #FFFFFF;
64  }
```

També modificarem el fitxer que carrega els estils amb el component *Style Manager*, en concret hem de modificar el fitxer */style/default/style.css* i afegir-li en la clau

modules → *external* un array que descriurà els estils a carregar per a les accions del mòdul *webservices*

Listing 19.4: Definicions d'estils per el mòdul webservices

```
<?php

// definicio d'estils per altres moduls externs
"webservices"  => array (
  "default"=> array (
    0 => array("file"=>"reset.css","media"=>array("screen"),
        "alternate"=>false),
    1 => array("file"=>"blueprint/screen.css","media"=>array
        ("screen"),"alternate"=>false),
    2 => array("file"=>"application/webservices/style.css","
        media"=>array("screen"),"alternate"=>false)
  )
),
// altres definicions d'estils per a altres moduls externs
?>
```

Capítol 20

REST

20.1 Introducció

REST, acrònim de **Representational State Transfer** és una arquitectura de programari per a continguts hipermèdia distribuïts sobre Internet, seguint la filosofia de la WWW. El terme es va originar l'any 2000, apareixent per primera vegada a una tesi doctoral sobre la web escrita per *Roy Fielding*, un dels principals autors de l'especificació de l'HTTP. El terme REST es refereix originalment a un conjunt de principis per al disseny d'arquitectures en xarxa. Aquests principis resumeixen com els recursos són definits. El terme freqüentment és emprat en el sentit de descriure qualsevol interfície que transmet dades específiques d'un domini sobre HTTP sense una capa addicional, com fa SOAP. Aquests dos significats poden xocar i fins i tot solapar-se. És possible dissenyar un sistema de programari de gran volum, d'acord amb l'arquitectura proposada per Fielding sense emprar HTTP o sense interactuar amb la Web. O també es pot dissenyar una interfície XML sobre HTTP que no segueixi els principis REST, i en canvi seguir un model RPC. Cal destacar que REST no és un estàndard, és una arquitectura. Encara que REST no sigui un estàndard, està basat en la utilització dels següents estàndards com HTTP, URL, representació de recursos (XML/HTML/GIF/JPEG/...) i tipus MIME (text/xml, text/html, ...). La motivació de REST és de seguir les mateixes característiques de la web que l'han feta esdevenir una vertadera revolució a nivell mundial. La web és la única aplicació distribuïda que ha aconseguit ser escalable a nivell de tot internet. L'èxit és degut a l'ús de formats de missatges extensibles i estàndards, amb un adreçament global (estàndard i extensible a la vegada).

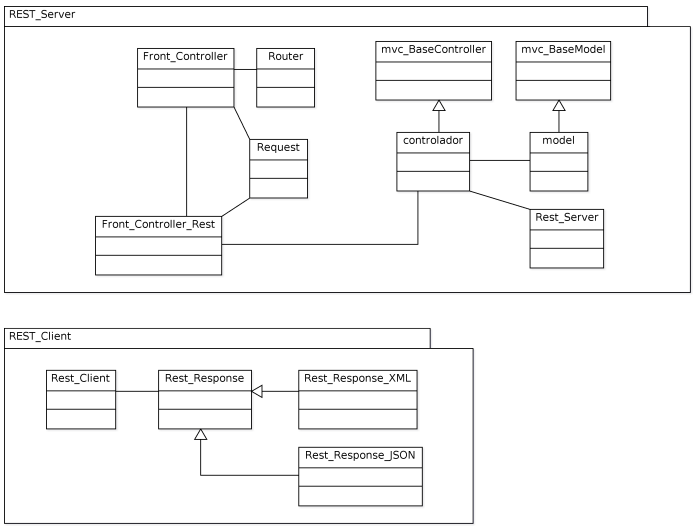


Figura 20.1: UML del sistema REST

20.2 Consumir serveis web REST

20.2.1 Introducció

En aquesta secció desenvoluparem una mini-aplicació web que consumirà serveis web REST, en concret consumirem dades en format XML i s'explicarà també l'API del client REST.

20.2.2 API del client REST

El client de REST és un component compost per un conjunt de classes que generen, comproven i decodifiquen una petició HTTP a un servei web.

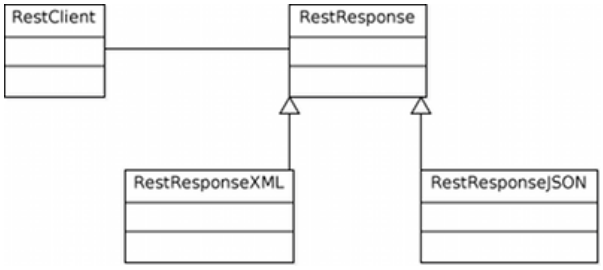


Figura 20.2: Diagrama de classes del client de REST

Construcció (constructor) del client de REST

Listing 20.1: Constructor del client de REST

```
1  <?php
2  // constructor d'un client de rest
```

```
3   __construct(FW_Container_Parameter $parameters=null){
4
5   // exemple
6
7   // creem els parametres
8   $parameters = new FW_Container_Parameter();
9   $parameters->endpoint = "http://servidor.com/recurs/obtindre
        ?format=xml";
10  $parameters->method   = "GET";
11  $parameters->type     = "XML";
12
13  // si el servei web necessita autenticacio (HTTP)
14  $parameters->username = "usuari";
15  $parameters->password = "contrasenya";
16
17  // creem el client
18  $client     = new FW_Rest_Client($parameters);
19  ?>
```

Cridar al servei web

Per l'arquitectura del serveis web implementats amb l'arquitectura REST, per cridar a un servei web REST només cal la URL amb els paràmetres corresponents i/o dades enviades via mètodes *POST/PUT*. A continuació es presenta el codi font per consumir un servei web de «echo» (que torna les dades tal com van ser enviades) enviant dades via POST.

Listing 20.2: Consumir un servei web enviant dades amb POST

```
1   <?php
2   // Consumir un servei web REST
3
4   // creem els parametres
5   $parameters = new FW_Container_Parameter();
6   $parameters->endpoint = "http://localhost/framework/rest/
        webservices/rest/demo/echo";
7   $parameters->method   = "POST";
8   $parameters->type     = "XML";
9
10  // dades a enviar via POST
11  $parameters->data     = array("string"=>"Hello World!");
12
13  // creem el client
14  $client     = new FW_Rest_Client($parameters);
15
16  // enviem la peticio HTTP
17  $client->exec();
18
19  // obtenim la resposta amb les dades en format XML
20  $response   = $client->getResponse();
21
22  var_dump($response->getBody());
23
24  /*
25    object(SimpleXMLElement)[8]
```

```
26        public 'response' => string 'Hello World!' (length=12)
27    */
28
29    ?>
```

En el cas d'haver d'enviar dades amb mètodes *PUT* o *DELETE*, el codi és el mateix però canviant el paràmetre ≪method≫ de la configuració del client de REST per el mètode adequat al servei web.

Atenció: La classe *FW_Rest_Response* pot decodificar només respostes en formats *XML* o *JSON*, per altres formats ha de ser el desenvolupador qui es faça càrrec de la manipulació i decodificació de les dades. Aquestes format pot especificar-se mitjançant el paràmetre ≪type≫ (tipus de dades esperats amb la resposta REST) i pot tindre com a valors ≪XML≫ o ≪JSON≫, qualsevol altre valor per aquest paràmetre farà que el contingut de la resposta s'entregue en brut tal i com es reben les dades (útil quan la resposta és un fitxer binari, pdf, una imatge o un document en un format ≪estrany≫).

20.2.3 Consumir serveis web REST en diversos formats

En aquesta secció consumirem un servei web que retorna una imatge, en concret consumirem el servei ≪Google Static Maps≫ (més informació en `http://code.google.com/intl/ca-ES/apis/maps/documentation/staticmaps/`).

Demanarem a Google un mapa de la ≪Universitat Politècnica de València≫ en format PNG en el que ficarem dos marcadors sobre l'ETSINF.

Listing 20.3: Consumir un servei web que genera imatges

```
1   <?php
2   // creem els parametres
3   $parameters = new FW_Container_Parameter();
4   $parameters->endpoint = "http://maps.google.com/maps/api/
        staticmap?center=Universitat%20Polit%C3%A8cnica%20de%20
        Val%C3%A8ncia&zoom=17&size=400x400&maptype=hybrid&sensor=
        true&markers=color:blue|label:A|39.4828,-0.3472&markers=
        color:red|label:B|39.4825,-0.3485";
5   $parameters->method   = "GET";
6   $parameters->type     = "PNG";
7
8   // creem el client
9   $client     = new FW_Rest_Client($parameters);
10
11  // enviem la peticio HTTP
12  $client->exec();
13
14  // obtenim la resposta amb les dades en format PNG
15  $response   = $client->getResponse();
16
17  // enviem al navegador una capcalera de imatge PNG per poder
        mostrar la imatge
18  header("Content-Type: image/png");
19  // imprimim el contingut rebut (la imatge)
20  print $response->getBody();
21
22  ?>
```

Figura 20.3: Resultat d'executar el codi anterior

20.2.4 Consumir servei web REST en format XML: Un mini-agregador de feeds RSS

Introducció

Un feed RSS (*Really Simple Sindication*) és un document XML que conté informació sobre les darreres entrades (posts) d'un bloc o un lloc web. Aquest document XML és un document fàcilment processable del que es coneix bé l'estructura - donat que és un estàndard prou utilitzat - .

Aprofitarem els següents feeds per fer el nostre mini-agregador de feeds RSS:

- ETSINF: `http://tv.inf.upv.es/?feed=rss2`
- Universitat Politècnica de València: `http://www.upv.es/pls/oalu/sic_rss.rss_ver20?p_rss_feed=7&p_idioma=v`
- Universitat Jaume I: `http://www.uji.es/cocoon/xpfpub/rss?p_channel=2269`
- 20Minutos: `http://fxfeeds.mozilla.com/es-ES/firefox/headlines.xml`

Totes les funcionalitats d'aquesta demostració les crearem en el controlador *newsPorta* que hem creat en el capítol d'introducció a la creació de serveis web.

Connexió i descàrrega d'un feed RSS

Per a descarregar-nos un feed RSS crearem un nou client REST al que li assignarem el *endpoint* al valor de la URL del feed que vulguem descarregar-nos indicant-li a més a més que el contingut que esperem està en format XML. Després executarem el mètode ≪exec()≫ del client que ens tornarà un objecte *FW_Rest_Response_XML* amb les dades del feed i sobre aquest objecte que ens torna haurem d'executar el mètode ≪getBody()≫ per obtindre el document XML que conté les dades del feed RSS. Així doncs, amb aquest pedaç de codi aconseguirem les dades del feed RSS.

Listing 20.4: Creacio i inicializacio d'un client de serveis web REST

```
<?php
// Creem un contenidor de parametres
$parameters = new FW_Container_Parameter();

// URL del feed RSS de la ETSINF
$parameters->endpoint = "http://tv.inf.upv.es/?feed=rss2";

// Indiquem metode GET per obtindre les dades
$parameters->method   = "GET";

// Indiquem que la resposta sera de tipus XML
$parameters->type     = "XML";

// Creem el client de REST
$client  = new FW_Rest_Client($parameters);
// Executem la peticio a REST
$client->exec();

// Obtenim la resposta
$response   = $client->getResponse();
// Obtenim les dades en XML
$dades      = $response->getBody();
?>
```

Processament dels feeds RSS

Una vegada s'han descarregat les dades del feed RSS podem passar a processar-lo. Per a això ens ajudarem de la biblioteca *SimpleXML* que ve de sèrie amb PHP.

Primerament processarem les dades del primer element *Channel* disponible en cada feed RSS, per a això ens crearem el següent mètode privat dins el controlador:

Listing 20.5: Obtindre les dades del canal RSS

```
<?php
private function _getChannelInfo(SimpleXMLElement $data) {
  $channel = array (
    "title"       => (string) $data->channel[0]->title,
    "description"=> (string) $data->channel[0]->description,
    "date"        => (string) $data->channel[0]->pubDate
  );
  return $channel;
```

```
9   }
10  ?>
```

Amb aquest mètode obtindrem totes les dades necessàries del primer canal RSS disponible en les dades XML que hem descarregat mitjançant el client REST. Una vegada obtingudes, les guardarem en un array associatiu i se les tornarem a l'acció del controlador.

Després processarem per el primer canal de cada feed RSS les entrades o items que aquest continga. Per aquesta tasca, també crearem un mètode privat dins el controlador:

Listing 20.6: Obtindre les entrades del canal RSS

```
1   <?php
2   private function _getChannelItems(SimpleXMLElement $data) {
3     $items   = array();
4     $pattern = '/<\s*img [^\>]*src\s*=\s*[\""\']?([^\""\'\s
          >]*)/i';
5     // si hi ha entrades dins el canal
6     if (isset($data->channel[0]->item)) {
7       // per a cada entrada del canal ...
8       foreach ($data->channel[0]->item as $item) {
9
10        // obtindre la primera imatge del contingut de l'
              entrada
11        $image = "";
12        if (preg_match($pattern, (string) $item->description,
              $matches)) {
13          $image = $matches[1];
14        }
15        // si no hi ha imatges, mostrar noimage.png
16        if (strlen($image)==0) {
17          $image = BASE_URL."images/noimage.png";
18        }
19
20        // obtindre les dades de cada entrada
21        $aux = array(
22          "title"       => (string) $item->title,
23          "description" => (string) $item->description,
24          "date"        => (string) $item->pubDate,
25          "link"        => (string) $item->link,
26          "image"       => $image
27        );
28
29        // guardar les dades de cada entrada en array
30        $items []= $aux;
31      }
32    }
33    return $items;
34  }
35  ?>
```

Juntant tot el codi

A continuació possarem tot el codi font junt en el controlador *newsPortal*:

Listing 20.7: Codi font per consumir un feed rss

```
<?php
public function newsPortal($source) {
      $feed     = "";
      $source   = $this->escape($source);
      $data     = "";
      $channels = array();
      $feed     = array();

      switch ($source) {

          default:
          case "etsinf":
              $feed []= "http://tv.inf.upv.es/?feed=rss2";
              break;

          case "upv":
              $feed []= "http://www.upv.es/pls/oalu/sic_rss.
                  rss_ver20?p_rss_feed=7&p_idioma=v";
              break;

          case "uji":
              $feed []= "http://www.uji.es/cocoon/xpfpub/rss
                  ?p_channel=2269";
              break;

          case "news":
              $feed []= "http://20minutos.feedsportal.com/c
                  /32489/f/478284/index.rss";
              break;

          case "all":
              $feed []= "http://20minutos.feedsportal.com/c
                  /32489/f/478284/index.rss";
              $feed []= "http://www.uji.es/cocoon/xpfpub/rss
                  ?p_channel=2269";
              $feed []= "http://www.upv.es/pls/oalu/sic_rss.
                  rss_ver20?p_rss_feed=7&p_idioma=v";
              $feed []= "http://tv.inf.upv.es/?feed=rss2";
              break;
      };

      if (count($feed)>0) {
          foreach ($feed as $url) {
              $parameters = new FW_Container_Parameter();
              $parameters->endpoint = $url;
              $parameters->method   = "GET";
              $parameters->type     = "XML";

              $client     = new FW_Rest_Client($parameters);
              $client->exec();
              $response   = $client->getResponse();
              $data       = $response->getBody();
```

```
47
48                  $channel    = $this->_getChannelInfo($data);
49                  $items      = $this->_getChannelItems($data);
50                  $channels []= array(
51                    "channel"=>$channel,
52                    "items"=>$items
53                  );
54              }
55          }
56          $this->setSlot("content","rest/consume/newsPortal",
                array("data"=>$channels));
57          $this->renderLayout("default");
58      }
59  }
60  ?>
```

Rutes

A continuació descriurem les rutes necessàries per a utilitzar l'acció que hem generat abans:

Listing 20.8: Rutes per accedir al servei web

```
1   <?php
2   FW_Router::Connect(
3   array (
4     'url' => '/webservices/rest/consume/news/:source',
5     'type' => 'app',
6     'cache' => false,
7     'parameters' => array (
8           'source' => array (
9         'name' => 'source',
10        'type' => 'string',
11        'format' => false,
12      )
13    ),
14    'authentication' => false,
15    'module' => 'webservices',
16    'controller' => 'newsPortal',
17    'action' => 'newsPortal',
18    'internal' => false,
19    'pattern' => '#^/webservices/rest/consume/news/(?:([^/]+))
          [/]*$#',
20    'parameterOrder' =>
21        array (
22          0 => 'source',
23        ),
24    )
25  );
26  ?>
```

S'ha de desar aquesta ruta en el fitxer *app/modules/webservices/config/routes.php* i un cop desada, netejar la caché de rutes.

Creació de les vistes i finalització del mini-agregador de feeds RSS

1. Creació de la vista *newsPortal*: Aquesta és la vista principal de l'agregador de feeds que es composa de les vistes parcials *channels*, *channel* i *item*. Aquesta vista recorrerà tota la col·lecció de canals RSS que s'haja descarregat el controlador, mostrant les dades del canal per a cada canal i les dades de cada entrada amb les vistes parcials corresponents.

Listing 20.9: Vista newsPortal

```
<div class="span-5 colborder column" style="padding: 15
    px;">
  <?php $this->renderView("rest/consume/channels"); ?>
</div>
<div class="span-16 column last">
  <?php
  if (count($data)>0) {
    foreach ($data as $channel) {
      $this->renderView("rest/consume/channel",array("
          channel"=>$channel["channel"]));
      if (count($channel["items"])>0) {
          foreach ($channel["items"] as $item) {
              $this->renderView("rest/consume/item",
                  array("item"=>$item));
          }
      }
      else { ?>
        <p>No hi ha noticies disponibles</p>
      <?php
      }
  }
}
?>
</div>
```

Aquest codi el desarem en el fitxer
`/app/modules/webservices/view/rest/consume/newsPortal.php`

2. Creació de la vista *channels*: Aquesta vista ens indica quines fonts RSS tenim disponibles, és una vista parcial que es situa en la part esquerra de la pàgina i mostra una llista desordenada amb les fonts RSS disponibles.

Listing 20.10: Vista channels

```
<h3>Fonts disponibles</h3>
<ul>
  <li><?php print html::link_to_internal("webservices","
      newsPortal","newsPortal","Totes les fonts",array("
      source"=>"all")); ?></li>
  <li><?php print html::link_to_internal("webservices","
      newsPortal","newsPortal","ETSINF",array("source"=>
      "etsinf")); ?></li>
```

```
5     <li><?php print html::link_to_internal("webservices","
          newsPortal","newsPortal","UPV",array("source"=>"
          upv")); ?></li>
6     <li><?php print html::link_to_internal("webservices","
          newsPortal","newsPortal","UJI",array("source"=>"
          uji")); ?></li>
7     <li><?php print html::link_to_internal("webservices","
          newsPortal","newsPortal","Noticies",array("source"
          =>"news")); ?></li>
8   </ul>
```

Aquest codi el desarem en el fitxer
`/app/modules/webservices/view/rest/consume/channels.php`

3. Creació de la vista *channel*: Aquesta vista ens mostrarà les dades de la font sel·leccionada per a visualitzar els articles que hi ha en ella.

Listing 20.11: Vista channels

```
1   <h3><?php print $channel["title"]; ?></h3>
2
3   <p><?php print $channel["description"]; ?></p>
4   <hr class="space" />
5   <p style="font-size: smaller;">Darrera modificacio <?php
        print $channel["date"]; ?></p>
6   <hr class="separator" />
7
8   <h6>Noticies</h6>
9   <hr class="space" />
```

Aquest codi el desarem en el fitxer
`/app/modules/webservices/view/rest/consume/channel.php`

4. Creació de la vista *item*: Aquesta vista parcial s'encarregarà de mostrar una entrada d'un canal.

Listing 20.12: Vista item

```
1   <div class="span-16 container last post">
2     <div class="span-3">
3       <a href="<?php print $item["link"]; ?>" title="<?php
            print $item["title"]; ?>"> <img src="<?php
           print $item["image"];?>" width="100" alt="<?php
           print $item["title"]; ?>" /> </a>
4     </div>
5     <div class="span-12 last">
6       <a href="<?php print $item["link"]; ?>" title="<?php
            print $item["title"]; ?>"><h4> <?php print
           $item["title"]; ?></h4> </a>
7       <p class="alt">Publicat el <?php print $item["date"
           ]; ?></p>
8       <hr/>
9       <?php print $item["description"]; ?>
10      <br />
```

```
11        <a href="<?php print $item["link"]; ?>"title="<?php
              print $item["title"]; ?>">Llegir mes</a>
12      </div>
13    </div>
```

Aquest codi el desarem en el fitxer
`/app/modules/webservices/view/rest/consume/item.php`

Amb aquestes quatre vistes que acabem de crear, podrem obtindre una aplicació web d'aspecte similar a la següent captura de pantalla:

Figura 20.4: Demostració de serveis web REST XML 'NewsPortal'

20.3 Creació de serveis web REST

20.3.1 Introducció

En aquesta secció es mostrarà com es poden crear serveis web amb el marc de treball.

20.3.2 Autenticació

Els serveis web REST suporten autenticació estàndard via el protocol *HTTP*. L'autenticació d'un servei web REST s'ha de configurar en la definició de rutes de cada operació del servei web en el paràmetre *authentication*. Existeixen tres opcions per aquest paràmetre:

- Valor a **false**: Desactivar autenticació.
- Valor a **true**: Autenticació activa amb qualsevol usuari que no estiga blocat.
- Valor a un **array**: Autenticació activa amb qualsevol usuari que tinga un rol dels definits en el array.

20.3.3 Creació d'un servei web REST senzill

Per introduir la creació de serveis web REST crearem un servei web molt senzill que consta de tres operacions:

- Obtindre el temps: *getTime*
- Saludar: *greet*
- Tornar el text enviat: *echoBack*

Totes aquestes funcionalitats es crearan en el mòdul **webservices** en el controlador *restController*

Operació «getTime»

Aquesta operació és molt senzilla, consistirà en tornar el temps (data i hora) del moment en el que s'instància el servei web.

Listing 20.13: Operació «getTime»

```
<?php
public function getTime() {
  $time = (string) date("d/m/Y H:i:s");
  return array("time"=>$time);
}
?>
```

Operació «greet»

Aquesta operació rep un paràmetre de tipus *string* anomenat «name» i retorna la cadena «Hello {$name}».

Listing 20.14: Operació «greet»

```
<?php
public function greet($name) {
  $name = utf8_encode($name);
  return array("response"=>"Hello {$name}!");
}
?>
```

Operació «echoBack»

Aquesta operació rep un paràmetre de tipus *string* via mètode *HTTP POST* i retorna el text enviat.

Listing 20.15: Operació «echoBack»

```
<?php
public function echoBack() {
  $string = "";

  // obtenim la instancia del servidor de REST
```

```
6     $server = FW_Rest_Server::getInstance();
7
8     // la cridada es amb el metode POST?
9     if ($server->isPOST()) {
10
11      // obtindre els continguts enviats
12      $contents = $server->contents();
13      if (!empty($contents)) {
14        $string = $contents["string"];
15      }
16    }
17    // respondre
18    return array("response"=>$string);
19  }
20  ?>
```

Rutes per aquesta demostració

A continuació es pressenten les rutes necessàries per a connectar aquest ≪mini≫ servei web amb el món exterior.

Listing 20.16: Rutes necessàries per aquest servei web

```
1   <?php
2   FW_Router::Connect(
3   array (
4     'url'     => '/webservices/rest/demo/time',
5     'type'    => 'rest',
6     'cache'   => false,
7     'parameters' => array (),
8     'authentication' => false,
9     'module' => 'webservices',
10    'controller' => 'rest',
11    'action' => 'getTime',
12    'method' => 'GET',
13    'internal' => false,
14    'pattern' => '#^/webservices/rest/demo/time[/]*$#',
15    'parameterOrder' => array ()
16    )
17  );
18
19  FW_Router::Connect(
20  array (
21    'url'     => '/webservices/rest/demo/saludar/:name',
22    'type'    => 'rest',
23    'cache'   => false,
24    'parameters' => array (
25        'name' => 'name',
26        'type' => 'string',
27        'format' => false
28    ),
29    'authentication' => false,
30    'module' => 'webservices',
31    'controller' => 'rest',
```

```
32      'action' => 'saludar',
33      'method' => 'GET',
34      'internal' => false,
35      'pattern' => '#^/webservices/rest/demo/saludar/(?:([^/]+))
            [/]*$#',
36      'parameterOrder' => array (0=>'name')
37      )
38    );
39
40    FW_Router::Connect(
41    array (
42      'url'    => '/webservices/rest/demo/echo',
43      'type'   => 'rest',
44      'cache' => false,
45      'parameters' => array (),
46      'authentication' => false,
47      'module' => 'webservices',
48      'controller' => 'rest',
49      'action'=> 'echoBack',
50      'method'=> 'POST',
51      'internal' => false,
52      'pattern' => '#^/webservices/rest/demo/echo[/]*$#',
53      'parameterOrder' => array()
54      )
55    );
56    ?>
```

20.3.4 Creació d'un servei web REST complet

Per aquesta demostració utilitzarem el controlador *restController*, emprat anteriorment i crearem un servei web REST que imite a una biblioteca molt bàsica amb cinc operacions bàsiques (obtindre tots els llibres, obtindre un llibre, crear un llibre, reemplaçar la col·lecció de llibres i esborrar un llibre).

Creació de la base de dades i del model

La base de dades per aquesta demostració serà molt senzilla. Crearem una taula *blog_book* (aprofitant el prefixe de la base de dades en la que hem creat el capítol del desenvolupament) amb els tres atributs (**isbn** de tipus *varchar (20)* i clau primària, **title** de tipus *varchar (255)* i **author**, també de tipus *varchar (255)*.

El codi SQL per a la creació de la taula *blog_book* és el següent:

Listing 20.17: Codi font per crear la taula *blog_book*

```
1   CREATE TABLE blog_book (
2     isbn   VARCHAR(20) PRIMARY KEY NOT NULL,
3     title  VARCHAR(255) NOT NULL,
4     author VARCHAR(255) NOT NULL
5   );
```

A continuació crearem el model *book*. Crearem un fitxer anomenat *book.class.php* dins el directori */app/lib/models* (el directori corresponent als models de les nostres

aplicacions web). En ell crearem el model *book* amb les propietats vistes anteriorment en la creació de la taula *blog_book*.

Listing 20.18: Codi font per crear el model *book*

```php
<?php

  class book extends FW_ActiveRecord_Model {
    public $isbn;
    public $title;
    public $author;
  };

?>
```

A continuació ja podrem crear totes les operacions previstes (accions del controlador *restController*).

Obtindre tots els llibres *getBooks*

Aquesta operació tornarà - si hi ha llibres en la base de dades - una llista de llibres emmagatzemats. Si no existeix cap llibre en la base de dades, tornarà un error avisant de que no hi ha llibres en la bbdd. Tornarem un array amb els llibres emmagatzemats en la base de dades.

Aquesta operació l'implementarem amb el següent codi:

Listing 20.19: Codi font per implementar l'acció *getBooks* al controlador i model

```php
<?php

// al controlador restController
public function getBooks() {
  $data  = array("book"=>array());

  // obtindre els llibres
  $books = $this->model()->getBooks();

  // si hi ha llibres ..
  if ($books->hasResult()) {
    // per a cada llibre obtindre la seva representacio en
       array
    foreach ($books as $book) {
      $data  ["book"][]=$book->toArray();
    }
    // tornem els llibres
    return array("books"=>$data);
  }
  else {
    // tornem un error
    return array("error"=>"Bookshelf is empty");
  }
}

// al model restModel
```

```
26  public function getBooks() {
27    $books  = book::find();
28    return $books;
29  }
30  ?>
```

Obtindre un llibre *getBook*

Aquesta operació tornarà el llibre corresponent al ISBN passat com a paràmetre (si existeix), en cas que no existeix-ca tornarà un error.

Listing 20.20: Codi font per implementar l'acció *getBook* al controlador i model

```
1   <?php
2
3   // al controlador restController
4   public function getBook($isbn) {
5     // obtindre el llibre amb el isbn ...
6     $book  = $this->model()->getBook($isbn);
7
8     // si s'ha trobat el llibre
9     if ($book!==null) {
10      // codificar en utf8 el autor del llibre
11      $book->author = utf8_encode($book->author);
12
13      // retornem informacio sobre el llibre
14      return array("book"=>$book->toArray());
15    }
16    else {
17      // retornem un error
18      return array("error"=>"The book you have requested doesn
            't exists");
19    }
20  }
21
22  // al model restModel
23  public function getBook($isbn) {
24
25    // condicions per cercar el llibre
26    $conditions = array(
27      array (
28        "name"     => "isbn",
29        "operator" => "=",
30        "value"    => $isbn
31      )
32    );
33    // cerquem el llibre
34    $book = book::find($conditions);
35
36    // si hem trobat el llibre, retornar el primer resultat
37    if ($book!=null && $book->hasResult()) {
38        return $book->first();
39    }
40  }
```

```
41  ?>
```

Esborrar un llibre de la col.lecció *deleteBook*

Aquesta operació esborrarà un llibre (segons el ISBN passat per paràmetre) de la base de dades. Tornarà una resposta exitosa o un error depenent si s'ha esborrat o no el llibre demanat. Per aquesta operació utilitzarem el mètode *HTTP DELETE*.

Listing 20.21: Codi font per implementar l'acció *deleteBook* al controlador i model

```
1   <?php
2
3   // al controlador restController
4   public function deleteBook($isbn) {
5
6     // cerquem el llibre que tinga l'isbn
7     $book = $this->model()->getBook($isbn);
8
9     // si no es troba el llibre
10    if ($book===null) {
11        return array("error"=>"The book you have requested
              doesn't exists");
12    }
13
14    // si s'esborra el llibre amb exit
15    if ($this->model()->deleteBook($book)) {
16        return array("success"=>"Book has been successfully
              removed from the bookshelf!");
17    }
18
19    // si hi ha algun error ...
20    else {
21        return array("error"=>"Book can't be removed from the
              bookshelf by some unknown error");
22    }
23  }
24
25  // al model restModel
26  public function deleteBook(book $book) {
27    return $book->delete();
28  }
29  ?>
```

Crear un llibre en la col.lecció *createBook*

Aquesta operació crearà el llibre passat com a paràmetre si les dades d'aquest són correctes (no existeix un llibre amb el mateix ISBN). Tornarà una resposta afirmativa o error depenent si s'ha creat el llibre o no. Per aquesta operació utilitzarem el mètode *HTTP POST*.

Listing 20.22: Codi font per implementar l'acció *createBook* al controlador i model

```
1   <?php
```

```
2
3   // al controlador restController
4   public function createBook() {
5
6     $isbn   = "";
7     $title  = "";
8     $author = "";
9
10    // obtenim instancia del servidor de REST
11    $server = FW_Rest_Server::getInstance();
12    // obtenim el format de la peticio (XML/JSON)
13    $format = $server->getFormat();
14
15    // si les dades van ser enviades en format XML
16    if ($format==="xml") {
17        $data   = $server->getContentsAsXML();
18
19        if ($data===null) {
20            return array("error"=>"Error 500, sent data was in
                  unknown format");
21        }
22
23        $data   = $data["book"];
24        $isbn   = (string) $data->isbn;
25        $title  = (string) $data->title;
26        $author = (string) $data->author;
27    }
28
29    // si les dades van ser enviades en format JSON
30    if ($format==="json") {
31        $data   = $server->getContentsAsJSON();
32
33        if ($data===null) {
34            return array("error"=>"Error 500, sent data was in
                  unknown format");
35        }
36
37        $data   = $data["book"];
38        $isbn   = $data["isbn"];
39        $title  = $data["title"];
40        $author = $data["author"];
41    }
42
43    // comprovem si el llibre existeix en la base de dades
44    if ($this->model()->getBook($isbn)!==null) {
45        return array("error"=>"There is an existing book with
              this ISBN {$isbn} {$data}");
46    }
47
48    $book         = new book();
49    $book->isbn   = $isbn;
50    $book->title  = $title;
51    $book->author = $author;
52    if ($this->model()->createBook($book)) {
```

```
53          // llibre creat amb exit
54          return array("success"=>"Book has been successfully
                added to bookshelf!");
55      }
56      else {
57          // error al crear el llibre
58          return array("error"=>"Book can't be added to the
                bookshelf by some unknown error");
59      }
60  }
61
62  // al model restModel
63   public function createBook(Book $book) {
64    if ($book->save()) {
65          return true;
66      }
67      return false;
68  }
69  ?>
```

Reemplaçar la col.lecció de llibres sencera *createBooks*

Aquesta operació esborrarà tots els llibres de la col·lecció i crearà una nova col·lecció amb tots els llibres amb les que s'envien les dades. Tornarà una resposta afirmativa o error depenent si s'han creat els llibres o no. Per aquesta operació utilitzarem el mètode *HTTP PUT*.

Listing 20.23: Codi font per implementar l'acció *createBooks* al controlador i model

```
1   <?php
2
3   // al controlador restController
4   public function createBooks() {
5
6     $books  = array();
7     $result = array();
8
9     // Obtenim la instancia del servidor REST
10    $server = FW_Rest_Server::getInstance();
11    // obtenim el format de la peticio HTTP (XML/JSON)
12    $format = $server->getFormat();
13
14    // si el format es XML
15    if ($format==="xml") {
16        $data   = $server->getContentsAsXML();
17        if ($data===null) {
18            return array("error"=>"Error 500, sent data was in
                  unknown format");
19        }
20        if (isset($data->book)) {
21            foreach ($data->book as $book) {
22                $isbn   = (string) $book->isbn;
23                $title  = (string) $book->title;
24                $author = (string) $book->author;
```

```
25                  $books []= array("isbn"=>$isbn,"title"=>$title
                        ,"author"=>$author);
26              }
27          }
28      }
29
30      // si el format es JSON
31      if ($format==="json") {
32          $data   = $server->getContentsAsJSON();
33
34          if ($data===null) {
35              return array("error"=>"Error 500, sent data was in
                     unknown format");
36          }
37          foreach ($data as $book) {
38              $isbn   = $book["isbn"];
39              $title  = $book["title"];
40              $author = $book["author"];
41              $books []= array("isbn"=>$isbn,"title"=>$title,"
                    author"=>$author);
42          }
43      }
44
45      // si hem obtingut llibres amb les dades enviades ...
46      if (count($books)>0) {
47          // esborrem tots els llibres
48          $this->model()->clearBooks();
49
50          // per a tots els llibres 'nous' ...
51          foreach ($books as $aux) {
52              // comprovem que no estiga duplicat el llibre
53              if ($this->model()->getBook($aux["isbn"])!==null)
                    {
54                  $result []= array("error"=>"There is an
                        existing book with this ISBN {$aux["isbn"]}
                        ");
55                  continue;
56              }
57
58              // creem el llibre
59              $book         = new book();
60              $book->isbn   = $aux["isbn"];
61              $book->title  = $aux["title"];
62              $book->author = $aux["author"];
63
64              // si es desa el llibre en la bbdd
65              if ($this->model()->createBook($book)) {
66                  $result []= array("success"=>"Book {$aux["isbn
                        "]} has been successfully added to
                        bookshelf!");
67              }
68              else {
69                  $result []= array("error"=>"Book {$aux["isbn"
                        ]} can't be added to the bookshelf by some
```

```
                    unknown error");
            }
        }
    }
    return array("results"=>$result);
}

// al model restModel
public function clearBooks() {
    $books = book::find();
    if ($books!==null && $books->hasResult()) {
      foreach ($books as $book) {
        $book->delete();
      }
    }
}
?>
```

Rutes

A continuació descriurem les rutes necessàries per a utilitzar les accions que hem generat abans:

Listing 20.24: Rutes per accedir al servei web

```
<?php
FW_Router::Connect(
array (
  'url' => '/webservices/rest/provide/books',
  'type' => 'rest',
  'cache' => false,
  'parameters' => array (),
  'authentication' => false,
  'module' => 'webservices',
  'controller' => 'rest',
  'action' => 'getBooks',
  'method' => 'GET',
  'internal' => false,
  'parameterOrder' => array()
  )
);

FW_Router::Connect(
array (
  'url' => '/webservices/rest/provide/books/:isbn',
  'type' => 'rest',
  'cache' => false,
  'parameters' => array (
      'isbn' => array (
        'name'    => 'isbn',
        'type'    => 'string',
        'format' =>false
      )
  ),
```

```
30      'authentication' => false,
31      'module' => 'webservices',
32      'controller' => 'rest',
33      'action' => 'getBook',
34      'method' => 'GET',
35      'internal' => false,
36      'parameterOrder' => array(0=>'isbn')
37      )
38  );
39
40  FW_Router::Connect(
41  array (
42      'url' => '/webservices/rest/provide/books',
43      'type' => 'rest',
44      'cache' => false,
45      'parameters' => array (),
46      'authentication' => false,
47      'module' => 'webservices',
48      'controller' => 'rest',
49      'action' => 'createBook',
50      'method' => 'POST',
51      'internal' => false,
52      'parameterOrder' => array()
53      )
54  );
55
56  FW_Router::Connect(
57  array (
58      'url' => '/webservices/rest/provide/books',
59      'type' => 'rest',
60      'cache' => false,
61      'parameters' => array (),
62      'authentication' => false,
63      'module' => 'webservices',
64      'controller' => 'rest',
65      'action' => 'createBooks',
66      'method' => 'PUT',
67      'internal' => false,
68      'parameterOrder' => array()
69      )
70  );
71
72  FW_Router::Connect(
73  array (
74      'url' => '/webservices/rest/provide/books/:isbn',
75      'type' => 'rest',
76      'cache' => false,
77      'parameters' => array (
78          'isbn' => array (
79              'name'    => 'isbn',
80              'type'    => 'string',
81              'format' =>false
82          )
83      ),
```

```
84      'authentication' => false,
85      'module' => 'webservices',
86      'controller' => 'rest',
87      'action' => 'deleteBook',
88      'method' => 'DELETE',
89      'internal' => false,
90      'parameterOrder' => array(0=>'isbn')
91      )
92  );
93  ?>
```

S'han de desar aquestes rutes en el fitxer *app/modules/webservices/config/routes.php* i un cop desades, netejar la caché de rutes.

Capítol 21

SOAP

21.1 Introducció

SOAP (Simple Object Access Protocol o Protocol Simple d'Accés a Objectes) és un protocol de comunicació dissenyat per intercanviar missatges en format XML en una xarxa d'ordinadors, normalment sobre el protocol HTTP. Habitualment s'usa per accedir a Serveis web. Està pensat per facilitar la comunicació entre aplicacions, independentment de la plataforma on s'executin i del llenguatge de programació en què estiguin implementades. És senzill i fàcilment extensible.

SOAP té els següents objectius:

1. Establir un protocol estàndard d'invocació de serveis remots basada en procotols estàndards d'internet: HTTP per la transmissió i XML per la codificació de les dades.

2. Independència de la plataforma, llenguatge de programació i sistema operatiu.

El protocol HTTP és el emprat en gran part de les connexions a internet. Garanteix que qualsevol client amb un navegador estàndard puga connectar-se amb un servidor remot. Per tant s'empra el protocol estàndard HTTP donades les moltes facilitats que aquest proporciona. La transmissió de dades s'empaqueta amb XML, que s'ha convertit en l'estàndard d'intercanvi de dades. SOAP basa completament l'intercanvi de missatges en documents XML amb un cert esquema i estàndard definits per la *W3C* en les seves recomanacions per SOAP. En la bibliografia corresponent a aquesta part s'exposen els enllaços a les recomanacions i estàndards de *SOAP* i *WSDL* definits per la **W3C**

21.1.1 Els components

En el *package* de SOAP podem distingir els següents components:

Client : La classe *FW_SOAP_Client* permet consumir serveis web proveïts per nosaltres o per tercers. És un simple embolcall sobre el client de SOAP proveït per PHP. El client és configurable i pot utilitzar-se en mode *WSDL* o mode simple, a més a més de poder emprar autenticació HTTP per poder utilitzar el servei web.

Server : La classe *FW_SOAP_Server* és un embolcall sobre el servidor SOAP proveït per el paquet *PEAR::SOAP* que permet construir servidors de SOAP sobre PHP

i emprant el protocol HTTP com a transport. Aquesta classe és capaç de generar automàticament (a través de PEAR::SOAP), el document *WSDL* necessari per descriure el servei web. Per descriure el servei web (que és el únic mode de funcionament del servidor SOAP). Aquest es basa en una sèrie de definicions de rutes amb un mateix endpoint (paràmetre URL) que enrutats per el sistema d'enrutat després de passar per el sistema de *Filtres* i el *FrontController* arriben al servidor SOAP; que munta un adaptador (classe *FW_SOAP_Adapter*) per cada ruta i analitza la seva descripció. Un cop analitzades totes les rutes i muntats els adaptadors corresponents li passa aquestos a *PEAR::SOAP* que genera tot el document *WSDL* que describeix el servei web. Per acabar, mitjançant la configuració de SOAP, situada en el fitxer *soap.php* es pot configurar cada servei web, afegir-li autenticació (només en el protocol HTTP i per a cada petició), descriure el servei i fixar el espai de noms de cada servei web.

Adapter : La classe *FW_SOAP_Adapter* permet fer exposar funcionalitats implementades en els nostres mòduls del sistema *MVC* a un servei web. Cada adaptador proveïx un accés a l'objecte que adapta (un controlador) i fa de pont - de forma transparent - entre el servidor de SOAP i el la funcionalitat. A més a més, cada adaptador descriu una operació del servei web, amb els seus paràmetres d'entrada, els paràmetres d'eixida i també els tipus de dades complexos que poden generar-se per encapsular tipus de dades complexos. El adaptador analitza la definició de la ruta (en PHP) que conté el servei que es vol adaptar i obté tota la informació esmentada abans i necessària per descriure l'operació del servei web.

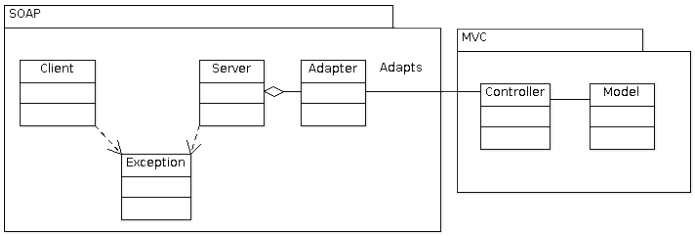


Figura 21.1: Diagrama de classes del component *Soap*

21.1.2 WSDL: *Web Services Description Language*

WSDL és l'acrònim de *Web Services Description Language*, un document amb una definició especial en format *XML* que és emprat per descriure serveis web i és ≪recomanació≫ per el consorci *W3C*.

WSDL descriu la interfície pública d'accés a serveis web basats en *SOAP*. En aquest document també apareixen la forma de comunicació, els requeriments del protocol i les operacions amb els missatges o paràmetres d'entrada i eixida així com el tipus de dades lligat a cada missatge per a cada operació de les que ofereix el servei web.

Amb un document WSDL ben format (és a dir, vàlid i sense errors lèxics segons l'esquema XML) un client pot llegir-lo, determinar quines operacions estan disponibles en el servei web, les definicions dels tipus de dades no escalars - o complexos - necessaris per a enviar o rebre missatges i els paràmetres d'entrada i d'eixida de cada operació.

Aquest tipus de document està avui en dia molt estès - ja que és la millor forma de descriure un servei web i fer-lo funcionar amb qualsevol plataforma o llenguatge de programació (principis bàsics d'interoperabilitat) - per tant, un client de serveis web SOAP pot llegir el document WSDL i saber quines operacions estan disponibles en el servidor remot. Amb aquesta informació, el client proporciona una interfície basada en el patró de disseny proxy, que permet cridar a operacions remotes com si estigueren implementades en el codi del sistema local, el client s'encarrega de transformar les dades i cridades als formats i transports necessaris per fer-les arribar al servidor de SOAP situat en una màquina remota, rebre les respostes, processar-les i fer-ho tot transparent al sistema local, és a dir «com si estigués dins el propi programa implementat tot».

21.2 Consumir serveis web SOAP

21.2.1 Introducció

Consumir serveis web que facen us de l'estàndard SOAP és ben senzill. Donada l'arquitectura dels serveis web i dels clients de SOAP és possible consumir serveis web escrits en qualsevol llenguatge de programació, en qualsevol sistema operatiu i que estiguen servint en qualsevol màquina remota o de la nostra xarxa. El client de SOAP analitza el servei web (si es disposa del document WSDL) i crea una sèrie d'objectes que representen l'estructura dels objectes que formen el servei web en el servidor.

Així doncs, el client de SOAP (utilitzant el document WSDL) és capaç de muntarnos l'estructura per fer les peticions al servei web de forma transparent, com si foren objectes del nostre entorn. També cal esmentar que el client de SOAP - a través del document WSDL - és capaç d'inferir els tipus de dades i nombres i ordres d'arguments de cada operació del servei web, per tant, és capaç de crear un representant del servei web remot en el nostre entorn.

21.2.2 API del client SOAP

El client de SOAP és un embolcall sobre el client de SOAP proporcionat per PHP amb algunes modificacions per fer més senzilla l'API.

Construcció (constructor) del client de SOAP

Listing 21.1: Constructor del client de SOAP

```
1  <?php
2  // constructor de FW_Soap_Client
3  __construct($wsdl=null,$username="",$password="",array
       $options=array());
4
5  /**
6   * Opcions de construccio:
7   */
8
9  // Nomes amb el fitxer WSDL
10 $wsdl   = "http://servidor.com/soap?wsdl";
11 $client = new FW_Soap_Client($wsdl);
12
```

```
13  // Amb wsdl,usuari i contrasenya (autenticacio http)
14  $wsdl    = "http://servidor.com/soap?wsdl";
15  $client = new FW_Soap_Client($wsdl,$usuari,$contrasenya);
16
17  // Altres combinacions (consultar http://www.php.net/manual/
        en/soapclient.soapclient.php )
18  $options = array(
19    'location' => "http://localhost/soap.php",
20    'uri'      => "http://test-uri/"
21  );
22  $client  = new FW_Soap_Client(null,$usuari,$contrasenya,
        $options);
23  ?>
```

Si estem provant un servei web desenvolupat per nosaltres i que està canviant constantment el *document WSDL*, haurem d'especificar-li al client de SOAP que no volem que utilitze la cache de documents WSDL (té una cache de 86400 segons que són 24h i manté el document WSDL durant eixe temps).

Listing 21.2: Constructor del client de SOAP sense caché de documents WSDL

```
1  <?php
2  // Client de SOAP sense cache WSDL
3  $wsdl    = "http://servidor.com/soap?wsdl";
4  $options = array (
5    "cache" => false
6  );
7  $client = new FW_Soap_Client($wsdl,$usuari,$contrasenya,
       $options);
8  ?>
```

Consultar `http://www.php.net/manual/en/soapclient.soapclient.php` per més opcions del constructor a través de l'array *options*.

Obtindre les operacions disponibles en el servei web

Obtindre les operacions disponibles en el servei web pot ser molt útil quan estem començant a utilitzar el servei web o l'estem depurant. Aquesta operació ens tornarà un array de cadenes de text amb la signatura de cada operació disponible.

Listing 21.3: Obtindre les operacions disponibles en el servei web

```
1  <?php
2  // constructor de FW_Soap_Client
3  array getAvailableOperations(void);
4
5  $wsdl       = "http://servidor.com/soap?wsdl";
6  $client     = new FW_Soap_Client($wsdl);
7
8  $operations = $client->getAvailableOperations();
9  if (count($operations)) {
10   foreach ($operations as $operation) {
11     print "Operacio: {$operation}";
12   }
13 }
14
```

```
15
16  // Eixida d'exemple
17  // Operacio: ArrayOfBooks getBooks()
18  // Operacio: Book getBook(string $isbn)
19  // Operacio: boolean deleteBook(string $isbn)
20  // Operacio: boolean createBook(Book $book)
21  ?>
```

Obtindre els tipus de dades complexos disponibles en el servei web

Obtindre els tipus de dades complexos disponibles en el servei web pot ser molt útil quan estem començant a utilitzar el servei web o l'estem depurant. Aquesta operació ens tornarà informació sobre cada tipus de dades complex disponible en el servei web.

Listing 21.4: Obtindre els tipus de dades complexos disponibles en el servei web

```
1   <?php
2   // constructor de FW_Soap_Client
3   array getComplexTypes(void);
4
5   $wsdl       = "http://servidor.com/soap?wsdl";
6   $client     = new FW_Soap_Client($wsdl);
7
8   $types      = $client->getComplexTypes();
9
10  var_dump($types);
11
12  /* Exemple d'eixida
13  array(2) {
14    [0]=>
15      string(60)
16        "struct Book {
17          string isbn;
18          string title;
19          string author;
20        }"
21    [1]=>
22      string(36)
23        "struct ArrayOfBooks {
24          Book items;
25        }"
26  }
27  */
28  ?>
```

Cridar a operacions del servei web

Per cridar a operacions del servei web, farem servir el client de SOAP cridant dins d'ell a les operacions, com si fos el client de SOAP l'objecte sobre el que realitzem les operacions. Cal recordar que el client de SOAP al implementar el *patró de disseny Proxy* trasllada la cridada al servei web esperant la resposta i les converteix a tipus de dades del nostre sistema transparentment.

Listing 21.5: Cridar a operacions del servei web

```
<?php
// Fitxer WSDL
$wsdl   = "http://servidor.com/soap?wsdl";

// Creem el client de SOAP
$client = new FW_Soap_Client($wsdl);

// Obtenim els llibres
$books  = $client->getBooks();
foreach ($books->item as $book) {
  print "{$book->title} ISBN: {$book->isbn}, per {$book->
      author} \n\n";
}

// Obtenim informacio d'un llibre per el ISBN
$book   = $client->getBook("978-84-415-2514-6");
if ($book!==null) {
  print "Dades del llibre amb ISBN 978-84-415-2514-6: ";
  print "{$book->title} ISBN: {$book->isbn}, per {$book->
      author} \n\n";
}

?>
```

Altres cridades d'utilitat

Per terminar, amb el client de SOAP podem obtindre informació sobre la darrera petició enviada o la resposta rebuda en el client. Són mètodes que ens poden ajudar a depurar el servei web.

Listing 21.6: Altres cridades d'utilitat en un client de SOAP

```
<?php
/* Altres operacions */

// Obtindre la darrera peticio
mixed getLastRequest(void);

// Obtindre la darrera resposta
mixed getLastResponse(void);
?>
```

21.2.3 Exemple: Consumir un servei web d'informació meteorològica

Per aquest exemple anem a obtindre dades d'un servei web d'informació meteorològica. Dissenyarem una xicoteta pàgina que ens permeta veure el temps de les principals ciutats del món. El servei web és accessible i gratuït a tothom a través del seu fitxer WSDL situat a `http://www.webservicex.net/globalweather.asmx?wsdl`. Observant aquest document WSDL es pot inferir que existeixen les següents operacions:

GetCitiesByCountry Aquesta operació obté un llistat de les principals ciutats d'un país.
Paràmetres:

CountryName : Nom del país en anglés (exemple: ≪Spain≫)

GetWeather Aquesta operació obté l'oratge d'una ciutat.
Paràmetres

CityName : Nom de la ciutat (exemple: ≪Valencia≫)

CountryName : Nom del país en anglés (exemple: ≪Spain≫)

Consumició del servei web

Per a consumir aquest servei web crearem les següents accions dins el controlador *soapController*

displayCitiesByCountry Aquesta acció mostrarà una llista de les ciutats principals de cada país fent us de l'operació *GetCitiesByCountry* del servei web. Per a cada ciutat a mostrar es mostrarà un enllaç que vaja a l'acció *displayCityWeather* del mateix controlador.

El codi font per crear aquesta acció és el següent:

Listing 21.7: Codi font de l'acció *displayCitiesByCountry*

```
1  <?php
2   public function displayCitiesByCountry($country) {
3          // Fitxer WSDL que descriu el servei web
4          $wsdl   = "http://www.webservicex.net/globalweather.
               asmx?wsdl";
5
6          // Creem un client de SOAP
7          $client = new FW_Soap_Client($wsdl);
8          // Cridem a la operacio GetCitiesByCountry
9          $data   = $client->GetCitiesByCountry(
10                 array("CountryName"=>$country)
11         );
12         // Un cop obtingudes les dades, les carreguem amb
               SimpleXML
13         $data   = @simplexml_load_string($data->
               GetCitiesByCountryResult);
14
15         // Si falla SimpleXML significa que ha ocorregut un
               error, mostrar error
16         if (!$data) {
17             $this->setSlot("content","soap/consume/oratge/
                   noCities",array("country"=>$country));
18         }
19         else {
20
21             $cities = array();
22             // Per a cada resultat obtenim el nom de la
                   ciutat i creem un enllac
```

```
23                 foreach ($data as $city) {
24                     $city     = str_replace('/','_',( (string)
                           $city->City));
25                     $cities []= html::link_to_internal("
                           webservices","soap","displayCityWeather",
                           $city,array("city"=>$city,"country"=>
                           $country));
26                 }
27                 // Mostrem la llista de ciutats
28                 $this->setSlot("content","soap/consume/oratge/
                       cities",array("country"=>$country,"cities"=>
                       $cities));
29             }
30
31             $this->renderLayout("default");
32         }
33     ?>
```

displayCityWeather Aquesta acció mostrarà l'oratge d'una ciutat fent us de l'operació *GetWeather* del servei web.

Listing 21.8: Codi font de l'acció *displayCityWeather*

```
1   <?php
2   public function displayCityWeather ($city,$country) {
3           // Transformem el nom de la ciutat
4           $city    = str_replace('_','/',urldecode($city));
5
6           // Fitxer WSDL que descriu el servei web
7           $wsdl    = "http://www.webservicex.net/globalweather
                .asmx?wsdl";
8           // Creem un client de SOAP
9           $client  = new FW_Soap_Client($wsdl);
10
11          // Cridem a la operacio GetWeather
12          $data    = $client->GetWeather(array("CityName"=>
                $city,"CountryName"=>$country));
13          // Transformem les dades
14          $data    = str_replace('utf-16','utf-8',$data->
                GetWeatherResult);
15          // Un cop obtingudes les dades, les carreguem amb
                SimpleXML
16          $data    = @simplexml_load_string($data);
17
18          // Si no hi ha dades, mostrem error
19          if (!$data) {
20              $this->setSlot("content","soap/consume/oratge/
                    noWeather",array("city"=>$city,"country"=>
                    $country));
21          }
22          else {
23              // Mostrem les dades de l'oratge en la ciutat
                    sel.leccionada
```

```
24                  $this->setSlot("content","soap/consume/oratge/
                        oratge",array("city"=>$city,"country"=>
                        $country,"data"=>$data));
25              }
26              $this->renderLayout("default");
27          }
28      ?>
```

Creació de les vistes

A continuació creem quatre vistes:

cities Aquesta vista mostrarà una llista sense ordre de les ciutats obtingudes després d'haver executat l'acció *displayCitiesByCountry* amb èxit, és a dir, després d'haver obtingut un resultat.

Listing 21.9: Codi font de la vista *cities*

```
1   <h3>Ciutats del pais <?php print $country; ?></h3>
2   <hr/>
3   <!-- Utilitzem el helper unorderedList de la classe HTML -->
4   <?php print html::unorderedList($cities,"list-style: none;")
        ;?>
```

noCities Aquesta vista mostrarà un error al no haver pogut obtindre dades l'acció *displayCitiesByCountry*.

Listing 21.10: Codi font de la vista *noCities*

```
1   <h3>Error</h3>
2   <hr/>
3   <p>No hi ha ciutats per el pais seleccionat (<?php print
        $country; ?>)</p>
```

weather Aquesta vista mostrarà una taula amb les dades obtingudes mitjançant l'acció *displayCitiesByCountry*.

Listing 21.11: Codi font de la vista *weather*

```
1   <h3>Oratge per a <?php print $city; ?> (<?php print $country
        ; ?>)</h3>
2   <hr/>
3   <table style="width: 75%; border: 0.5px solid black;">
4           <thead>
5                   <tr>
6                           <th>Dada</th>
7                           <th>Valor</th>
8                   </tr>
9           </thead>
```

```
10          <tbody>
11
12      <!-- Per a cada dada que ens retorna el servei web,
            mostrem el nom de la dada i el valor -->
13      <?php
14          foreach ($data as $key=>$value):
15      ?>
16          <tr>
17                  <td> <?php print $key;   ?> </td>
18                  <td> <?php print (string) $value; ?> </td>
19          </tr>
20      <?php
21          endforeach;
22      ?>
23
24          </tbody>
25  </table>
```

weather Aquesta vista mostrarà una taula amb les dades obtingudes mitjançant l'acció *displayCityWeather*.

Listing 21.12: Codi font de la vista *weather*

```
1   <h3>Oratge per a <?php print $city; ?> (<?php print $country
        ; ?>)</h3>
2   <hr/>
3   <table style="width: 75%; border: 0.5px solid black;">
4           <thead>
5                   <tr>
6                           <th>Dada</th>
7                           <th>Valor</th>
8                   </tr>
9           </thead>
10          <tbody>
11
12      <!-- Per a cada dada que ens retorna el servei web,
            mostrem el nom de la dada i el valor -->
13      <?php
14          foreach ($data as $key=>$value):
15      ?>
16          <tr>
17                  <td> <?php print $key;   ?> </td>
18                  <td> <?php print (string) $value; ?> </td>
19          </tr>
20      <?php
21          endforeach;
22      ?>
23
24          </tbody>
25  </table>
```

noWeather Aquesta vista mostrarà un error ja que l'acció *displayCityWeather* no ha pogut obtindre dades de l'oratge en la ciutat i país sel·leccionats.

Listing 21.13: Codi font de la vista *noWeather*

```
<h3>Error</h3>
<hr/>
<p>L'oratge no es disponible o no existeix la ciutat <?php
    print $city; ?> (<?php print $country; ?>)</p>
```

Rutes

A continuació descriurem les rutes necessàries per a utilitzar les dos accions que hem generat abans:

Listing 21.14: Rutes per accedir a les accions de l'exemple

```
<?php

FW_Router::Connect(
array (
  'url' => '/webservices/soap/consume/oratge/:country',
  'type' => 'app',
  'cache' => false,
  'parameters' =>
  array (
    'country' =>
    array (
      'name' => 'country',
      'type' => 'string',
      'format' => false,
    )
  ),
  'authentication' => false,
  'module' => 'webservices',
  'controller' => 'soap',
  'action' => 'displayCitiesByCountry',
  'internal' => false,
  'pattern' => '#^/webservices/soap/consume/oratge
     /(?:([^/]+))[/]*$#',
  'parameterOrder' =>
      array (
        0 => 'country',
      ),
  )
);

FW_Router::Connect(
array (
  'url' => '/webservices/soap/consume/oratge/veure/:city/:
     country',
  'type' => 'app',
  'cache' => false,
```

```
35      'parameters' =>
36      array (
37        'city' =>
38            array (
39            'name' => 'city',
40            'type' => 'string',
41            'format' => false,
42        ),
43        'country' =>
44            array (
45              'name' => 'country',
46              'type' => 'string',
47              'format' => false,
48            )
49      ),
50      'authentication' => false,
51      'module' => 'webservices',
52      'controller' => 'soap',
53      'action' => 'displayCityWeather',
54      'internal' => false,
55      'pattern' => '#^/webservices/soap/consume/oratge/veure
            /(?:([^/]+))/(?:([^/]+))[/]*$#',
56      'parameterOrder' =>
57          array (
58            0 => 'city',
59            1 => 'country',
60          ),
61      )
62  );
63
64  ?>
```

S'han de desar aquestes rutes en el fitxer *app/modules/webservices/config/routes.php*i un cop desades, netejar la caché de rutes.

Captures de pantalla

Per finalitzar aquesta demostració, aquestes dos captures de pantalla:

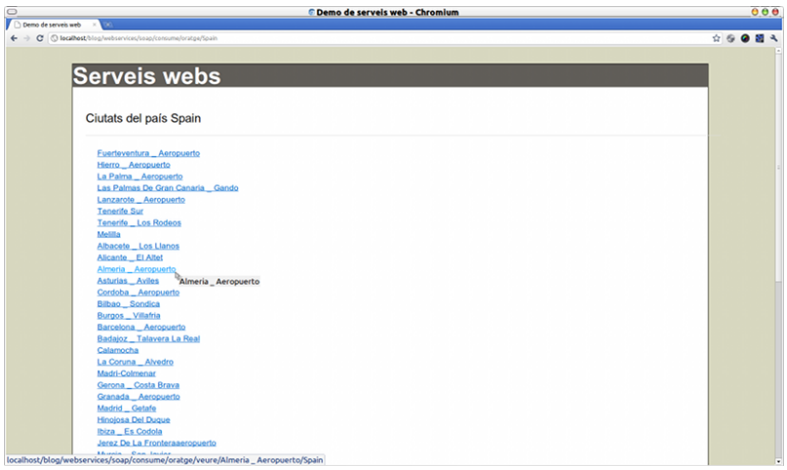


Figura 21.2: Captura de pantalla corresponent al resultat de l'acció *displayCitiesBy-Country* amb el país Espanya

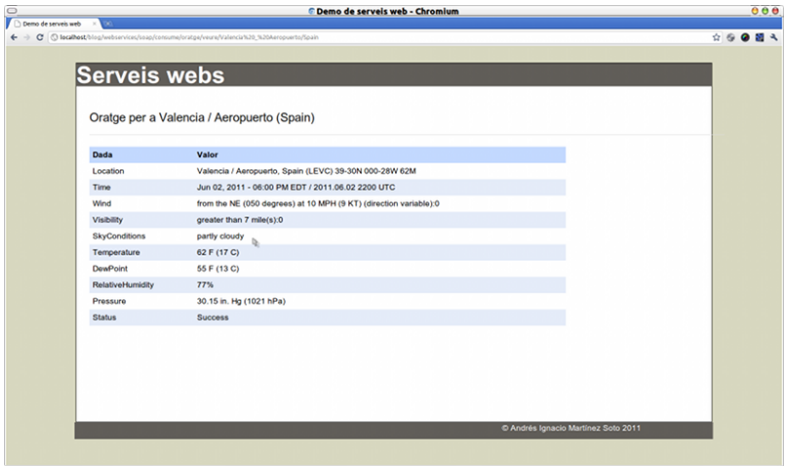


Figura 21.3: Captura de pantalla corresponent al resultat de l'acció *displayCityWeather* amb la ciutat de València i el país Espanya

21.3 Generació de serveis web SOAP

21.3.1 Introducció

En aquesta secció aprendrem a generar serveis web que funcionen sota el protocol SOAP. Per facilitar-nos la tasca de generar serveis web, el marc de treball proporciona - a través de la biblioteca externa *PEAR::SOAP* - tot el necessari per crear serveis web a partir d'objectes de PHP i per generar el document *WSDL* automàticament.

21.3.2 Funcionament

El servidor de SOAP (classe *FW_SOAP_Server*) rep una sèrie de rutes que prèviament ha seleccionat el component *Router* i que procedeixen de totes les rutes que tenen el **paràmetre url** al mateix valor. La idea és agrupar totes les funcionalitats d'un servei web en rutes amb la mateixa URL o *endpoint*. Una vegada es tenen totes aquestes rutes, el servidor de SOAP crea per a cadascuna d'elles el adaptador *FW_SOAP_Adapter* que analitzarà cada ruta i obtindrà els paràmetres d'entrada, eixida, tipus de dades complexos, descripcions, ... Després d'analitzar-se totes les rutes aportades, s'afegeixen al servidor de SOAP (classe *PEAR::SOAP_Server* i aquesta a través de les seves subclasses crea el servidor de SOAP i el *document WSDL*.

En els següents diagrames UML de seqüència es pot apreciar el procés per iniciar i configurar el servidor de SOAP [1].

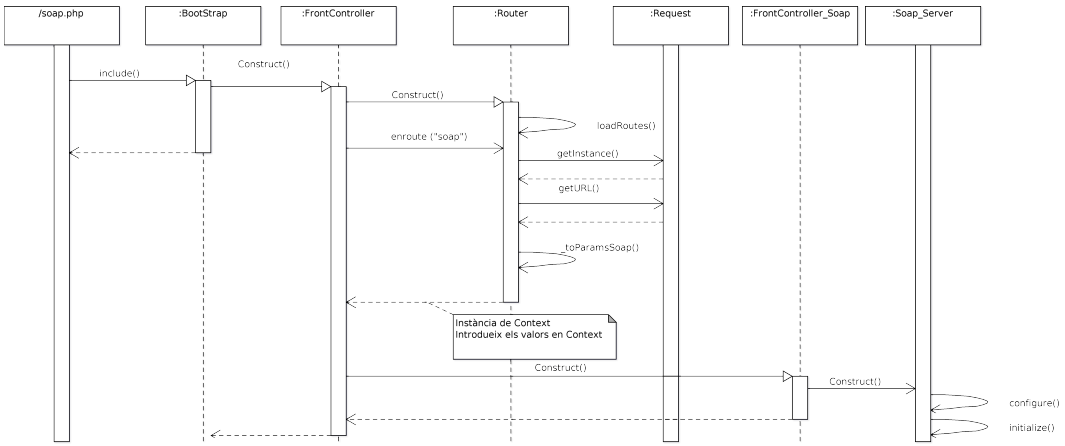


Figura 21.4: Diagrama UML de seqüència que ilustra el funcionament del servidor de SOAP (Part 1/2)

[1] Nota: S'ometen diversos processos per falta d'espai i mancança de claritat

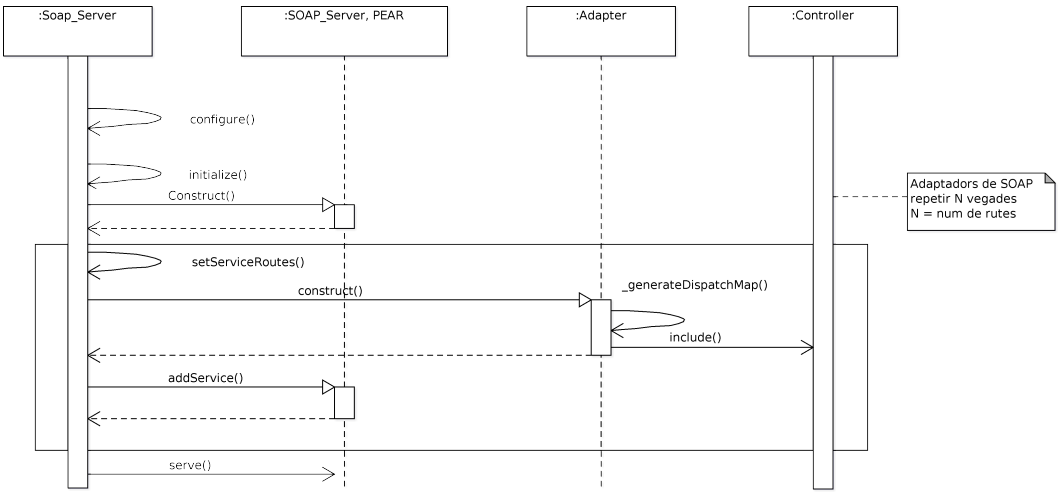


Figura 21.5: Diagrama UML de seqüència que ilustra el funcionament del servidor de SOAP (Part 2/2)

21.3.3 Descripció de serveis web

Com totes les funcionalitats del nostre sistema, el generar un servei web comença per generar les rutes corresponents al fitxer de rutes del mòdul corresponent, en aquest cas el mòdul «webservices», així que utilitzarem el fitxer */app/modules/webservices/config/routes.php* i allí posarem la informació de les nostres rutes .

Cada ruta que generem per un servei web SOAP ha de tindre la mateixa URL o endpoint. Amb aquesta URL el que farem és definir el endpoint per a que altres sistemes es puguen connectar al nostre servei web. A més a més, si al final de la URL (en una petició d'un client o consumint el nostre servei web), li afegim el paràmetre **?wsdl** obtindrem el document WSDL del nostre servei web. Podem tindre tots els serveis web amb totes les operacions que vulguem, la única diferència entre tots ells serà la URL, ja que el sistema agrupa operacions d'un servei web per URL.

Una vegada posada la informació de les rutes, s'ha d'esborrar la caché de rutes (veure apèndix sobre el tema) i recarregar el sistema per a que carregue les noves rutes.

A continuació veurem amb detall com crear paràmetres i tipus de dades complexos per a les nostres operacions de serveis web.

Paràmetres

Els paràmetres per a cada ruta es defineixen com en totes les rutes, mitjançant arrays dins de la pròpia definició de la ruta:

Listing 21.15: Configuració de paràmetres en rutes

```
<?php
  // en una definicio d'una ruta ...
  'parameters' => array (
      'isbn' => array (
          'name' => 'isbn',
          'type' => 'string',
          'format' => false,
```

```
        ),
        'Book' => array (
            'name' => 'book',
            'type' => '{urn:proves}Book',
            'format' => false,
        )
    ),
    // segueix la definicio de la ruta
?>
```

Tipus de dades complexos

Per crear tipus de dades complexos s'ha d'incloure una nova clau anomenada *typedef* dins la definició d'una ruta. Dins d'aquesta clau crearem un array en la que definirem els tipus de dades complexes de SOAP de la manera que es pot veure en l'exemple:

Listing 21.16: Configuració de tipus de dades complexos en rutes

```
<?php
  // en una definicio d'una ruta ...
 'typedef' => array (
    'Book' => array (
      'isbn'  =>'string',
      'title' =>'string',
      'author'=>'string'
     ),
     'ArrayOfBooks' => array(
        'items' => '{urn:proves}Book'
    )
),
// segueix la definicio de la ruta
?>
```

21.3.4 Configuració del servei web

La configuració d'un servei web es farà dins el fitxer */framework/config/soap.php*. En el fitxer, es trobarà una matriu d'arrays en PHP, que permetrà afegir dades relatives a cada servei web que creen. Dins l'array es distingixen dos arrays (**global** - que contindrà únicament la clau "active- i **sections** - en la que cada clau serà la descripció del servei web que duga el nom de la clau -). Aquest fitxer té el següent aspecte:

Listing 21.17: Configuració d'un servei web

```
<?php
$config = array(
    "sections" => array (
        "proves" => array (
            "name"           => "proves",
            "namespace"      => "proves",
            "description"    => "Servei web de proves",
            "authentication" => true,
            "realm"  => "Aquest servei web necessita
                autenticacio",
```

```
10                "cancel" => "No esta autoritzat a utilitzar
                      aquest servei web",
11                "roles" => array ( "editor","admin","client" )
12            )
13        ),
14        "global" => array (
15            "active" => true
16        )
17    );
18
19    FW_Config::createConfig("soap");
20    FW_Config::setConfig("soap",$config);
21    ?>
```

Descripció del servei web

Per a la descripció del servei web disposem dels següents paràmetres:

name : El nom del servei web. Aquest nom coincidirà amb el últim pedaç de la *URL* o *endpoint* emprada per accedir al servei web. Exemple: ≪http://elmeuservidor.com/soap/comprar?wsdl≫ correspondrà a ≪comprar≫.

namespace : L'espai de noms per el servei web.

description : Una descripció del servei web.

Cadascun d'aquestos paràmetres ha de ser configurat en aquest fitxer. De no ser configurats, el servidor de SOAP aplicarà la configuració per defecte, quedant-se tots aquestos paràmetres en el valor ≪default≫

Autenticació

Aquest servidor de SOAP suporta autenticació únicament via el protocol *HTTP*, per tant, per accedir-hi tant al document *WSDL* com al servei web en sí, es demanarà una autenticació bàsica HTTP mitjançant usuari i contrasenya, amb un missatge personalitzable per el desenvolupador que informarà als usuaris del servei que aquest necessita autenticació.

Per tal d'habilitar i configurar l'autenticació existeixen els següents paràmetres dins el fitxer de configuració:

authentication : Valor de true o false

realm : Missatge que apareixerà a l'usuari del servei web demanant-li l'autenticació.

cancel : Missatge que apareixerà a l'usuari del servei web si cancel·la el procés d'autenticació.

roles : Rols del sistema d'autenticació ACL amb els que es pot utilitzar el servei web. Aquest paràmetre ha de ser obligatòriament un array. Si volem permetre que qualsevol usuari enregistrat en el sistema puga fer us d'aquest servei web, caldrà deixar el paràmetre roles a un array buit (*"roles"=> array();*)

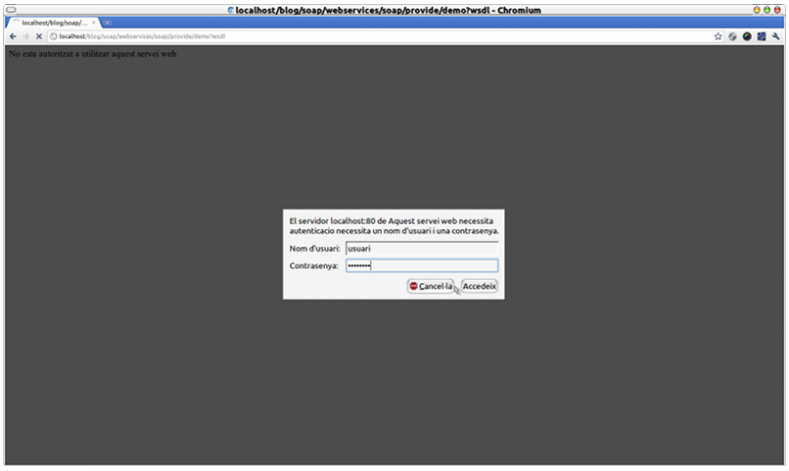


Figura 21.6: Autenticació HTTP bàsica per a serveis web SOAP

21.3.5 Generació d'un servei web

Per aquesta demostració utilitzarem el controlador *soapController*, emprat anteriorment amb la demo de consumició d'un servei web SOAP. Per aquesta demostració crearem un servei web SOAP que imite a una biblioteca molt bàsica amb quatre operacions bàsiques (obtindre tots els llibres, obtindre un llibre, crear un llibre i esborrar un llibre).

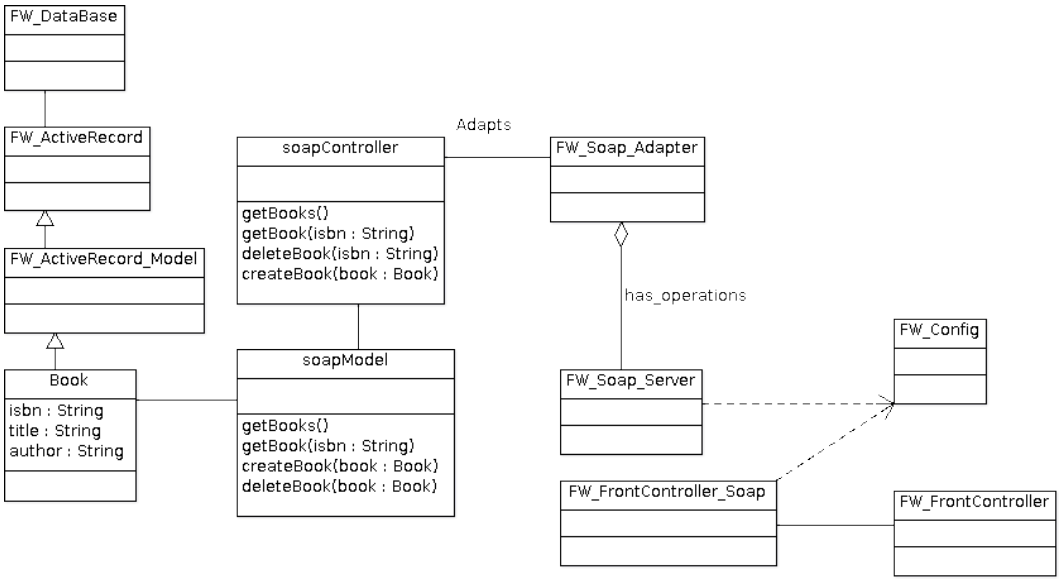


Figura 21.7: Diagrama UML de les classes implicades en la demostració

Configuració i rutes del servei web

Creem les rutes per accedir-ne a les funcionalitats del servei web:

Listing 21.18: Rutes del servei web de demostració

```
<?php

/* BEGIN Demo SOAP Server */
FW_Router::Connect(
  array (
    'url'         => '/webservices/soap/provide/demo',
    'type'        => 'soap',
    'cache'       => false,
    'authentication' => false,
    'module'      => 'webservices',
    'controller' => 'soap',
    'action'      => 'getBooks',
    'internal'    => false,
    'description' => 'Torna un array de llibres',
    'namespace'    => 'proves',
    'service'      => 'proves',
    'parameters'   => array(),
    'typedef' =>
        array (
          'Book'          => array ('isbn'=>"string",'title'
              =>'string','author'=>'string'),
          'ArrayOfBooks' => array("items"=>"{urn:proves}Book
              ")
        ),
    'return' =>
        array ('result' => '{urn:proves}ArrayOfBooks'),
    'pattern'         => '',
    'parameterOrder' =>
     array (),
  )
);

FW_Router::Connect(
  array (
  'url'           => '/webservices/soap/provide/demo',
  'type'          => 'soap',
  'cache'         => false,
  'authentication' => false,
  'module'        => 'webservices',
  'controller' => 'soap',
  'action'      => 'getBook',
  'internal'    => false,
  'description' => 'Torna un llibre',
  'namespace'    => 'proves',
  'service'      => 'proves',
  'parameters' => array (
    'isbn' => array (
        'name' => 'isbn',
        'type' => 'string',
        'format' => false,
    )
  ),
```

```
51      'typedef' => array (
52            'Book' => array ('isbn'=>'string','title'=>'string',
                  'author'=>'string')
53      ),
54      'return' => array ( 'result' => '{urn:proves}Book'),
55      'pattern' => '',
56      'parameterOrder' =>
57      array (0=>"isbn"),
58    )
59    );
60
61    FW_Router::Connect(
62    array (
63      'url'          => '/webservices/soap/provide/demo',
64      'type'         => 'soap',
65      'cache'        => false,
66      'authentication' => false,
67      'module'       => 'webservices',
68      'controller' => 'soap',
69      'action'     => 'deleteBook',
70      'internal'   => false,
71      'description' => 'Esborra un llibre',
72      'namespace'   => 'proves',
73      'service'     => 'proves',
74      'parameters' => array (
75        'isbn' => array (
76            'name' => 'isbn',
77            'type' => 'string',
78            'format' => false,
79        )
80      ),
81      'return' => array ( 'result' => 'bool'),
82      'pattern' => '',
83      'parameterOrder' =>
84      array (0=>"isbn"),
85    )
86    );
87
88    FW_Router::Connect(
89    array (
90      'url'          => '/webservices/soap/provide/demo',
91      'type'         => 'soap',
92      'cache'        => false,
93      'authentication' => false,
94      'module'       => 'webservices',
95      'controller' => 'soap',
96      'action'     => 'createBook',
97      'internal'   => false,
98      'description' => 'Crea un llibre',
99      'namespace'   => 'proves',
100     'service'     => 'proves',
101     'parameters' => array (
102       'book' =>  array (
103         'name' => 'book',
```

```
104         'type' => '{urn:demo}Book',
105         'format' => false
106       )
107     ),
108     'return' => array ( 'result' => 'bool'),
109     'pattern' => '',
110     'parameterOrder' =>
111     array (0=>"book"),
112   )
113   );
114
115   /* END Demo SOAP Server */
116
117   ?>
```

Configurem després el servei web en el fitxer */framework/config/soap.php*:

Listing 21.19: Configuració servei web de demostració

```
1   <?php
2   $config = array(
3       "sections" => array (
4           "demo" => array (
5               "name"           => "demo",
6               "namespace"      => "demo",
7               "description"    => "Servei web de proves",
8               "authentication" => false,
9               "realm"          => "",
10              "cancel"         => "",
11              "roles"          => array ()
12          )
13      ),
14      "global" => array (
15          "active" => true
16      )
17  );
18
19  FW_Config::createConfig("soap");
20  FW_Config::setConfig("soap",$config);
21  ?>
```

Creació de la base de dades i del model

La base de dades per aquesta demostració serà molt senzilla. Crearem una taula *blog_book* (aprofitant el prefixe de la base de dades en la que hem creat el capítol del desenvolupament) amb els tres atributs (**isbn** de tipus *varchar (20)* i clau primària, **title** de tipus *varchar (255)* i **author**, també de tipus *varchar (255)*.

El codi SQL per a la creació de la taula *blog_book* és el següent:

Listing 21.20: Codi font per crear la taula *blog_book*

```
1   CREATE TABLE blog_book (
2     isbn   VARCHAR(20) PRIMARY KEY NOT NULL,
```

```
3     title  VARCHAR(255) NOT NULL,
4     author VARCHAR(255) NOT NULL
5   );
```

A continuació crearem el model *book*. Crearem un fitxer anomenat *book.class.php* dins el directori */app/lib/models* (el directori corresponent als models de les nostres aplicacions web). En ell crearem el model *book* amb les propietats vistes anteriorment en la creació de la taula *blog_book*.

Listing 21.21: Codi font per crear el model *book*

```
1   <?php
2
3     class book extends FW_ActiveRecord_Model {
4       public $isbn;
5       public $title;
6       public $author;
7     };
8
9   ?>
```

A continuació ja podrem crear totes les operacions previstes (accions del controlador *soapController*).

Atenció: Si ja hem creat aquest model i taula de la base de dades al fer la demostració de serveis web REST no cal tornar-la a crear.

Obtindre tots els llibres *getBooks*

Aquesta operació tornarà - si hi ha llibres en la base de dades - una llista de llibres emmagatzemats. Si no existeix cap llibre en la base de dades, tornarà un *SoapFault* avisant de que no hi ha llibres en la bbdd. El tipus de dades tornat per aquesta operació serà el tipus de dades complex *ArrayOfBooks* en el que cada element de l'estructura serà del tipus de dades complex *Book*. Aquesta operació l'implementarem amb el següent codi:

Listing 21.22: Codi font per implementar l'acció *getBooks* al controlador i model

```
1   <?php
2
3   // soapController.class.php
4   public function getBooks() {
5     $data  = array();
6     // obtenim els llibres de la base de dades
7     $books = $this->model()->getBooks();
8     if ($books->hasResult()) {
9         foreach ($books as $book) {
10          // codifiquem cada llibre com un tipus complex
11          $data  []=  new SOAP_Value("Book","{urn:demo}Book",
                $book->toArray());
12        }
13        // retornem els llibres com un tipus complex '
              ArrayOfBooks'
14        return new SOAP_Value("ArrayOfBooks","{urn:demo}
              ArrayOfBooks",array($data));
```

```
15        }
16        else {
17          // si no hi ha llibres en la base de dades, retornem
                un error
18          return new SoapFault("No hi ha llibres disponibles en
                aquesta biblioteca");
19        }
20    }
21
22
23    // soapModel.class.php
24    public function getBooks() {
25      // obtenim tots els llibres
26      $books  = book::find();
27      return $books;
28    }
29
30
31    ?>
```

Obtindre un llibre *getBook*

Aquesta operació tornarà el llibre corresponent al ISBN passat com a paràmetre (si existeix), en cas que no existeix-ca tornarà un *SoapFault*. El tipus de dades tornat per aquesta operació serà el tipus de dades complex *Book*.

Listing 21.23: Codi font per implementar l'acció *getBook* al controlador i model

```
1   <?php
2
3   // soapController.class.php
4   public function getBook($isbn) {
5     if ($isbn!=="") {
6       // obtindre el llibre amb el isbn demanat
7       $book = $this->model()->getBook($isbn);
8
9       if ($book!==null) {
10        // si existeix el llibre, tornar-lo
11        return new SOAP_Value("Book","Book",$book->toArray());
12      }
13      else {
14        // si no existeix el llibre, error
15        return new SoapFault("1","Error: No existeix cap
              llibre amb isbn {$isbn}");
16      }
17
18    }
19    else {
20       // si el parametre esta buit, error
21      return new SoapFault("2","Error: El parametre isbn es
            buit");
22    }
23  }
24
```

```
25
26  // soapModel.class.php
27  public function getBook($isbn) {
28    // creem les condicions de la cerca
29    $conditions = array(
30      array (
31        "name"     => "isbn",
32        "operator" => "=",
33        "value"    => $isbn
34      )
35    );
36    // cerquem el llibre
37    $book = book::find($conditions);
38    if ($book!=null && $book->hasResult()) {
39      // si existeix el llibre, el retornem
40      return $book->first();
41    }
42  }
43
44  ?>
```

Esborrar un llibre de la col.lecció *deleteBook*

Aquesta operació esborrarà un llibre (segons el ISBN passat per paràmetre) de la base de dades. Tornarà una resposta amb true o false depenent si s'ha esborrat o no el llibre demanat.

Listing 21.24: Codi font per implementar l'acció *deletBook* al controlador i model

```
1   <?php
2   // soapController.class.php
3   public function deleteBook($isbn) {
4     // intentem cercar el llibre amb el isbn
5     $book = $this->model()->getBook($isbn);
6     if ($book!==null) {
7       // si trobem el llibre amb el isbn, intentem esborrar-lo
8       if ($this->model()->deleteBook($book)) {
9         // si s'esborra el llibre
10        return true;
11      }
12      // si no s'ha pogut esborrar el llibre
13      return false;
14    }
15    else {
16        // si no s'ha trobat el llibre, error
17        return new SoapFault("1","Error: No existeix cap
              llibre amb isbn {$isbn}");
18    }
19  }
20
21  // soapModel.class.php
22  public function deleteBook(Book $book) {
23    // intentem esborrar el llibre
24    if ($book->delete()) {
```

```
25          return true;
26      }
27      return false;
28  }
29  ?>
```

Crear un llibre en la col.lecció *createBook*

Aquesta operació crearà el llibre passat com a paràmetre si les dades d'aquest són correctes (no existeix un llibre amb el mateix ISBN). Tornarà una resposta amb true o false, depenent si s'ha creat el llibre o no.

Listing 21.25: Codi font per implementar l'acció *createBook* al controlador i model

```
1   <?php
2
3   // soapController.class.php
4   public function createBook($book) {
5     // comprovem si ja existeix un llibre amb el mateix isbn
6     if ($this->model()->getBook($book->isbn)!==null) {
7       // retornem un error
8       return new SoapFault("1","Ja existeix un llibre amb
            aquest isbn {$isbn}");
9     }
10    else {
11      // creem un llibre nou
12      $b = new Book();
13      $b->isbn = $book->isbn;
14      $b->title = $book->title;
15      $b->author = $book->author;
16      // intentem desar el llibre
17      if ($this->model()->createBook($b)) {
18          return true;
19      }
20    }
21    return false;
22  }
23
24  // soapModel.class.php
25  public function createBook(Book $book) {
26    // intentem desar el llibre
27    if ($book->save()) {
28      return true;
29    }
30    return false;
31  }
32  ?>
```

Creació d'un client de SOAP per provar el servei web de demostració

Creem un client de SOAP que ens permeta provar el servei web que acabem de crear. Aquest client el crearem en un fitxer a banda anomenat *books.php* i el posarem en el directori arrel on hem instal·lat el framework.

Listing 21.26: Client de SOAP per a provar el servei web de demostració

```php
<?php
  // forcem al client de soap a no cachear el WSDL
  ini_set("soap.wsdl_cache_enabled","0");

  class Book {
    public $isbn;
    public $title;
    public $author;
  };

  // adreca del fitxer wsdl
  $wsdl   = "http://localhost/blog/soap/webservices/soap/
      provide/demo?wsdl";
  // creem un client web
  $client = new SoapClient($wsdl);

  // provem accio getBooks
  $books  = $client->getBooks();
  foreach ($books->item as $book) {
    print "{$book->title} ISBN: {$book->isbn}, per {$book->
        author} \n\n";
  }

  // obtenim el llibre amb el ISBN ...
  $book   = $client->getBook("978-84-415-2514-6");
  if ($book!==null) {
    print "Dades del llibre amb ISBN 978-84-415-2514-6: ";
    print "{$book->title} ISBN: {$book->isbn}, per {$book->
        author} \n\n";
  }

  // creem un nou llibre
  $b         = new Book();
  $b->isbn   = "1";
  $b->title  = "Proves";
  $b->author = "Author1";

  // creem el libre
  $result = (bool) $client->createBook($b);
  if ($result===true) {
    print "Llibre creat\n\n";
  }

  // esborrem el llibre creat
  $result = (bool) $client->deleteBook("1");
  if ($result===true) {
    print "Llibre esborrat\n\n";
  }

?>
```

Listing 21.27: Possible eixida al executar el client

```
usuari@maquina:/var/www/framework$ php books.php
Ajax, JavaScript y PHP ISBN: 978-84-415-2514-6, per Ballard, Phil | Moncur,
    Michael | Gomez del Castillo, Rosario

Mi libro ISBN: 15, per blahblah

Dades del llibre amb ISBN 978-84-415-2514-6: Ajax, JavaScript y PHP ISBN:
    978-84-415-2514-6, per Ballard, Phil | Moncur, Michael | Gomez del
    Castillo, Rosario

Llibre creat

Llibre esborrat
```

Part VII

Temes avançats de desenvolupament

Capítol 22

Introducció

22.1 Introducció

En aquest capítol es presenten temes avançats de desenvolupament.

Capítol 23

Internacionalització

23.1 Introducció

La internacionalització (i18n) d'una aplicació web és una forma d'adaptar l'interfície de la mateixa a la llengua i costum de l'usuari, és a dir, fer que una aplicació estiga disponible en molts idiomes arribant així a molts usuaris.
Per a l'internacionalització es fa servir la biblioteca *GNU Gettext* a través de la versió de PHP i un programa que genera fitxers *PO* on s'indiquen quines cadenes són traduïbles i la seva traducció.

23.2 Marcatge de cadenes com a traduïbles

Per a poder traduir una cadena de text, aquesta ha d'estar marcada amb els caràcters _("..."); o mitjançant una cridada a la funció $\$string = gettext(\$cadena)$; cal notar que $\ll\_ \gg$ és un àlies de la funció gettext per tant, es recomana cridar a la funció _ en comptes de gettext ja que és més senzill d'escriure.

En el següent fragment de codi es resumeixen les posibilitats de marcatge de cadenes com a traduïbles.

Listing 23.1: Marcatge de cadenes com a traduïbles

```
1  <?php
2
3  $cadena = _("Cadena internacionalitzable");
4  $cadena = gettext("Cadena internacionalitzable");
5
6  print _("Aixo es una cadena traduible");
7  print gettext("Aixo es una cadena traduible");
8
9  ?>
```

Més documentació sobre Gettext per a PHP en l'adreça `http://php.net/manual/en/function.gettext.php`

23.3 Generació dels fitxers *PO* de traducció

Per tal de que l'aplicació funcione de manera internacionalitzable s'han d'arreplegar totes les cadenes marcades com a traduïbles en tota l'aplicació web i generar amb elles un fitxer *PO* per a cada idioma dels que es pensa traduir. Per fer aquest pas ens ajudarem amb el programa ≪PoEdit≫ descarregable des de `http://www.poedit.net/` o instal·lable amb l'ordre

Listing 23.2: Aspecte d'un fitxer PO

```
msgid ""
msgstr ""
"Project-Id-Version: Poedit 1.5\n"
"Report-Msgid-Bugs-To: poedit-devel@lists.sourceforge.net\n"
"POT-Creation-Date: 2010-05-07 08:40+0200\n"
"PO-Revision-Date: 2009-02-06 13:44+0100\n"
"Last-Translator: Fulanito de tal <fulanito@domini.com>\n"
"Language-Team: Fulanito de tal <fulanito@domini.com>\n"
"MIME-Version: 1.0\n"
"Content-Type: text/plain; charset=UTF-8\n"
"Content-Transfer-Encoding: 8bit\n"
"X-Poedit-Language: Catalan\n"

#: ../src/edframe.cpp:1999
msgid " (modified)"
msgstr " (modificat)"

#. TRANSLATORS: This is version information in about dialog,
     it is followed
#. by version number when used
#: ../src/edframe.cpp:2351
#, fuzzy
msgid " Version "
msgstr "versio"

#: ../src/edframe.cpp:1964
#, fuzzy, c-format
msgid "%i %% translated, %i strings"
msgstr "S'han traduit %u cadenes automaticament"
```

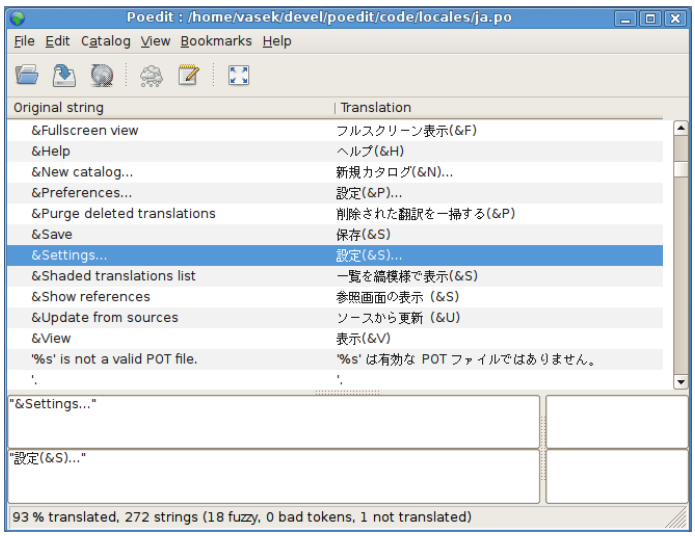


Figura 23.1: Aspecte de Poedit

Una vegada instal·lat el programa Poedit, crearem un projecte nou amb el nom ≪blog≫ al que li afegirem el directori */var/www/blog* o el directori en el que hagem instal·lat el marc de treball i l'aplicació de demostració.

Crearem un els següents directoris dins del directori *app/resources/locales* marc de treball *ca_ES/LC_MESSAGES* (per a l'idioma català) i *es_ES/LC_MESSAGES* per a l'idioma castellà i altres directoris per altres idiomes que vullguem traduir l'aplicació web.

Després crearem un fitxer ≪messages.po≫ (que contindrà les cadenes marcades com a traduïbles) en cadascun dels directoris creats anteriorment amb ajuda de les següents línies d'ordres.

Listing 23.3: Ordres per generar els fitxers .po

```
1  find . -type f -iname '*.php' | xargs  xgettext -n *.php --
       from-code utf-8 -o app/resources/locales/ca_ES/
       LC_MESSAGES/messages.po
2  find . -type f -iname '*.php' | xargs  xgettext -n *.php --
       from-code utf-8 -o app/resources/locales/es_ES/
       LC_MESSAGES/messages.po
```

Una vegada estiguen generats tants fitxers com siguen necessaris (un fitxer per idioma), s'obriran amb el programa *Poedit* i s'escomençaran a editar i traduir segons convinga. Cada vegada que es genere una nova cadena traduïble o a cada vista que es complete, és necessari actualitzar els fitxers *.po* per a que aquestos incloguen les noves cadenes que es volen traduir.

Capítol 24

Cache

24.1 Introducció

En aquesta secció es mostra l'utilització del component *Cache* del marc de treball per a millorar i optimitzar els temps de càrrega.

La cache és una petita - però ràpida - memòria temporal que permet desar dades molt utilitzades o que s'estan emprant en un moment determinat, en un indret del sistema amb tal d'obtindre-les amb el menor temps possible. En aquest projecte la cache permet desar dades temporals o permanents amb tal d'obtindre-les ràpidament i sense tornar-les a haver d'obtindre de les fonts habituals (bases de dades, serveis web, càlculs complicats,...), incrementant la velocitat i el rendiment de l'aplicació web.

24.2 Tipus de dades emmagatzemables en memòria cache

La cache pot desar dos tipus de dades:

Resultats d'accions : Es pot desar el resultat d'una acció (el codi font xhtml, document xml, document json, textos, ...) en la memòria cache.

Dades d'usuari : Es poden desar les dades que el desenvolupador considere en la memòria cache; tals com objectes, resultats de càlculs o dades externes al sistema.

24.3 Mecanismes d'emmagatzematge de dades en cache

La caché disposa de quatre mecanismes per desar les dades:

- En **fitxers** en el disc dur: Desa les dades en fitxers en el disc dur.
- En **bases de dades**: Desa les dades en una taula especialment dissenyada en la base de dades que utilitza el sistema.

- En base de dades **SQLite3** en memòria: Desa les dades en una base de dades en memòria en el gestor de bases de dades SQLite3, que escrit en C i molt optimitzada és una de les bases de dades més ràpides existents avui en dia.
- En **Memcache**: Utilitza una extensió de PHP per el sistema *Memcached*, que empra com a sistema d'emmagatzematge la memòria RAM, reduïnt així la necessitat d'accessos a disc dur o a bases de dades.

24.4 Identificació de dades en cache

Les dades en caché son identificades per tres paràmetres:

id : ID cadena de text que identifica unívocament dades desades en la cache. Aquest identificador ha de ser únic per a cada dada que s'haja de desar en la cache. Una bona forma de generar aquesta ID és aplicant l'algorisme *MD5* a una representació en text de les dades *serialitzades* o a una cadena de text. Donat que aquest algorisme produeix sempre el mateix *hash* per a les mateixes dades són ideals per identificar unívocament unes dades.

namespace : Cadena de text que indica *l'espai de noms* - o categoria - de les dades desades.

lifetime : *Temps de vida* en segons de la dada en caché. Pot variar des de un segon fins molts segons. Aquest temps de vida és el que determinarà si les dades han 'caducat' o segueixen essent vàlides. En cas que les dades hagen caducat, les dades hauran de ser regenerades i tornades a desar en caché.

24.4.1 Configuració

La configuració de la Cache resideix tota en el fitxer de configuració *framework/config/data/cache.php*.

En aquest fitxer - com podem comprovar al codi font d'exemple - es defineixen una sèrie d'entrades dins el array *sections*. Cada entrada contindrà informació sobre una configuració diferent d'aquest component, però totes aquestes entrades contindran almenys la següent informació:

enabled : Valor booleà que determina si aquesta configuració de caché està o no activa.

driver : Driver de caché emprat amb aquesta configuració.
Valors possibles (file,SQLite,memcache,database).

parameters : Array amb els paràmetres de configuració del driver corresponent (si escau).

Paràmetres per el driver **memcache**:

host : Nom del servidor on està en funcionament el servei de *Memcached*

port : Número sencer que indica el port del servei de *Memcached* en el servidor

Paràmetres per el driver **database**:

table : Nom de la taula de la base de dades en la que es desaran les dades de la caché.

Listing 24.1: Configuració del component *Cache*

```
<?php
$config = array(
  "sections" => array (
      "default"   => array (
          "enabled"    => true,
          "driver"     => "file",
          "parameters" => array ()
      ),
      "database" => array (
          "enabled"    => true,
          "driver"     => "database",
          "parameters" => array (
              "table" => "cache_object"
          )
      ),
      "memcache" => array (
          "enabled"    => true,
          "driver"     => "memcache",
          "parameters" => array (
              "host" => "localhost",
              "port" => 11211
          )
      ),
      "sqlite" => array (
          "enabled"    => true,
          "driver"     => "SQLite",
          "parameters" => array ()
      )
  ),
  "global" => array()
);

FW_Config::createConfig("cache");
FW_Config::setConfig("cache",$config);
?>
```

Caching de dades d'accions

Per a cada acció que s'haja de desar en la caché se li han d'afegir uns paràmetres en el fitxer de rutes corresponent a l'acció que la configuraran per a que automàticament es dese en caché.

Listing 24.2: Configuració d'una ruta com a cacheable

```
<?php
  // desactivar el cache de dades per aquesta ruta
  "cache" => false,
  // cachear l'eixida produida per aquesta accio 10 minuts
  "cache" => 600
?>
```

Si la clau *cache* de la definició de la ruta té com a valor false, aquesta ruta no serà cacheable, en cas contrari serà cacheable durant el temps en segons indicat al valor

d'aquesta clau. Es poden cachear accions de qualsevol tipus, que es desaran i regeneraran automàticament - quan el temps expire - en la caché que estiga configurada.

24.4.2 API

Obtindre la instància de la caché

Per obtindre la instància de la caché (donat que Caché és un component que implementa el patró de disseny *Singleton* s'emprarà el següent mètode:

Listing 24.3: Obtenció de l'instància del component *Cache*

```
<?php
 // Obtindre la instancia del component Cache
 FW_Cache static getInstance(void);

 $cache = FW_Cache::getInstance();
?>
```

Obtindre dades emmagatzemades en caché

Per obtindre unes dades emmagatzemades en la caché cal emprar el següent mètode

Listing 24.4: Obtenció de de dades de *Cache*

```
<?php
// Obtindre dades de cache
mixed get (string $id, string $namespace);

$data  = null;
$cache = FW_Cache::getInstance();
$dades = $cache->get("id","les_meves_dades");
if ($dades!==null) {
  if (!$dades->hasExpired()) {
    $data = $dades->getContents();
  }
}
?>
```

El mètode *get* torna **null** o un objecte de tipus **cache_object**.

Desar dades en caché

Per desar dades en caché cal emprar el següent mètode:

Listing 24.5: Desament de dades en la Caché

```
<?php
// Desar dades en cache
mixed save (string $id, string $namespace, mixed $value,
    double $lifetime)
// o el seu alias ...
mixed set (string $id, string $namespace, mixed $value,
    double $lifetime)
```

```
6
7   $cache = FW_Cache::getInstance();
8   $dades = array(1,2,3,4,5,6,8,9,10);
9   $cache->set("dades","les_meves_dades",serialize($dades),600)
        ;
10  ?>
```

Esborrar dades de caché

Per esborrar dades de caché cal emprar el següent mètode:

Listing 24.6: Esborrament de dades en la Caché

```
1   <?php
2   // Esborrar dades en cache
3   bool remove (string $id, string $namespace)
4
5   $cache = FW_Cache::getInstance();
6
7   $dades = array(1,2,3,4,5,6,8,9,10);
8   $cache->set("dades","les_meves_dades",serialize($dades),600)
        ;
9
10  $cache->remove("dades","les_meves_dades");
11  ?>
```

Esborrar totes les dades de caché

Per esborrar totes les dades d'un espai de noms de caché cal emprar el següent mètode:

Listing 24.7: Esborrament de dades d'un espai de noms en la Caché

```
1   <?php
2   // Esborrar dades d'espai de noms en cache
3   bool clean (string $namespace)
4
5   $cache = FW_Cache::getInstance();
6   $cache->clean("les_meves_dades");
7   ?>
```

Obtindre les dades d'un *cache_object*

El mètode *getContents()* d'un *cache_object* retorna les dades emmagatzemades en caché en eixe objecte.

Listing 24.8: Obtindre les dades d'un *cache_object*

```
1   <?php
2   // Obtindre les dades emmagatzemades en un cache_object
3   mixed getContents ()
4
5   $dades = null;
6   $obj   = $cache->get("id","les_meves_dades");
```

```
7   if ($obj!==null) {
8     $dades = $obj->getContents();
9   }
10  ?>
```

Verificar si ha expirat el contingut d'un *cache_object*

El mètode *hasExpired()* d'un *cache_object* proporciona informació sobre si ha expirat o no la informació desada en eixe objecte segons el paràmetre lifetime (temps de vida) amb el que es va crear.

Listing 24.9: Verificar si ha expirat el contingut d'un *cache_object*

```
1   <?php
2   // Verificar si les dades emmagatzemades en un cache_object
        han expirat
3   bool hasExpired ()
4
5   $dades = null;
6   $obj   = $cache->get("id","les_meves_dades");
7   if ($obj!==null) {
8     if (!$obj->hasExpired()) {
9       $dades = $obj->getContents();
10    }
11    else {
12      // regenerar les dades
13    }
14  }
15  ?>
```

24.4.3 Un exemple complet

En aquest exemple obtindrem el *feed RSS* de l'*ETSINF* i el desarem el caché per 600 segons (10 minuts):

Listing 24.10: Exemple complet

```
1   <?php
2   $dades = null;
3
4   // url del feed RSS de la UPV
5   $feed  = "http://tv.inf.upv.es/?feed=rss2"
6
7   // component cache
8   $cache = FW_Cache::getInstance();
9   $obj   = $cache->get(md5($feed),"rss");
10
11  if ($obj!==null) {
12      // tenim les dades
13      if (!$obj->hasExpired()) {
14        // no han expirat les dades
15        $dades = $obj->getContents();
16      }
```

```
17        else {
18          // regenerem les dades, ja que han expirat
19          $dades = file_get_contents($feed);
20          if (strlen($dades)>0) {
21            $cache->remove(md5($feed),"rss");
22            $cache->set(md5($feed),"rss",serialize($dades),600);
23          }
24        }
25    }
26    else {
27        // no tenim les dades, toca obtindre-les
28        $dades = file_get_contents($feed);
29        if (strlen($dades)>0) {
30          $cache->set(md5($feed),"rss",serialize($dades),600);
31        }
32    }
33    ?>
```

Capítol 25

Configuracions d'aplicació personalitzades

25.1 Introducció

A vegades és necessari que les aplicacions web tinguen una sèrie de valors relacionats amb la lògica de negoci de la pròpia aplicació. Aquestos valors poden ser constants, restriccions, valors computats anteriorment, regles de negoci, ...

Amb la finalitat de no emprar constants per representar aquestos valors en la nostra aplicació web i també amb una finalitat d'estar tots aquestos valors organitzats i accessibles a través d'una interfície comuna a les aplicacions d'usuari se'ls permet crear els seus fitxers de configuració i accedir-ne a aquestos amb una simple cadena de text de l'estil *negoci.sections.produccio.preu* o a través d'una classe derivada de *FW_Config_Handler* (classe ajudant que permet fer operacions sobre configuracions).

25.2 Creació de configuracions personalitzades

Per crear una configuració personalitzada s'ha de crear un fitxer (que la continga) en el directori */app/config* (exemple */app/config/proves.php*). Sempre és recomanable que el fitxer - i per tant el nom de la configuració - estiga format d'una paraula i sense accents,espais o punts que causen problemes; per tant, un nom no vàlid seria «proves.de.configuració» o «la meua configuració».

25.2.1 Esquelet d'un fitxer de configuració

Un cop creat el fitxer de configuració passarem a crear en ell l'esquelet d'una configuració, que ha d'estar composta **com a mínim** per un array amb dos claus **sections** i **global** . L'esquelet d'un fitxer de configuració seria el següent:

Listing 25.1: Esquelet d'un fitxer de configuració

```
1  <?php
2  $config = array(
3      "sections" => array (
4        // aci anira la configuracio detallada
```

```
        ),
        "global"    => array (
          // aci anira la configuracio global
        )
    );

    FW_Config::createConfig("proves");
    FW_Config::setConfig("proves",$config);
    ?>
```

25.2.2 Creació de valors dins la configuració

Després d'haver creat l'esquelet del fitxer de configuració ja podrem crear valors dins ell. En l'esquelet haviem distingit dues seccions (*global* i *sections*). En la secció global és on crearem els nostres valors que siguen «molt comuns a tota la nostra aplicació web» (exemple: permetre desar el log d'operacions, engegar o apagar un component,...), per altra banda, en la secció *sections* crearem valors concrets per a cada àrea que ens interesse (exemple: usuari i contrasenya del terminal de punt de venda virtual (TPVV)). En tot cas, els valors que podem donar-lis seran de la forma (**clau-valor**, **clau-array** o combinacions d'ambdues formes) d'altra manera no funcionarà.

Listing 25.2: Exemple de fitxer de configuració

```
<?php
$config = array(
    "sections" => array (
      "banc_del_parc" => array (
        "username" => "bpark102030",
        "password" => "bpk10224521"
      ),
      "banc_a_rrota"  => array (
        "username" => "bkrt9035",
        "key"      => "adsfasfadsjkfp1i1"
      )
    ),
    "global"    => array (
      "permetreTPVV" => true,
      "defaultTPVV"  => "banc_del_parc"
    )
);

FW_Config::createConfig("proves");
FW_Config::setConfig("proves",$config);
?>
```

25.2.3 Accés a valors de la configuració

Accedir a valors de la configuració és molt senzill. La forma d'accedir-ne a un valor de la configuració és mitjançant una clau (cadena de caràcters) de l'estil *proves.sections.banc_del_parc.username*.
Tota configuració té sempre dos seccions (*sections* o *global*), així totes les claus comencen per *proves.**sections***. o *proves.**global***. i després del últim punt el nom de la clau

a recuperar.
Si el valor de la clau de la configuració és un array poden presentar-se dos situacions:

1. Es vol obtindre **l'array complet**: Posarem en la clau el nom de l'array:
   Exemple, volem recuperar la configuració de banc_del_parc , llavors la clau seria *proves.sections.**banc_del_parc***.

2. Es vol obtindre un valor concret de l'array d'una configuració: Posarem la clau amb el nom de l'array i li afegirem un punt «.» i el nom de la clau de l'array que volem recuperar:
   Exemple, volem recuperar el nom d'usuari de banc_del_parc , llavors la clau seria *proves.sections.**banc_del_parc.username***.

Així, la configuració és totalment flexible (utilitzant *arrays clau-valor*), la informació es pot emmagatzemar d'eixes dues formes o de forma mixta (arrays amb valors que són arrays anidats o valors que són escalars). Per tant, és igual de senzill obtindre una informació que estiga dins de 5 arrays anidats que una informació que siga una entrada en la secció global, tot depèn del contingut de la clau i aquesta identificarà unívocament el valor al que es vol accedir.

A continuació es pot veure un exemple de com accedir a una configuració personalitzada:

Listing 25.3: Accés a una configuració personalitzada

```
<?php
  // metode get del component Config
  mixed get($key)

  // en un accio d'un controlador o en un metode d'un model
     ...
  /**
    Suposem que tenim una variable anomenat tipus (de
       pagament)
    que vindria determinada per certes condicions ..
  **/

  // obtenim la instancia del component Config
  $config       = FW_Config::getInstance();
  $clau         = "pagaments.sections.{$tipus}";

  // obtenim el valor que hi ha desat en eixa clau
  $configuracio = $config->get($clau);
  if ($configuracio!==null) {
      $tpvv = tpvv::factory($tipus,$configuracio);
      ...
  }
  else {
    throw new FW_User_Exception("No s'ha pogut trobar la
       forma de pagament {$tipus}");
  }
?>
```

Per veure més mètodes d'accés a una configuració personalitzada sense emprar-ne una classe ajudant, llegiu la documentació corresponent la classe *FW_Config*. És

recomanable - sempre que es puga - utilitzar una classe ajudant per a gestionar configuracions.

25.3 Creació d'ajudants per a configuracions personalitzades

Un ajudant o *helper* per una configuració personalitzada és una classe que permet gestionar un fitxer de configuració personalitzat amagant la lògica necessària per accedir o generar configuracions i proporcionant una interfície senzilla per accedir-ne a aquesta. Això facilita una separació completa entre l'aplicació web i la configuració personalitzada, de manera que l'accés o modificació d'aquesta no es fa directament des d'una acció d'un controlador sinó que la classe ajudant pot tindre una lògica que determinant algunes condicions d'accés o donades unes condicions horàries puga escollir una clau o altra de la configuració, ...

Per a tots aquestos problemes, podem crear una classe ajudant que farà el treball per nosaltres. Una classe ajudant és una classe de PHP derivada de la classe *FW_Config_Handler* (gestionador de configuració i que conté els mètodes necessaris per accedir, modificar-ne i afegir valors de/a la configuració) que s'ha de crear en el directori */app/config/Handler* i que ha de tindre de nom en majúscules el nom de la configuració a la que ≪ajuda≫. Exemple:

- configuració: *pagaments* , nom de la classe: *FW_Config_Handler_**Pagaments***, nom del fitxer: */app/config/Handler/Pagaments.class.php*

- configuració: *serveis_SMS* , nom de la classe: *FW_Config_Handler_**Serveis_SMS***, nom del fitxer: */app/config/Handler/Serveis_SMS.class.php*

25.3.1 Creació de la classe ajudant

A continuació es mostra el codi font mínim d'una classe ajudant per poder gestionar la configuració ≪pagaments≫:

Listing 25.4: Codi font mínim per a crear una classe ajudant de configuració

```
1  <?php
2    class FW_Config_Handler_Pagaments extends
         FW_Config_Handler {
3
4          //  Aci aniran les propietats
5
6          //  Aci aniran les operacions
7    };
8  ?>
```

Com es pot veure, aquesta classe deriva de la classe *FW_Config_Handler* que conté totes les operacions i propietats per actuar directament sobre un fitxer de configuració determinat. Com s'ha comentat abans, cada classe ajudant rep el nom del fitxer de configuració i de la configuració (tot en minúscules) sobre la que actua.

25.3.2 API de l'ajudant de configuració

Obtenció de valors

Per obtindre valors de la configuració a través d'un ajudant de configuració tenim el mètode *getParameter* que amb la clau corresponent s'encarrega d'obtindre dins la configuració. Aquest mètode pot tornar un valor escalar o array (si es troba la configuració amb la clau) o null si no troba la configuració amb la clau.

Listing 25.5: Obtenció de dades de la configuració dins una classe ajudant de configuració

```
<?php
  // obtindre un valor de la configuracio
  mixed getParameter(string $key);

  // exemples
  $valor = $this->getParameter("sections.{$tipus}.usuari");
  $valor = $this->getParameter("global.status");
?>
```

Modificació de valors

Per modificar-ne un valor existent a través d'un ajudant de configuració s'ha d'emprar el mètode *setParameter* que amb la clau del valor que es vol modificar i el valor a modificar canviarà el valor al que apunta eixa clau en la configuració.

Listing 25.6: Modificació de dades d'una configuració dins una classe ajudant de configuració

```
<?php
  // modificar un valor existent de la configuracio
  void setParameter(string $parameter,mixed $value);

  // exemples
  $this->setParameter("global.status","enabled");
  $this->setParameter("sections.{$tipus}.usuari",'');
?>
```

Creació de valors

Per afegir-ne o crear-ne un valor a través d'un ajudant de configuració s'ha d'emprar el mètode *addParameter* que amb la clau del valor que es vol crear i el valor a modificar crearà el valor al que apunta eixa clau en la configuració. Nota, cal esmentar que aquest mètode només funcionarà si el valor previ (el pare del valor que anem a crear) existeix i aquest és un valor de tipus array, sino estariem intentant crear una clau en un element que no és de tipus array i per tant no és possible crear elements dins d'un valor escalar.

Listing 25.7: Creació de dades d'una configuració dins una classe ajudant de configuració

```
<?php
  // afegir un valor a una configuracio
```

```
3     void addParameter(string $parameter,mixed $value);
4
5     // exemples
6     $this->addParameter("global.mailbox","enabled");
7     $this->setParameter("sections.mailbox.username","anmarso4"
          );
8   ?>
```

Recàrrega del fitxer de configuració

Es pot recarregar el contingut del fitxer de configuració (exemple: hem fet canvis que no són correctes, volem refrescar el contingut de la configuració, ...). Aquesta operació es fa a través del mètode *reload* de l'ajudant, esborra les dades de la configuració i de la caché de configuració, per tant, qualsevol dada no desada en el fitxer desapareixerà.

Listing 25.8: Recàrrega d'un fitxer de configuració dins una classe ajudant de configuració

```
1   <?php
2     // recarregar un fitxer de configuracio
3     void reload(void);
4
5     // exemples
6     $this->addParameter("global.mailbox","enabled");
7     print $this->getParameter("global.mailbox"); // -> enabled
8
9     $this->reload();
10    print $this->getParameter("global.mailbox"); // -> null
11  ?>
```

Desament de valors

Per desar el valor actual d'una configuració (si l'hem canviat), es pot fer servir el mètode *save*, que desarà totes les dades de la nostra configuració al fitxer de configuració sobre el que estem treballant.

Listing 25.9: Desament de les dades d'una configuració dins d'un ajudat de configuració

```
1   <?php
2     // desar les dades en un fitxer de configuracio
3     void save(void);
4
5     // exemples
6
7     // al principi no tenim valor
8     print $this->getParameter("global.mailbox"); // -> null
9
10    // afegim el parametre i li donem valor
11    $this->addParameter("global.mailbox","enabled");
12    print $this->getParameter("global.mailbox"); // -> enabled
13
```

```
14      // desem el fitxer de configuracio
15      $this->save();
16
17      // recarreguem el fitxer de configuracio
18      $this->reload();
19
20      // el valor existeix
21      print $this->getParameter("global.mailbox"); // -> enabled
22    ?>
```

25.3.3 Exemple complet

Listing 25.10: Exemple complet

```
1   <?php
2     class FW_Config_Handler_Pagaments extends
          FW_Config_Handler {
3
4       public function calcularPagament($valor,$tipus) {
5         if ($tipus==="llibre") {
6           $taxa = $this->getParameter("sections.taxes.llibres"
                );
7         }
8         if ($tipus==="menjar") {
9           $taxa = $this->getParameter("sections.taxes.menjar")
                ;
10        }
11        // calcul per a la taxa+valor ...
12      }
13
14      public function adjustarTaxa($tipus,$valor) {
15        $taxa = $this->getParameter("sections.taxes.{$tipus}")
              ;
16        if ($taxa!==null) {
17          $this->setParameter("sections.taxes.{$tipus}",$valor
                );
18          $this->save();
19          $this->reload();
20        }
21      }
22
23    };
24  ?>
```

Capítol 26

Creació i configuració dels estils CSS d'una aplicació web

26.1 Introducció

El marc de treball amb l'ajuda del component *Style* és capaç de gestionar la càrrega de fitxers d'estil *CSS* a demanda segons l'acció que s'estiga executant.

26.2 Crear un tema

Crear un tema és cosa fàcil. Per crear un tema crearem un fitxer de text en el que desarem la configuració del tema en forma d'arrays de PHP. Cada fitxer de definició dels nostres temes estarà situat en el directori */style/NOM_DEL_NOSTRE_TEMA* i tindrà de nom de fitxer *style.php* Els següents paràmetres són obligatoris:

name : Nom del tema.

version : Versió del tema.

description : Informació sobre el tema (descripció,...) .

developer : Nom i/o correu del desenvolupador del tema.

screenshot : Captura de pantalla del tema (per mostrar-ne en el canviador de temes).

isDefaultTheme : Indica si aquest tema és el tema per defecte.

enabled : Indica si el tema es pot utilitzar o no.

Després s'han de crear els arrays de PHP amb les claus *default* (els estils per defecte per a tot el sistema en cas que no tinga estils per defecte el mòdul i el controlador de l'acció que s'execute tampoc tinga estils definits) i *modules* dins de l'array que defineix el tema. Dins de l'array que té clau *modules*, crearem altres dos arrays amb les claus *external* (per a mòduls de la nostra aplicació web) i *internal* (per a mòduls del sistema).

Per cada mòdul (siga *internal* o *external*), definirem un array amb la informació del mòdul d'acord amb el següent codi font:

Listing 26.1: Descripció dels estils d'un mòdul

```
<?php
  "NOM_MODUL"  => array (

    "default"=> array (
      // estils per defecte per accions d'aquest modul
    ),
    "controllers" => array (
      // estils per cada controlador d'aquest modul

      "controlador1" => array (
        // estils per el controlador1
      ),
      "controlador2" => array (
        // estils per el controlador2
      )
    )
  )
?>
```

Juntant tots els paràmetres anteriors amb les definicions per a cada mòdul - no és obligatori tindre'ls tots definits; si no està definit un, passarà a l'estil per defecte - que tinguem de la nostra aplicació tindrem un fitxer amb contingut similar al següent codi font:

Listing 26.2: Descripció d'un tema

```
<?php
    $theme = array (
        // dades generals
        "name"           => "default",
        "version"        => "1.0",
        "description"    => "default theme",
        "developer"      => "Andres Ignacio Martinez Soto",
        "screenshot"     => "screen.png",
        "isDefaultTheme" => true,
        "enabled"        => true,

        // dades per a cada modul
        "modules"        => array (

          "external" => array(

            "index"  => array (
              "controllers" => array (),
              "default"=> array (
                0 => array("file"=>"reset.css","media"=>
                    array("screen"),"alternate"=>false),
                1 => array("file"=>"blueprint/screen.css","
                    media"=>array("screen"),"alternate"=>
                    false),
                2 => array("file"=>"application/index/style.
                    css","media"=>array("screen"),"alternate"
                    =>false)
```

```
23                  )
24                ),
25                "content"  => array (
26                  "controllers" => array (
27                    "post"=> array (
28                      0 => array("file"=>"reset.css","media"=>
                            array("screen"),"alternate"=>false),
29                      1 => array("file"=>"blueprint/screen.css",
                            "media"=>array("screen"),"alternate"=>
                            false),
30                      2 => array("file"=>"application/content/
                            style.css","media"=>array("screen"),"
                            alternate"=>false)
31                    ),
32                    "page" => array (
33                      0 => array("file"=>"reset.css","media"=>
                            array("screen"),"alternate"=>false),
34                      1 => array("file"=>"blueprint/screen.css",
                            "media"=>array("screen"),"alternate"=>
                            false),
35                      2 => array("file"=>"application/content/
                            pages/style.css","media"=>array("screen
                            "),"alternate"=>false),
36                    )
37                  )
38                ),
39              ),
40
41              "internal" => array()
42            ),
43            "default"  => array (
44                // dades dels estils per defecte per a tota l'
                    aplicacio web
45              0 => array("file"=>"reset.css","media"=>array("
                  screen"),"alternate"=>false),
46              1 => array("file"=>"blueprint/screen.css","media"
                  =>array("screen"),"alternate"=>false),
47              2 => array("file"=>"blueprint/print.css","media"=>
                  array("print"),"alternate"=>false),
48              3 => array("file"=>"application/style.css","media"
                  =>array("screen"),"alternate"=>false),
49              4 => array("file"=>"application/alternate_style.
                  css","media"=>array("screen","print"),"
                  alternate"=>true),
50            )
51
52        );
53    ?>
```

26.3 Configuració del component Style

La configuració del component *Style* resideix en el fitxer de configuració */framework/-config/style.php* . En ell es defineixen els paràmetres bàsics del component *Style*, com són la configuració de la base de dades per carregar els temes de l'usuari a demanda, el tema per defecte i alguns altres paràmetres.

Els paràmetres bàsics a configurar són:

enabled : Valor booleà que defineix si està actiu o no el component *Style*.

defaultTheme : Cadena de caràcters que defineix el nom del tema per defecte.

userThemeHasPriority : Valor booleà que indica si el tema d'usuari té prioritat sobre el tema de l'aplicació.

database : Defineix les dades de la base de dades en la que resideix les dades del tema d'usuari.
Es composa de:

- **source** : Cadena de caràcters que indica el nom de la taula de la base de dades.
- **column** : Cadena de caràcters que indica el nom de la columna que conté la informació del tema d'usuari.
- **username** : Cadena de caràcters que indica el nom de la columna que conté el nom d'usuari.

El següent exemple mostra un codi de configuració del component *Style* per defecte.

Listing 26.3: Configuració del component *Style*

```
<?php
$config = array(
  "sections" => array (),
  "global"    => array (
    "enabled" => true,
    "database" => array (
      "source"   => "user",
      "column"   => "theme",
      "username" => "username"
    ),
    "defaultTheme"         => "default",
    "userThemeHasPriority" => true
  )
);
FW_Config::createConfig("style");
FW_Config::setConfig("style",$config);
?>
```

26.3.1 Configuració dels temes d'usuari

Un tema d'usuari és aquell tema que es carrega segons l'usuari escollix. Per a poder carregar el tema escollit per l'usuari hem de desar el nom del tema a la base de dades. El lloc ideal per desar aquesta informació és la taula que continga l'informació dels

usuaris del sistema. Per a això a aquesta taula li afegim una columna de tipus *VARCHAR* o *TEXT* que anomenarem *style*, en aquesta columna, cada usuari tindrà el seu tema, és a dir, per a cada usuari que estiga en el sistema desarem també el tema amb el que vulga veure l'aplicació web.

Capítol 27

Tasques programades: CRON

27.1 Introducció

A vegades és necessari que determinades tasques en una aplicació web siguen realitzades automàticament i d'una manera regular cada determinats minuts/hores.

Per a poder fer funcionar aquestes tasques, aquest projecte incorpora un mètode de funcionament anomenat *CRON* que permet exposar una determinada funcionalitat a la interfície del programador de tasques dels sistemes operatius Unix, GNU-Linux o Windows. El programador de tasques CRON dels sistemes operatius tipus UNIX o el programador de tasques de Windows permeten la programació de tasques a executar a intervals de temps; aquestos programadors permeten tant executar un *script* o programa per línia d'ordres, com fer una petició a una determinada pàgina web.

Exemples de tasques programades CRON són:

- Fer còpia de seguretat de la base de dades diàriament.
- Generar informes de vendes automàticament cada 5h.
- Enviar «newsletters» cada primer dia de mes.

27.2 Preparació d'ordres cron

Per generar una funcionalitat i exposar-la a la interfície CRON només cal escriure el codi de la funcionalitat i especificar en la seva definició de ruta que aquesta funcionalitat és de tipus cron.

Listing 27.1: Exemple d'acció de tipus *CRON*

```
1  <?php
2  public function writeLog() {
3    $time = (string) date("d/m/Y H:i:s");
4    $fp   = fopen("/tmp/log","a+");
5    if ($fp) {
6      fwrite($fp,"{$time}\n");
7      fclose($fp);
8      return 0;
9    }
```

```
10      return -1;
11    }
12    ?>
```

Per últim, la ruta necessària per utilitzar aquesta funcionalitat CRON.

Listing 27.2: Ruta necessària per accedir a la funcionalitat *CRON*

```
1   <?php
2   FW_Router::Connect(
3       array (
4         'url'       => '/cron/demo',
5         'type'      => 'cron',
6         'cache'     => false,
7         'authentication' => true,
8         'module'      => 'cron',
9         'controller' => 'cron',
10        'action'      => 'writeLog',
11        'internal'    => false,
12        'parameters' => array (),
13        'pattern'     => '#^/cron/demo[/]*$#',
14        'parameterOrder' => array()
15      )
16  );
17  ?>
```

27.3 Autenticació en tasques CRON

L'autenticació per a tasques CRON està implementada com a autenticació estàndard mitjançant el protocol HTTP, per tant, si volem que siga necessària autenticació per executar una tasca CRON haurem de configurar-ho en la seva definició de ruta amb els valors corresponents (*true* si volem autenticar amb qualsevol usuari del sistema o un *array* amb els rols que hauran de tindre els usuaris).

27.4 Execució de tasques CRON

Per executar tasques CRON tenim dos opcions, les quals utilitzarem segons convinga o tinguem limitat el servidor:

27.4.1 Execució de tasques CRON des de línia d'ordres

Aquesta és l'opció clàssica i que podriem emprar amb el programador de tasques del sistema operatiu. Mitjançant aquesta opció executem l'script *cron.php* al que li passarem com a paràmetre la URL de la tasca CRON i l'usuari i contrasenya - si escau - . La tasca CRON en cas que incloga paràmetres els posarem en la URL i li passarem a la tasca CRON la URL amb els paràmetres.

Listing 27.3: Execució de tasques *CRON* des de línia d'ordres

```
1   usuari@maquina:/var/www/framework$ php cron.php /cron/demo
2
3   usuari@maquina:/var/www/framework$ php cron.php /cron/demo/param1/param2
        usuari contrasenya
```

27.4.2 Execució de tasques CRON des del navegador

Si el sistema o el nostre hosting no ens permet executar tasques cron des de consola o programar directament el programador de tasques però sí ens permet programar tasques a través d'una interfície web simple que permet fer una petició a una URL cada determinat temps, utilitzarem aquesta opció. Aquesta opció consisteix en fer una petició a la URL de la tasca CRON amb els paràmetres inclosos. Si la tasca necessita autenticació, aquesta es demanarà a través del navegador o es permetrà inserir en la configuració del programador de tasques del nostre hosting.

Part VIII

Apèndixos: Instal·lació, configuració

Capítol 28

Instal·lació

28.1 Instal·lació

La instal·lació d'aquest marc de treball és molt senzilla i similar per a tots els sistemes operatius. Només s'han de complir el següents requeriments:

- Servidor web amb suport per a PHP (recomanat Apache).
- PHP versió 5.3 o superior.
- Base de dades relacional (MySQL recomanada)

Una vegada cerciorats que tenim els requeriments, instal·larem les dependències (servidor web, php i base de dades) per a fer funcionar el marc de treball. Aquesta instal·lació es farà mitjançant l'inserció de les següents ordres en la línia d'ordres:

Listing 28.1: Instal.lació de PHP en GNU-Linux

```
usuari@maquina:$ sudo aptitude install apache2
usuari@maquina:$ sudo aptitude install php5
usuari@maquina:$ sudo aptitude install mysql-server
usuari@maquina:$ sudo aptitude install libapache2-mod-auth-mysql
usuari@maquina:$ sudo aptitude install php5-mysql
usuari@maquina:$ sudo aptitude install php5-cli
usuari@maquina:$ sudo aptitude install php5-curl
usuari@maquina:$ sudo aptitude install libapache2-mod-php5
usuari@maquina:$ sudo aptitude install php5-pear
usuari@maquina:$ sudo pear install soap
```

Es recorda que *ha de ser root* o tindre permisos de sudo per a poder instal·lar programari en GNU-Linux.

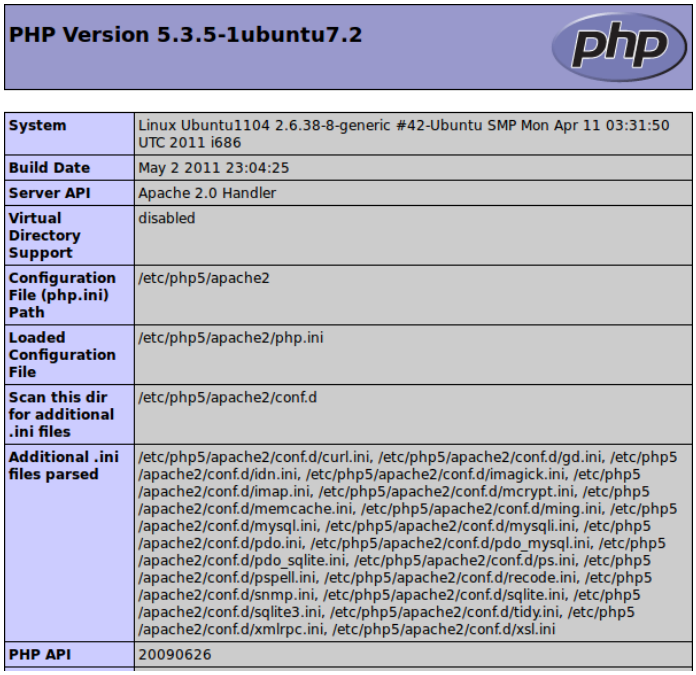
28.1.1 Comprovació de que PHP funciona

Per comprovar que PHP funciona correctament crearem un fitxer anomenat *prova.php* en el directori `/var/www` amb el següent contingut:

Listing 28.2: Fitxer per verificar que PHP funciona

```
<?php
  print phpinfo();
?>
```

Si tot ha estat correctament instal·lat, després d'accedir-ne amb el navegador a l'adreça
`http://localhost/prova.php` hauria d'eixir-nos una informació similar a la següent:



PHP Version 5.3.5-1ubuntu7.2

System	Linux Ubuntu1104 2.6.38-8-generic #42-Ubuntu SMP Mon Apr 11 03:31:50 UTC 2011 i686
Build Date	May 2 2011 23:04:25
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/curl.ini, /etc/php5/apache2/conf.d/gd.ini, /etc/php5/apache2/conf.d/idn.ini, /etc/php5/apache2/conf.d/imagick.ini, /etc/php5/apache2/conf.d/imap.ini, /etc/php5/apache2/conf.d/mcrypt.ini, /etc/php5/apache2/conf.d/memcache.ini, /etc/php5/apache2/conf.d/ming.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/mysqli.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini, /etc/php5/apache2/conf.d/pdo_sqlite.ini, /etc/php5/apache2/conf.d/ps.ini, /etc/php5/apache2/conf.d/pspell.ini, /etc/php5/apache2/conf.d/recode.ini, /etc/php5/apache2/conf.d/snmp.ini, /etc/php5/apache2/conf.d/sqlite.ini, /etc/php5/apache2/conf.d/sqlite3.ini, /etc/php5/apache2/conf.d/tidy.ini, /etc/php5/apache2/conf.d/xmlrpc.ini, /etc/php5/apache2/conf.d/xsl.ini
PHP API	20090626

Figura 28.1: Prova de que tot ha estat correctament instal.lat

28.2 Instal·lació del marc de treball

Per instal·lar el marc de treball seguirem els següents passos:

1. Descomprimir el marc de treball en un directori.
2. Copiar el directori descomprimit al directori `/var/www`
3. Accedir a través de la línia d'ordres a eixe directori i donar-li permisos d'escriptura a l'usuari *www-data* en el directori
   `/framework/cache`

28.3 Reescriptura de URLs utilitzant Apache *mod_rewrite*

Mod_Rewrite és un mòdul del servidor web Apache que permet fer reescriptures de les URLs de les peticions web al vol.
Es basa en l'existència d'un fitxer (anomenat *.htaccess*) que descriu una sèrie de regles basades en expressions regulars.
Per activar aquest mòdul cal amb introduir en la línia d'ordres la següent ordre: « sudo a2enmod rewrite » , el sistema ens demanarà la contrasenya i ens activarà aquest mòdul.

Listing 28.3: Regles d'Apache mod_rewrite

```
<IfModule mod_rewrite.c>
  RewriteEngine On RewriteBase /
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteRule ^(.*)$ index.php/$1        [L,QSA]
  RewriteRule ^app/(.*)$  index.php      [L]
  RewriteRule ^framework/(.*)$ index.php [L]
  RewriteRule ^rest/(.*)$ rest.php/$1    [L,QSA]
  RewriteRule ^soap/(.*)$ soap.php/$1    [L,QSA]
  RewriteRule ^cron/(.*)$ cron.php/$1    [L,QSA]
</IfModule>

# RewriteCond %{REQUEST_FILENAME} !-f
# RewriteCond %{REQUEST_FILENAME} !-d
# RewriteRule ^(.+)$ /index.php/$1 [L,QSA]
```

Capítol 29

Configuració del projecte

La configuració del projecte resideix en el fitxer de configuració *core.php* situat en el directori `/framework/config`. Té un aspecte semblant al següent codi font:

Listing 29.1: Configuració bàsica del projecte

```
<?php
$config = array(
    "sections" => array (
        "locale" => array (
            "default" => "ca_ES",
            "hack"    => ".UTF-8"
        )
    ),
    "global"    => array (
        "baseURL" => "http://localhost/blog",
        "baseURI" => "/blog",
        "basePath"=> "/var/www/blog",
        "title"   => "Blog"
    )
);

FW_Config::createConfig("core");
FW_Config::setConfig("core",$config);
?>
```

En ell, s'han de configurar el següents paràmetres com a mínim:

29.1 Locales

Els paràmetres mínims a configurar per a les locales (idiomes) són:

sections.locale.default : Locale (idioma) per defecte (en aquest cas «ca_ES» (català d'Espanya).

sections.locale.hack : 'Hack' per sistemes ubuntu.

29.2 Directoris

Els paràmetres mínims a configurar els directoris són:

global.baseURL : Adreça web del framework (`http://localhost/blog`

global.baseURI : URI web del framework `/blog`

global.basePath : Directori on està instal·lat el framework.

global.title : Títol per defecte en totes les pàgines.

Capítol 30

Base de dades del projecte

Amb el següent codi SQL construirem la base de dades utilitzada en aquest projecte:

Listing 30.1: Taules de la base de dades utilitzada en aquest projecte

```
1  -- phpMyAdmin SQL Dump
2  -- version 3.3.10deb1
3  -- http://www.phpmyadmin.net
4  --
5  -- Servidor: localhost
6  -- Versio del servidor: 5.1.54
7  -- Versio de PHP : 5.3.5-1ubuntu7.2
8
9  SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
10
11
12 /*!40101 SET @OLD_CHARACTER_SET_CLIENT=
       @@CHARACTER_SET_CLIENT */;
13 /*!40101 SET @OLD_CHARACTER_SET_RESULTS=
       @@CHARACTER_SET_RESULTS */;
14 /*!40101 SET @OLD_COLLATION_CONNECTION=
       @@COLLATION_CONNECTION */;
15 /*!40101 SET NAMES utf8 */;
16
17 --
18 -- Base de dades: `blog`
19 --
20 CREATE DATABASE blog;
21
22 -- --------------------------------------------------------
23
24 --
25 -- Estructura de la taula `book`
26 --
27
28 CREATE TABLE `blog_book` (
29   `isbn` varchar(100) NOT NULL,
30   `title` varchar(255) NOT NULL,
31   `author` varchar(255) DEFAULT NULL,
```

```
32    PRIMARY KEY (`isbn`),
33    KEY `author` (`author`)
34  );
35
36  -- --------------------------------------------------------
37
38  --
39  -- Estructura de la taula `cache_object`
40  --
41
42  CREATE TABLE `blog_cache_object` (
43    `id` varchar(256) NOT NULL,
44    `namespace` varchar(256) NOT NULL,
45    `value` text NOT NULL,
46    `lifetime` int(11) NOT NULL,
47    `cached_at` datetime NOT NULL,
48    PRIMARY KEY (`id`)
49  );
50
51  -- --------------------------------------------------------
52
53  --
54  -- Estructura de la taula `comment`
55  --
56
57  CREATE TABLE `blog_comment` (
58    `id` int(11) NOT NULL AUTO_INCREMENT,
59    `id_post` int(11) NOT NULL,
60    `created_at` datetime NOT NULL,
61    `content` text NOT NULL,
62    `author` varchar(50) NOT NULL,
63    PRIMARY KEY (`id`)
64  );
65
66  -- --------------------------------------------------------
67
68  --
69  -- Estructura de la taula `plugin_option`
70  --
71
72  CREATE TABLE `blog_plugin_option` (
73    `plugin` varchar(50) NOT NULL,
74    `name` varchar(50) NOT NULL,
75    `value` varchar(512) NOT NULL,
76    PRIMARY KEY (`plugin`,`name`,`value`)
77  );
78
79  -- --------------------------------------------------------
80
81  --
82  -- Estructura de la taula `post`
83  --
84
85  CREATE TABLE `blog_post` (
```

```
86    `id` int(11) NOT NULL AUTO_INCREMENT,
87    `title` varchar(512) NOT NULL,
88    `excerpt` text NOT NULL,
89    `content` text NOT NULL,
90    `created_at` datetime NOT NULL,
91    `author` varchar(50) NOT NULL,
92    `is_commentable` int(11) NOT NULL,
93    `permalink` varchar(255) NOT NULL,
94    `status` int(11) NOT NULL,
95    PRIMARY KEY (`id`),
96    UNIQUE KEY `permalink` (`permalink`)
97  );
98
99  -- --------------------------------------------------------
100
101 --
102 -- Estructura de la taula `role`
103 --
104
105 CREATE TABLE `blog_role` (
106   `role` varchar(20) NOT NULL,
107   `enabled` int(1) NOT NULL,
108   PRIMARY KEY (`role`,`enabled`)
109 );
110
111 -- --------------------------------------------------------
112
113 --
114 -- Estructura de la taula `user`
115 --
116
117 CREATE TABLE `blog_user` (
118   `username` varchar(50) NOT NULL,
119   `password` varchar(50) NOT NULL,
120   `email` varchar(50) NOT NULL,
121   `role` varchar(50) NOT NULL,
122   `name` varchar(50) NOT NULL,
123   `activation_key` varchar(50) NOT NULL,
124   `date_register` datetime NOT NULL,
125   `language` varchar(10) NOT NULL,
126   `theme` varchar(50) NOT NULL,
127   `display_name` varchar(128) NOT NULL,
128   `status` int(1) NOT NULL,
129   `url` varchar(128) NOT NULL,
130   `bio` text NOT NULL,
131   `image` varchar(256) NOT NULL,
132   `notifications` varchar(4) NOT NULL,
133   PRIMARY KEY (`username`)
134 );
135
136 -- --------------------------------------------------------
137
138 --
139 -- Estructura de la taula `user_has_roles`
```

```
140 --
141
142 CREATE TABLE `blog_user_has_roles` (
143   `username` varchar(50) NOT NULL,
144   `role` varchar(20) NOT NULL,
145   PRIMARY KEY (`username`,`role`)
146 );
```

Capítol 31

Taules de figures i llistat de codi font

Índex de figures

Listings