UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

R.O.W.D.S - Diseño de un Framework para el diseño y desarrollo rápido de aplicaciones web en PHP, Javascript y MySQL para entornos emprendedores

Proyecto Final de Carrera

Ingeniería Informática.

Autor: Juan Vilar Sanchis

Director: Moisés Pastor i Gadea

18 Septiembre 2013

Resumen

De la necesidad de poder crear una aplicación de gran calidad de software sin tener que requerir el uso de frameworks complejos de terceros que pueden ralentizar el desarrollo de la aplicación así como retrasar los plazos en cada una de las iteraciones del desarrollo del software nace R.O.W.D.S (Rapid Online Web Deployment System). La idea es ofrecer al programador o al desarrollador de la empresa un framework ligero, rápido y sencillo de utilizar, con una curva de aprendizaje muy corta y que pueda producir resultados de alta calidad, dejando total libertad al desarrollador de programar como el desee, pero ofreciéndole una pequeña API y una metodología de trabajo integrada en el propio framework que sea capaz de ordenar y agilizar el desarrollo de su aplicación.

ROWDS trata de aplicar de forma sencilla y práctica metodologías básicas de programación en un framework que puedan utilizar los desarrolladores y que ofrezca soluciones y funcionalidades modulares que reduzcan al máximo el acoplamiento entre componentes y aumenten la cohesión interna de cada uno de ellos. El framework, basado en PHP 5.3 (muy común entre la mayoría de servidores web que cuenten con apache o IIS o similar) trata de ser ligero y preciso a la hora de programar , y trata de ofrecer al programador la facilidad de conceptos como el modelo vista controlador o el uso de middlewares y handlers para la correcta comunicación entre la capa de presentación de la aplicación y la abstracción de la base de datos

Agradecimientos

Quisiera aprovechar este pequeño espacio para agradecer a las personas que ,a lo largo de mi carrera académica me han apoyado y han hecho posible de un modo u otro este proyecto.

- A mis padres, por haberme brindado la posibilidad de elegir la vida que he querido y los estudios que he cursado.

- A Moisés Pastor, por ser un magnífico profesor y tutor.

- A Alejandro Pérez Segura y Carmen Mora Hueso, por la ayuda y la amistad que me han brindado siempre a lo largo de mis estudios universitarios.

- A las becas Talentum Startup de telefónica y en especial a los chicos de Bandness.com, por ofrecerme un entorno de trabajo único y porque sin los conocimientos que me obtuve de ellos nunca hubiese podido llevar a cabo este proyecto

Tabla de contenidos

R.O.W.D.S - Diseño de un Framework para el diseño y desarrollo rápido de aplicaciones web en PHP, Javascript y MySQL para entornos emprendedores.

1. Problema general

A menudo en entornos de emprendedores o cuando se planea diseñar una nueva aplicación o formar una nueva empresa o startup, dado que muchos de los negocios y de las startups que se crean hoy en día basan su negocio en un modelo web, muchos de los emprendedores se enfrentan al problema de cómo crear y diseñar la aplicación web. A menudo al formar una startup los socios que hay en ella son pocos, no todos tienen la experiencia o la cualificación técnica que se requiere para llevar a cabo y también (sobretodo aquí en España) carecen de los recursos para contratar a programadores y desarrollar la parte técnica de su empresa. A menudo se encuentran con equipos de desarrollo muy reducidos de dos o tres personas que deben coordinarse y trabajar juntos para desarrollar el sistema web y la aplicación que implementará su idea y su modelo de negocio. La escasez de recursos humanos técnicos, así como la casi inexistencia de financiación a corto plazo que ayude a financiar a la gran mayoría de estas startups hacen que les sea muy complicado a estas empresas emprendedoras sobrevivir.

Estas dificultades a las que se someten las empresas emprendedoras en España hacen que la toma de decisiones sea un factor clave y vital para la subsistencia de cualquiera de ellas. Con escasos recursos y a menudo escaso tiempo para construir la aplicación y poder desarrollar su idea, la elección de la tecnología, los sistemas a usar así como los lenguajes de programación que van a utilizarse se convierte en uno de los primeros pasos más decisivos de la futura empresa.

¿Pero cómo tomar decisiones tan importantes en un entorno tan complejo? Principalmente la toma de decisión de las tecnologías a emplear depende de los conocimientos de la parte técnica de la startup. La elección más acertada siempre será aquel lenguaje de programación con el que la parte técnica en general se sienta más cómoda, cuyo aprendizaje para otros miembros que lo desconocen sea rápido y sencillo, y cuyos ciclos de desarrollo sean más cortos y más rápido. Al tratarse de aplicaciones web hoy en día la mayor parte de estos desarrolladores eligen trabajar en lenguajes de muy alto nivel como Ruby on Rails, Java Server Pages, Active Server Pages, .NET y sobretodo, PHP.

PHP (Hypertext Preprocessor) es un lenguaje multiparadigma, imperativo, orientado a objetos, procedural y reflexivo que está muy extendido a lo largo de internet hoy en día. El gran parecido que posee PHP con los lenguajes más comunes de programación estructurada, como C y Perl, permiten a la mayoría de los programadores crear aplicaciones complejas con una curva de aprendizaje muy corta. También les permite involucrarse con aplicaciones de contenido dinámico sin tener que aprender todo un nuevo grupo de funciones y todo su diseño está orientado a facilitar la creación de sitios webs cada vez

más complejos y funcionales. Sumado al hecho de que es ampliamente utilizado en todo internet y que posee una comunidad y unas fuentes de aprendizaje y de código muy robustas y numerosas, lo convierten en el lenguaje de programación ideal para que cualquier startup comience a desarrollar su aplicación web.

Si además hablamos de entornos de aceleración de startups, dónde se deben fijar una serie de objetivos y fechas límite para que la aplicación (o al menos un prototipo de ella) esté lista, el uso de PHP por uso relativamente sencillo se convierte en la elección más usada por muchos de los desarrolladores.

Por supuesto, no solo se tienen que decidir los lenguajes de programación a emplear, sino también los ciclos de trabajo, las iteraciones y si se va a utilizar algún framework para el desarrollo de la aplicación (otra de las elecciones críticas en el desarrollo de cualquier sistema).

Pero de nuevo se debe tener en cuenta la escasez de recursos y de tiempo para tomar estas decisiones. A menudo el cambio que ha habido en los desarrollos basados en sesiones de cliente / servidor tradicional hacia los desarrollos libres, abiertos, carentes de una sesión y colaborativos como la web 2.0 han incrementado la necesidad de iteraciones de desarrollo más rápidas y cortas en las fases de cualquier proceso de desarrollo de software. Si a esto le sumamos el cada vez más incipiente presencia de frameworks y tecnologías de código libre en los núcleos de desarrollo de muchos sistemas comerciales, muchos de los desarrolladores directamente buscan una metodología de desarrollo rápido de aplicaciones.

RAD (Rapid application development) es una metodología de desarrollo de software muy presente en entornos emprendedores, que favorece el uso de un prototipado rápido en detrimento de la planificación de la aplicación. En esta metodología, la planificación del software a desarrollar se solapa con la generación del código para la aplicación. La falta de una planificación extensiva generalmente favorece que el software sea escrito de una forma mucho más rápida y hace que sea más fácil cambiar sus requisitos y funcionalidades. Por supuesto pese a que la mayoría de metodologías RAD fomentan la reutilización de código así como una estructura basada en equipos de trabajo pequeños (lo cual es el caldo de cultivo de cualquier startup), es bastante evidente señalar que en definitiva, ninguna metodología rápida de trabajo puede proporcionar una mejora sustancial sobre cualquier otra metodología de desarrollo, y pese a que todas ellas tienen el potencial para proporcionar un buen framework para el desarrollo más rápido de productos, rara vez se verá a grandes compañías como Microsoft o Google utilizar metodologías RAD para el desarrollo de sus productos estrella.

Cualquier ciclo RAD se compone de cuatro fases principales:
1) Fase de planificación de requisitos: En esta fase se debe decidir cuáles son las funcionalidades a realizar, cuales son necesarias y cuáles no. Hay que tomar las decisiones de saber cuáles son los requisitos del sistema y cuáles son las más críticas e importantes y en cuales hay que centrar la mayor parte del desarrollo.

2) Fase de diseño de usuario: En esta fase es cuando se desarrolla el producto. Por lo general trabajan de forma conjunta todos los miembros del equipo de desarrollo. Por una parte la aplicación debe ser funcional y cumplir con los requisitos que se proponen. También es importante la usabilidad de la misma, el diseño final y como lucirá. Hoy en día en el entorno tan innovativo moderno y competitivo que hay en internet para estas startups saber como luce la aplicación final es también muy importante, por lo que (sobre todo en cuanto a diseño web se refiere) de normal es importante contar con la aportación de artistas y diseñadores gráficos que ayuden a hacer que la aplicación luzca bien y sea agradable de usar para los usuarios finales.

3) Fase de construcción: La fase que más tiempo acapara de todo el proceso, se centra en el programa y en el desarrollo de la aplicación. Los programadores programan los módulos, comprueban la cohesión y el acoplamiento entre los componentes y comprueban la funcionalidad de los mismos, realizando diversos testing al finalizar cada apartado funcional. También en diseño web se deben maquetar los diseños en HTML5 y CSS para que queden acordes al diseño original y esto se debe integrar con la parte programativa de la propia aplicación.

4) Fase final de diseño y usabilidad: En esta fase se realizan los ajustes finales, se ensamblan todos los módulos que hayan sido preparados por separado, se prueba la aplicación y se realiza un testing más riguroso para comprobar que la aplicación es estable y robusta. Comparado con los métodos de programación más tradicionales, el proceso entero se comprime y como resultado el nuevo sistema se construye y se pone en ejecución en un tiempo mucho menor.

Ciclo de vida de una metodología RAD (Rapid Application Development)

Pese a las ventajas de las metodologías RAD en entornos emprendedores y startups, hay que tener en cuenta alguna de las desventajas. Esta metodología tiene una fuerte dependencia en equipos de trabajo fuertemente cohesionados y cuyo compromiso individual con el proyecto sea muy grande. Además debido

a esto el producto puede perder su ventaja competitiva debido a la falta de funcionalidades en el núcleo de la aplicación, y si los plazos en el ciclo de desarrollo son muy cortos (que a menudo lo son) la aplicación puede acabar teniendo una apariencia pobre y de baja calidad. A menudo tener poca funcionalidad en iteraciones a corto plazo implica enormes retrasos en la programación de funcionalidades completas en las iteraciones finales, por lo que encontrar un balance de cuánta funcionalidad se debe implementar en cada una de las iteraciones es crítico en los desarrollos de software de estas empresas de nueva creación.

En cuanto a la elección de un framework, debido al poco tiempo de que disponen, muchas startups se decantan por una programación tradicional sin ninguna metodología de trabajo, programando “a pelo” sin orden y con mucho caos entre las diferentes piezas de código que generan cada uno de los miembros del equipo. Pese a que esto favorece la creación rápida de piezas de software y la agilidad de crear el producto poco a poco con pequeños incrementos en sus funcionalidades, a menudo el software acaba siendo poco o nada reutilizable, de bastante baja calidad y dado que producen poca documentación escrita en el momento, acaban requiriendo de una cantidad considerable de documentación post proyecto (para saber cómo está programada cada una de las partes del software) y por lo general deben reescribir todo el código (o gran parte del mismo) una vez se acaba la iteración y deben comenzar una nueva versión del programa.

Las empresas que si se atreven a desarrollar utilizando un framework (en este caso podemos citar como ejemplo a Zend, Joomla o Symfony2[1]) pueden acabar demorando la programación de sus funcionalidades y teniendo retrasos imprevistos (cabe recordar que la fase de análisis de requisitos así como la planificación de costes del proyecto muy a menudo es mínima o casi nula). Muy a menudo estas empresas acaban dándose cuenta que carecen de una plantilla suficiente con conocimientos técnicos para llevar a cabo sus proyectos con celeridad. Además, debido al desconocimiento de todos los aspectos técnicos del framework empleado (dado que cada vez son más y más complejos) y a la falta de formación de muchos de sus miembros en el mismo (así como la falta de recursos para contratar a gente nueva que si posea experiencia en estos ambientes) parte del código que producen deberá ser reprogramado o cambiado por completo en un futuro, con lo cual no se aprovecha toda la potencia de estos frameworks.

Cabe recalcar que en estos entornos empresariales con personal muy limitado y con pocos recursos muchos de los emprendedores deben tomar decisiones críticas en poco tiempo para que el desarrollo sea lo más rápido posible y poder acercarse a una ronda de inversores con algo más que una idea que poder enseñarles para agradarles, coger su confianza y quizás conseguir algo de seed capital (capital semilla de inversión que se paga generalmente vendiendo un gran porcentaje de la empresa emprendedora) con lo que empezar la siguiente iteración. Es importante destacar lo crítico de los plazos porque muchas de las personas que comienzan a trabajar en estas startups lo hacen por amor a su idea y sin cobrar ningún tipo de sueldo. Y como solo se puede trabajar sin cobrar dinero durante un periodo determinado de tiempo (en

función de la manutención, sustento o ahorro de cada uno de los integrantes del equipo), cuanto más tiempo pasa entre cada iteración y desarrollo de cada una de las funcionalidades críticas, mayor es la probabilidad de que cualquiera de sus miembros integrantes se vea ahogado en una situación económica personal y se vea obligado a abandonar el proyecto (lo cual puede ser mortal para la startup).

Otro problema que puede surgir de no usar un framework y de realizar una programación más “artesanal” consiste en la dificultad de luego integrar todos los archivos correctamente para el uso y corrección del resto de miembros del equipo. Por lo general y sobretodo tratandose de PHP, se mezcla mucho el desarrollo de la aplicación a nivel programativo con el desarrollo del diseño de la aplicación. Lo problemático del desarrollo de una aplicación web moderna por lo general es que se emplean muchos tipos distintos de tecnología (MYSQL, PHP, CSS, JS, HTML, XML..) en un mismo proyecto, por lo que llevar un orden adecuado y no sumir la aplicación en el caos puede facilitar mucho la correcta distribución de trabajo y agilizar la programación de código.

2. Solución propuesta

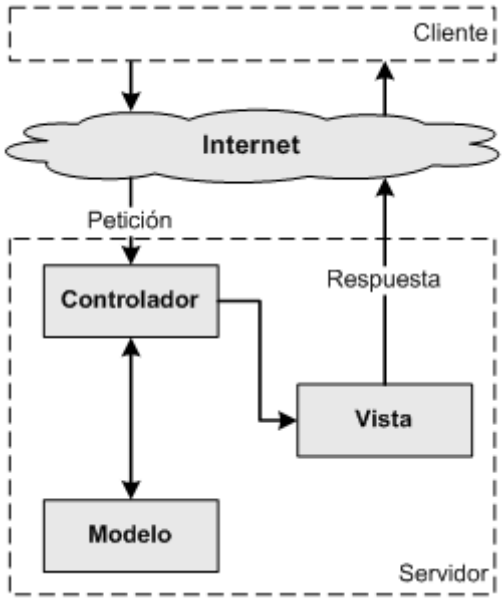
De la necesidad de poder crear una aplicación de gran calidad de software sin tener que requerir el uso de frameworks complejos de terceros que pueden ralentizar el desarrollo de la aplicación así como retrasar los plazos en cada una de las iteraciones del desarrollo del software nace R.O.W.D.S (Rapid Online Web Deployment System). La idea es ofrecer al programador o al desarrollador de la empresa un framework ligero, rápido y sencillo de utilizar, con una curva de aprendizaje muy corta y que pueda producir resultados de alta calidad, dejando total libertad al desarrollador de programar como el desee, pero ofreciéndole una pequeña API y una metodología de trabajo integrada en el propio framework que sea capaz de ordenar y agilizar el desarrollo de su aplicación.

ROWDS trata de aplicar de forma sencilla y práctica metodologías básicas de programación en un framework que puedan utilizar los desarrolladores y que ofrezca soluciones y funcionalidades modulares que reduzcan al máximo el acoplamiento entre componentes y aumenten la cohesión interna de cada uno de ellos. El framework, basado en PHP 5.3 (muy común entre la mayoría de servidores web que cuenten con apache o IIS o similar) trata de ser ligero y preciso a la hora de programar , y trata de ofrecer al programador la facilidad de conceptos como el modelo vista controlador o el uso de middlewares y handlers para la correcta comunicación entre la capa de presentación de la aplicación y la abstracción de la base de datos

2.a Modelo Vista Controlador (MVC)

El modelo vista controlador (MVC) es un patrón muy común y muy util en la arquitectura del softare (especialmente en la arquitectura de aplicaciones web) que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario. El patrón se basa en las ideas fundamentales de reutilización de

código y separación de conceptos, lo cual facilita las tareas del desarrollo y posterior mantenimiento y cambio de las mismas aplicaciones en un futuro. El patrón MVC fue una de las primeras ideas en el campo de interfaces gráficas de usuario y uno de los primeros trabajos en describir e implementar aplicaciones software en términos de sus diferentes funciones. La idea de organizar correctamente la vista y el controlador y separarlos facilita enormemente la programación a realizar. En una aplicación web como las de hoy en día dónde la capa de presentación cobra cada vez más y más importancia es necesario tener clara dicha separación para que el mantenimiento de la propia aplicación en un futuro sea más sencillo y esto a su vez haga que la aplicación sea más robusta.



Esquema de un MVC (Modelo Vista Controlador)

De manera genérica, los componentes del modelo vista controlador se podrían definir de la siguiente manera:

El modelo: Es tanto la representación de la información con la cual opera el sistema como la forma de acceder y gestionar los datos. Gestiona todos los accesos a dicha información y es de forma abstracta la que da significado a lo que el usuario está haciendo. Guarda las relaciones entre los datos y es la encargada de implementar también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación o en la lógica de negocio y por lo general incluye un driver que responde a las órdenes que le da el controlador.

El controlador: Responde a los eventos e invoca peticiones al modelo. Por lo general se encarga de actuar como un “driver” y de gestionar las peticiones que realiza el usuario para transcribirlas al modelo. Desde la inserción de un formulario hasta el registro de un usuario, el controlador se asegura de pasar toda la información que ha usado el usuario en la vista y procesarla correctamente al modelo o sistema de la base de datos. En la parte del controlador es dónde se manipulan todos los datos, se realizan los cálculos

complejos y se consigue todo lo que luego se usará para la vista. Actúa como un middleware entre la lógica de negocio y la capa de presentación.

La vista: La capa final, la capa de presentación y lo que el usuario ve. Presenta el "modelo" en un formato adecuado para interactuar. Comprende toda la interfaz gráfica de usuario y todo lo que el usuario puede ver o tocar.

Muchos sistemas informáticos (y en general todos los sistemas en entornos de emprendedores) requieren de utilizar un Sistema de Gestión de Base de Datos, siendo el más popular en entornos emprendedores MYSQL. Dicha gestión en el MVC corresponde a la parte del modelo. La unión entre la capa de presentación y la capa de negocio representa la integración entre la vista y su correspondiente controlador. El mayor objetivo del MVC es separar la capa visual gráfica de su correspondiente programación y acceso a datos, algo que mejora notablemente el desarrollo y mantenimiento de la aplicación, ya que tanto la vista como el controlador en paralelo cumplen ciclos de vida y funcionalidades muy diferentes entre sí y ayuda a aplicar un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de la aplicación.

MVC ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Los primeros frameworks MVC para desarrollo web planteaban un enfoque de cliente ligero en el que la mayor parte de la funcionalidad tanto de la vista como del controlador recaían principalmente en el servidor. No obstante, con la maduración de las tecnologías web y de JavaScript cada vez las aplicaciones web ejecutan más parte del código en el navegador, haciendo que ciertos componentes MVC se ejecuten parcial o totalmente en el cliente. Muchos frameworks que han triunfado en este aspecto pueden ser JavaScriptMVC, mooTools o jQuery.

2.b Middleware

Un middleware es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware o sistemas operativos. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones que son necesarias en sistemas distribuidos.

En el caso de ***ROWDS***, el middleware se trata de una capa que gestiona las comunicaciones entre el controlador y la base de datos. Para implementar esto se diseñó el sistema con la posibilidad de utilizar un handler para la base de datos que interactuase con la misma mediante el driver PDO de MYSQL. PDO es una extensión ligera y de acceso consistente para acceder a bases de datos en PHP. Proporciona una interfaz para cada uno de los sistemas de gestión de bases de datos a los que se tenga que conectar (en el ejemplo práctico de ROWDS se conectará con MYSQL.

La aplicación (como se detallará más adelante) se ha diseñado con el propósito de poder ampliar el middleware a otros tipos de sistemas de gestión de base de datos como mongodb o oracle, o incluso sistemas de gestión no-sql como mongoDB o prolog.

El concepto de middleware es importante porque por lo general suele ser un aspecto crítico de cualquier aplicación web y tener bien localizada la sección desde la que se accede al sistema de gestión de base de datos agiliza muchísimo la programación y permite que luego el código sea reutilizable y de fácil mantenimiento.

2.c Componentización

ROWDS se caracteriza por ser un sistema modulable por componentes. Cada componente es totalmente independiente del resto y tiene sus propios archivos, imágenes, clases, eventos, controladores y handlers. De tal manera que cada módulo funciona por separado y no dependen entre sí. Para lograr esto a nivel de una aplicación web, ROWDS cuenta con un enrutador que a partir de la ruta redirige al componente pertinente y a la acción que debe realizar el módulo. Además también permite el uso de templates para que cada vista pueda tener asociado un diseño diferente, o que incluso una llamada a través de la página web pueda no lanzar código HTML, sino objetos JSON, XML, SQL, SOAP o cualquier formato que se desee utilizar. Esto hace que el sistema sea mucho más plurivalente, pues un mismo módulo en función de sus diferentes acciones puede o bien lanzar una respuesta a una petición HTML, o devolver un objeto parseado en JSON (JavaScript Object Notation) para su posterior utilización en la capa de usuario de la aplicación.

2.d Gestión de Usuarios

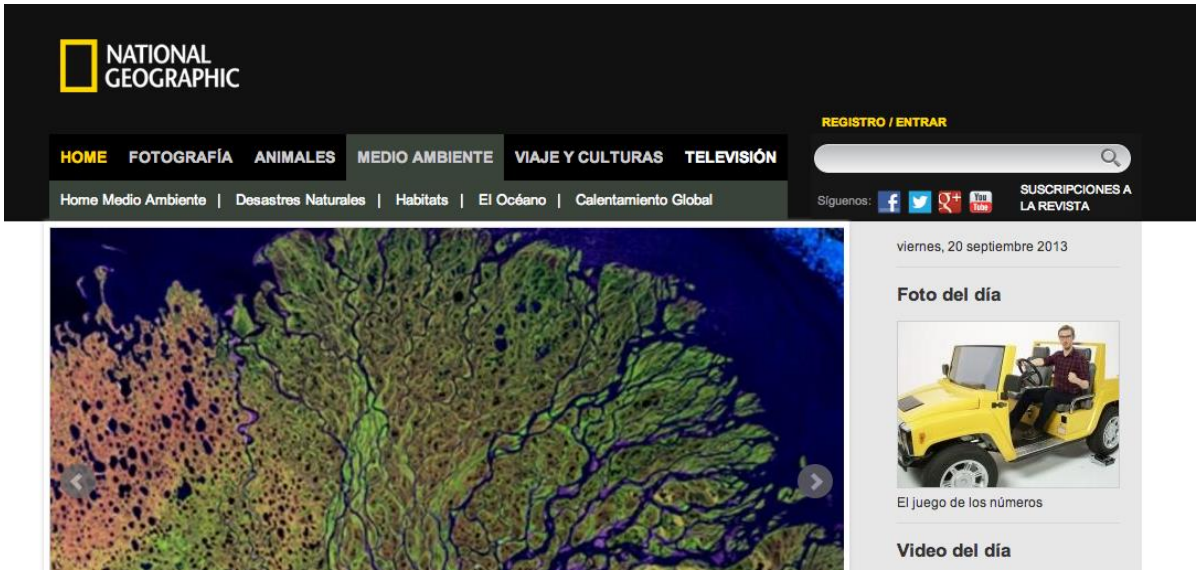
La gestión de usuarios puede ser un tema muy complejo y variable para cada una de las páginas web. Por lo general siempre es deseable que el usuario se registre, pero con la web siendo cada vez más social, es posible que la web requiera disponer de conexiones a través de terceros, como OpenID, Google u otras redes sociales como Facebook o Twitter. La capacidad de registrarse con herramientas de identificación de redes sociales es un elemento muy importante para que la aplicación web registre visitas y gane tracción, logrando

así sustentar un posible modelo de negocio basado en la captación de usuarios. No obstante, cada aplicación puede disponer de unas necesidades totalmente diferentes. Lo que ROWDS propone como modelo para gestionar las bases de datos es la conjunción de un sistema de sesión acompañado de un sistema middleware que interactúe con la base de datos. De esta manera podemos obtener información bien sea porque el usuario decide registrarse en el sistema, o bien porque decide hacerlo a través de una herramienta de terceros que haya sido incrustada en la página web. En cualquiera de los casos, el sistema permite total libertad al programador para que , dentro de un marco metodológico sea capaz de diseñar de forma sencilla el sistema de registro de usuarios que más y mejor le convenga.

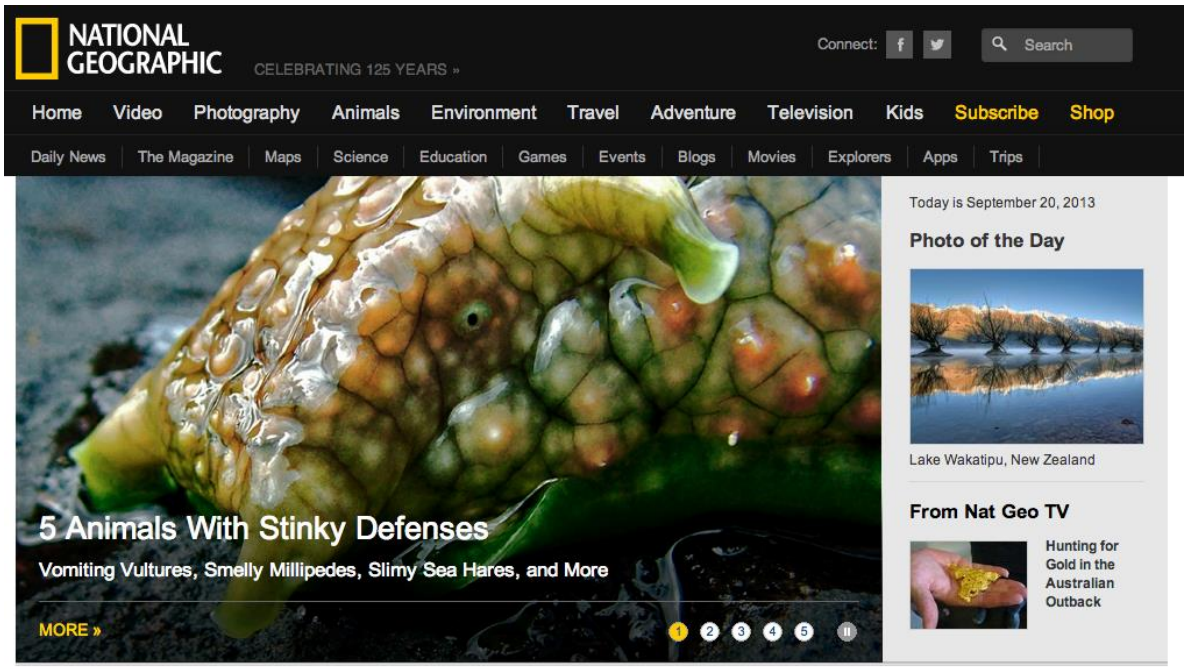
2.e Sistema multilingüe de Idiomas.

La mayor parte de las páginas web acaban teniendo la necesidad de poder ser multilenguaje por la necesidad de conseguir una mayor audiencia. El tema de la traducción es bastante complejo ya que, pese a que por lo general solamente es necesario cambiar la cadena de texto con el contenido, puede ser necesario que además de ello sea necesario cambiar la vista entera ya que en un idioma (o para un país específico) puede ser necesario cambiar tanto los contenidos como la forma en la que se muestran los mismos.

Pongamos por ejemplo la web de National Geographic en su versión Española.



Y ahora la compararemos con su versión homónima en Inglés.



Podemos darnos cuenta que tanto los contenidos como parte de su distribución son diferentes, pese a que el sistema web es el mismo (la web ha sido diseñada utilizando synfony2). Dado que los conceptos de usabilidad de una página web pueden variar mucho entre país y país (por ejemplo en china se lee de abajo a arriba, y en Árabe de derecha a izquierda), es necesario tener clara esta distinción y el sistema debe ser capaz de controlar esta diferencia y poder aplicarla en cada caso.

ROWDS propone un mecanismo sencillo de traducción mezclado con la gestión de vistas para poder lograr este efecto sin el mayor de los problemas. Existe una clase encargada de gestionar las traducciones, y a partir de ella y de cómo se configure la página web mediante el control de vista y el enrutador, pueden existir diferentes vistas en todo momento que se encarguen de gestionar el idioma de la página web y la forma en la que distribuye el contenido para cada uno de los idiomas.

2.f Control de errores.

El control de errores forma una parte importante de cualquier aplicación web. Es importante para el mantenimiento del código saber en qué ha fallado la web, cómo dónde y por qué. A nivel de control de errores ROWDS posee un módulo de control de errores que se encarga de gestionar los errores e imprimirlos en pantalla, cortando después la ejecución del script y evitando que los errores se propaguen por toda la vista y lleguen al usuario en forma de cascada. ROWDS controla todos los errores que el desarrollador desea registrar con la intención de ayudar a depurar su código y de hacer la aplicación web más robusta a fallos y a intentos de acceso no autorizados. De esta manera los sistemas son mucho más fáciles de mantener y la reusabilidad del código aumenta.

2.g Metodología estándar propuesta.

ROWDS propone sobre todo una metodología de tipo RAD para desarrollar el software basado en cuatro fases:

1) Fase de diseño del modelo y análisis de requisitos: Se diseña el modelo a emplear y los requisitos funcionales que deben ser cubiertos.

2) Fase del diseño de la aplicación: Se diseña la aplicación a nivel gráfico y de usabilidad. Se debe dar prioridad a la usabilidad ya que hoy en día con las tecnologías web tan ampliamente extendidas y con la feroz competitividad que hay en el mercado, una usabilidad errónea o incómoda para el usuario puede desembocar en que el público no utilice la herramienta y esto es crítico para la supervivencia de una startup

3) Fase del desarrollo de la aplicación: Ligeramente solapada con la fase de diseño anterior. La aplicación se desarrolla. Se construyen los controladores, las vistas y los handlers. Se maqueda la web y se testea cada uno de los componentes.

4) Fase final de comprobación del acabado: Se junta toda la aplicación. se repasan y se ultiman los detalles y se corrigen los bugs y fallos menores que puedan haberse generado durante la programación de los componentes.

Por supuesto el uso de ROWDS no tiene que verse obligado a utilizar estar metodología en cuestión, siendo compatible el uso de cualquier otra para el desarrollo del producto. No obstante, utilizar este tipo de metodologías RAD permite ahorrar y comprimir tiempo en la fase del análisis y el diseño de la aplicación, dando más tiempo a que los desarrolladores se centren en programar código de la mejor calidad posible. Por supuesto sin una fase concienciada en la que los desarrolladores realicen testings y se centren en arreglar bugs, y sin la capacidad productiva que proporciona tener un equipo grande de personas dedicadas a la aplicación, la calidad del software en principio se vería reducida por la escasez de recursos, pero esto se compensa reduciendo el número de funcionalidades a realizar. Cabe destacar que , como ya se ha comentado anteriormente, siempre que se acortan las fases en un proyecto de desarrollo de software, suele llevar acarreado errores en la fase de planificación y diseño del software que se abordarán en el siguiente apartado.

Basándome en mi experiencia personal en entornos emprendedores que desarrollé durante mi colaboración con la aceleradora de empresas "*Business Booster*" mediante la beca TALENTUM Startups de *Telefónica* al igual que en otros proyectos personales emprendedores que he realizado, pude comprobar que uno de los grandes problemas que existen a la hora del desarrollo de aplicaciones web es la poca práctica de ingeniería del software que se utiliza, así como lo poco definidas que acaban quedando las fases de cada proyecto el cual, durante su desarrollo, acaba mezclando todas las fases simultáneamente.

Un problema muy común es que debido a la falta de tiempo y a las prisas acarreadas por tratar de cumplir los plazos que establecen las bolsas de inversores y el negocio, no existe una clara metodología en el desarrollo del software. Generalmente las etapas del ciclo de vida del desarrollo se solapan

las unas con las otras. Se deciden nuevas funcionalidades cuando la aplicación ya está en su fase de diseño, y esta fase se suele solapar también con la fase de desarrollo y construcción cuando al ver el diseño ya implementado no está a las expectativas de lo que en principio se quiso desarrollar. Nuevos feedbacks llegan de clientes , amigos o conocidos que obligan a cambiar algún aspecto del software con los que no se contó al principio de la fase de análisis, y si se trata de un entorno de coworking donde las ideas de varios emprendedores fluyen conjuntamente en un solo lugar, los cambios que pueden llegar a proponerse durante la fase de desarrollo aumentan exponencialmente por el feedback que se recibe de los demás emprendedores.

No será necesario señalar los retrasos que esto puede conllevar al desarrollo de un producto que ya de por sí tiene unas fechas muy ajustadas. Tras varios meses pude comprobar que la teoría de la metodología del desarrollo del software es muy diferente de lo que realmente se lleva a cabo. Los análisis de coste y tiempo nunca se aproximan para nada a la realidad y siempre llevan asociados o bien retraso en las fechas de entrega o bien la falta de funcionalidades que aún no han podido ser programadas. El uso de diagramas UML y BPMN queda totalmente descartado por la falta de tiempo y planificación y por lo general se suelen decidir muchas de las funcionalidades del modelo en base a los conocimientos de los desarrolladores en vez de a la auténtica necesidad del usuario o la aplicación. Por lo general en este tipo de entornos es difícil diferenciar una fase de otra.

Ciclo de vida del desarrollo de software en entornos emprendedores

En mi opinión personal, era una lástima que en un entorno multi empresarial emprendedor tan rico donde la imaginación y el talento fluían por cada rincón de las oficinas y dónde tantos emprendedores tenían buenas ideas y buenos conocimientos hubiese tanto caos y tanto desorden a la hora de desarrollar las aplicaciones web, los modelos de negocio y de establecer ciertos criterios a la hora de aplicar una metodología que realmente se llevase a cabo y fuese efectiva en la práctica. Esto ha motivado la necesidad de crear una metodología de trabajo de tipo RAD que pueda aprovechar tanto la idea inicial del propio emprendedor como el feedback de las personas ajenas que de vez en cuando echarán un cable y aportarán su pequeño granito de arena al desarrollo del sistema. Aportes externos que deben quedar patentes en el ciclo de vida del desarrollo del producto porque muy a menudo la visión de un tercero puede traer mejoras sustanciales en la aplicación o incluso en el propio modelo de negocio. Más aún cuando el propio modelo de negocio establece normas sobre el proyecto a desarrollar, como la captación de usuarios a través de redes sociales que requiere que la página web gire entorno a ello.

Se debe formular una metodología que esté más orientada a la realidad de estos entornos y que se pueda adaptar a las exigencias y necesidades de un entorno con pocos recursos y tiempo limitado. Algo que no solo quede en teoría sobre cómo debería de ser, sino que refleje cómo se trabaja en estos entornos y ofrezca herramientas y ayudas a los desarrolladores de software que la empleen para calcular sus proyectos.

2.h Metodología acelerada propuesta.

La metodología RAD que ROWDS propone para este tipo de entornos, como ya se ha explicado anteriormente es de 4 fases con prototipado. Adoptaremos esta metodología para que pueda ejecutarse en un entorno con pocos recursos y cuyas iteraciones sean cortas:

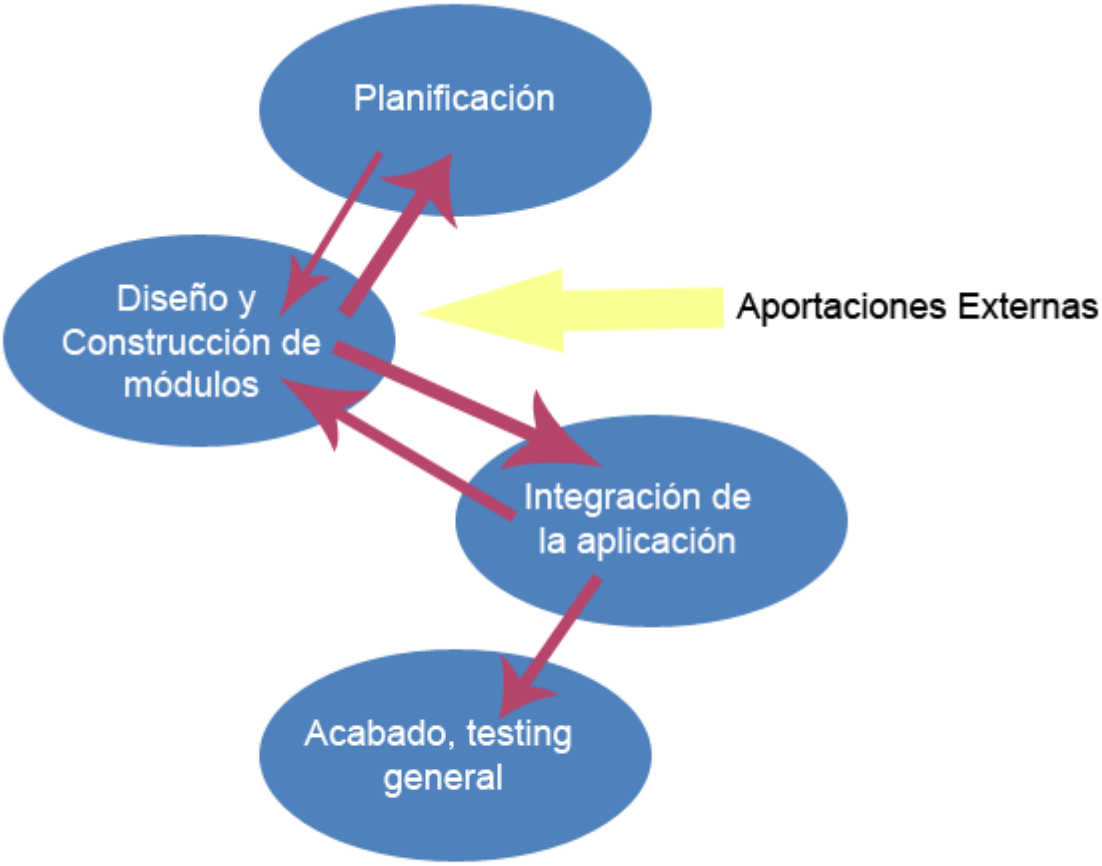
1) Fase de diseño del modelo y análisis de requisitos: En esta fase se debe diseñar el modelo de la aplicación y sus requerimientos. Es una fase corta y breve que no debe durar más de un día de trabajo. Los socios y desarrolladores se sientan en la mesa y discuten el plan de negocio, cómo va a construirse la aplicación, que requisitos debe tener, como se van a organizar los trabajos y la repartición del trabajo. Si en un breve periodo de tiempo como puede ser un día de trabajo no se es capaz de realizar esta tarea de planificación puede significar que la idea del proyecto esté muy difuminada o que no se haya sido realista a la hora de especificar las funcionalidades a implementar de esta iteración

2) Fase del diseño de la aplicación, construcción de controladores y módulos: La aplicación se comienza a diseñar a nivel gráfico. Los artistas o grafistas trabajan diseñando el entorno de usuario, el diseño, los menús etc. Por otro lado los desarrolladores comienzan directamente a crear los controladores y a interactuar con la base de datos. Durante esta fase se deja abierto el diseño del modelo para realizar cambios que puedan surgir mientras se realizan las primeras implementaciones cuando aún hay tiempo de introducir nuevas ideas y nuevos conceptos a los requerimientos iniciales. A partir de

esta fase cualquier innovación o cambio probablemente tendrá que esperar a la siguiente iteración del software.

3) Fase de integración de la aplicación: Ligeramente solapada con la fase de diseño anterior. Se desarrollan las vistas de los módulos y componentes cuya vista ya esté terminada , se van terminando las vistas y se va integrando el sistema, con el modelo, la vista y el controlador.

4) Fase final de comprobación del acabado: Se junta toda la aplicación. se repasan y se ultiman los detalles y se corrigen los bugs y fallos menores que puedan haberse generado en las etapas anteriores.



Ciclo de vida RAD acelerado

2.i Metodología propuesta para el cálculo de costes

Otro de los grandes problemas que existen en este tipo de entornos es un enorme desconocimiento del cálculo de costes que existe. Los emprendedores pueden llegar a ser muy ambiciosos y poner fechas de entrega irreales que no

se corresponden con la realidad y cuyas entregas están muy por encima de las posibilidades y de las capacidades del equipo.

Debido a esta presión por las fechas de entrega muchos equipos de trabajo se ven desbordados por la cantidad de trabajo que tienen delante. Destacando que los inicios de estas empresas son tan duros, y que normalmente no suele haber financiación para pagar el prototipo, la presión por sacar la aplicación a tiempo y dentro de las rondas de inversión suele causar exceso de trabajo y estrés en el equipo de programación, lo cual puede desgastar la moral del equipo de producción y por tanto acarrear la creación de un software de peor calidad.

El uso de softwares de gestión de proyectos como Doolphy o Scrum Manager (www.doolphy.com) sería una buena idea para poder recoger feedback sobre la cantidad de horas de programación que se les ha dedicado a cada uno de los apartados de la web, y poder medir la efectividad y las horas invertidas en cada uno de estos proyectos. No obstante debido a la presión de la que hablábamos antes, muchas de estas herramientas suelen pasar a un segundo plano y no utilizarse.

La solución propuesta es un sencillo modelo matemático para calcular el coste que acarrea programar un módulo o una unidad funcional. Para calcularlo, primero se deben dividir las tareas en unidades funcionales que se han realizado para un periodo de tiempo determinado (una semana, un mes, o un ciclo de entrega), y se deben contemplar únicamente aquellas unidades funcionales que estén completamente terminadas (para penalizar las planificaciones muy ambiciosas).

A cada una de estas tareas se le asigna una complejidad y un valor numérico. Si el programador considera que la tarea era sencilla, se le asigna el valor de 4 puntos. Si la complejidad era media, el valor será de 7 puntos, y si la tarea era compleja y dificil de realizar entonces serán 15 puntos. Cada tarea tendrá entonces un valor arbitrario que se medirá en puntos de función. Hay que destacar que estos valores siguen una escala no lineal totalmente arbitraria y pueden ser alterados conforme el equipo vaya avanzando en el proyecto.

Después, se calcula el número de horas que ha trabajado cada programador a lo largo del periodo y se multiplican por un coeficiente en función de si el programador es novato, al que se le asignará un valor de 0.8, si el programador tiene experiencia, que se le asignará un valor de 1, y si el programador es un programador senior, al que se le asignará un valor de 1.2

Estos valores se sumarán para, posteriormente dividirlo entre el sumatorio de las tareas multiplicadas por el valor que se le ha asignado por la complejidad. De esta manera se obtiene una estimación de las horas que se invierten en programación por punto de función y este dato se guarda. Se sumará un pequeño porcentaje a las horas calculadas de entre el 5% y el 15% para tener un colchón con el que poder trabajar en caso de que algo salga mal (de esta manera obtenemos un pequeño margen de error)

Conforme se van realizando más iteraciones o períodos temporales y se van anotando las horas por punto de función. Se realiza una media a partir de los períodos para , a partir de las mismas, poder estimar con relativa precisión cuánto costará programar cada una de las unidades funcionales en un futuro.

Ejemplo

Explicaremos este método con un ejemplo más sencillo. Supongamos que la empresa E tiene las siguientes tareas completadas a lo largo del primer periodo de medición, con su dificultad asociada:

1. Crear el registro de usuarios - complejo
2. Conectar la vista del registro de usuarios con la base de datos - medio
3. Diseñar un módulo de noticias - medio
4. Implementar la base de datos para el módulo de noticias - sencillo
5. Crear el controlador y el middleware para la base de datos - medio

Se dispone de 4 programadores que trabajan 40 horas a la semana. Dos de los programadores son novatos, un programador es experimentado y el equipo cuenta con un programador sénior. Por tanto tenemos:
40*0.8 + 40*0.8 + 40*1 + 40*1.2, lo cual nos da un total de 152 horas de programación ponderadas.
Por otro lado, tenemos que los puntos de función son 15 + 7 +7 + 4 = 33

Realizamos la división de 152 /33 = 4.606 horas por punto de función.

Tras realizar este cálculo durante tres periodos de medición adicionales, obtenemos los valores de 5.54, 4.43 y 6.01 horas por punto de función. Tras calcular la media de estos tres períodos obtenemos una media de 5.36 por punto de función.

Con estos datos ya podemos realizar una estimación de cuánto tiempo nos va a costar realizar el trabajo de las siguientes iteraciones. Supongamos que las tareas a realizar la siguiente semana son las siguientes:

1. Integración de una IA - complejo - 15 puntos
2. Creación de un controlador para interactuar con los usuarios de una red social - medio - 7 puntos
3. Implementar la posibilidad de cambiar la contraseña - sencillo - 4 puntos

La media de horas por punto de función es de 5.36 horas, y tenemos 15 + 7 + 4 = 26 puntos de función, Multiplicando obtenemos una estimación de que realizar todas estas tareas será de 139.36 horas. Se le sumará un margen de error del 10%, por lo que al total de horas calculadas se le sumarán 13.9 horas, lo cual hacen un total de 153.26 horas. De esta manera podemos estimar que las funciones se podrán realizar en un periodo de una semana, si se realizan un par de horas extra, o en ocho días para ir un poco más holgados.

Finalizado el periodo, las mediciones indican que se han invertido 126,33 horas reales de programación, lo cual supone un 82% de las horas inicialmente estimadas. En el siguiente cálculo se restará un 18% a las horas estimadas para ser más preciso.

Este modelo matemático es una forma muy sencilla de estimar los costes en tiempo que acarreará cada una de las unidades funcionales que se deseen programar. Por supuesto los valores son muy arbitrarios y la estimación puede ser imprecisa si a las tareas no se les asigna un valor real de complejidad o si no se ha sido objetivo con la dificultad de la tarea , pero con el tiempo el sistema va obteniendo más datos y las estimaciones van mejorando con cada iteración o con cada medición temporal. De esta manera se puede solventar de forma rápida y sencilla cuánto tiempo costará programar las unidades y se podrá evitar en un futuro realizar predicciones muy ambiciosas a la hora de presentar funcionalidades en cada fecha de entrega.

3. El framework

Pasaremos ahora a explicar el framework comenzando con su estructura de directorios. Como se ha indicado anteriormente, ROWDS es un framework ligero escrito en PHP para entornos unix. La idea principal del proyecto es separar los contenidos del diseño web, proporcionar un entorno con modelo vista controlador que sea sencillo de utilizar, sólido, rápido, con una curva de aprendizaje muy corta y ligero.

Los directorios de ROWDS son los siguientes:

dependencies: En este directorio se guardan todas las dependencias que el framework tiene que cargar para ejecutar la web. Todas las clases que son necesarias para que la aplicación web funcione se meten aqui dentro en subcarpetas. Las clases como la encargada de gestionar la Sesión o la base de datos se introducirán dentro de esta carpeta.

essential: dentro de esta carpeta se guarda la clase del AutoLoader(), que es la encargada de inicializar todo el sistema , junto con los parámetros de configuración y los parámetros y definiciones que sean necesarios. Esta es la carpeta que el framework cargará primero.

lang: Como su propio nombre indica, aquí se almacenan todos los archivos de traducción de texto para cada uno de los módulos que así lo requieran. El traductor cargará desde esta carpeta el archivo de traducción correspondiente a cada vista. De esta manera las traducciones se tienen centralizadas dentro de un único directorio, lo cual ayuda al mantenimiento de la aplicación y a la posterior traducción de la misma. Dentro de la carpeta lang se introduce cada uno de los idiomas dentro de subcarpetas, y estas subcarpetas contendrán el

archivo de traducción correspondiente al idioma que esté empleando la vista en ese momento.

templates: La carpeta templates se encarga de gestionar y guardar los templates necesarios para la web. Cada template que se crea se introduce aqui para que después la web tenga acceso total a ellos. De esta manera, los archivos de template están totalmente externalizados de la aplicación y de los controladores y de esta manera se facilita la programación mediante el modelo vista controlador.

views: Aqui van cada uno de los componentes que forman la página web a nivel de vista. Las vistas se generan aquí, con sus controladores, se realizan las acciones y se implementan las funcionalidades a nivel de usuario y de usabilidad.

examples: es una carpeta totalmente ajena al funcionamiento del framework donde se ponen varios ejemplos realizados de algunas conexiones a bases de datos y maneras de trabajar.

En cada uno de los directorios que se creen dentro del proyecto , siempre se incluirá un archivo en blanco llamado index.html. Esta es una medida muy sencilla de seguridad para evitar que nadie pueda acceder al listado de carpetas de dentro de la aplicación web y pueda averiguar así la estructura interna de nuestra aplicación. De esta manera reducimos el riesgo de que pueda producirse ningún tipo de vulnerabilidad.

index.php

Pasaremos ahora a analizar el archivo index.php necesario para arrancar la página web. En cualquier aplicación web, se requiere un archivo desde el cual la aplicación inicia y comienza a cargar la funcionalidad requerida:

```
1. <?
2. //cargamos los defines ,dependencies y las clases
3. include("essential/defines.php");
4. include("essential/config.php");
5. include("essential/AutoLoader.php");
6. $loader= new AutoLoader();
7. $loader->loadDir('dependencies');
8. //$loader->loadDir('views');
9. $Session = new Session();
10. //Start the module router
11. $router = new ModuleRouter();
12. $route = $router->loadRoute();
13.
14. //load the default route and it's handlers
15. include("views/$route/$route.php");
16. $loader->loadHandlers("views/$route/handlers");
```

```
17. $loader->loadScripts($route);
18. $module = new $route;
19. $action = $router->loadAction($module);
20.
21. //Load the action
22. $loader->setAction($action);
23. $loader->setRoute($route);
24. $module->$action();
25.
26. ?>
```

Al principio cargamos la configuración y los parámetros. Después se se carga el autoloader. La clase autoloader es una clase que se encarga de cargar directorios y que se puede utilizar de forma auxiliar para cargar los archivos que una clase pueda requerir.

Se utiliza el AutoLoader para cargar el directorio de dependencias y se crea la Sesión y el Enrutador que se encargará de cargar la ruta y averiguar qué acción y qué módulo se debe cargar.

Con la ruta cargada, cargamos el módulo de la ruta y ejecutamos la acción que hay que realizar.

Pasaremos ahora a ver cómo están programadas las clases que hacen que el framework pueda funcionar por orden de ejecución. Para ello analizaremos las funciones que tienen lugar en cada una de ells

Class AutoLoader - essential/AutoLoader/Autoloader.php

```
1. public function loadDir($dir)
2.         {
3.                 $classes = scandir ($dir);
4.                 $bad = array(".", "..", ".DS_Store",
   "_notes", "Thumbs.db");
5.                 $classes = array_diff($classes, $bad);
6.                 foreach ($classes as $class)
7.                 {
8.                         if(is_dir("$dir/".$class))
9.                         {
10.                                 if ($class[0]!='.' &&
   !is_dir($class))
11.                                 {
12.                                         require_once
   ("$dir/$class/$class.php");
13.                                 }
14.                          }
15.                 }
16.         }
```

La llamada a loadDir sencillamente carga el directorio que se le indica por el parámetro. Por lo general puede haber muchos archivos ocultos que el sistema operativo introduzca en el directorio pero que nosotros no deseemos cargar, por lo que excluimos los archivos ocultos en unix, el .DS_Store de Apple (que se encarga de guardar la información referente a la posición de los iconos en un entorno MAC) o el Thumbs.db de sistemas Windows. A continuación, se crea la sesión mediante la clase Session.

Class Session - dependencies/Session/Session.php

```
1.      public function Session()
2.       {
3.               Session_start();
4.               if (!isset($_SESSION['login']) )
5.               {
6.                       $_SESSION['login'] = 'anon';
7.               }
8.       }
```

El objeto sesión se encarga de inicializar la sesión del lado del servidor mediante PHP y después comprueba si existe una sesión anterior. en caso de que no exista, se identifica la sesión como una persona anónima. Es importante destacar que este método impide de ninguna manera que un usuario pueda llamarse anon, ya que esta se define como una palabra reservada del sistema para describir a los usuarios anónimos que no se han registrado en el sistema y de los cuales no se dispone de información. La Sesión tiene también una pequeña API integrada que sirve, por ejemplo, para comprobar si el usuario es anónimo o está registrado.

Class Session - dependencies/Session/Session.php

```
1. public function checkAnon()
2.       {
3.               if ( !strcmp($_SESSION['login'] ,'anon') )
4.               {
5.                       return true;
6.               }
7.               return false;
8.       }
```

De nuevo comprueba si la sesión es anónima o no y devuelve verdadero o falso en función del resultado.

El siguiente paso que realiza la web es cargar la ruta para averiguar que módulo hay que cargar y poner en marcha. Esto se consigue mediante la funcion loadRoute del AutoLoader

Class AutoLoader - essential/AutoLoader/AutoLoader.php

```
1. public function loadRoute()
2.         {
3.                 if (isset($_GET["option"]))
4.                 {
5.                         $variable = $_GET["option"];
6.                         if
   (file_exists("views/$variable/$variable.php"))
7.                         {
8.
9.                                 return $variable;
10.                         }
11.                         else
12.                         {
13.
14.                                 return MODULE_NOT_FOUND;
15.                         }
16.                 }
17.                 else
18.                 {
19.
20.                         return MODULE_DEFAULT;
21.                 }
22.         }
```

La forma en la que se elige el módulo mediante la ruta es con el parámetro “option” que se pasa por la URL. La clase carga la ruta de la carpeta “views” que sea se devuelve y en caso de que no exista la carpeta te devuelve el valor del módulo 404 (MODULE_NOT_FOUND). De esta manera podemos además personalizar cada vez que no se encuentra una ruta.

Después de eso, se carga la ruta que ha devuelto el método loadRoute() y con el método loadHandlers() se cargan los manejadores que requiere la vista.

```
1. $loader->loadHandlers("views/$route/handlers");
```

Class AutoLoader - essential/AutoLoader/AutoLoader.php

```
1.         public function loadHandlers($dir)
2.         {
3.                 $classes = scandir ($dir);
```

```
4.                $bad = array(".", "..", ".DS_Store",
   "_notes", "Thumbs.db");
5.                $classes = array_diff($classes, $bad);
6.                foreach ($classes as $class)
7.                {
8.                        if ($class[0]!='.' &&
   !is_dir($class))
9.                        {
10.
11.                               require_once
   ("$dir/$class");
12.                       }
13.
14.               }
15.       }
```

De la misma manera que con los directorios ,cargamos los handlers de la clase de la vista que vamos a necesitar para el controlador. La siguiente parte que hacemos es cargar los scripts de JavaScript que vamos a necesitar para la vista.

Class AutoLoader - essential/AutoLoader/AutoLoader.php

```
1. public function loadScripts($view)
2.        {
3.                if (is_dir("views/$view/js"))
4.                {
5.                        $this->scripts = scandir
   ("views/$view/js");
6.                        $bad = array(".", "..", ".DS_Store",
   "_notes", "Thumbs.db");
7.                        $this->scripts = array_diff($this-
   >scripts, $bad);
8.                }
9.                else $this->scripts = null;
10.       }
```

Cargamos todos los scripts necesarios y los mantenemos guardados para imprimirlos en el template más adelante.

Class ViewHelper- dependencies/ViewHelper/ViewHelper.php

```
1. public function loadAction($m)
2.        {
3.                if (isset($_GET["action"]))
```

```
4.                {
5.                        $variable = $_GET["action"];
6.                        if
   (method_exists($m,$variable."Controller"))
7.                        {
8.
9.                                return
   $variable.'Controller';
10.                        }
11.                        else
12.                        {
13.
14.                                return ACTION_DEFAULT;
15.                        }
16.                }
17.                else
18.                {
19.                        return ACTION_DEFAULT;
20.                }
21.        }
```

De la misma manera que con la elección del módulo. Realizamos el mismo paso para la Acción y averiguar así que controlador de la vista debemos cargar. El método comprueba si el método de la acción que estamos invocando existe y en caso afirmativo devuelve el nombre de la acción para poder ejecutarla. Por último, ejecutamos esa acción

El modelo vista - controlador

Una vez llegados a este punto, podemos empezar a programar como queramos el controlador.

Ejemplo de views/$view/$view.php

```
1. <?php
2.
3. class main extends ViewHelper
4. {
5.
6.        public function __construct__()
7.        {
8.                super();
9.        }
10.
11.
12.        public function mainController()
13.        {
14.                //This loads the main page
15.                $this->generate();
16.        }
```

Para entender mejor este concepto, cabe destacar que toda vista que se construye debe extender la clase padre ViewHelper.

Class ViewHelper - dependencies/ViewHelper/ViewHelper.php

```php
<?php
global $view;
global $translator;
global $name;
global $actionView;
use Dependencies\ErrorModule;
use Dependencies\Translator;
class ViewHelper
{
	protected $view;
	protected $errors;
	protected $translator;
	protected $template;

	public function getView()
	{
		return $this->view;
	}

	public function setView($v)
	{
		$this->view = $v;
	}

	public function ViewHelper()
	{
		$this->errors = new ErrorModule();
		$this->translator = new Translator();
		$this->translator->load(get_class($this));
		$this->template = "default";
		$this->view = new stdClass;
	}

	function super() {
		$par = get_parent_class($this);
		$this->$par();
	}

	public function getTemplate()
	{
		return $this->template;
```

```
42.         }
43.
44.         public function setTemplate($t)
45.         {
46.                 $this->template = $t;
47.         }
48.
49.         public function mainController()
50.         {
51.                 $this->generate();
52.         }
53.
54.         public function generate($file ='view')
55.         {
56.                 if ($this->errors->hasErrors())
57.                 {
58.                         $this->errors->printErrors();
59.                         die();
60.                         //TODO , this has to be made better
   on the next iteration
61.                 }
62.                 $view = $this->view;
63.                 $translator = $this->translator;
64.                 $name = get_class($this);
65.                 $template = $this->getTemplate();
66.                 $actionView = $file;
67.                 include("templates/$template/$template.php"
   );
68.
69.         }
70.
71. }
72. ?>
```

Las funciones getTemplate() y setTemplate() servirán para cargar los templates correspondientes. Por defecto al provenir de la clase ViewHelper, toda vista tendrá un template por defecto que será el que se reciba con el método getTemplate(), que es el que se utilizará para renderizar la página. De esta manera, si en cualquier parte de la vista (como por ejemplo en el constructor) se desea alterar el template por defecto para que incluya un template diferente, bastará con que se emplee el método setTemplate() para que el template cambie por completo y poder tener así un diseño completamente diferente. La función super() se agrega también en la clase hijo porque en PHP no se ejecuta el constructor de la clase padre, sino solamente la de la clase hija.

Si no se establece un controlador principal (en este caso se ha elegido main como el controlador por defecto) , dado que en PHP, cuando se extiende una clase, la subclase hereda todos los métodos públicos y protegidos de la clase padre, a menos que una clase sobrescriba esos métodos, mantendrán su funcionalidad original. Por tanto una clase que extienda del ViewHelper tendrá el método por defecto del controlador (main) que no realizara absolutamente

nada más que invocar al método generate, que imprimirá la página. Con esta versatilidad podemos reescribir el método main en la clase hija para que haga lo que nosotros queramos antes de llamar a generate().
Puede ocurrir que no nos interese generar código HTML, y que nuestro controlador quiera devolver un objeto en formato XML o JSON. En tal caso al final del controlador sencillamente no agregaremos la función generate(), por lo que la vista no se ejecutará nunca y podremos ser nosotros quienes decidamos que deseamos imprimir en pantalla., Por ejemplo, para aplicaciones web modernas una llamada ajax podría buscar la respuesta de un controlador de nuestra aplicación,el cual le devolvería un objeto en formato JSON que la vista parsearía y utilizaría para sus propias necesidades. También podría crear una cookie y usarla como controlador de sesión en el lado del cliente. Esto es totalmente posible porque hasta este punto, la aplicación todavía no ha impreso nada por la salida estándar. El buffer todavía se encuentra vacío y no se han enviado todavía las cabeceras del archivo, por lo que se puede modificar la cabecera y que la petición no necesariamente tenga por qué devolver código HTML. La utilidad de esto permite que nuestra herramienta sea tremendamente versatil. Pasaremos ahora a analizar la función generate:

Class ViewHelper - dependencies/ViewHelper/ViewHelper.php

```
1.      public function generate($file ='view')
2.      {
3.              if ($this->errors->hasErrors())
4.              {
5.                      $this->errors->printErrors();
6.                      die();
7.                      //TODO , this has to be made better
   on the next iteration
8.              }
9.              $view = $this->view;
10.              $translator = $this->translator;
11.              $name = get_class($this);
12.              $template = $this->getTemplate();
13.              $actionView = $file;
14.              include("templates/$template/$template.php"
   );
15.
16.      }
```

La función generate se encarga de realizar todas las tareas necesarias antes de comenzar a imprimir la vista. Si el módulo de errores de la vista ha devuelto errores, los imprime y termina la ejecución. Podemos ver un TODO comentado en parte del método. Esto está pensado para que en un futuro, quizás podría ser interesante que el módulo no se limitase a imprimir errores, sino que pudiera enviar el reporte de errores en cualquier otro formato (un error 500 , un objeto json) y enviarlos al lugar de dónde procede la petición (de normal sería un navegador, pero podría ser una petición ajax). Esta es una implementación que podría implementarse en un futuro.

El atributo view contiene los objetos que queremos pasarle a la vista a través del controlador. Supongamos que hemos cargado los nombres y apellidos del usuario y hay que imprimirlos por pantalla. Lo guardaríamos como un atributo del objeto $view para que luego, en la vista pudiésemos sencillamente imprimirlo. Todo lo que se guarde referenciado en el atributo view pasará a la vista para que se pueda utilizar. Veremos un par de ejemplos más adelante.

El translator también se pasa a la vista para mantener el sistema de traducción en la vista y que todas las cadenas de texto puedan ser traducidas al idioma correspondiente en la vista.

Por último, guardamos el nombre de la clase en el atributo $name y el atributo $file (que podemos ver que por defecto, la vista será view.php, pero que esto se puede cambiar independientemente para que cada acción tenga una vista diferente) para utilizarlo en el template más adelante. Seleccionamos el template mediante la acción getTemplate de la que se ha hablado anteriormente y se procede a cargar el template.

Procederemos ahora a enseñar el template por defecto de una aplicación ya programada para que sirva como ejemplo sobre qué pinta puede tener un template.

Default Template - templates/default/default.php

```
1. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
   Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
   transitional.dtd">
2. <html>
3. <head>
4. <title><?php echo WEB_NAME ?></title>
5. <!--<link
   href='http://fonts.googleapis.com/css?family=Prosto+One'
   rel='stylesheet' type='text/css'> -->
6. <link rel="stylesheet" href="<?php echo
   "templates/$template/css/style.css" ?>" type="text/css" />
7. <!--[if lte IE 6]>
8. <link rel="stylesheet" href="<?php echo
   "templates/$template/css/ieonly.css" ?>" type="text/css" />
9. <![endif]-->
10. <?php global $loader; global $route; $loader-
   >printScripts($route); ?>
11. </head>
12. <body>
13.         <div id="tablecontent" >
14.         <?php include("views/$name/$actionView.php");?>
15.                 <div class="clearer"></div>
```

```
16.         </div>
17. </body>
18. </html>
```

Aqui podemos ver claramente como ya empezamos a imprimir en pantalla código HTML, con su cabecera y demás. Esta es la primera clara separación que hay entre la vista y el controlador. Hemos separado ambos contenidos para que los programadores puedan trabajar por un lado tranquilamente y los maquetadores y diseñadores puedan trabajar por otro sin ningún problema.

Pondremos especial atención ahora a la parte PHP del código html.

Default Template - templates/default/default.php

```
1. <?php global $loader; global $route; $loader-
   >printScripts($route); ?>
2. </head>
3. <body>
4.         <div id="tablecontent" >
5.         <?php include("views/$name/$actionView.php");?>
```

la primera cosa que está ocurriendo aquí es que se imprimen los javascripts que hemos cargado anteriormente con el método loadScripts().

Autoloader Class - essential/Autoloader.php

```
1.         public function printScripts($view)
2.         {
3.                 foreach($this->scripts as $script)
4.                 {
5.                         echo "<script language='javascript'
   src='views/$view/js/$script'></script>";
6.                 }
7.         }
```

Podemos comprobar sencillamente que para cada script, se incluye la ruta que hemos cargado en la propia vista. Una forma muy sencilla y a la vez elegante de separar los scripts js que necesita cada uno de los módulos. de forma genérica y sin tene que preocuparse de incluirlos manualmente. Por supuesto se pueden seguir añadiendo más javascripts de forma manual en el propio template. Por último, es la siguiente línea de código la que llama a la vista de la acción

```
<?php include("views/$name/$actionView.php");?>
```

Con este último include llamamos a la variable $actionView que hemos declarado antes y que es la vista de la acción que vamos a realizar. Dado que está parametrizado, siempre podemos elegir cualquier vista, e incluso hacer que un mismo controlador pueda elegir varias vistas en función de algún

parámetro de entrada o una variable de control. Procederemos ahora a enseñar una vista genérica

módulo genérico main, acción : register - views/main/register.php

```
<div class = "imglogo"><img
src="views/main/img/killercorp.png"></img></div>

<div class ="fieldinput">
        <div class = "img"><img src="views/main/img/<?php
echo $view->imageURL; ?>.jpg"></img></div>
        <div class = "name"><?php echo $view->name;
?>,<span> <?php echo utf8_decode($translator-
>trans('INVITED'));?></span></div>
        <div class = "info"></div>
        <div class = "mail"><input type="text"
placeholder='<?php echo utf8_decode($translator-
>trans('ENTER_EMAIL'));?>'></input></div>
        <div class = 'button'><?php echo
utf8_decode($translator->trans('PARTICIPATE'));?></div>
</div
```

De nuevo podemos ver código html impreso en la salida estándar. También podemos ver como se acceden a los elementos que se han guardado en la vista de forma sencilla, y como se accede al módulo traductor de una manera sencilla, para que este cargue las variables que sean necesarias traducir y se imprima el texto traducido correctamente. Cabe destacar también que en este ejemplo, esta vista se ha cargado desde el registerController, que es el controlador de la acción register.

Vamos a proceder a ver como ejemplo el controlador del registro, que forma parte del módulo main que carga por defecto. Podemos apreciar que si el módulo a cargar es main y la acción a realizar es la que viene por defecto, la vista no será la misma que si el módulo a cargar es main y la vista a realizar es register. De nuevo, esta es una herramienta muy cómoda y muy potente para los desarrolladores.

RegisterController - Class Main - views/main/main.php

```
public function registerController()
        {
                if (!isset($_GET['k']))
                {
                        $this->errors->addError($this-
>translator->trans('NOACCESS'));
                }
                $var = $_GET['k'];
                if (strlen($var) > 13 )
```

```
9.                {
10.                       $this->errors->addError($this-
   >translator->trans('HACK_ATTEMPT'));
11.                }
12.
13.                $db = new DatabaseHandler();
14.                if ( $db->checkSecurityCode($var) != TRUE)
15.                {
16.                       $this->errors->addError($this-
   >translator->trans('NO_USER'));
17.                }
18.                $assassin = $db->getAssassinInfo($var);
19.                if (!$assassin)
20.                {
21.                       $this->errors->addError($this-
   >translator->trans('NO_USER'));
22.                }
23.                $this->view->mail = $assassin['mail'];
24.                $this->view->info = $assassin['info'];
25.                $this->view->name = $assassin['name'];
26.                $this->view->imageURL =
   $assassin['imageURL'];
27.
28.                $this->generate('register');
29.         }
```

Aquí ya podemos ver como se utilizan handlers para generar la vista, y de cómo se carga y se emplea el middleware, del que hablaremos más adelante. También se puede apreciar cómo se utiliza el translator para poder incluso añadir errores en el módulo de control de errores.

El middleware.

La capa middelware intermedia es la solución que utiliza la aplicación para crear un enlace entre la base de datos y el controlador. El middleware se encarga de funciones específicas que el controlador le pide. De esta manera se focaliza el código de conexión con la base de datos y el del controlador en archivos y lugares diferentes y el tratamiento de errores está más distribuido y el mantenimiento de la aplicación resulta más sencillo. Pasemos a ver el controlador de la base de datos estándar..

Class DBConn - dependencies/DBConn/DBConn.php

```
1. <?php
2.
3. Class DBConn
4. {
5.        protected $username, $password,$database,$host,
   $DBODB;
6.
7.        public function DBConn($host,$db,$user,$pw)
```

```
8.        {
9.                $this->host = $host;
10.                $this->database = $db;
11.                $this->username = $user;
12.                $this->password = $password;
13.
14.                $this->DBODB = new
   PDO("mysql:host=$host;dbname=$db;charset=utf8", $user,
   $pw);
15.        }
16.
17. ?>
```

El constructor de la capa middleware acepta los argumentos necesarios para poder realizar la conexión a la base de datos y guarda un objeto de la conexión realizada. A través de este objeto se realizarán todas las operaciones en la base de datos que en este caso es un sistema de gestión de base de datos de MYSQL, pero que gracias al driver PDO podría tratarse de cualquier otro sistema de gestión sql de bases de datos.

Este archivo en si no se utiliza nunca, y se encuentra en el framework meramente como un archivo de ayuda para la construcción del propio middleware como handler de cada una de las vistas y para poder explicar el código en este proyecto.

Con esta parte del código realizada, observaremos ahora como se realiza una conexión para obtener datos de usuario de ejemplo.

DatabaseHandler class - views/main/handlers/DatabaseHandlers.php

```
1. public function checkSecurityCode($security)
2.        {
3.                $stmt = $this->DBODB->prepare("SELECT * FROM
   users WHERE security=?");
4.                $stmt->execute(array($security));
5.                $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
6.                if ( $stmt->rowCount() == 1)
7.                {
8.                        $this->assassin = $rows[0];
9.                        return TRUE;
10.                }
11.                else
12.                {
13.                        return FALSE;
14.                }
15.        }
```

En este ejemplo podemos apreciar como el driver PDO accede a la base de datos y ejecuta una consulta SQL. El sistema de gestión de la base de datos devuelve un resultado que el middleware analiza y devuelve al controlador

dependiendo de la función que el controlador necesita. En este caso, por ejemplo, el controlador necesita saber si el código de seguridad que se le ha pasado como parámetro existe y tiene algún usuario en la base de datos. La función devuelve true o false en función de si el usuario existe o no. La gestión de errores de la base de datos la lleva el propio sistema de gestión. Quizás la forma más correcta sería englobar ese apartado del código en una sentencia try/catch para poder capturar el error en caso de que la base de datos no estuviera disponible, pero los bloques try/catch en PHP son bastante pesados de cargar. Elevan notoriamente la carga de transacciones en el lado del servidor y el resultado al final será el mismo (que la base de datos siga sin estar accesible). ROWDS deja a discreción del programador si insertar el código de la base de datos en un try catch y poder tratar el error que daría la aplicación de forma personalizada y enmascararlo en vez de usar el propio control de errores que lleva el propio sistema de gestión de la base de datos.

Implementación del módulo de errores.

El control de errores es algo muy básico en cualquier sistema. Es necesario saber qué errores pueden tener lugar en la aplicación. Saber identificarlos antes de que ocurran y tener un tratamiento adecuado de los mismos para que depurar el código y averiguar dónde está el error sea más sencillo.

ROWDS implementa un control de errores muy sencillo e intuitivo de utilizar para el programador.

ErrorModule class - dependencies/ErrorModule/ErrorModule.php

```
1. <?php
2. namespace Dependencies;
3. class ErrorModule
4. {
5.         protected $number;
6.         protected $errors;
7.
8.         public function ErrorModule()
9.         {
10.                 $this->errors = array();
11.                 $this->number = 0;
12.         }
13.
14.         public function getNumber()
15.         {
16.                 return $number;
17.         }
18.
19.         public function addError($msg)
20.         {
21.
22.                 $this->errors[$this->number]= $msg;
```

```
23.                 $this->number++;
24.         }
25.
26.         public function getErrors()
27.         {
28.                 return $errors;
29.         }
30.
31.         public function printErrors()
32.         {
33.                 echo '<p class="errors"> ';
34.                 foreach ($this->errors as $error)
35.                 {
36.                         echo $error;
37.                         echo '<br/>';
38.                 }
39.                 echo '</p>';
40.         }
41.
42.         public function hasErrors()
43.         {
44.                 if ($this->number) return true;
45.                 else return false;
46.         }
47. }
48. ?>
```

El módulo de errores guarda un vector de errores. Cada vez que se añade un error, la cadena de texto se añade a este vector y se suma uno al número de errores existente. Podemos ver que hay un método para comprobar si existen errores y otro método que imprime en pantalla mediante código HTML personalizable con CSS los errores que hayan tenido lugar. En un principio ROWDS, como ya se ha explicado con anterioridad, corta la ejecución de la aplicación web una vez se han listado todos los errores. Hay que diferenciar un error de la aplicación (buscar un código de usuario que no existe o no poder conectarse a la base de datos) de un error de usuario (equivocarse a la hora de escribir una contraseña). Los errores de aplicación implican que sin la información necesaria o con el error recién ejecutado, la ejecución de la aplicación no puede seguir teniendo lugar.

La vista en el lado del cliente

En aplicaciones web, cada vez es más y más importante el uso de HTML5 y frameworks JS como JQUERY para conseguir una funcionalidad y una usabilidad mayor. Las funcionalidades que el avance en las tecnologías web han permitido realizar han incrementado exponencialmente el uso de aplicaciones web que están cada vez más presentes en más aplicaciones y dispositivos móviles. Todo esto no sería posible de no ser por la sustancial mejora en los entornos de desarrollo de software de aplicaciones web. El código que se ejecuta es más rápido y de mayor calidad y se pueden programar una mayor cantidad de funcionalidades en menor tiempo

Como medida adicional para organizar los archivos, cada vista contiene una carpeta separada llamada js donde se guardan y se gestionan los archivos JavaScript. La vista en el lado del cliente se compone así pues de un archivo .php donde se genera el html y la información necesaria, y cada una de las acciones del controlador tiene a su vez un archivo xxxController.js en la carpeta js donde carga su propio controlador JS y donde el usuario puede introducir todo el código JavaScript que le sea necesario. De esta manera se mantiene un mejor control sobre donde va cada archivo y se aumenta el mantenimiento de los cada dia más complejas aplicaciones web. Además, también se incluye una subcarpeta common donde se guardan archivos javascript comunes a todas las vistas, para que la reutilización de código sea mayor y la estructuración y organización de archivos sea más óptima.

El framework jQuery

jQuery es una libreria cross-browser de JavaScript diseñada para simplificar el lado del cliente de la programación de código HTML/JS. Es usado en el 65% de las 10.000 páginas más visitadas de internet y a día de hoy es la biblioteca JavaScript más usada de todos los tiempos.

La sintaxis de jQuery ha sido diseñada para hacer más sencillo el navegar a través de un documento HTML, seleccionar elementos del DOM, crear animaciones, manejar eventos y desarrollar aplicaciones en Ajax. jQuery también proporciona capacidades para que desarrolladores creen plugins encima de la librería JavaScript. La forma de elegir elementos del DOM y manipularlos ha creado una nueva forma de programar, fusionando algoritmos y estructuras de datos DOM.

Ejemplos de uso de jQuery y AJAX en ROWDS

Se mostrarán ahora algunos ejemplos de utilización de jQuery y AJAX en ROWDS. Como hemos indicado antes, el entorno RODWS no tiene porque siempre devolver código HTML, sino que puede perfectamente devolver cualquier objeto e imprimir por pantalla cualquier script, ya sea XML, HTML u objetos JSON. El ejemplo que se mostrará a continuación es un botón de login que una vez se aprieta realiza una llamada ajax a un controlador de ROWDS que procesa la petición y devuelve un objeto JSON con la respuesta, que el código del lado del cliente deberá interpretar como un acceso autorizado o por el contrario un acceso no autorizado, y en función de ello imprimir un error o crealizar la transición del login correctamente.

Loader.js - views/main/js/loader.js

```
1. $('.loginButton').on('click', function(){
2. var jqxhr = $.ajax( {url:
   "index.php?option=main&action=login", data: { code:
   $('.login input').val() }} )
3.                  .done(function() {
4.                    console.log( "success" );
5.                    if (jqxhr.responseText == "-1")
6.                    {
```

```
                        $('.info').addClass('error');
                        $('.info').html('Acceso no
autorizado');
                        $('.info').fadeIn();
                         return;
                    }
                    else
                    {
                         var tmp =
$.parseJSON(jqxhr.responseText);
                         localStorage['id'] =
tmp.id;
                         //.. se omite el código en el
ejemplo ....//
                    }

                })
                .fail(function() {
                  console.log( "error" );
                })
                .always(function() {
                  console.log( "complete" );
                });
});
```

Se puede ver fácilmente cómo se programa con jQuery en este tipo de entornos y como se usa ajax para enviar peticiones HTTP y recibir peticiones en objetos JSON. La llamada en ajax se realiza a la URL "index.php?option=main&action=login" y se le pasan los datos obtenidos de un input de HTML. Una vez que recibe la petición, si el objeto devuelto no es -1 y se trata en realidad de un objeto JSON, lo parsea y realiza el resto de la acción. El propio jQuery se encarga de contemplar los errores en caso de que por alguna razón la llamada ajax no pueda realizar, pero no se se puede encargar de gestionar los errores de acceso a la base de datos, ya que este código se ejecuta en el cliente. Esos errores los gestiona directamente la aplicación web en la parte del servidor.

Se mostrará ahora el controlador en ROWDS que recibe esta petición

Main class - views/main/main.php

```
        public function loginController()
        {

                if (!isset($_GET['code']))
                {
                        json_encode(-1);
                        return;
                }
```

```
9.                $var = $_GET['code'];
10.                $db = new DatabaseHandler();
11.                $tmp = $db->checkLogin($var);
12.                if ( !$tmp)
13.                {
14.                        echo json_encode(-1);
15.                        return;
16.                }
17.                else
18.                {
19.                        global $Session;
20.                        $Session-
   >set('login',$tmp['name']);
21.                        $Session->set('id',$tmp['id']);
22.                        $Session->set('code',$tmp['code']);
23.                        echo json_encode($tmp);
24.                        return;
25.                }
26.        }
```

Como se puede observar, el controlador comprueba si existe el código, crea un nuevo Handler de la base de datos y realiza una llamada al método checkLogin del middleware. En caso afirmativo, carga la sesión y guarda los parámetros en la sesión que son necesarios para la correcta identificación del usuario. Observemos como este controlador no llama a la función generate(), sino que directamente hace uso de la función json_encode() para enviar los datos de vuelta a la petición ajax, que utilizara el método parseJSON (JavaScript Object Notation) para convertir el objeto recibido en un objeto JavaScript con el que poder interactuar, obtener sus datos etc.

Para finalizar, mostraremos el código del middleware encargado de interactuar con la base de datos.

DatabaseHandler class - views/main/handlers/DatabaseHandler.php

```
1.        public function checkLogin($code)
2.        {
3.                $stmt = $this->DBODB->prepare("SELECT * FROM
   users WHERE code=?");
4.                $stmt->execute(array($code));
5.                $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
6.                if ( $stmt->rowCount() == 1)
7.                {
8.                        $this->assassin = $rows[0];
9.                        return $this->assassin;
10.                }
11.                else
12.                {
13.                        return FALSE;
14.                }
15.        }
```

Aqui de nuevo se puede ver como se comprueba si existe un usuario con el código escrito y se devuelve el objeto de usuario que devuelve la base de datos o falso en caso de que el usuario no exista.

Se procederá ahora a ver otro ejemplo posible de uso sin utilizar notación JSON ni el framework jQuery.

Ejemplo de conexión sin jQuery

```
function nuevoTurno()
{
	var url = "?option=ajax&action=newTurn&aparam";
	http.open('get',url);
	http.onreadystatechange = handleNewTurnResponse;
	http.send(null);
}

function handleNewTurnResponse()
{
	 if(http.readyState == 4 && http.status == 200)
	 {
		var response = http.responseText;
		if(response)
		{
			document.getElementById("Turns").innerHTML = response;
		}
	 }
}
```

de nuevo aquí podemos ve que se llama al módulo ajax y a la acción newTurn del componente ROWDS. En este ejemplo se ha creado un componente adrede para gestionar todas las llamadas AJAX.

```
	public function newTurnController()
	{
		$animaHandler = new animaHandler();
		$this->view['response'] = $animaHandler->generateNewTurn();
		$this->generate();
	}
```

En este caso no se devuelve ningún objeto JSON, lo que se va a devolver es código HTML puro. En este caso, el handler devuelve una lista HTML de nombres que se acoplan al parámetro 'response' del atributo view, que será lo que luego se acabe imprimiendo en la respuesta AJAX. El código html que se devuelve sustituye al código original en la web mediante la linea

document.getElementById(“Turns”).innerHTML = response . Para evitar que hayan cabeceras innecesarias, se crea un template en blanco al que no se le añade nada más que la propia respuesta que da el módulo.

White Template - templates/white/white.php

```
1. <?php include("views/$name/$actionView.php");?>
```

De esta manera el middleware y/o el controlador se encarga de devolver código HTML en la respuesta. En este ejemplo, la vista en el lado del cliente lo único que tiene que hacer imprimir la respuesta en el contenedor html adecuado, mientras que en el anterior ejemplo la vista del lado del cliente recibe un objeto JSON que debe procesar y reemplazar cada uno de los elementos o introducirlos dónde corresponda.

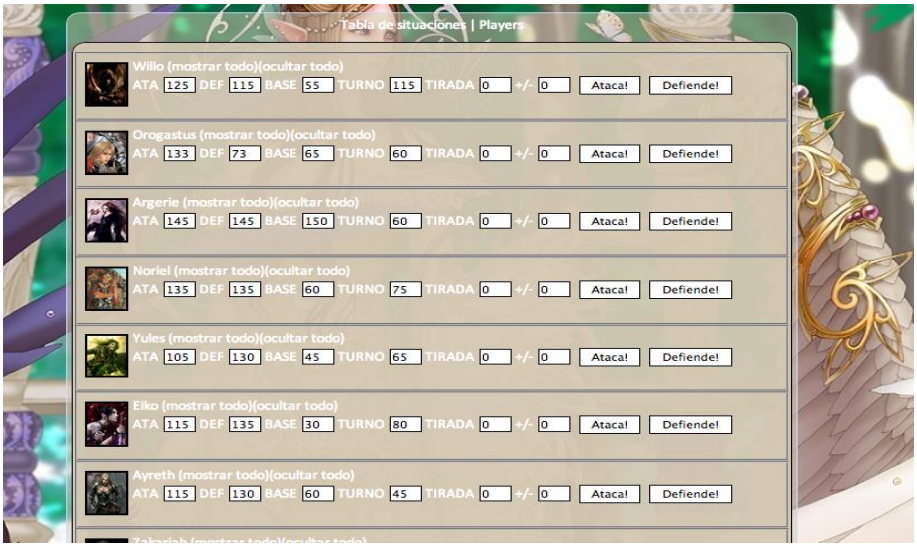
Esta segunda forma de pasar los datos está en desuso, ya que se le pasa al controlador y al middleware una funcionalidad que debería realizar la vista. Esta forma de recibir y procesar llamadas AJAX no respeta completamente el modelo vista controlador. Pese a ello, ambas formas de hacerlo son perfectamente válidas y posibles y se le deja al desarrollador la libertad de poder hacerlo de una manera u otra.

4. Aplicaciones ya diseñadas en un sistema ROWDS.

Para demostrar la potencia de ROWDS mostraremos dos aplicaciones sencillas ya diseñadas con la metodología y el framework ROWDS. La primera aplicación es un gestor de personajes y combates del juego de rol “Anima: Beyond Fantasy”. Utiliza una las ventajas de la tecnología ajax con una capa middleware que se conecta con un sistema de gestión de bases de datos MYSQL . La segunda aplicación es un gestor de usuarios para un juego de rol en vivo llamado “Killer: el juego de los asesinos” en el que los jugadores juegan a eliminarse entre ellos con armas ficticias.

Cabe destacar que el framework continúa abierto actualmente se está empleando para desarrollar dos aplicaciones adicionales, pero la descripción de ambas no tienen cabida en este proyecto, siendo su mención solamente para ayudar a describir el potencial de este framework así como sus usos futuros en entornos de desarrollo muy diversos.

4.a Anima: Beyond Fantasy - Gestor de personajes.



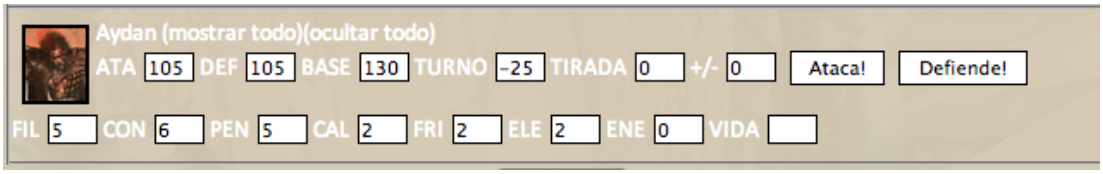
Captura de pantalla de la aplicación “Anima Helper”

“Anima: Beyond fantasy” es un juego de rol en el que los jugadores encarnan a un personaje con poderes sobrenaturales y un director del juego les guía a través de una aventura. Para realizar los combates se deben realizar una serie de tiradas que complican el desarrollo del juego. La aplicación “Anima Helper” ayuda a realizar estos cálculos matemáticos para permitir que los jugadores se centren en el juego y no en el funcionamiento de las mecánicas. La aplicación lleva una interfaz ajax que se conecta con la base de datos para obtener los atributos de los personajes y permitir su modificación en cualquier momento de forma persistente. Cada tirada que realizan los personajes se anota en la aplicación y al apretar el botón de “Ataca” o “Defiende” se realizan los cálculos y se obtienen los resultados pertinentes al combate.

De esta manera el proceso del combate se reduce enormemente y se agilizan las partidas. Además la aplicación también cuenta con la funcionalidad de

calcular los turnos en los que le corresponde a cada uno de los jugadores actuar.

Para realizar los cálculos, el sistema guarda todos los atributos del personaje, que son ocultables cuando no es necesario tenerlos a la vista mediante los botones de mostrar u ocultar todo.



Se procederá a mostrar parte del handler que se encarga de la conexión con la base de datos.

AnimaHandler.php - views/principal/handlers/AnimaHandler.php

```
1. class animaChar
2. {
3.         var $ID;
4.         var $NOMBRE;
5.         var $ATA;
6.         var $DEF;
7.         var $BASE;
8.         var $FIL;
9.         var $CON;
10.         var $PEN;
```

```
11.         var $CAL;
12.         var $ELE;
13.         var $FRI;
14.         var $ENE;
15.         var $PB;
16.         var $RF;
17.         var $RV;
18.         var $RM;
19.         var $RP;
20.         var $PV;
21.         var $TURNO;
22.         var $LEFT;
23.         var $next = null;
24.
25.         function get($var)
26.         {
27.                 return $this->$var;
28.         }
29.
30.         function set($var,$value)
31.         {
32.                 $this->$var = $value;
33.         }
34.
35. }
36.
37. class animaHandler
38. {
39.         var $dbConn;
40.         var $error;
41.         var $chars;
42.         var $enemies;
43.         var $charnum=0;
44.         var $enemynum=0;
45.
46.         public function animaHandler()
47.         {
48.                 $this->error = new errorModule();
49.                 $this->dbConn = new MyCon();
50.                 $this->dbConn->connect("localhost", "root",
    "", "anima");
51.                 $this->chars = new animaChar();
52.                 $this->enemies = new animaChar();
53.
54.                 //$this->loadChars();
55.         }
56.
57.         public function updateChar($param,$value, $name)
58.         {
```


```
59.                 $sql = "UPDATE pj SET $param =$value WHERE
   nombre = '$name'";
60.                 $this->dbConn->execute($sql);
61.         }
62.
63.         public function updateEnemy($param,$value, $name)
64.         {
65.                 $sql = "UPDATE enemy SET $param =$value
   WHERE nombre = '$name'";
66.                 $this->dbConn->execute($sql);
67.         }
```

En este caso, la aplicación no utiliza el driver PDO para MYSQL, sino que utiliza una clase personalizada que implementa una interfaz similar a ADO y ODBC mediante las funciones que PHP tiene para interactuar con MYSQL.

La clase de la base de datos que implementa el driver ODBC se muestra a continuación

Connclass.php - dependencies/conclass/Connclass.php

```
1. <?php
2. class Myvaluerecordset{
3.         var $result;
4.         var $value;
5. }
6.
7. class Recordset{
8.         var $result;
9.         var $row;
10.         var $contador=0;
11.         var $EOF=FALSE;
12.         var $registros;
13.
14.         function moveNext(){
15.                 if($this->contador < $this->registros )
16.                 {
17.                         $this->contador++;
18.                 }
19.                 $this->row=mysql_fetch_object($this-
   >result);
20.                 $this->EOF=($this->row == NULL);
21.         }
22.
23.         function moveLast(){
24.                 if ($this->contador)
25.                 {
26.                         $this->contador--;
27.                 }
```

```
28.                 mysql_data_seek($this->result,$this-
   >contador);
29.                 $this->row=mysql_fetch_object($this-
   >result);
30.                 $this->EOF=($this->row == NULL);
31.         }
32.
33.         function move($registros){
34.                 $this->contador=registros;
35.                 mysql_data_seek($this->result,$registros);
36.                 $this->row=mysql_fetch_object($this-
   >result);
37.                 $this->EOF=($this->row == NULL);
38.         }
39.
40.         function fields($field){
41.                 $aux=new Myvaluerecordset();
42.                 $aux->result=$this->result;
43.                 $aux->value=$this->row->$field;
44.                 $this->EOF=($this->row == NULL);
45.                 return $aux;
46.         }
47.
48.         function close() {
49.                 mysql_free_result($this->result);
50.         }
51.
52.         function recordCount(){
53.                 return $this->registros;
54.         }
55. }
56.
57. class MyCon{
58.
59.         var $Conexion_ID = 0;
60.         var $contador=0;
61.
62.
63.         function connect($hostname, $username,
   $password,$dbname){
64.                 $this-
   >Conexion_ID=mysql_connect($hostname,$username,$password);
65.                 mysql_select_db($dbname);
66.         }
67.
68.         function execute($instruccion){
69.                 $rs = new Recordset();
70.                 //$instruccion =
   mysql_real_escape_string($instruccion);
```

```
71.                 $rs->result = mysql_query($instruccion);
72.                 if ($instruccion[0]=='S' ||
    $instruccion[0]=='s'){ // SI ES UN SELECT
73.                         $rs->row=mysql_fetch_object($rs-
    >result);
74.                         $rs->registros=mysql_num_rows($rs-
    >result);
75.                         $rs->EOF=($rs->row == NULL);
76.                 }
77.                 return $rs;
78.         }
79.
80.         function close() {
81.                 mysql_close($this->Conexion_ID);
82.         }
83.
84. }
85. ?>
```

Esta aplicación tiene un middleware que actúa de la misma manera que se ha descrito a lo largo de este documento. No obstante como ya se ha comentado, la conexión no se realiza mediante el driver PDO de MYSQL, sino con el la interfaz recién explicada. La clase cuenta con los métodos moveNext(), moveLast(), moveFirst() etc para moverse entre los registros que devuelve la base de datos. Otros sistemas se limitan a devolver un array asociativo y se itera sobre ellos con un bucle for o foreach.

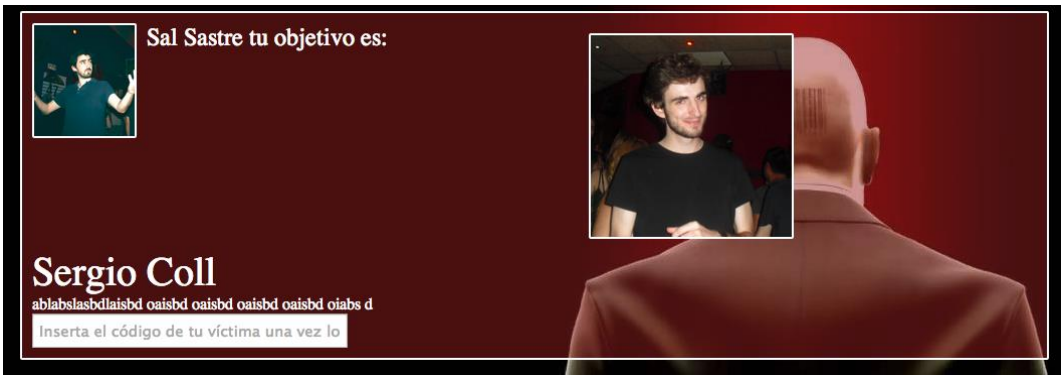
4.b Aplicación Killer

Killer: The Game of Assassination (traducido oficialmente en castellano como Killer, juego de rol en vivo) es un juego de rol en vivo en el que los personajes interpretan a personajes que tienen que eliminarse entre ellos de uno en uno. La partida de rol en vivo empieza una vez que se haya fijado una fecha de inicio. A partir de esa fecha los jugadores se abandonan a sus ocupaciones diarias, con la diferencia de que en cualquier momento otro jugador del juego puede matarles con uno de los objetos establecidos previamente como un arma (una pinza de tender la ropa, una pistola de agua, una cuchara etc). En realidad cualquier objeto inofensivo puede bastar para que el director de juego acepte que uno de los jugadores cuente como asesinado, siempre y cuando dicho objeto haya sido consensuado antes del inicio de la partida. La partida avanza a medida que los jugadores se van encontrando en la ciudad en la que viven (aunque si lo desean pueden buscarse los unos a los otros para ir “asesinándose” progresivamente. El juego termina cuando solo queda uno en pie. El juego del asesino es, pues, un juego concebido para aportar humor y diversión a la rutina cotidiana.

El sistema permite que el director de juego invite a los jugadores a la partida, y si estos aceptan, se les entrega una clave que deben guardar y memorizar para averiguar cuales irán siendo sus objetivos.

Captura del registro de la aplicación Killer

Una vez se han registrado y comienza la partida, los personajes se identifican en el sistema y el sistema les indica su objetivo



Capturas de pantalla de la aplicación Killer

De esta manera cuando ocurre una muerte , la “víctima” le da a su “asesino” su clave para que este la introduzca en el sistema y de esta manera se anota su “asesinato” y se le asigna una nueva víctima (la que tenía la víctima asignada).

Se mostrará ahora el código del middleware que realiza este cambio en el sistema de gestión de la base de datos

DatabaseHandler class - views/main/handlers/DatabaseHandler.php

```
1.      public function kill($code)
2.       {
3.               global $Session;
4.
5.               //$id es el asesino.
6.               $id = $Session->get('id');
7.               $stmt = $this->DBODB->prepare("SELECT id
   FROM users WHERE code=?");
8.
9.               $stmt->execute(array($code));
10.               $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
11.               if ( $stmt->rowCount() != 1)
12.               {
13.                       //si el código no existe
14.                       $stmt = $this->DBODB-
   >prepare("UPDATE users SET count=count +1 WHERE id=?");
15.                       $stmt->execute(array($id));
16.                       return -1;
17.               }
18.
19.               $stmt = $this->DBODB->prepare("SELECT id
   FROM asesina WHERE victima=?");
20.               $stmt->execute(array($rows[0]['id']));
21.               $rows2 = $stmt->fetchAll(PDO::FETCH_ASSOC);
22.
23.               if ( $rows2[0]['id'] != $id)
24.               {
25.                       //o pones el que no toca...
```

```
26.                         $stmt = $this->DBODB-
   >prepare("UPDATE users SET count=count +1 WHERE id=?");
27.                         $stmt->execute(array($id));
28.                         return -1;
29.                 }
30.
31.                 $stmt = $this->DBODB->prepare("SELECT
   victima FROM asesina WHERE id=?");
32.                 $stmt->execute(array($rows[0]['id']));
33.                 $rows3 = $stmt->fetchAll(PDO::FETCH_ASSOC);
34.
35.                 $stmt = $this->DBODB->prepare("UPDATE
   asesina SET killedby=? WHERE id=?");
36.                 $stmt->execute(array($id,$rows[0]['id']));
37.                 $stmt = $this->DBODB->prepare("UPDATE
   asesina SET victima=? WHERE id=?");
38.                 $stmt-
   >execute(array($rows3[0]['victima'],$id));
39.
40.                 return $this->getVictim($id);
41.         }
```

Podemos ver cómo a partir del código de la víctima, se comprueba si la víctima era la correcta (la que el jugador debía eliminar) y que el código existe. A continuación se procede a “eliminar” a la víctima y asignarle al asesino la víctima de su víctima.

5. Visión de conjunto

Estas dos aplicaciones muestran la versatilidad del sistema. Un desarrollador puede aprender a utilizar el software en apenas una hora y comenzar de forma rápida el desarrollo de su aplicación, olvidándose de los problemas que pueden dar las instalaciones de frameworks más complejos que a menudo incluyen muchas herramientas que probablemente el desarrollador no vaya a necesitar utilizar en su aplicación.

Con otros frameworks, las necesidades del sistema a la hora de instalarlos pueden ser bastante elevadas y complejas de instalar y configurar. Con ROWDS solamente se necesita PHP >= 5.3 (y el driver PDO para mysql si se va a utilizar PDO) , que es la configuración más común en la mayoría de servidores web basados en sistemas UNIX hoy en día. Cualquier sistema XAMP, MAMP o LAMP servirá para que la aplicación funcione y su equipo comience a desarrollar la aplicación.

5.1 El futuro – R.O.W.D.S en GitHub

Para que el proyecto crezca se ha decidido emplear como plataforma de difusión y desarrollo GitHub.

GitHub es una forja para alojar proyectos utilizando el sistema de control de versiones Git. El código de almacena de forma pública para que la comunidad de desarrolladores la difunda, corrija errores, actualice y realice mejoras y sugerencias en el sistema. Actualmente se puede visitar el proyecto en GitHub en la siguiente dirección web https://github.com/troindx/rowds