UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Web social para la gestión de actividades deportivas

Proyecto Final de Carrera
Ingeniería Informática

Autor: Juan Raúl Garcia Canet

Director: José Vicente Busquets Mataix

Septiembre 2015

Resumen

El presente documento describe la memoria del proyecto fin de carrera consistente en una aplicación web social para gestionar actividades deportivas.
El proyecto se ha enfocado a proporcionar a los usuarios, una plataforma donde poder crear espacios web personalizados enfocados a actividades deportivas o deportes, que sirva de punto de encuentro para toda la gente que comparta afición por ese deporte. Posibilitando la creación y gestión de grupos que organicen eventos y/o actividades.
Desde el punto de vista técnico se ha usado la última versión de CakePHP, es decir, la versión 3 y como sistema de gestión de bases de datos se ha optado por MySQL 5.6.
Los diferentes apartados del documento corresponden a cada una de las fases del ciclo de vida de desarrollo de un diseño, implementación y evaluación

Palabras clave: PFC, CakePHP, PHP, web social, deporte, actividades.

Tabla de contenido

1 Introducción 7
2 Especificación de requisitos software 9
2.1 Introducción 9
2.2 Descripción General 12
2.3 Requerimientos específicos 13
3 Análisis 15
3.1 Casos de uso 15
3.2 Descripción de los Casos de uso 16
4 Diseño 21
4.1 Diagrama de clases 21
4.2 Arquitectura MVC (Modelo, Vista, Controlador) 22
4.3 Diagrama de Actividad modelo MVC 23
4.4 Diagrama Actividad del Registro de usuario 23
4.5 Diagrama de Actividad de Login de usuario 24
4.6 Diagrama de Actividad de Creación de espacio 24
5 Implementación 25
5.1 Tecnologías 25
5.2 Herramientas 26
5.3 Detalles de implementación 27
5.4 El prototipo 31
5.5 Instalación del entorno de Desarrollo 41
5.6 Configuración del entorno de Desarrollo 43
5.7 Creación de las tablas de la bases de datos para SPORTSNET 57
5.8 Creación de los Modelos, Controladores y Vistas para SPORTSNET 61
5.9 Un sistema de login para nuestra aplicación 70
5.10 La página de entrada de SPORTSNET 76
5.11 El Sistema de enrutado (routing) de CakePHP. 76
5.12 Implementación de los casos de uso 77
6 Evaluación y Pruebas 115
6.1 Test Unitarios 115
6.2 Fixtures 117
6.3 Generando Tests 117
6.4 Errores durante los tests 121
6.5 Consideraciones finales 121
7 Conclusiones 122
8 Bibliografía y enlaces 124

Tabla de imágenes

Imagen 1: Diagrama CU-1 16
Imagen 2: Diagrama CU-2 18
Imagen 3: Diagrama CU-5 19
Imagen 4: Diagrama CU-16 20
Imagen 5: Diagrama de clases 21
Imagen 6: Flujo de una petición MVC 22
Imagen 7: Diagrama de actividad MVC 23
Imagen 8: Diagrama actividad registro usuario 23
Imagen 9: Diagrama de Actividad de Login de usuario 24
Imagen 10: Diagrama de creación de espacio 24
Imagen 11: Configuración de la BD 27
Imagen 12: Configuración BD CakePHP2 28
Imagen 13: Instalación de la consola bake 29
Imagen 14: Instalación de DebugKit 29
Imagen 15: Instalación readline 29
Imagen 16: Instalación de DebugKit 29
Imagen 17: Instalar pdo_sqlite 30
Imagen 18: Configuración DB DebugKit 30
Imagen 19: Página principal 33
Imagen 20: Lista de espacios: Una vez autentificado el usuario será redirigido a esta página 34
Imagen 21: Página interna - Vista de uno de los "Espacios" del usuario 35
Imagen 22: Vista de uno de los espacios del usuario con respuestas en los comentarios 36
Imagen 23: Vista del Formulario de contacto del usuario autentificado 37
Imagen 24: Vista del Formulario de contacto del usuario NO autentificado 37
Imagen 25: Vista del perfil del usuario 38
Imagen 26: estructura del directorio del prototipo 39
Imagen 27: Fichero hosts 43
Imagen 28: Pantalla principal de phpMyAdmin 46
Imagen 29: Creación de la BD 46
Imagen 30: Error al ejecutar composer 47
Imagen 31: Directorios CakePHP 48
Imagen 32: Estructura de directorios CakePHP 2 49
Imagen 33: Listado de Módulos de Apache 51
Imagen 34: Habilitar módulos en Apache 51
Imagen 35: Directorio para compass 52
Imagen 36: Ejecución de compass 54
Imagen 37: htaccess directorio principal CakePHP 55
Imagen 38: htaccess directorio webroot CakePHP 55
Imagen 39: Página principal de CakePHP 56
Imagen 40: Icono de DebugKit 57
Imagen 41: Barra de herramientas de DebugKit 57
Imagen 42: Relación "muchoas a muchoes" 57
Imagen 43: Relación "a muchos" 58
Imagen 45: Ventana SQL en phpMyAdmin 59
Imagen 46: Consola bake 64
Imagen 47: bake model 64
Imagen 48: Creación del modelo para Anuncios 65
Imagen 49: Creación de controladores 66
Imagen 50: Crear todos los objetos a la vez 67
Imagen 51: Ficheros creados por bake 68
Imagen 52: Gestion de Espacios 69
Imagen 53: Nuevo usuario. 69
Imagen 54: Activando y configurando controlador de autenticación 70

Imagen 56: Configuración AuthComponent por defecto 70
Imagen 56: Nuestra configuración AuthComponent 70
Imagen 57: Permisos sobre acciones 71
Imagen 58: Login/logout....71
Imagen 59: Hashing passwords....72
Imagen 60: Hashing passwords CakePHP 2.x 73
Imagen 61: Formulario de login 73
Imagen 62: Formulario de Login 74
Imagen 63: Error tabla no encontrada 74
Imagen 64: Indicar a Auth que tabla usar para la auteticación 75
Imagen 65: Routing por defecto 76
Imagen 66: Definir nuestra propia página de inicio. 77
Imagen 67: CU-1 - Controlador de login/logout 78
Imagen 68: CU-1 - Vista de la página principal - Parte 1 80
Imagen 69: CU-1 - Vista de la página principal - Parte 2 81
Imagen 70: CU-1 - Elemento "login" 82
Imagen 71: CU-1 - Fichero SCSS página principal....84
Imagen 72: CU-2 - Controlador añadir usuario....85
Imagen 73: CU-2 - Vista añadir Usuario....87
Imagen 74: CU-2 - SCSS formulario Registro 88
Imagen 75: CU-3 - Controlador de edición del perfil de usuario 89
Imagen 76: CU-3 - Vista de edición del Perfil....91
Imagen 77: CU-3 - SCSS edición del perfil del usuario 92
Imagen 78: CU-4 - Controlador de espacios 93
Imagen 79: CU-4 - Controlador de espacios - SQL 94
Imagen 80: CU-4 - Vista de espacios 95
Imagen 81: CU-4 - Ficheo SCSS....96
Imagen 82: CU-16 - Controlador de Vista espacio 97
Imagen 83: CU-16 Vista Espacio - Datos del Espacio 99
Imagen 84: CU-16 Vista Espacio - Datos de los comentarios 100
Imagen 85: CU-16 Vista Espacio - Respuestas a los comentarios....101
Imagen 86: CU-16 Vista espacio - Datos Anuncios....102
Imagen 87: CU-16 Vista espacio - Respuestas a los anuncios 103
Imagen 88: CU-16 Vista espacio - Eventos 104
Imagen 89: CU-16 Vista espacio - Miembros....105
Imagen 90: CU-16 Sidebar 106
Imagen 91: CU-16 SCSS - Sección Espacio y comentarios 108
Imagen 92: CU-16 SCSS - Sección Respuestas a los comentarios 108
Imagen 93: CU-16 SCSS - Sección Anuncios y respuestas....109
Imagen 94: CU-16 SCSS - Sección Eventos y miembros 110
Imagen 95: CU-16 Fichero JS....111
Imagen 96: CU-5 - Controlador nuevo comentario....112
Imagen 97 configuración BD testing: 116
Imagen 98: TestCase - función básica setUp() 117
Imagen 99: Generación de tests unitarios....118
Imagen 100: Generación de fixtures 118
Imagen 101: resultado test unitario....119
Imagen 102: Testcase generado por CakePHP 120

1 Introducción

Facebook, Google+, Twitter, etc. son ejemplos de redes sociales dónde diariamente interactuamos con otras personas, compartimos vivencias, materiales y todo tipo de información. Todas son de propósito general y en cualquiera de ellas se puede conseguir información de todo tipo.
Lo que se pretende en este proyecto es la realización de una aplicación web enfocada a las actividades deportivas y que posea características de las redes sociales, como la posibilidad de poner en contacto usuarios, la posibilidad de organizar eventos y, en resumen, proporcionar un espacio localizado dónde los usuarios con las mismos inquietudes deportivas puedan encontrarse y compartir todo lo referente a dichas actividades. No se trata en sí de una red social completa, pero podría considerarse una base para la creación de una futura red social dedicada a las actividades deportivas.

Dado que se trata de una aplicación web social enfocada a los deportes, la llamaremos SPORTSNET y su cometido principal será reunir usuarios, sean deportistas profesionales o no, entorno a un deporte o actividad deportiva permitiéndoles desde la simple compartición de información hasta la creación y gestión de eventos deportivos en los que usuarios podrán tomar parte.
Además de esto, en las actividades deportivas, se utilizan ciertos tipos de instrumentos u objetos destinados a la práctica de este deporte. Ofreceremos la posibilidad de que los usuarios puedan realizar pequeñas ventas/intercambios de materiales u objetos a través de una sección de "Segunda mano".

Esta sección, sin embargo, se limitará únicamente a permitir la publicación de anuncios y a poner en contacto vendedores y compradores. Pero no permitirá, al menos en esta primera versión, la gestión de pagos. Es decir, quedarán excluidas las funciones específicas de una aplicación E-Commerce, como por ejemplo, gestión de medios de pago, pasarelas de pago, etc. dado que no es la finalidad última de esta aplicación. El carácter social de la web, impone además, que los contenidos puedan ser compartidos en otras redes sociales. Por tanto, se dispondrá de dichas opciones en cualquier sitio susceptible de poder ser compartido.

En cuanto al diseño, deberemos tener en cuenta el uso de la web desde cualquier dispositivo y con cualquier resolución. Esto nos lleva a optar por un Diseño adaptable o adaptativo (RWD[1])

En lo referente a las tecnologías utilizadas, se ha la nueva versión 3 de CakePHP por los siguientes motivos:

- Obtener una visión general de las diferencias entre las versiones 2 y 3 del Framework de desarrollo CakePHP.
- Establecer un precedente de desarrollo para futuros proyectos que utilicen el mismo framework que se usa en este proyecto. En este caso CakePHP 3.
- Implícitamente se consigue, con estos objetivos, que este proyecto sirva de guía para una posible migración de proyectos realizados en CakePHP 2. Al menos, hasta que las herramientas de migración ofrecidas para este fin, alcancen la madurez suficiente para realizar este proceso con garantías.

El motor de base de datos MySQL además de ser gratuito es uno de los más extendidos en los servidores web comerciales, con lo no nos será difícil alojar la web en alguno de ellos.

[1] El **diseño web adaptable** o **adaptativo**, conocido por las siglas **RWD** (del inglés, Responsive Web Design) es una filosofía de diseño y desarrollo cuyo objetivo es adaptar la apariencia de las páginas web al dispositivo que se esté utilizando para visualizarla. (fuente: Wikipedia)

Al igual que MySQL, PHP y Apache son tecnologías que se encuentran disponibles por defecto en todos los proveedores de alojamiento web.
A continuación se puede ver descripción general del entorno y las herramientas utilizadas para el desarrollo de la aplicación.

Sistema Operativo	Linux – Kubuntu 14.04
Servidor Web	Apache/2.4.7
Servidor Bases de Datos	MySQL 5.5.41
Control de la BBDD	PHPMyAdmin – 4.0.10
Modulos PHP:	mbstring, curl, xdebug, pdo, gd, intl, mcrypt, pdo_mysql, xdebug
Debugger	XDebug 2.2.3 (módulo xdebug de php)
Entorno de Desarrollo (IDE)	Netbeans 8.0.2
Framework de Desarrollo	CakePHP V3
Lenguaje de Backend	PHP 5.5.9
Lenguajes de Frontend	HTML5, CSS3 (SASS), JQuery 2.1.3
Framework Frontend	Compass 1.0.1
Otras herramientas:	Composer 1.0-dev, Firebug 2.0.8, The easiest Xdebug 2.0.1

Tabla 1: Herramientas y módulos de desarrollo

En el capítulo 5 puede encontrar una descripción más detallada de la utilidad de estos módulos, así como las configuraciones específicas, que en su caso, se hayan hecho para este proyecto, tanto de estos módulos como del resto de herramientas.

2 Especificación de requisitos software

2.1 Introducción

En este apartado definiremos los requisitos que nuestra aplicación deberá cumplir, de forma que cualquier usuario pueda saber qué hace la aplicación y cómo funciona. Además estaremos en condiciones de establecer un diseño que se ajuste a los requerimientos aquí expuestos.

Ámbito

La aplicación web a construir está dirigida a cualquier persona interesada en practicar algún tipo de deporte y que deseen encontrar a gente que comparta sus aficiones y con la que pueda reunirse para practicar dicho deporte y compartir experiencias y conocimientos sobre el mismo.

En principio todos los usuarios de nuestra aplicación deberán tener una cuenta propia y por tanto no existirán, en este caso, usuarios anónimos.

Cada usuario registrado dispondrá de un espacio propio dónde podrá consultar todos los datos referentes a su propia cuenta. Entre otros datos, podrá ver los las publicaciones que haya hecho, los eventos a los que está apuntado, los procesos de compra/venta en los que está participando y por supuesto las opciones necesarias para crear y gestionar todos estos datos.

Como usuario registrado podrá crear “Espacios propios” (en adelante Espacio) que se dedicaran a un deporte concreto. El usuario que crea el espacio será automáticamente administrador del mismo y será el encargado de gestionarlo y gestionar sus contenidos. Únicamente el creador del Espacio podrá nombrar como administradores a otros usuarios miembros del mismo.

Los contenidos de los espacios serán visibles sólo por sus miembros. Por este motivo, para que un usuario pueda ver el contenido de un espacio, éste deberá solicitar a un administrador la pertenencia a ese espacio. El administrador podrá aprobar o denegar dicha solicitud.

Para que los usuarios eviten crear Áreas duplicadas, se proporcionarán opciones de búsqueda avanzada y diferentes tipos de listados de los contenidos existentes en la web.

Definiciones y Acrónimos

Definiciones

- Actividad deportiva: deporte

- **Usuario Registrado:** Cualquier usuario de la web que posea una cuenta y acceda usando un nombre de usuario y una contraseña.

- **Usuario Administrador:** Cualquier usuario que son capaces de gestionar a todos los usuarios y en general la propia aplicación.

- **Espacio deportivo:** Espacio destinado a difundir información sobre una actividad deportiva o deporte. Un usuario registrado puede crear tantas áreas deportivas como desee y será automáticamente administrador de las mismas.

- **Framework:** En terminología informática definimos framework como un conjunto de librerías que proporcionan ayuda para facilitar el trabajo de desarrollo, posibilitando que el desarrollador se centre en su aplicación y deje el resto al framework.

- Por ejemplo: el uso de ORM en CakePHP evita que el desarrollador tenga que construir sus propias funciones de acceso a datos con todo lo que ello conlleva (comprobación de la corrección de los datos, etc.). Ese trabajo se deja a la libraría. El desarrollador sólo tiene que usar las funciones apropiadas para su trabajo.

- **Publicar anuncio:** Acción realizada por el usuario que pone a disposición de los demás usuarios un anuncio.

- **Despublicar anuncio:** Acción realizada por el usuario que oculta un anuncio a los demás usuarios.

- **Evento:** Reunión organizada en una fecha y en un lugar a la que los usuarios y/o grupos pueden inscribirse como asistentes y, por supuesto, asistir.

- **CakePHP:** Framework para el desarrollo aplicaciones web escrito en PHP, creado sobre los conceptos de Ruby on Rails.

Acrónimos

- **COMPASS:** Es un framework open-source para CSS que utiliza el lenguaje SASS.

- **CSS:** Cascading Style Sheets. Hojas de estilo, se usan para definir la presentación de un documento estructurado.

- **DNS**: Acrónimo de Domain Name Server o Servidor de Nombres de Dominio. Es un servidor que se encarga d erealizar las traducciones entre direcciones IP y direcciones web. Por ejemplo, cuando el DNS recibe www.google.es lo traduce a la dirección IP 74.125.133.94[2]

- **HTML:** HyperText Markup Language: Lenguaje de marcad para la elaboración de páginas web.

- **ORM:** Object-Raltional mapping. En espanol **mapeo objeto-relaciona**, es una técnica de programación que permite convertir datos entre sistemas de tipos. En la práctica consiste en crear una Base de Datos Orientada a Objetos virtual.

- Es decir, las tablas, relaciones y otros elementos de la base de datos se representan con clases, atributos y funciones en la aplicación. La librería ORM, se encargará pues de manejar la conversión entre objetos de la aplicación y las tablas/relaciones de la base de datos.

- **PHP:** PHP Hypertext Pre-processor. Es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.

- **SASS:** Syntactically Awesome Style Sheets. Es un metalenguaje de Hojas de Estilo en Cascada (CSS). Es un lenguaje de script que es traducido a CSS. SassScript es el lenguaje de script en sí mismo.

[2] Este IP de Google no es única ya que Google tiene muchas direcciones IP en sus servidores. Esta, en concreto, se ha obtenido desde el ordenador con el que se ha escrito este documento mediante el comando "ping www.google.es". Probablemente el lector de este documento obtenga otra dirección IP usando el mismo comando.

2.2 Descripción General

Perspectiva del Producto

Se trata de una aplicación web. Por tanto, siempre que el dispositivo disponga de un navegador y conexión a Internet, la aplicación funcionará correctamente.

Funciones del Producto

Desde la perspectiva del usuario podemos definir las siguientes funciones que la aplicación podrá realizar.

- Usuarios registrados
    - Gestión de datos personales.
    - Gestión de áreas propias.
    - Gestión de eventos
    - Gestión de mensajes propios en las áreas propias y ajenas.
    - Gestión de operaciones de compra-venta.
    - Solicitud de acceso a un área de otro usuario.

- Administradores: Dado que cada usuario que crea un espacio será automáticamente designado como creador/administrador de ese espacio. Estos usuarios administradores hacen referencia a los Administradores de la aplicación y no de un espacio en concreto. Sus acciones comprenden:
    - Lo mismo que un usuario registrado.
    - Gestionar a todos los usuarios y todo lo relacionado con ellos.
    - Gestionar la aplicación.

Características del Usuario y Obligaciones Generales

Como ya se ha indicado tendremos en el sistema, dos tipos de usuarios:

- Usuarios registrados: dado que cualquier usuario podrá ser administrador de su propio espacio, éstos deberán tener conocimientos de usuario avanzado en el uso de Internet. Deberán controlar en todo momento aquello que ocurra en su espacio y en sus grupos y tomar las decisiones oportunas para mantener el orden y el decoro en los mismos.

- Administradores: Estos usuarios, no sólo controlaran a todos los demás, sino que gestionaran la propia aplicación, con lo que deberán tener una visión global del funcionamiento de la aplicación. Además, puesto que serán capaces de gestionar usuarios, bloquearlos, etc. deberán contar con cierto grado de independencia, siendo objetivos en las decisiones que afecten a otros usuarios.

2.3 Requerimientos específicos

Requerimientos de interfaces externos

Interfaces de usuario

Podemos considerar interfaces externas, aquellos procesos que permitan al usuario interaccionar con la aplicación. Así pues se considerarán interfaces de usuarios desde los enlaces que permitan acceder a diferentes partes de la aplicación hasta los dispositivos de entrada habituales: teclado, ratón y/o pantallas táctiles en los dispositivos que las posean.

Interfaces hardware

No se requieren interfaces hardware más allá de las que se encuentran en cualquier equipo que sea capaz de acceder a Internet.

Interfaces software

Al ser una aplicación web, la única interfaz software requerida es un navegador web.

Restricciones de diseño

Estándares cumplidos

El producto cumple el estándar W3C.org, y utilización de estilos y colores atractivos para garantizar la atención de los usuarios sobre el sitio.
La notación, definiciones y acrónimos utilizados son los estándares para los portales Web, así como el lenguaje utilizado es el castellano.

Limitaciones hardware

Las limitaciones hardware en el lado del servidor de la aplicación se limitan a las restricciones que sufre cualquier sistema Windows, Linux, Android o iOS.
Desde el lado del cliente las limitaciones son menores, reduciéndose tan solo al mínimo hardware necesario para ofrecer navegación por Web.

Limitaciones Software

Dado que se quiere proporcionar la mejor experiencia de usuario posible, en el desde el punto de vista de la accesibilidad y usabilidad web, se usaran tecnologías de última generación que pueden no estar soportadas en navegadores antiguos. Por ello y para evitar problemas futuros, establecemos como requisito mínimo recomendado, el uso de nuestra aplicación con las últimas versiones de los navegadores actuales.

Atributos

Seguridad

Se trata de una aplicación web que maneja datos de usuario con diferentes niveles de privilegios. La aplicación deberá controlar el acceso y las acciones que el usuario puede hacer en cada área según los privilegios del usuario.
Además, dado que los usuarios acceden con un nombre de usuario y un password, éste deberá almacenarse mediante una función de cifrado de vía única. Es decir, que no pueda descifrarse nuevamente o que las técnicas utilizadas para romper el cifrado tengan un coste temporal inasumible por los computadores actuales.

Mantenimiento

Aparte del mantenimiento habitual de cualquier aplicación para la corrección de fallos, no será necesario realizar ningún otro tipo de mantenimiento.
Caso aparte puede considerarse el mantenimiento para agregar nuevas funcionalidades.

3 Análisis

3.1 Casos de uso

REF.	DESCRIPCIÓN	REALIZADO*
CU-1	Como usuario quiero ver una página principal desde la que poder loguearme, acceder a la página de registro o a la página de recordatorio de contraseña.	100%
CU-2	Como Usuario no registrado quiero tener la opción de poder registrarme en la página web con un proceso sencillo.	100%
CU-20	Como Administrador de la web quiero poder gestionar todos los espacios de la misma.	100%
CU-3	Como Usuario registrado quiero poder gestionar mis datos personales a través de una interfaz web amigable.	100%
CU-4	Como Usuario registrado quiero poder gestionar mi/s espacio/s web de una forma sencilla e intuitiva.	100%
CU-5	Como Usuario registrado quiero poder publicar comentarios.	100%
CU-6	Como Usuario registrado quiero poder gestionar mis comentarios.	100%
CU-7	Como Usuario registrado quiero poder crear anuncios de compra-venta.	80%
CU-8	Como Usuario registrado quiero poder gestionar mis anuncios de compra-venta.	80%
CU-9	Como Usuario registrado quiero poder responder a anuncios de compra-venta.	0%
CU-10	Como Usuario registrado quiero poder gestionar mis respuestas a anuncios de compra-venta.	0%
CU-11	Como Usuario registrado quiero poder crear eventos.	50%
CU-12	Como Usuario registrado quiero poder gestionar mis eventos.	50%
CU-13	**Como Usuario registrado quiero poder solicitar ser admitido como miembro en el espacio de otros usuarios.**	0%
CU-14	Como Administrador de mi espacio quiero poder gestionar las solicitudes de admisión hechas por otros usuarios.	0%
CU-15	**Como Administrador de mi espacio quiero poder ver el espacio con todos los datos relacionados**.	100%
CU-16	Como Administrador de mi espacio quiero poder gestionar los usuarios de mi espacio.	100%
CU-17	Como Administrador de mi espacio quiero poder gestionar todos los comentarios de mi espacio,	80%
CU-18	Como Administrador de mi espacio quiero poder gestionar todos los anuncios de compra-venta de mi espacio.	80%
CU-19	Como Administrador de mi espacio quiero poder gestionar todas las respuestas a anuncios de compra-venta de mi espacio.	80%
CU-20	Como Administrador de mi espacio quiero poder gestionar todos los eventos de mi espacio.	50%
CU-21	Como Administrador de la web quiero poder gestionar todos los espacios de la misma.	100%
CU-22	Como Administrador de la web quiero poder gestionar todos los usuarios de la misma.	100%

Tabla 1: Listado Casos de Uso

Nota*: La columna %realizado comprende tanto funcionalidad como diseño, siendo éste último el de menor peso. Así, una celda con un valor del 80%, indica que la funcionalidad está implementada pero el diseño no coincide con el presentado en el apartado “prototipo”. Es decir que no se ha convertido el código del prototipo en código CakePHP.

3.2 Descripción de los Casos de uso

A continuación representaremos algún caso de uso en forma gráfica, de manera que podamos observar un poco de la funcionalidad de la aplicación.

CU-1: Como usuario no registrado quiero ver una página principal desde la que poder loguearme, acceder a la página de registro a la página de recordatorio de contraseña.

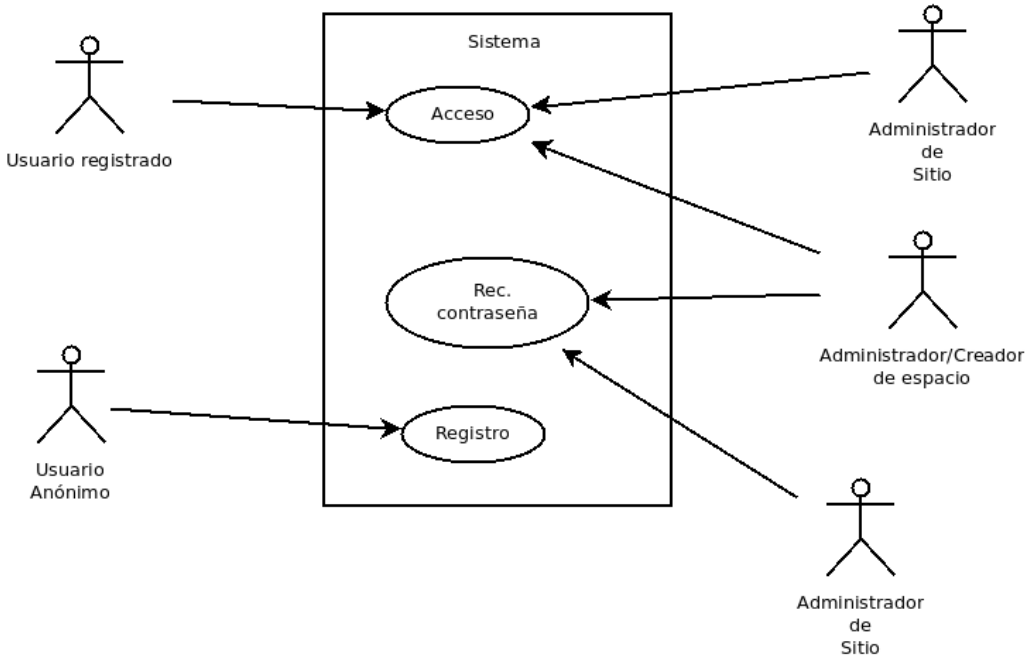


Imagen 1: Diagrama CU-1

RF-1	Login de usuario	
Objetivos asociados	**CU-1**	
Requisitos asociados	--	
Descripción	El sistema deberá comprobar si los datos facilitados por el usuario son válido	
Precondición	El usuario existe en el sistema	
Secuencia Normal	Paso	Acción
	1	El usuario introduce sus datos de acceso en el formulario de acceso
	2	El sistema comprueba si los datos son correctos.
Postcondición	El usuario es validado y accede a su página principal	
Excepciones	Paso	Acción
	4	Si el usuario ha introducido datos incorrectos o inexistentes el sistema le informará del hecho y le permitirá volver a intentarlo
Rendimiento	Paso	Cota de tiempo
	--	--
Frecuencia esperada	--	
Estabilidad	alta	
Comentarios	--	

Tabla 2: Descripción CU-1

RF-2	Recordatorio de contraseña	
Objetivos asociados	**CU-1**	
Requisitos asociados	--	
Descripción	El usuario accederá a un formulario dónde deberá introducir su dirección de correo. El sistema mandará por mail un correo un enlace con un token de un solo uso y con una validez determinada para confirmar la identidad del usuario solicitante. El usuario deberá usar el enlace para verificar la autenticidad de la solicitud. Al final el sistema solicitará al usuario una nueva contraseña.	
Precondición	El usuario existe en el sistema	
Secuencia Normal	Paso	Acción
	1	El usuario introduce su dirección de correo y pulsa "Recordar contraseña".
	2	El sistema comprueba que la dirección proporcionada existe.
	3	El sistema genera un enlace con un token de un solo uso y que tendrá una validez determinada y lo envía por correo al usuario.
	4	El usuario utiliza el enlace que ha recibido por correo para validar la solicitud.
	5	El sistema comprueba que el token sigue siendo válido y solicita al usuario una nueva contraseña
	6	El usuario introduce su nueva contraseña
	7	El sistema comprueba que la nueva contraseña cumple las condiciones de seguridad especificadas.
	8	El sistema almacena la nueva contraseña
	9	El sistema notifica al usuario sobre la operación.
Postcondición	El usuario ha cambiado su contraseña correctamente	
Excepciones	Paso	Acción
	5	Si el token ya no es válido, la operación será cancelada y el usuario será notificada.
	7	Si la contraseña no cumple los requisitos, no se guardará y el usuario será instado a que vuelva a introducir una contraseña valida.
	8	Si el sistema detecta un error en el guardado, los datos no se cambian y el usuario es notificado.
Rendimiento	Paso	Cota de tiempo
	--	--
Frecuencia esperada	--	
Estabilidad	alta	
Comentarios	--	

Tabla 3: Descripción CU-1b

CU-2: Como Usuario no registrado quiero tener la opción de poder registrarme en la página web con un proceso sencillo.

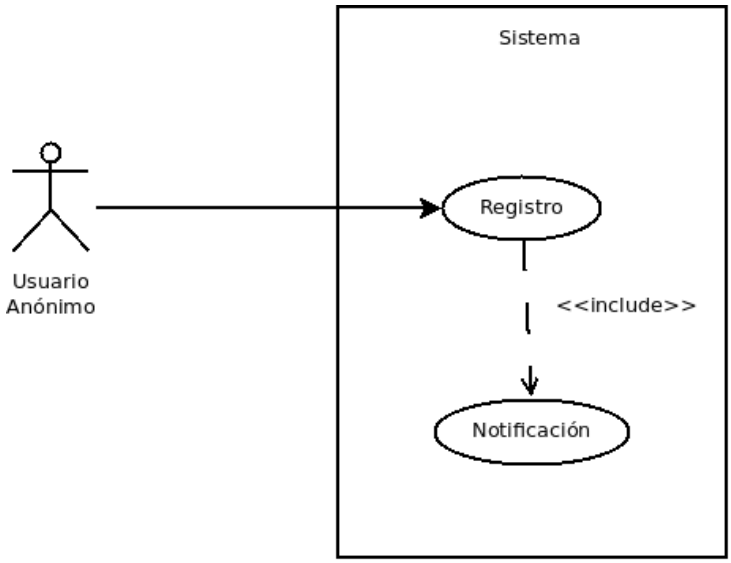


Imagen 2: Diagrama CU-2

RF-3	Registro de usuario	
Objetivos asociados	**CU-2**	
Requisitos asociados	--	
Descripción		
Precondición	El usuario no existe en el sistema	
Secuencia Normal	Paso	Acción
	1	El usuario introduce sus datos de en el formulario de registro
	2	El sistema comprueba si los datos son correctos.
Postcondición	El usuario es validado y accede a su página principal	
Excepciones	Paso	Acción
	4	Si el usuario ha introducido datos incorrectos o inexistentes el sistema le informará del hecho y le permitirá volver a intentarlo
Rendimiento	Paso	Cota de tiempo
	--	--
Frecuencia esperada	--	
Estabilidad	alta	
Comentarios	--	

Tabla 4: Descripción CU-3

CU-5: Como Usuario registrado quiero poder solicitar ser admitido como miembro en el espacio de otros usuarios.

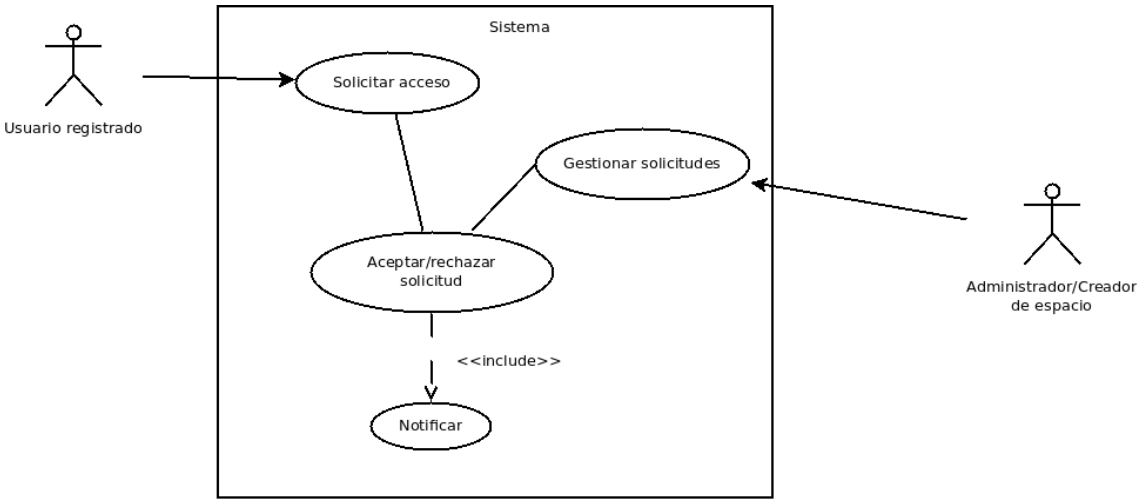


Imagen 3: Diagrama CU-5

RF-3	Solicitud de admisión es un espacio ajeno	
Objetivos asociados	**CU-5**	
Requisitos asociados	--	
Descripción	El usuario solicita ser admitido como miembro de un espacio.	
Precondición	El usuario existe en el sistema	
Secuencia Normal	Paso	Acción
	1	El usuario utiliza la opción disponible en el espacio del que quiere ser miembro “Solicitar admisión”.
	2	El sistema registra la solicitud y envía una notificación a los administradores del espacio.
	3	Un usuario administrador del espacio, acepta la solicitud mediante el botón “Aceptar”.
	4	El usuario aceptado será notificado de su aceptación.
	5	El sistema eliminará la solicitud de admisión.
Postcondición	El usuario es validado y accede a su página principal	
Excepciones	Paso	Acción
	3	Si el usuario administrador pulsa “Rechazar”, la solicitud será eliminada y el usuario será notificado de que su solicitud ha sido rechazada.
Rendimiento	Paso	Cota de tiempo
	--	--
Frecuencia esperada	--	
Estabilidad	alta	
Comentarios	--	

Tabla 5: Descripción CU-5

CU-15: Como Administrador de mi espacio quiero poder gestionar las solicitudes de admisión hechas por otros usuarios.

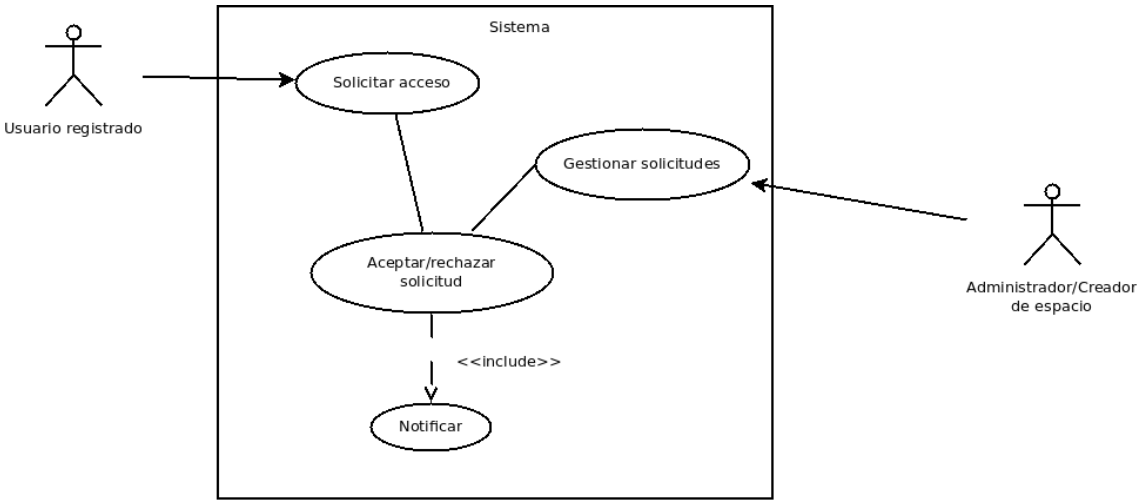


Imagen 4: Diagrama CU-16

RF-4	Gestiona solicitudes de admisión	
Objetivos asociados	**CU-15**	
Requisitos asociados	--	
Descripción	El usuario administrador del espació accede al listado de solicitudes de admisión. Cada solicitud presenta un botón “Aceptar” y un botón “Rechazar”.	
Precondición	El usuario debe ser administrador del espacio o administrador del sitio	
Secuencia Normal	Paso	Acción
	1	El usuario accede al listado de solicitudes a atrvés del enlace correspondiente del espacio.
	2	El sistema muestra un listado de solicitudes pendientes con dos opciones cada una: “Aceptar” y “Rechazar”.
	3	Si el usuario pulsa “Aceptar” la solicitud será eliminada y el usuario será notificado de que puede acceder al espacio. El usuario será registrado como miembro del espacio.
Postcondición	El usuario es admitido y puede acceder al espacio.	
Excepciones	Paso	Acción
	3	Si el usuario administrador pulsa “Rechazar”, la solicitud será eliminada y el usuario será notificado de que su solicitud ha sido rechazada.
Rendimiento	Paso	Cota de tiempo
	--	--
Frecuencia esperada	--	
Estabilidad	alta	
Comentarios	--	

Tabla 6: Descripción CU-16

4 Diseño

4.1 Diagrama de clases

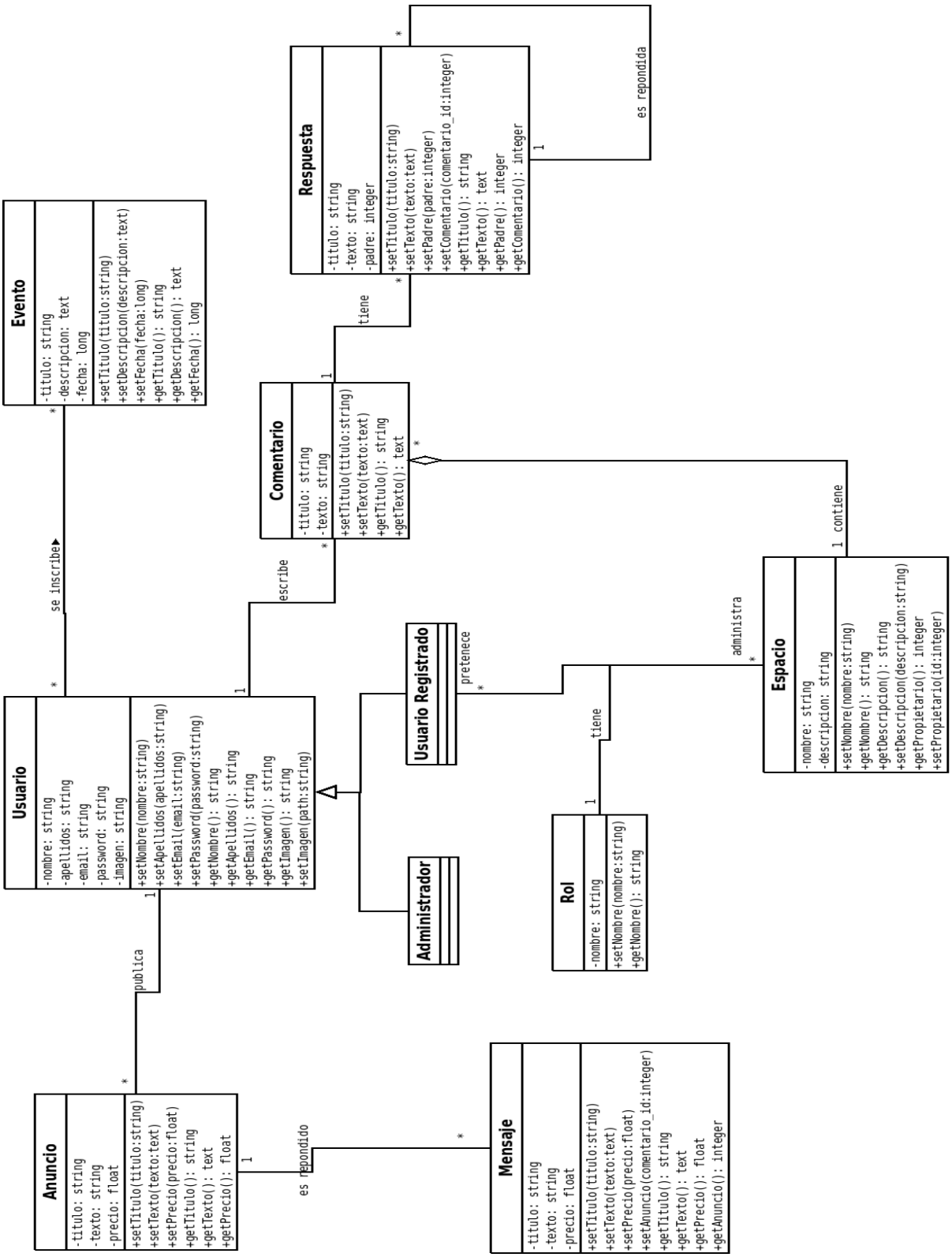


Imagen 5: Diagrama de clases

La clase “Administrador” que hereda de usuario, se ha dibujado como una clase independiente, para diferenciar a los usuarios administradores de la aplicación del resto de usuarios administradores de los espacios.

No significa esto que se almacenaran por separado, sino que simplemente se han dibujado de esta manera para clarificar el diagrama y la comprensión de los diferentes roles de la aplicación.

4.2 Arquitectura MVC (Modelo, Vista, Controlador)

La arquitectura MVC es un patrón de diseño que permite la separación de la lógica de negocio, los datos y la interfaz de usuario.
A continuación definiremos cada una de las parte y veremos cómo interactúan entre ellas, pero sin entrar en demasiados detalles técnicos.

El Modelo

El modelo es una representación textual de la base de datos, es decir un conjunto de clases que representa las tablas y relaciones de la base de datos.
Lo habitual es que cada clase represente una tabla de la Base de Datos y que los atributos de la misma representan a las columnas.
Las operaciones directas sobre la base de datos se representan mediante los métodos de la clase, aunque en este caso, no tiene por qué haber un método para cada operación.
Para obtener el modelo se suele recurrir al diagrama de clases (ver apartado anterior), puesto que éste contiene tanto los atributos como los métodos de los que consta cada clase.

La vista: Layout y plantillas

La vista es la encarga de representar los datos que recibe desde el controlador. Habitualmente se conoce como Layout y en CakePHP se utilizan las plantillas para representar las vistas.

El controlador: Módulos y acciones

El controlador es el que dirige que hacer y dice quién lo hace. Es el encargado de comunicar el modelo con la vista. A través de los métodos del controlador se envían/reciben datos al/del modelo, se procesan y se envían a la vista para que sean representados.

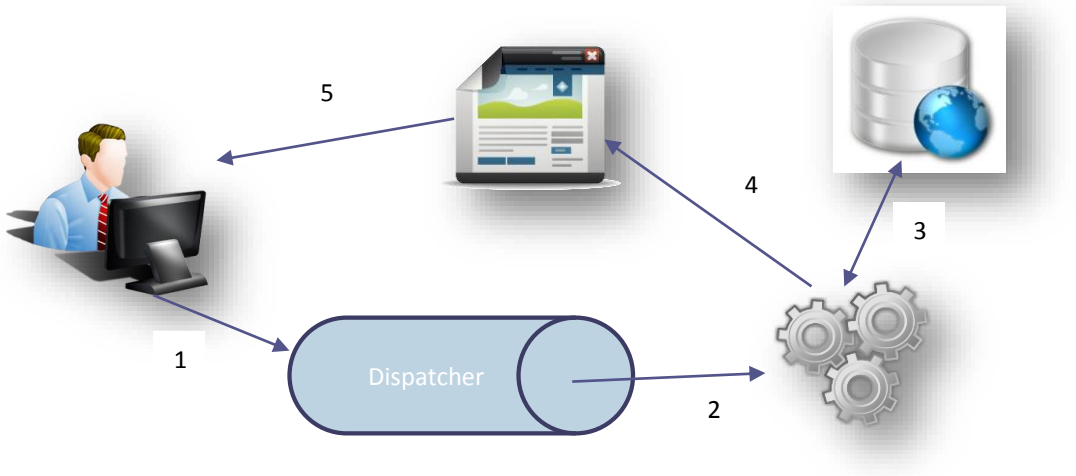


Imagen 6: Flujo de una petición MVC

Como funciona este modelo MVC:

1. El usuario solicita una URL a través del navegador.
2. El dispatcher recoge la petición y selecciona el controlador adecuado y le envía la orden de ejecutarse.
3. El controlador envía la solicitud al modelo y recupera los datos enviados por el modelo.
4. El controlador prepara los datos y los envía a la vista.
5. La vista se devuelve formatead al usuario

4.3 Diagrama de Actividad modelo MVC

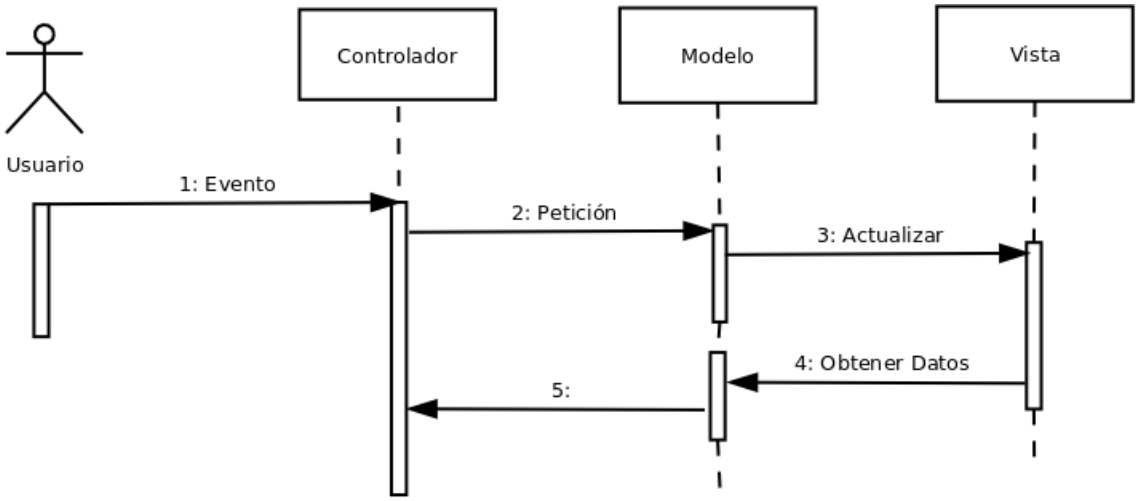


Imagen 7: Diagrama de actividad MVC

Pasos:

1. El usuario introduce el evento.
2. El Controlador recibe el evento y lo traduce en una petición al Modelo (aunque también puede llamar directamente a la vista).
3. El modelo (si es necesario) llama a la vista para su actualización.
4. Para cumplir con la actualización la Vista puede solicitar datos al Modelo.
5. El Controlador recibe el control

4.4 Diagrama Actividad del Registro de usuario

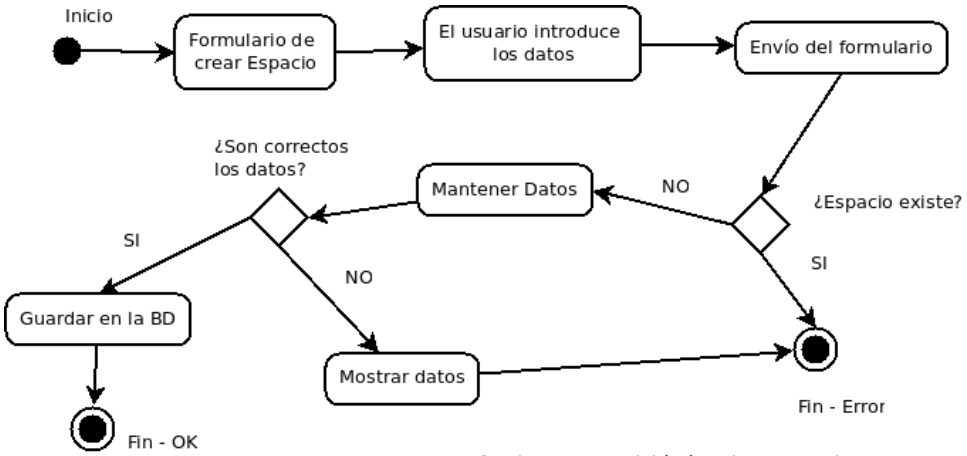


Imagen 8: Diagrama actividad registro usuario

4.5 Diagrama de Actividad de Login de usuario

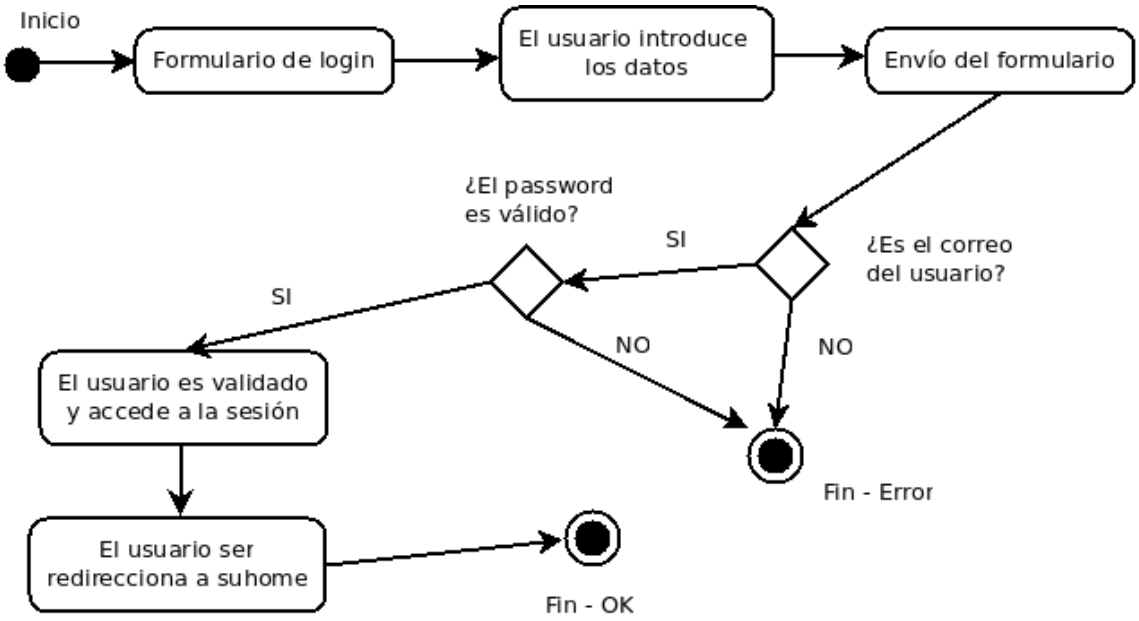


Imagen 9: Diagrama de Actividad de Login de usuario

4.6 Diagrama de Actividad de Creación de espacio

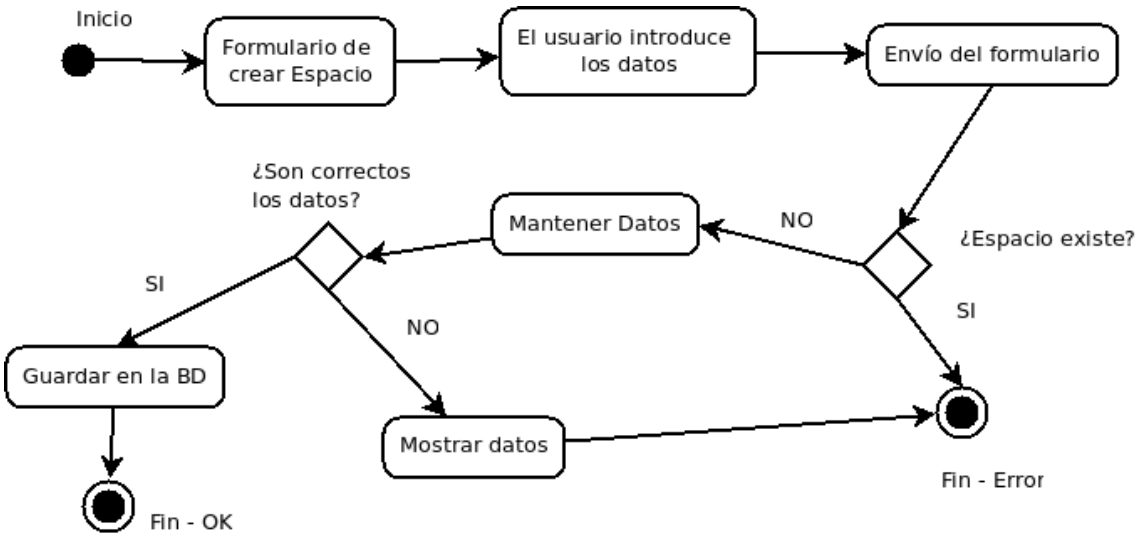


Imagen 10: Diagrama de creación de espacio

5 Implementación

5.1 Tecnologías

HTML5 ***(HyperText Markup Language)***: Es un lenguaje de marcas que constituye la base de cualquier página web. Este lenguaje se compone de marcas que identifican cada elemento que debe ser mostrado en una web. Posteriormente el navegador se encarga de darle representación gráfica. Cada vez que se carga la web, el navegador lee el código y dibuja los elementos de la forma que tiene internamente programada. Es por tanto un lenguaje interpretado[3]. Las páginas creadas en HTML puro son páginas estáticas, es decir, sólo se cambian si el programador las cambia. Actualmente se encuentra en su versión 5 y es la que se usará en este proyecto ya que aporta muchas ventajas y en comparación con las versiones anteriores y es un de uso obligatorio si se quiere desarrollar un sitio web resposive. Es desarrollado por la Wolrd Wide Web Consortium (W3C) por lo que es considerado software libre.

Más información en: https://es.wikipedia.org/wiki/HTML5 y en http://www.w3schools.com/html/html5_intro.asp

CSS3 ***(Cascading Style Sheets):*** Es un lenguaje que se usa para dar definir y crear la presentación del código html. La versión 3, que es la que se usa actualmente, aporta notables mejoras respecto a versiones anteriores. Dado que CSS3 soporta toda la sintaxis de versiones anteriores se recomienda encarecidamente su uso y es un de uso obligatorio si se quiere desarrollar un sitio web resposive. Es desarrollado por la Wolrd Wide Web Consortium (W3C) por lo que es considerado software libre.

Más información en: https://es.wikipedia.org/wiki/Hoja_de_estilos_en_cascada y en http://www.w3schools.com/css/css3_intro.asp

SASS ***(Syntactically Awesome StylesSheets):*** Es una extensión de CSS que permite el uso de variables, funciones, anidamiento de reglas y mucho más, convirtiendo el tradicional lenguaje CSS tradicional en una potente herramienta que comparte algunas características de los lenguajes de programación tradicionales.

Existen dos sintaxis para el uso de SASS, una más antigua llamada simplemente "*Sass*" y que se caracteriza por usar una sintaxis indentada en lugar de usar las típicas llaves de apertura/cierre ({}) y retornos de carro en lugar de ; para separar las líneas. Los ficheros con este forma to tiene extensión "*.sass*"

La segunda opción es la que más ampliamente se utiliza actualmente y se conoce como SCSS (Sassy CSS). Es una extensión de CSS3, lo que significa que todas las expresiones de CSS3 están soportadas y cualquier fichero CSS3 es también un fichero SCSS. Además soporta las extensiones de terceros[4] para las definiciones CSS. Los ficheros con este formato tienen extensión "*.scss*".

En nuestra aplicación usaremos este último formato por considerarse más similar al tradicional formato del CSS con el que el lector, a buen seguro, está familiarizado.
La implementación se encuentra bajo licencia del MIT[5]., por lo que se considera Software Libre.

Más información en: http://sass-lang.com/

[3] A diferencia de los lenguajes compilados (como C o Java), los ficheros ejecutados pueden ser leídos directamente en un editor de texto plano. Otros lenguajes interpretados son: PHP, Ruby, CSS, Javascript.
[4] -ms (Microsoft), -moz (Firefox), -o (Opera), -webkit (Safari y Chrome)

[5] MIT LIcense: http://opensource.org/licenses/MIT

PHP5 ***(PHP Hypertext Pre-processor):*** Es un lenguaje de programación del lado del servidor. Publicado bajo la PHP License, la Free Software Foundation considera esta licencia como software libre. Usualmente se utiliza para el desarrollo web y permite la creación de páginas dinámicas.
La diferencia fundamental, aparte de las nuevas funciones introducidas, entre PHP5 y las anteriores es la orientación a objetos. Actualmente ya no se desarrolla usando versiones anteriores a la 5. *La versión actual con la que se desarrolla este proyecto es la 5.5.9*.

Más información en: http://www.php.net

MySQL: Es un Sistema de gestión de bases de datos (SGBD), multihilo, multiusuario y multiplataforma. A pesar de que la empresa que desarrolla el producto (MySQL AB) es una subsidiaria de Oracle (desde Abril de 2009), MySQL se dritribuye bajo una doble licencia.
Por un lado se ofrece bajo la GNU GPL para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar a la empresa una licencia específica que les permita este uso.
En nuestro caso, no tendremos ningún tipo de problema en el uso ya que nuestro proyecto se enmarca bajo la licencia GNU GPL.
La versión actual de MySQL es la 5 y este proyecto se realiza sobre la versión 5.5.44.

Más información en: https://www.mysql.com/ y en https://es.wikipedia.org/wiki/MySQL

APACHE: Es un servidor web de código abierto multiplataforma más utilizado actualmente. Tiene una arquitectura modular que permite la instalación de diferentes módulos y que éstos puedan ser activados/desactivados según la necesidad.
Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido.
Es el encargado de interpretar y procesar el código PHP de las páginas web transformarlo en HTML y enviarlo al navegador para que sea representado.
La versión actual y recomendada del servidor apache es la 2. En el momento de desarrollar este proyecto, la versión de Apache es la 2.4.7

Más información en: http://www.apache.org

5.2 Herramientas

COMPASS: Es un framework open-source que utiliza Sass. Como todo framework, proporciona funcione, utilidades y librerías para hacer fácil la escritura de hojas de estilo. Podríamos decir que Compass es a SASS lo que CakePHP es a PHP. La salida generada por compass son ficheros CSS puros.

Más información en: http://compass-style.org/

CAKEPHP 3: Es un framework de desarrollo basado para PHP. O dicho de una forma sencilla, un conjunto de herramientas en forma de librarías que permiten a los programadores desarrollar aplicaciones web de una forma rápida y sencilla.
La principal ventaja que ofrece el desarrollo mediante un framework como CakePHP es que la mayoría de las operaciones habituales en el desarrollo de un proyecto ya están implementadas y sólo ha y que usarlas.

Pensemos, por ejemplo, en el tiempo que se tarda en desarrollar el sistema de autentificación de usuarios o el sistema de registro. Hay que crear todas y cada una de las funciones que recogen, validan, limpian

y/o guardan los datos de los usuarios. Además hay que crear una serie de funciones auxiliares que muestren mensajes de error o textos de ayuda que guíen a los usuarios.
Al usar un framework de desarrollo como CakePHP, nos ahorramos todo este trabajo porque estas funciones ya están implementadas y sólo tenemos que usarlas.

Más información en: http://www.cakephp.org

NETBEANS: Es lo que se llama comúnmente un IDE (Integrated Development Environment o Entorno Integrado de Desarrollo). Se trata de una herramienta que integra todas las herramientas necesarias para la creación, modificación, compilación, implementación y depuración de software. Es un producto libre y gratuito sin ningún tipo de limitaciones para uso.
Netbeans usa un sistema de plugins que permiten extender su funcionalidad y adaptarlo para que nos ayuda a ser más productivos en nuestro trabajo.

Más información en: http://www.netbeans.org

PHPMYADMIN: Es una aplicación web que permite controlar el servidor de bases de datos MySQL de una forma gráfica a través del navegador. Desde esta aplicación podemos controlar todas las operaciones que tengan que ver con el tratamiento de los datos (Creación, Actualización, Eliminación).
Además nos permite configurar una gran cantidad de parámetros de configuración del propio servidor.

Más información en: http://localhost/phpmyadmin

5.3 Detalles de implementación

Base de datos y configuración de CakePHP

CakePHP cuenta con un módulo ORM que nos evita el uso de SQL para el manejo de la base de datos y las operaciones que sobre ella se ejecutan (inserciones, ediciones, eliminaciones, etc.)
Antes de que el entorno pueda trabajar con la base de datos debe conocer de su existencia y tener los datos necesarios para poder acceder a ella. Esta configuración la haremos en el fichero *app.php:* <project_root>/config/app.php

```
        * Connection information used by the ORM to connect
        * to your application's datastores.
        * Drivers include Mysql Postgres Sqlite Sqlserver
        * See vendor\cakephp\cakephp\src\Database\Driver for complete list
        */
    'Datasources' => [
        'default' => [
            'className' => 'Cake\Database\Connection',
            'driver' => 'Cake\Database\Driver\Mysql',
            'persistent' => false,
            'host' => 'localhost',
            /*
             * CakePHP will use the default DB port based on the driver selected
             * MySQL on MAMP uses port 8889, MAMP users will want to uncomment
             * the following line and set the port accordingly
             */
            //'port' => 'nonstandard_port_number',
            'username' => 'sportsnet',
            'password' => 'sportsnet1234',
            'database' => 'sportsnet',
            'encoding' => 'utf8',
            'timezone' => 'UTC',
            'cacheMetadata' => true,
```

Imagen 11: Configuración de la BD

CakePHP permite la configuración de diferentes bases de datos, dependiendo del entorno de desarrollo. Es decir, que se podría configurar una base de datos para el entorno de desarrollo en local, otra para el servidor de pruebas y otra para el servidor de producción.
En nuestro caso usaremos la misma configuración para todos los entornos excepto para el servidor de producción. En ese momento deberemos editar este fichero y añadir una nueva sección con los datos de acceso correspondientes.

Se recomienda leer este fichero con detenimiento ya que hay muchas opciones que se pueden configurar, tales como el sistema de log, mail y las sesiones. Los comentarios son de gran ayuda para entender el funcionamiento de los parámetros que en este fichero se definen.

En CakePHP 2 la configuración de la base de datos se realizaba en el fichero *database.php*

app/Config/database.php

```
class DATABASE_CONFIG {
    public $default = array(
        'datasource'  => 'Database/Mysql',
        'persistent'  => false,
        'host'        => 'localhost',
        'login'       => 'cakephpuser',
        'password'    => 'c4k3roxx!',
        'database'    => 'my_cakephp_project',
        'prefix'      => ''
    );
}
```

Imagen 12: Configuración BD CakePHP2

Ahora CakePHP ya conoce la base de datos. A continuación deberemos definirle las estructuras de datos (tablas) con las que deberá trabajar y que nos servirán para definir los modelos que usaremos posteriormente.
Para crear las tablas de la base de datos usaremos phpMyAdmin (ver Anexo III).

A continuación generaremos los objetos que necesitaremos usando la consola "bake" de CakePHP. Estos objetos comprenden "Módelos", "Controladores" y "Vistas".

La consola "bake"

La consola Bake permite la generación rápida de código. A través de la consola bake pueden crearse cualquiera de los objetos que CakePHP puede manejar, tales como "modelos", "controladores", "vistas", "comportamientos", "helpers", "casos de prueba" y "plugins".
La instalación se puede llevar a cabo, si no se ha hecho ya, mediante el siguiente comando:

```
composer require --dev cakephp/bake:~1.0
```

Imagen 13: Instalación de la consola bake

El plugin DebugKit

Este plugin está mantenido por el equipo de desarrollo del núcleo de CakePHP y proporciona una barra de herramientas para facilitar la depuración de aplicaciones.

A diferencia de la consola bake, no viene instalado por defecto. Para la instalación nos situamos en el directorio principal del proyecto, en nuestro caso */var/www/sportsnet* y ejecutamos lo siguiente:

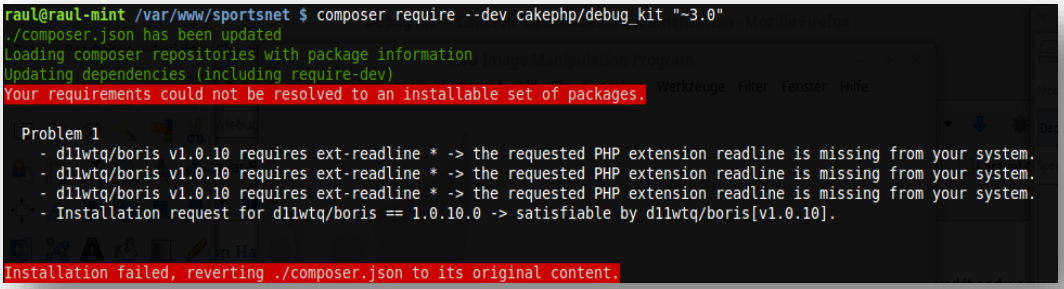


Imagen 14: Instalación de DebugKit

En este caso nos indica que una de las dependencias no se ha podido satisfacer y por ese motivo no se ha podido instalar el plugin. En este caso, se necesita la extensión de PHP "readline". La instalaremos desde la propia consola de comandos.

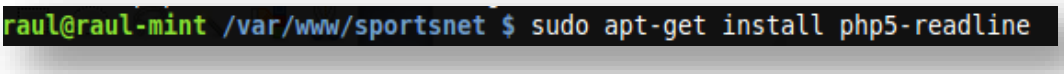


Imagen 15: Instalación readline

Después de la instalación, repetimos la instalación de DebugKit y ahora sí que se instalará correctamente.

Imagen 16: Instalación de DebugKit

Por defecto, DebugKit usa una pequeña base de datos SQLite para almacenar sus datos, así que deberíamos definir la conexión *debug_kit*. Esta conexión se definirá en el fichero **config/app.php**, al igual que la conexión con la base de datos de la aplicación.
La base de datos se almacenará en la carpeta **tmp** de nuestra aplicación.

Para instalar el controlador *pdo_sqlite*, ejecutaremos lo siguiente:

```
raul@raul-mint /var/www/sportsnet/config $ sudo apt-get install php5-sqlite
```

Imagen 17: Instalar pdo_sqlite

Tras la instalación, no olvidemos reiniciar Apache para que el nuevo módulo *pdo_sqlite* sea cargado.
La conexión con sqlite3 la definimos como sigue en el fichero **app.php**

```
/**
 * The test connection is used during the test suite.
 */
'test' => [
    'className' => 'Cake\Database\Connection',
    'driver' => 'Cake\Database\Driver\Mysql',
    'persistent' => false,
    'host' => 'localhost',
    //'port' => 'nonstandard_port_number',
    'username' => 'my_app',
    'password' => 'secret',
    'database' => 'test_myapp',
    'encoding' => 'utf8',
    'timezone' => 'UTC',
    'cacheMetadata' => true,
    'quoteIdentifiers' => false,
//'init' => ['SET GLOBAL innodb_stats_on_metadata = 0'],
],
/**
 * DebugKit connection
 */
'debug_kit' => [
    'className' => 'Cake\Database\Connection',
    'driver' => 'Cake\Database\Driver\Sqlite',
    'persistent' => false,
    'database' => '/var/www/sportsnet/tmp/debug_kit.sqlite3',
    'prefix' => '',
    'encoding' => 'utf8',
],
```

Imagen 18: Configuración DB DebugKit

Para desactivar el plugin tan solo cambiamos en el mismo fichero **app.php**:

```
'debug' => true,

por

'debug' => false,
```

5.4 El prototipo

Introducción

"Se come más con los ojos que con la boca"

Esta típica frase, aunque con otro sentido, se aplica perfectamente a las páginas web. Cuando un usuario llega a una página, lo que determina si esa página pasará a sus marcadores o caerá en el olvido es la primera impresión.

La página deberá cumplir con los siguientes criterios básicos:

- Combinación de colores adecuada: una web debe usar colores que evoquen y produzcan en el usuario sentimientos de confianza y al mismo tiempo seriedad.

- Ser sencilla tanto visualmente como a nivel de uso: Una web debe proporcionar información suficiente para mantener al usuario en ella y no abrumarlo con demasiada cantidad. Tener espacios en blanco en una *web ¡NO ES MALO!*

- *Usable* en el sentido más amplio de la palabra, es decir, el usuario debe poder interactuar con la página en cualquier situación y debe presentar un aspecto claro y limpio intentando destacar o poner en la línea de visión del usuario aquellas funcionalidades que ayuden al usuario a conseguir la mejor experiencia posible. De nada sirve usar un cursor con un icono de forma especialmente diseñada si éste es demasiado pequeño o se confunde con otros elementos de la web o produce efectos de parpadeo, etc.

- Información de calidad: La información de la web debe tener calidad y debe evitarse la repetición de la misma en diferentes lugares. Los enlaces deben validarse y evitar enlaces rotos o fraudulentos. Una web elegantemente diseñada pero información errónea, contrastada o simplemente inútil, será rápidamente eliminada de todos los navegadores de los usuarios.

Diseñar una página que cumpla con estos objetivos básicos, a priori tan sencillos, no es una tarea fácil y mucho menos en un proyecto como el que nos ocupa, en el que el tiempo es tan escaso.

Así pues nos conformaremos con diseñar una página que, al menos resulte agradable a la vista y se vea los más clara posible, dejando marcado el camino para futuras ampliaciones.
Lo primero que haremos será crear un prototipo en HTML + PHP plano, de forma que luego podamos usar el código creado directamente en las plantillas de CakePHP, pero sustituyendo las partes dinámicas con código HTML.

El motivo de hacerlo así es que, de esa forma podremos concentrarnos en el diseño, tener una visión más clara de la estructura de la página sin perder por ello la funcionalidad futura que tendrá la web, sin tener que preocuparnos, por el momento, de los posibles problemas que nos surgirán cuando usemos CakePHP. En su momento, una vez tengamos el prototipo, nos ocuparemos exclusivamente de implementar la funcionalidad en CakePHP.
Diseño: Prototipo HTML (Dummy HTML)

El prototipo HTML o dummy HTML es una versión totalmente funcional de la web. La mayoría de las veces desarrollada por completo en HTML puro, aunque es posible que se utilice algún lenguaje de programación como PHP para ayudar a dar dinamismo a la web o ahorrar la repetición de código HTML.
En realidad, la única diferencia entre la aplicación real y la aplicación dummy es que ésta última no usa base de datos, ni por supuesto las funciones de CakePHP.
Es una buena práctica desarrollar siempre un prototipo de estas características porque:

- Ayuda a ver dónde y bajo cuáles circunstancias aparecen o podrían aparecer los principales problemas durante la implementación.

- Todo el código, incluyendo CSS y javascript es reutilizable en la aplicación real. En el peor de los casos, los estilos CSS deberán ser adaptados al formato SCSS, en el caso de que queramos usar esta tecnología.

- Ayuda a los desarrolladores del frontend a ver exactamente cómo se comporta la web frente a la interacción del usuario y les facilita la toma de medidas, captura de colores y otros recursos que se necesitaran para el desarrollo.

- Ayuda al cliente a ver cuáles son las capacidades reales de funcionamiento y le proporciona una visión real del producto final, sobre la que tomar sus decisiones.

- Ayuda a descubrir los puntos fuertes y los débiles de cara a la usabilidad por parte de los usuarios.

- Proporciona facilidad de modificación en tiempo real: Durante la presentación al cliente, éste puede solicitar, por ejemplo, ver un texto en un color distinto. Esta modificación es fácilmente realizable usando las herramientas de desarrollo integradas de los navegadores.

- Ayuda a identificar las partes o elementos de los que se compone la web, permitiendo separar los elementos en objetos separados que pueden añadirse o no en función de las necesidades. Es decir, ayuda a identificar aquellas partes que son consideradas como parte del núcleo y aquellas que pueden ser añadidas en forma de widgets.

Diseño

Página principal

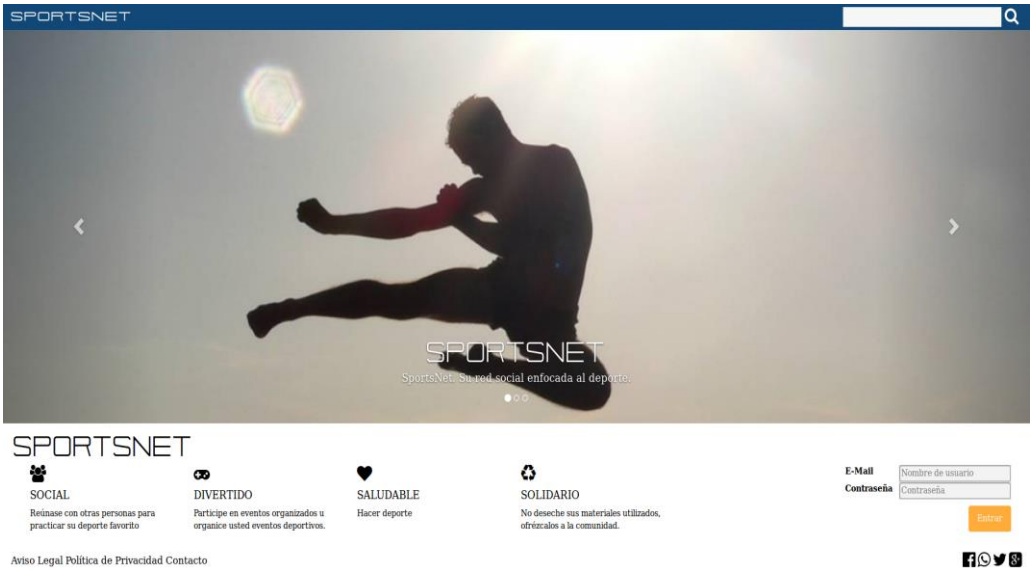


Imagen 19: Página principal

Como se aprecia en la imagen la página principal constará de una barra superior que mostrará el título del sitio, y un formulario de búsqueda que permitirá buscar contenidos que hayan sido publicados con acceso a usuarios no autentificados. En principio aparecerá una lista de sitios que coincidan con el texto de búsqueda, sin que ello conlleve que el acceso a dichos sitios esté permitido.

Bajo la barra superior se puede ver un "Slider" que automáticamente irá cambiado la imagen con la que, además, se mostrará un texto asociado. A cada lado se muestra un flecha que permitirá al usuario la interacción con el elemento, permitiendo que pueda desplazar las imágenes en la dirección que desee.

Una tercera sección compuesta por varios elementos informativos con texto imágenes que describen algunas de las características del sitio. A la derecha de estos elementos encontramos el formulario de login (autenticación) para usuarios registrados. La autenticación correcta dirigirá al usuario a una página con el listado de sus espacios.

Por último observamos en el pie de página dos secciones diferenciadas. A la derecha aparecen las páginas reglamentarias con el "Aviso legal", las "Políticas de Privacidad" y por supuesto el enlace al formulario de "Contacto".

En parte derecha los habituales enlaces hacia las otras redes sociales.

En principio, todos los objetos y textos que aparece serán configurables sólo por los Administradores del sitio (no confundir con los administradores de Espacio). Pero será en futuras versiones de la aplicación.

Páginas internas

Las páginas internas, en principio, tendrán siempre la misma estructura. Como vemos en las siguientes capturas de pantalla contendrán habitualmente: *barra superior, barra lateral de navegación (izquierda), barra lateral de herramientas (derecha) y una sección de contenido.*

Listado de espacios del usuario

Imagen 20: Lista de espacios: Una vez autentificado el usuario será redirigido a esta página

Vista de uno de los espacios del usuario

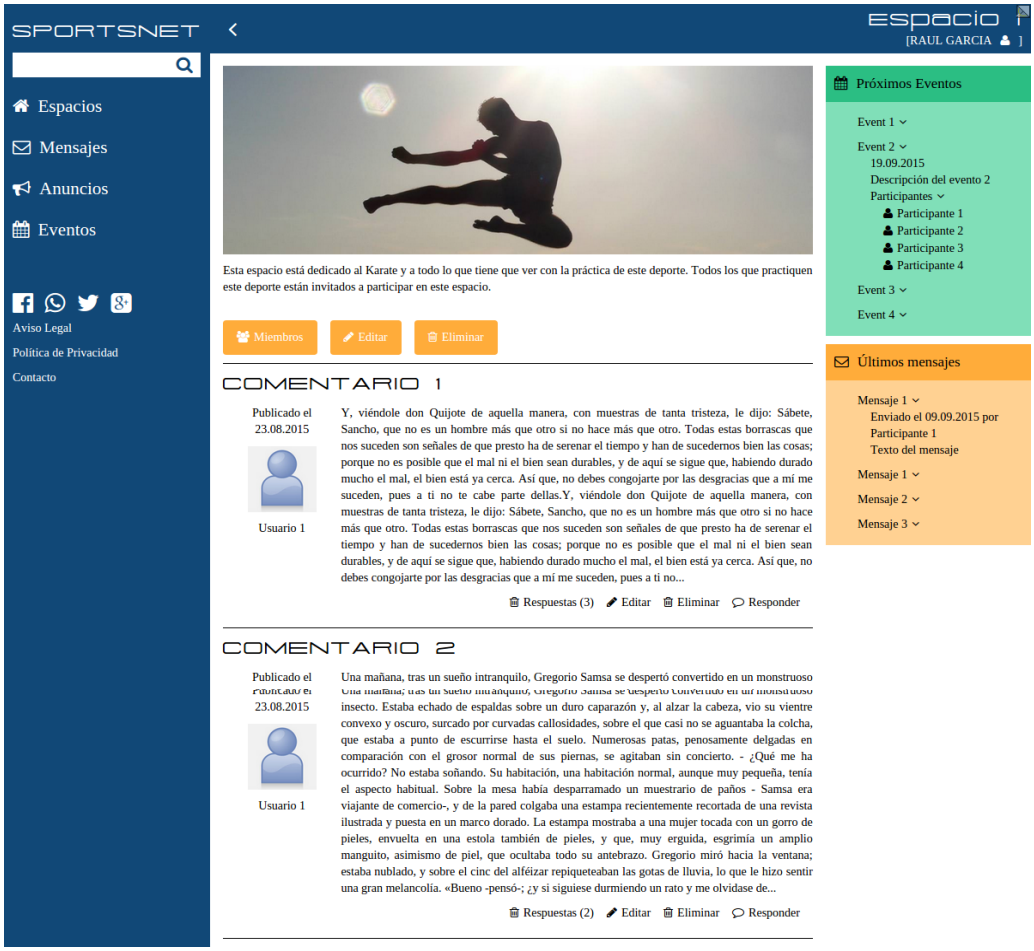


Imagen 21: Página interna - Vista de uno de los "Espacios" del usuario

Vista de uno de los espacios del usuario con respuestas en los comentarios

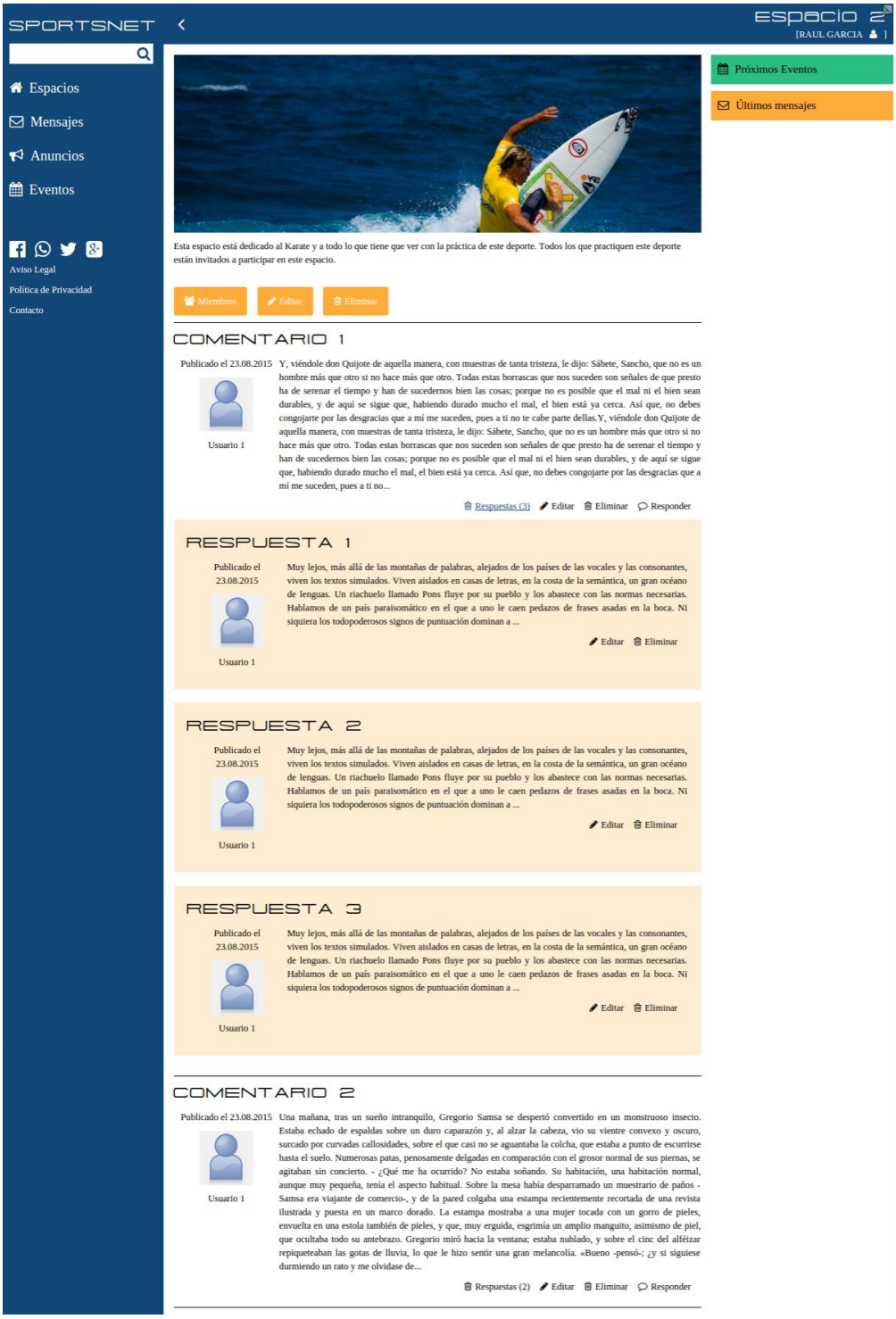


Imagen 22: Vista de uno de los espacios del usuario con respuestas en los comentarios

Vista del formulario de contacto del usuario autentificado

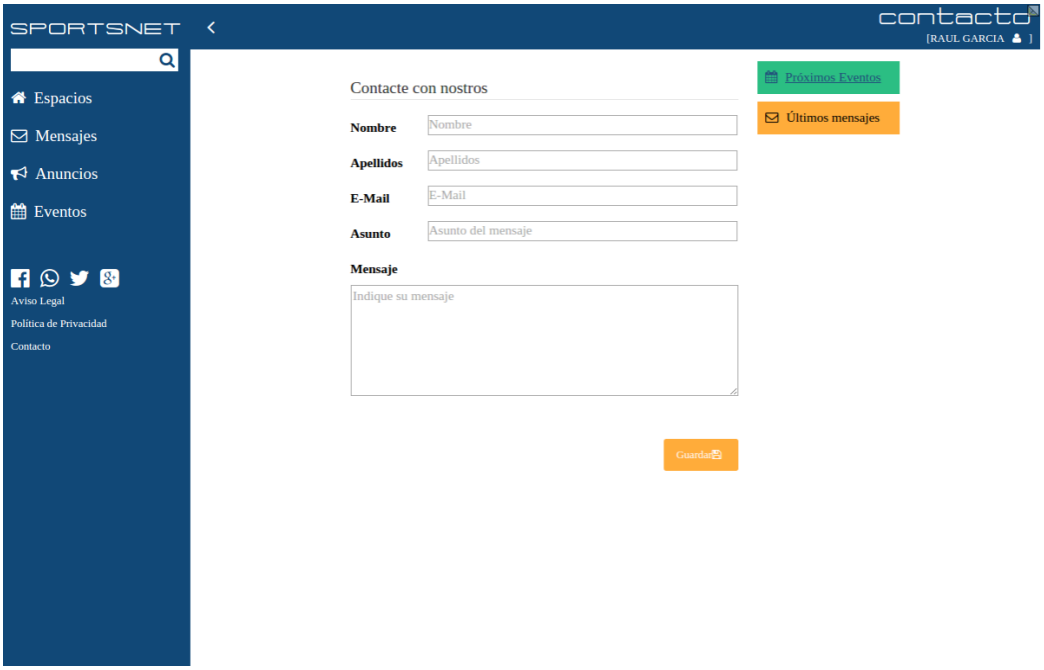


Imagen 23: Vista del Formulario de contacto del usuario autentificado

Vista del formulario de contacto del usuario NO autentificado

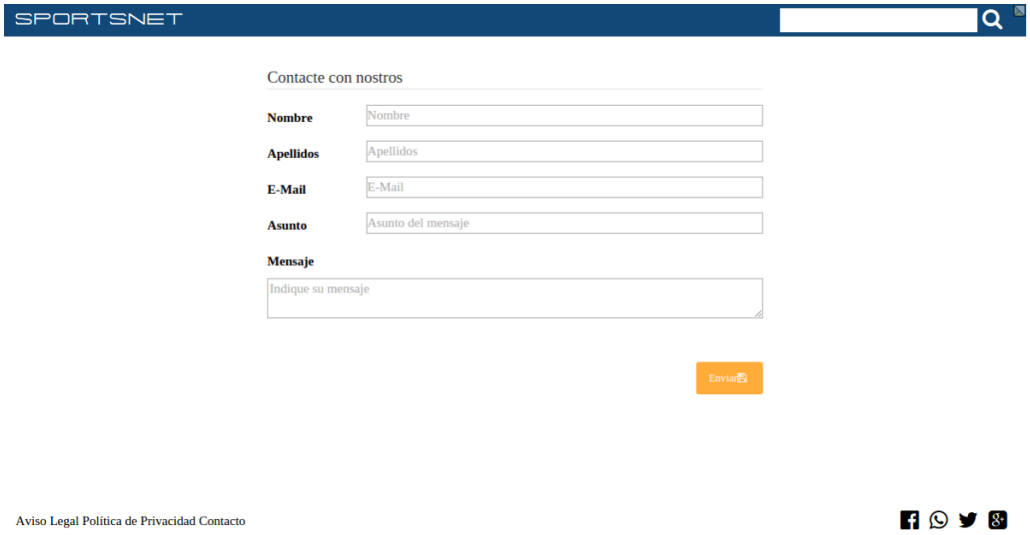


Imagen 24: Vista del Formulario de contacto del usuario NO autentificado

Vista del perfil del usuario

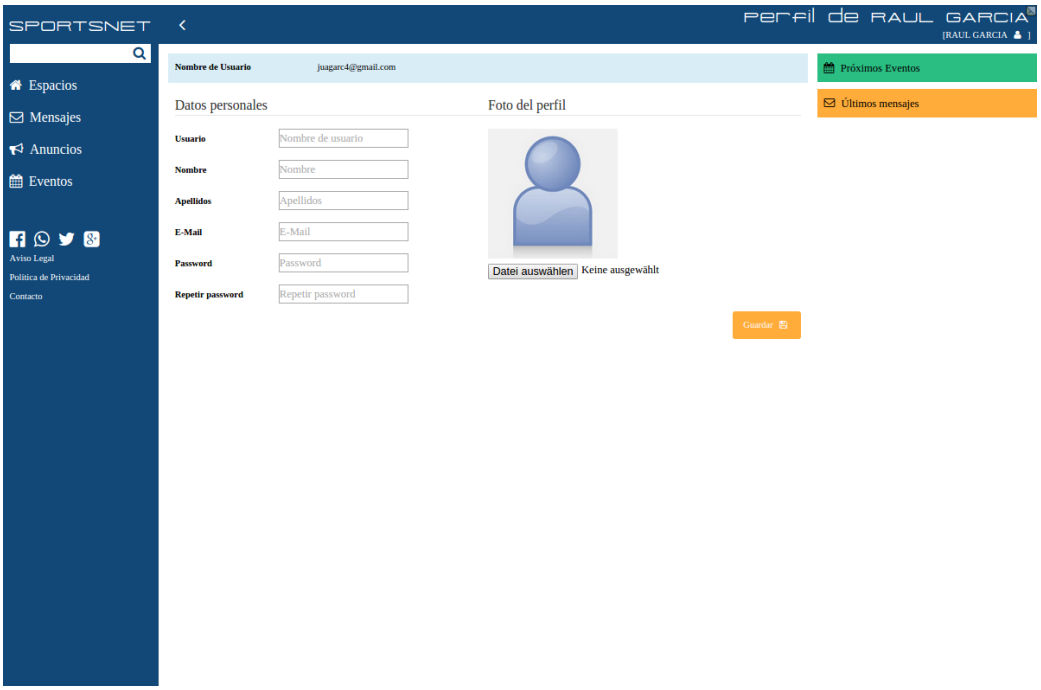


Imagen 25: Vista del perfil del usuario

Implementación

Para no extender demasiado el documento, se adjuntan en el fichero comprimido **prototipo.zip**, los archivos aquí descritos, de manera que el lector pueda aprovechar en la medida que lo requiera el código de los mismos.
Contenido del fichero **prototipo.zip**

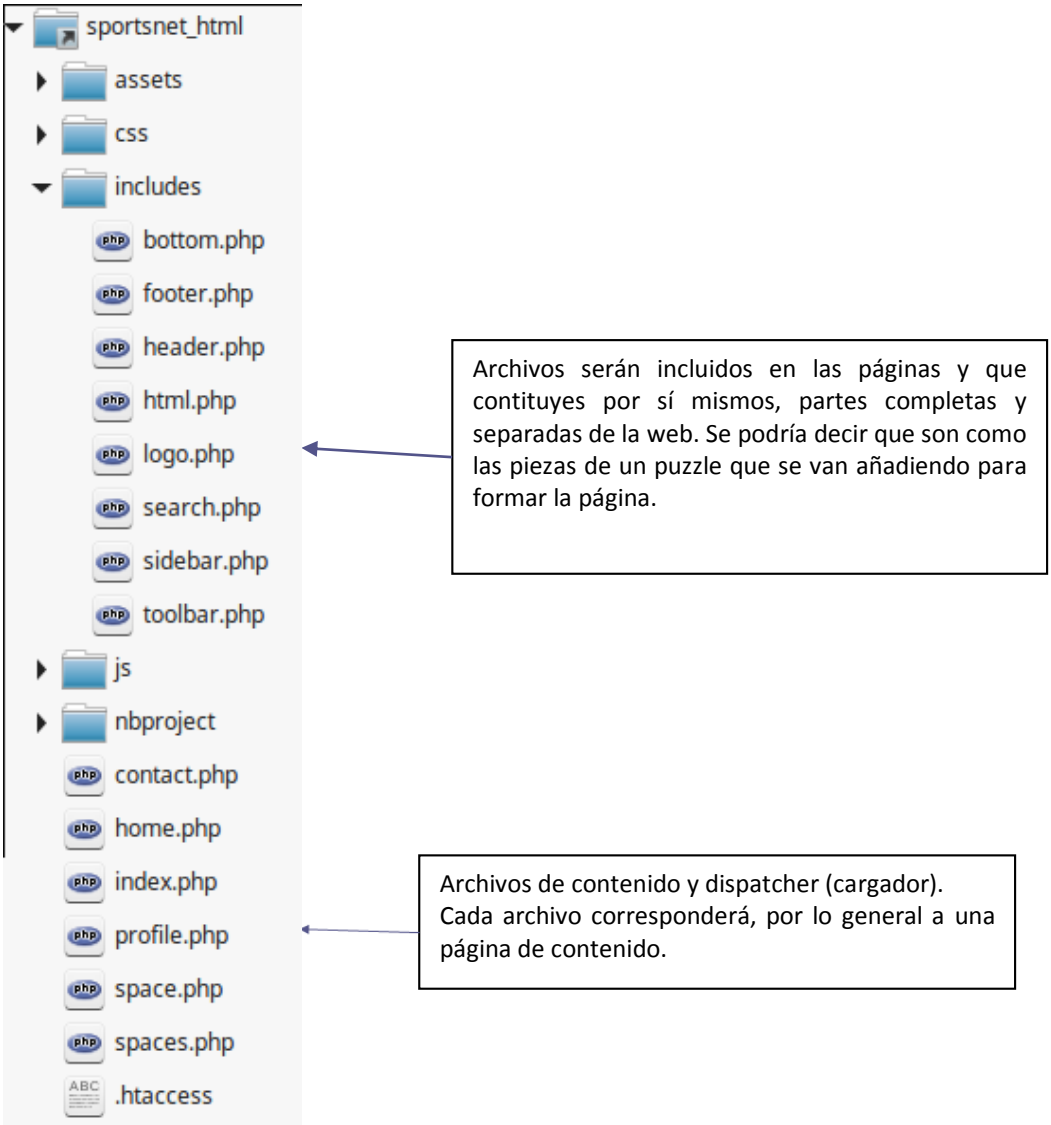


Imagen 26: estructura del directorio del prototipo

Página index (index.php)

Nuestra página index actuará de cargador para el resto de páginas. Este es el comportamiento habitual de cualquier CMS que use el patrón MVC. El fichero index.php será pues nuestro "*dispatcher*".

Encabezado del documento (html.php)

En este fichero se encuentra la definición del inicio del documento html, el tag de apertura, los tags “meta” que se utilizan en el inicio del mismo e incluso la etiqueta de apertura del cuerpo del contenido (el <body>). También se pueden ver en este archivo la inclusión de los ficheros CSS.

Cierre del documento (bottom.php)

En este fichero se encuentran los tags de cierre del contenido y del documento html, así como la inclusión de los ficheros javascript. Colocar la inclusión de los ficheros javascript al final del documento, justo antes del cierre de la etiqueta </body>. Esta es una práctica recomendable para mejorar el rendimiento en la carga de las páginas, puesto que permite que la página HTML sea presentada en cuanto llegue al navegador y el javascript se cargará a continuación ejecutándose sin retardos ya que los elementos sobre los que deberá actuar ya se han cargado.

Una excepción a esta regla con es la carga de ficheros javascript que influyen o pueden influir directamente en la presentación de los contenidos. Por ejemplo la biblioteca “modernizr6” que identifica que características de las nuevas tecnologías web (HTML, CSS3) puede soportar el navegador, permitiendo al desarrollador establecer el uso de características clásicas, en el caso de que el navegador no soporte las técnicas más nuevas.

Nótese que los ficheros CSS también se cargan mucho antes, justo al inicio del documento, porque estos documentos influyen directamente sobre la presentación del documento y por tanto deben cargarse cuanto antes.

Cabecera (header.php)

La cabecera será distinta dependiendo de si se muestra en página principal o en las páginas internas.

Pie (footer.php)

Al igual que la cabecera, el pie se mostrará de una forma o de otra dependiendo de la página que estemos viendo. Adicionalmente en el pie se mostrará además de las páginas habituales de contacto, privacidad, etc, los típicos iconos de contacto con otras redes sociales. El pie se mostrará en la parte inferior, únicamente en la página principal y en las páginas que no requieran autenticación. En el resto de páginas el contenido del pie se integrará en la barra lateral de navegación (sidebar.php).

Barra lateral de navegación (sidebar.php)

La barra lateral sólo se mostrará en páginas internas y tendrá la capacidad poder ser ocultada por el usuario.

[6] http://modernizr.com

Barra lateral de herramientas (toolbar.php)

Esta barra lateral se mostraré n las páginas internas únicamente y mostrará diferente tipos de objetos, llamados también widgets. Algunos ejemplos de widgets serían, un calendario un bloque de últimos eventos o de últimos mensajes, etc.

Contenido

En el contenido se mostrará, evidentemente, el contenido asociado a la sección que se esté visitando. Los ficheros correspondientes se encuentran en la raíz del directorio contenido en el archivo adjunto.

5.5 Instalación del entorno de Desarrollo

Nuestro entorno de desarrollo constará de los siguientes elementos básicos:

Servidor Web: Apache

Servidor de Bases de Datos: MySQL

Lenguaje de programación: PHP

Aparte de los elementos principales, necesitaremos algunos módulos extra. Algunos son requisitos que debemos tener para poder instalar CakePHP y otros nos ayudaran en el proceso de desarrollo. Antes de proceder a la instalación, como buenos informáticos, revisemos lo que vamos a instalar.

Módulos obligatorios

Para que CakePHP pueda trabajar correctamente, deberemos de instalar los siguientes módulos.

- *mbstring:* Proporciona funciones específicas para cadenas de texto multibyte que ayudan a tratar codificaciones multibyte en PHP .

- *mcrypt:* Es una interfaz para la biblioteca mcrypt, que proporciona funciones de encriptado y cifrado. Entre otros, admite los siguientes algoritmos de bloques: DES, TRipleDES, Blowfish (predeterminado), etc.

- *intl:* Extensión para la internacionalización. Permite a los programadores de PHP realizar un cotejo que cumple con el UCA y un formateo de fecha/hora/número/moneda en los scripts.

Durante la instalación de PHP5 se habrá instalado la biblioteca mbstring por lo que no tenemos que instalarla manualmente.

Extensiones de Base de Datos MySQL

Las extensiones de base de datos de MySQL (la extensión MySQL, *mysqli* y PDO MYSQL), se comunican con el servidor de MySQL.

En el pasado, esto lo realizaba la extensión utilizando los servicios prestados por la Biblioteca Cliente de MySQL (*libmysqlclient*). Las extensiones eran compiladas con la Biblioteca Cliente de MySQL con el fin de utilizar su protocolo cliente-servidor.

Ahora existe una alternativa: el Controlador Nativo de MySQL (*mysqlnd*), ya que las extensiones de bases de datos MySQL pueden ser compiladas para utilizar el Controlador Nativo de MySQL en lugar de la Biblioteca Cliente de MySQL.

Módulos de ayuda

- *curl:* Biblioteca que permite conectarse y comunicarse con diferentes tipos de servidores y diferentes tipos de protocolos. Actualmente, se admiten http, https, ftp, gopher, telnet, dict, file y ldap.

- *gd:* Permite que PHP pueda crear y manipular ficheros de imágenes en diferentes formatos. Actualmente están soportados: GIF, PNG, JPEG, WBMP y XPM. También permite que PHP pueda transferir flujos de imagen directamente al navegador, esto es, sin necesidad de generar un fichero de imagen.

- *xdebug:* Es una extensión que permite a PHP mostrar los errores en tiempo de ejecución en un formato amigable directamente en el navegador.

Otros Módulos

- *dbconfig-common:* Es una implementación del borrador "Mejores prácticas para aplicaciones de bases de datos". Su misión es facilitar a otros programas del Sistema el trabajo con bases de datos proporcionando un método sencillo pero consistente. Este paquete será usado por phpmyadmin y por mysql durante la instalación.

Instalación

```
sudo apt-get install apache2 php5 php5-intl php5-mcrypt php5-curl php5-gd php5-mysqlnd
php5-xdebug mysql-server mysql-common dbconfig-common phpmyadmin
```

Una vez instalado todo, ya seremos capaces de desarrollar aplicaciones en php que usen bases de datos. Si en cualquier momento necesitamos reiniciar alguno de los servidores (apache o mysql) usaremos los siguientes comandos:

Apache
sudo apache2ctrl restart / sudo service apache2 restart

MySQL
sudo service mysql restart

Referencias

mbstring: http://php.net/manual/es/book.mbstring.php
mcrypt: http://php.net/manual/es/book.mcrypt.php
intl: http://php.net/manual/es/book.intl.php
curl: http://php.net/manual/es/intro.curl.php
gd: http://php.net/manual/es/book.image.php
mysqlnd: http://es.php.net/manual/es/intro.mysqlnd.php
xdebug: http://xdebug.org/
dbconfig-common: https://people.debian.org/~seanius/policy/dbconfig-common.html/

5.6 Configuración del entorno de Desarrollo

Configuración del fichero hosts

Este fichero actúa como un DNS local y nos permite usar nuestras propias direcciones web. Cada dirección usada debe coincidir con un VirtualHost definido, concretamente debe coincidir con el parámetro "*ServerName"* del VirtualHost

Editamos el fichero con permisos de root y añadimos la linea correspondiente para nuestro proyecto:

```
sudo vim /etc/hosts
```

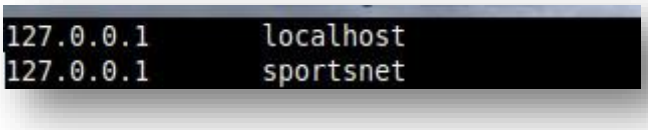


Imagen 27: Fichero hosts

Configuración de PHP

La configuración de php se realiza en el fichero */etc/php5/apache2/php.ini*
Aunque no es imprescindible su configuración, si es recomendable configurar los siguientes parámetros tal y como sigue. De esa forma seremos más productivos a la hora de desarrollar y nos permitirá detectar y corregir situaciones anómalas y errores que, inevitablemente, surgen durante el desarrollo.

La edición de este fichero debe hacerse con permisos de root.

```
sudo vim /etc/php5/apache2/php.ini
```

Parámetro desarrollo	Parámetro producción
error_reporting = E_ALL	error_reporting = E_ALL & ~E_DEPRECATED & ~E_STRICT
Esta variable permite que PHP captura los errores, según su tipo, que ocurran durante la ejecución de la página. Durante la etapa de desarrollo necesitaremos captar la mayor cantidad de errores posibles para que podamos subsanarlos antes de poner la página en producción.	
display_errors = On	display_errors = Off
Este parámetro controla la visibilidad de los mensajes de error capturados por "error_reporting". Si esta variable está activada, los mensajes serán mostrados directamente en el navegador. Si hemos instalado xdebug, los mensajes serán enviados a esta extensión antes de ser mostrados.	
error_log= /var/www/logs/php_errors.log	
Con este parámetro le decimos a PHP dónde almacenará el fichero de registro. Es recomendable siempre activar este parámetro ya que tendremos controlado dónde debemos mirar cuando se produzca alguna situación inesperada. Es importante tenerlo definido en producción, ya que allí tendremos la variable "display_errors=Off" y será en este fichero de dónde se guardaran los errores, en lugar de mostrarse en el navegador.	
date.timezone = 'Europe/Madrid'	
Este parámetro es obligatorio desde PHP 5.5 y debe definirse siempre para que PHP pueda trabajar bien con fechas. Se puede consultar una lista de posibles valores en: http://php.net/manual/es/timezones.php	

Tabla 7: Parámetros de PHP. Archivo php.ini

El resto de parámetros de PHP se pueden dejar en su valor por defecto.

Creación y configuración de la base de datos MySQL

Por simplicidad se utilizará la herramienta phpMyadmin para crear y configurar la base de datos. Antes de crear la base de datos vamos a configurar algunos parámetros para que sean tomados por defecto. Esta configuración la haremos en el fichero **/etc/mysql/my.cnf**

```
[client]
port                    = 3306
socket                  = /var/run/mysqld/mysqld.sock
default-character-set   = utf8

[mysqld]
#
# * Basic Settings
#
...
skip-external-locking
collation-server   = utf8_general_ci
character-set-server    = utf8
...
#
# Error log - should be very few entries.
#
log_error               = /var/log/mysql/error.log
```

El fichero debe editarse con permisos de root y únicamente añadiremos las líneas marcadas en color verde.

Guardamos y reiniciamos el servicio de mysql:

```
sudo service mysql restart
```

Para acceder a phpMyAdmin usaremos la URL: http://localhost/phpmyadmin
Durante la instalación en el Anexo I se nos habrá preguntado por los datos usuario y contraseña para MySQL y para phpMyAdmin. Aquí deberemos usar esos datos para acceder.

Imagen 28: Pantalla principal de phpMyAdmin

Nota:

Si hemos usado un usuario SIN contraseña durante la instalación de MySQL y phpMyAdmin deberemos ajustar la configuración de phpMyAdmin para que nos permita el uso de usuarios sin contraseña.
Esto lo haremos en el fichero */etc/phpmyadmin/config.inc.php*
En ese fichero buscaremos y descomentaremos la siguiente línea (la primera que encontremos):

```
$cfg['Servers'][$i]['AllowNoPassword'] = TRUE;
```

En la siguiente pantalla podremos crear la Base de datos:

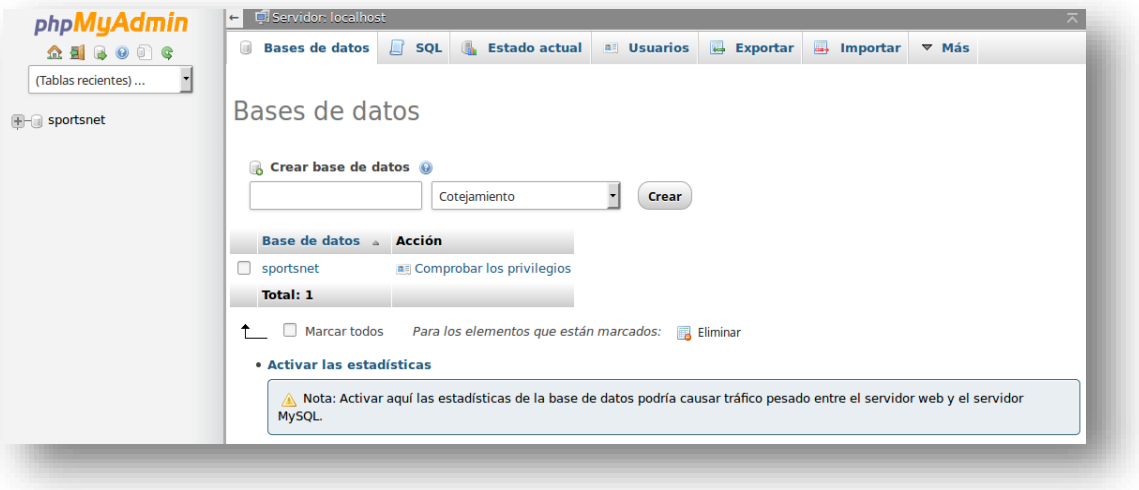


Imagen 29: Creación de la BD

Instalación de CakePHP

Instalando composer

Composer es un gestor de dependencias para PHP, del mismo modo que apt-get. yum, etc. lo son para Linux.
A través de Composer se pueden instalar proyectos como Symfony2, Zend Framework2, CakePHP3 y muchos otros más. Hay que resaltar que CakePHP2, por defecto, no sooprta la instalación a través de Composer, pero existe una forma de instalación avanzada que permite el uso de esta herramienta para la instalación.
Esta herramienta se encarga de descargar e instalar todas las dependencias que sean necesarias, incluyendo, por supuesto el propio framework.

En primer lugar deberemos situarnos en el directorio[7] dónde queramos instalar composer y una vez allí ejecutamos:

```
curl -s https://getcomposer.org/installer | php
```

Una vez ejecutado este comando tendremos un fichero *composer.phar*. Para facilitar su uso posterior crearemos un enlace en ese mismo directorio.

```
sudo ln –s composer.phar composer
```

En ese mismo directorio crearemos el directorio *.comporser.* El directorio dónde se encuentra composer y el directorio .composer, deben de ser accesibles y tener permisos de escritura para usuario que va a ejecutar composer.

Si en el momento de ejecutar composer vemos un mensaje de este tipo

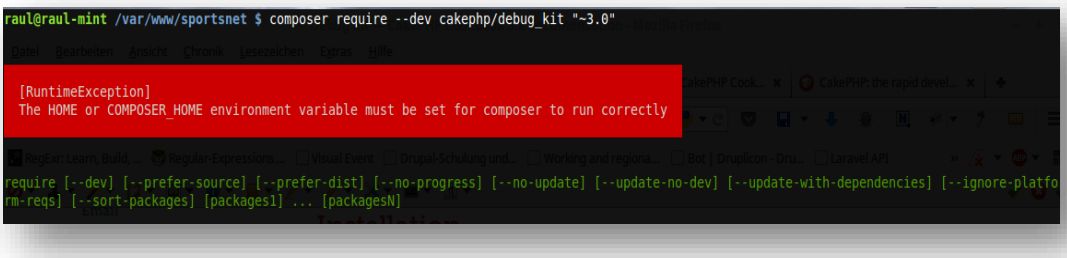


Imagen 30: Error al ejecutar composer

[7] Habitualmente se suele instalar en /usr/local/bin

Para eliminar este error editamos (si no existe, lo creamos) el fichero *.bashrc* del directorio del usuario que va a ejecutar composer y añadimos:

```
export COMPOSER_HOME=/usr/local/bin/.composer
```

Para que el cambio tenga efecto, debemos desloguearnos y volver a iniciar sesión.

Instalación de CakePHP a través de composer

Una vez tengamos composer instalado es muy sencillo instalar el framework php.

1.- Nos situamos en directorio que hayamos configurado como *documentRoot* en Apache. Habitualmente será */var/www/* o */var/htdocs*

2.- Desde ese directorio ejecutamos

```
composer create-project --prefer-dist cakephp/app [app_name]
```

Donde [app_name] es el nombre que tendrá nuestra aplicación, en ese caso "sportsnet"
Durante la instalación nos hará la siguiente pregunta:

Set Folder Permissions ? (Default to Y) [Y,n]?

Nos está preguntando si queremos que establezca los permisos de los directorios. De esta forma CakePHP establecerá los permisos para garantizar la seguridad de los directorios. Contestamos "Y" y la instalación finalizará. Ahora tendremos un directorio llamado "sportsnet" con la siguiente estructura:

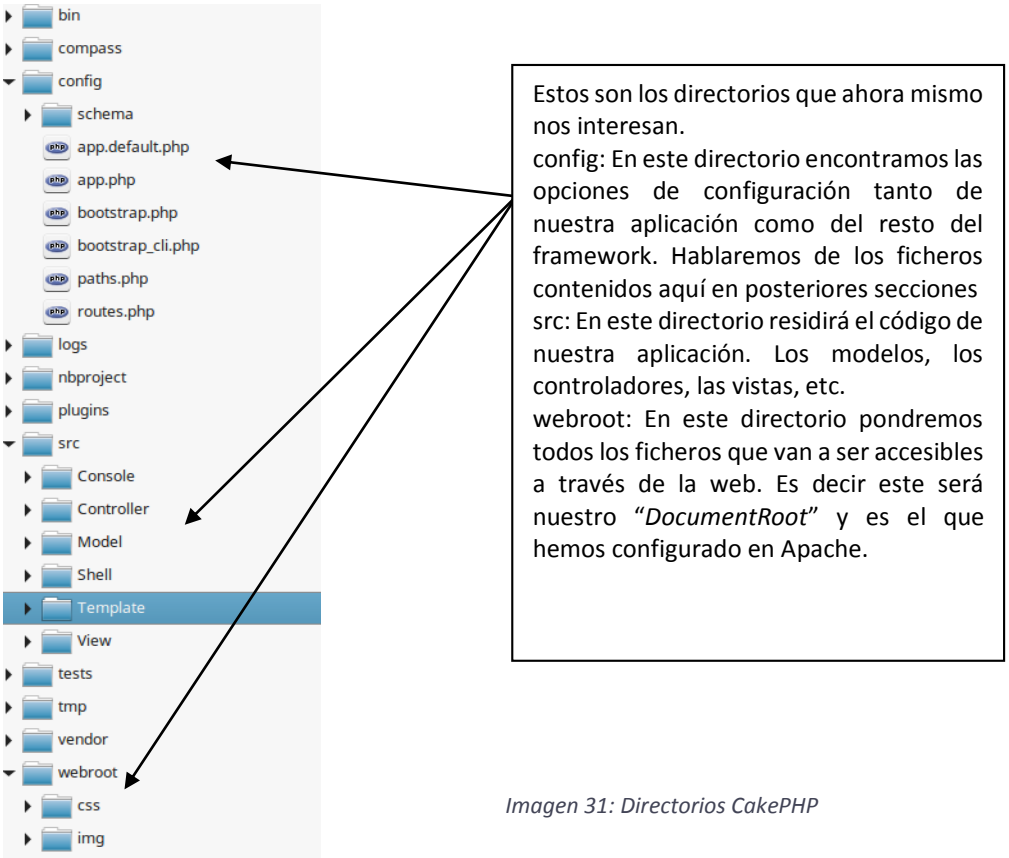
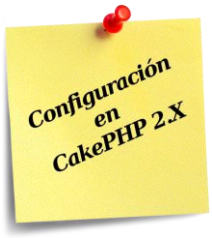


Imagen 31: Directorios CakePHP



En CakePHP 2 la estructura de directorios es bastante diferente.

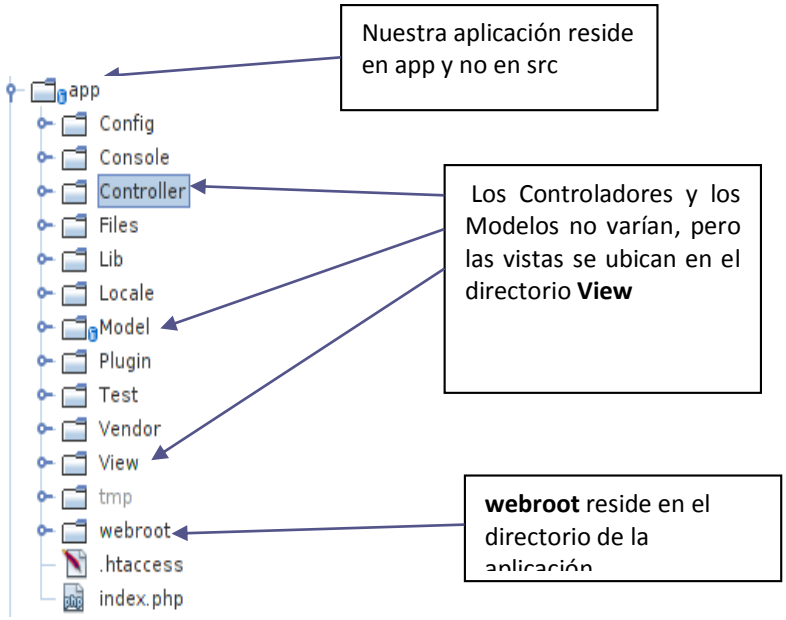


Imagen 32: Estructura de directorios CakePHP 2

Configuración de Apache

Por último configuraremos nuestro VirtualHost que le indicará a nuetgro servidor Apache en que directorio tiene que mirar cuando reciba la dirección http://sportsnet.local que hemos definido en el fichero hosts.
Realizamos este paso el último para asegurarnos que al reiniciar el servidor Apache, todo lo que necesitanmos (ÜHP, MySQL, CakePHP y el propio Apache) esté ya configurado. Ya que, de no hacerlo así y hacerlo antes de que el directorio sportsnet exista, Apache nos informaría de que el directorio no existe.

Crear el Virtuallhost:

Crear el fichero "*/etc/apache2/sites-avaliable/sportsnet.conf*"

```
sudo vim /etc/apache2/sites-avaliable/sportsnet.conf
```

Con el siguiente contenido:

```
<VirtualHost *:80>
        ServerAdmin webmaster@localhost
        ServerName sportsnet.local
        DocumentRoot /var/www/sportsnet/webroot
        <Directory /var/www/sportsnet/webroot>
          Require all granted
          Options Indexes FollowSymLinks
          AllowOverride All
        </Directory>
        ErrorLog /var/www/sportsnet-error.log
        CustomLog /var/www/sportsnet-access.log combined
</VirtualHost>
```

En posteriores secciones veremos que es el directorio webroot y porque lo hemos indicado aquí como DocumentRoot.

Una vez creado este fichero hay que habilitar el VirtualHost en Apache, de forma que éste sepa que este VirtualHost existe. Esto se puede hacer de dos formas:

Forma 1: manual

```
cd /etc/apache2/sites-enabled/
sudo ln -s ../sites-avaliable/sportsnet.conf .
```

Forma 2: automático

```
a2ensite  sportsnet   (habilitar sitio)
a2dissite  sportsnet  (deshabilitar sitio)
```

Por último hay que asegurarse que el modulo "*mod_rewrite*" de Apache está instalado y correctamente cargado. Habitualmente suele estarlo en cualquier instalación estándar de Apache.

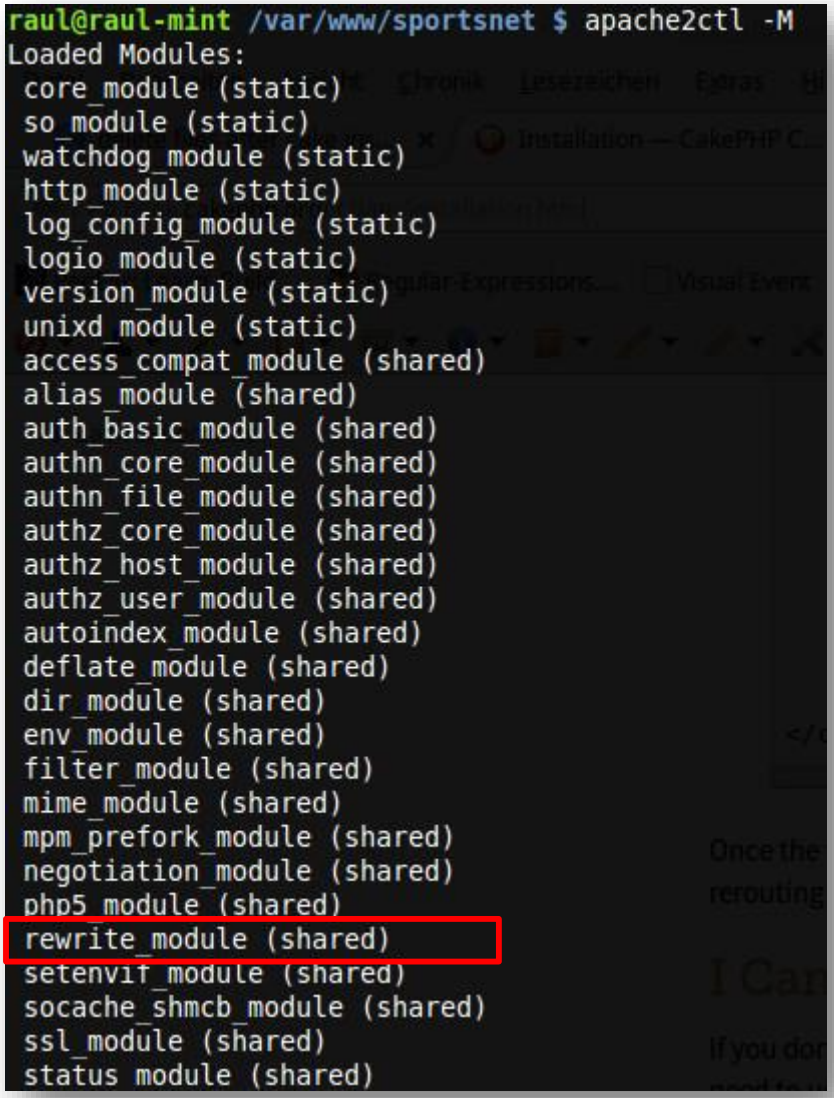


Imagen 33: Listado de Módulos de Apache

Si por casualidad no estuviera cargado el modulo "rewrite", ejecutaremos el siguiente comando:

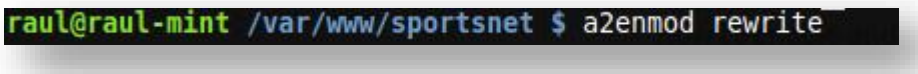


Imagen 34: Habilitar módulos en Apache

Instalación y configuración de SASS y Compass

Procedamos ahora a su instalación por el método habitual:

```
sudo apt-get install ruby-full rubygems1.9
sudo gem install sass
sudo gem install compass
```

Además de la instalación básica nos vendrá bien la instalación de la siguiente extensión de SASS.

```
sudo gem install sass-globbing
```

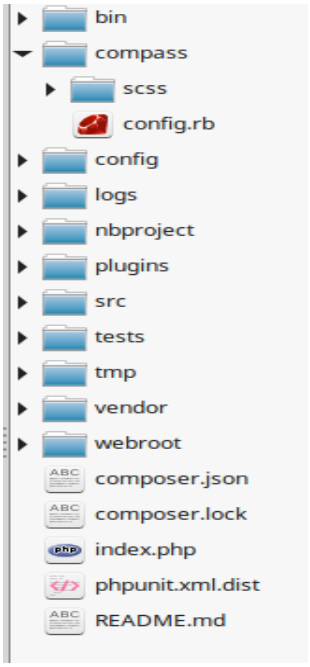
Esta extensión permite referenciar imports en compass usando comodines, como por ejemplo:

@import "componentes/*";

De esta forma nos aseguramos que todos los ficheros que estén en el directorio componentes sean agregados automáticamente sin tener que referenciarlos explícitamente en el fichero principal.

En el momento de la compilación es posible que compass nos informe de que no encuentra alguna librería. Para subsanar este error procederemos como hemos hecho para instalar sass-globbin, es decir, escribiendo en la línea de comandos

sudo gem install <libreria>



Para que compass sepa cómo trabajar con los ficheros .scss necesita ser configurado con ciertos parámetros. Dado que compass está basado en Ruby on Rails utiliza ficheros de configuración en formato Ruby y cuya extensión es *".rb"*

En el directorio principal de nuestra aplicación (/var/www/sportsnet o /var/htdocs/sportsnet) crearemos un nuevo directorio que llamaremos *"compass" y* dentro crearemos el fichero *"config.rb".* También crearemos el directorio *"scss"* que es dónde almacenaremos nuestros ficheros *".scss"*

Imagen 35: Directorio para compass

A continuación crearemos el fichero de configuración de compass para nuestra aplicación.

vim /var/www/sportsnet/compass/config.rb

```
# Default to development if environment is not set.
saved = environment
if (environment.nil?)
  environment = :development
else
  environment = saved
end

# Localizacion de los recursos

http_path = "/"
sass_dir = 'scss'
css_dir = '../webroot/css'
images_dir = '../webroot/img'
javascripts_dir = '../webroot/js'
http_stylesheets_path = 'css'
http_javascripts_path = 'js'
http_images_path = 'images'
environment = :development

# Si se necesitan reuqisito, indíquelos aquí. Las librerías deben estar instaladas en su sistema.
require 'compass-normalize'
require 'rgbapng'
require 'toolkit'
require 'breakpoint'
require 'singularitygs'
require 'susy'
require 'sass-globbing'
# Puede seleccionar el estilo de salida de compass (:expanded, :nested, :compact
# or :compressed).

output_style = (environment == :production) ? :expanded : :nested

# Output source maps in development mode.
sass_options = (environment == :production) ? {} : {:sourcemap => true}
```

Para hacer que compass esté observando los ficheros scss y genere el código automáticamente usamos el siguiente comando desde el mismo directorio dónde se almacena "config.rb". Es decir desde dentro de */var/www/compass/* ejecutaremos#

```
compass watch
```

y veremos la siguiente salida

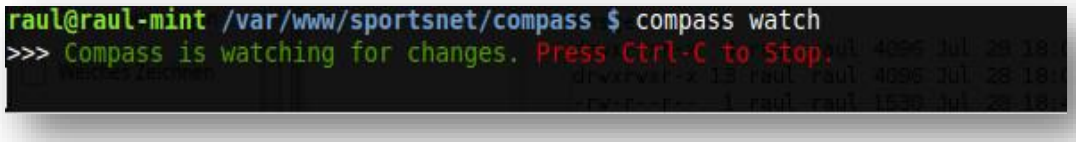


Imagen 36: Ejecución de compass

Ahora nos indica que compass está escuchando los cambios que se produzcan en el directorio que hemos indicado en el fichero *"config.rb"*, en el parámetro *"sass_dir"* y que la salida, en foma de ficheros css se producirá en el directorio definido en el parámetro *"css_dir".*

Para detener compass pulsaremos CTRL+C.

Reinicio de Apache y MySQL

Para finalizar tendremos que reiniciar Apche y MySQL. Si hubiérmos cometido algún error durante las configuraciones, ahora se mostrarán en la línea de comandos.

```
sudo service apache2 restart

sudo service mysql restart
```

Ajustando CakePHP y los ficheros .htaccess

Los ficheros **.htaccess** (*hypertext processor*) [8]son ficheros que usa el servidor web Apache y permiten definir diferentes directivas de configuración para cada directorio (con sus respectivos subdirectorios) sin necesidad de editar el archivo de configuración principal de Apache.

Todos los CMS contienen al algún/os lugar/es ficheros *.htaccess* que permiten controlar, entre otras cosas como se verán las direcciones URL que CakePHP manejará. Por ejemplo, permitirá sobreescribir direcciones como:
http://sportsnet.local/index.php/usuarios/ en http://sportsnet.local/usuarios

Apache, sabrá a partir del fichero **.htaccess,** que deberá poner el "index.php" entre la dirección base y la ruta solicitada y lo hará de forma transparente.

[8] http://httpd.apache.org/docs/current/howto/htaccess.html

Obervamos el fichero **.htaccess** de la carpeta principal de CakePHP en /var/www/sportsnet/.htaccess

```
<IfModule mod_rewrite.c>
    RewriteEngine on
    RewriteRule    ^$    webroot/    [L]
    RewriteRule    (.*) webroot/$1    [L]
</IfModule>
```

Imagen 37: htaccess directorio principal CakePHP

En este fichero no haremos, por el momento, ningún ajuste. En este fichero se le dice a Apache que cualquier dirección que contenga "webroot/" deberá sobreeescribirse y se omitirá mostrar este fragmento en la misma. Aunque la URL interna sí que contendrá ese fragmento. De esa forma tendremos direcciones como http://sportsnet.local/usuarios en lugar de http://sportsnet.local/webroot/usuarios.

El otro fichero que sí tendremos que ajustar se encuentra en la carpeta de nuestra aplicación: **/var/www/sportsnet/webroot/.htaccess**

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteBase /
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteCond %{REQUEST_URI} !^/(webroot/)?(img|css|js)/(.*)$
    RewriteRule ^ index.php [L]
</IfModule>
```

Imagen 38: htaccess directorio webroot CakePHP

En este fichero tendemos algunas configuraciones interesantes. No es el objetivo e este trabajo explicar en detalle las directivas de los ficheros .htaccess, pero en este caso, explicaremos las línea más interesantes.

RewriteCond %{REQUEST_URI} !^/(webroot/)?(img|css|js)/(.*)$

Esta línea evita que ciertos tipos de recursos incorrectos sean procesados por CakePHP (concretamente por el fichero index.php), mejorando así la productividad del servidor.

En CakePHP 2 la recomendación para la evitar que los ficheros con tipo incorrecto fueran procesados por CakePHP era crear un enlace simbólico en la carpeta *webroot* hacia la carpeta de recursos.

ln –s app/Plugin/YourPlugin/webroot/css/yourplugin.css app/webroot/css/yourplugin.css

Probando nuestra instalación de CakePHP

Si hemos hecho todo bien, ya podremos probar nuestra nueva instalación de CakePHP. Si visitamos la dirección URL: http://sportsnet.local veremos la página de inicio siguiente.

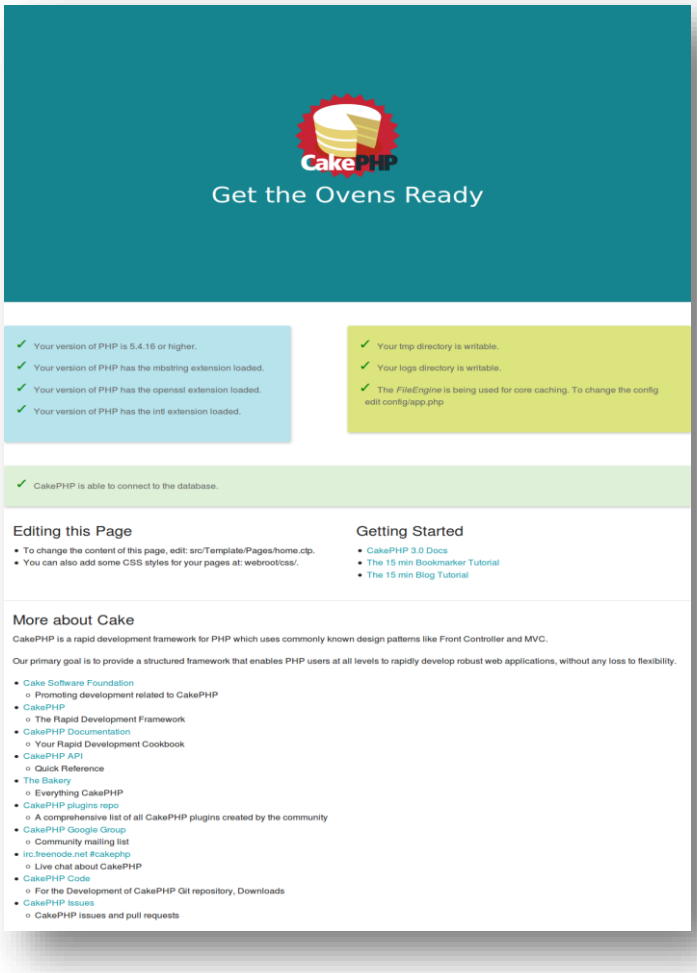


Imagen 39: Página principal de CakePHP

En la parte inferior de la ventana veremos el icono de la barra de herramientas de DebugKit.

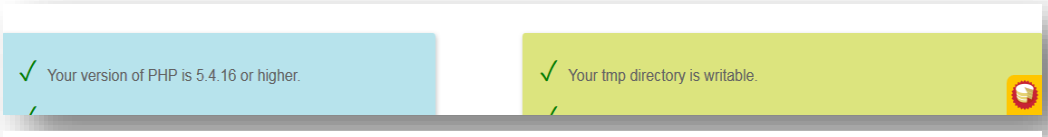


Imagen 40: Icono de DebugKit

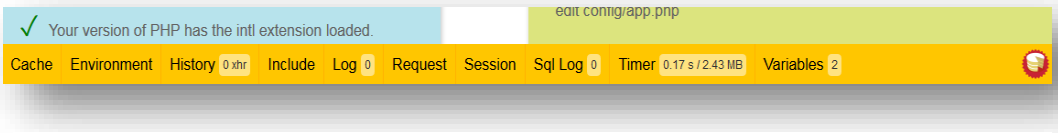


Imagen 41: Barra de herramientas de DebugKit

Más información sobre DebugKit en:

http://book.cakephp.org/3.0/en/debug-kit.html#database-configuration

5.7 Creación de las tablas de la bases de datos para SPORTSNET

La creación de las tablas de la base de datos se realizará a través de phpMyAdmin y se creará en base al diagrama de clases que presentamos en el capítulo 3

Tablas

Los nombres de las tablas y los campos de las mismas serán creados siguiendo las convenciones[9] de CakePHP (http://book.cakephp.org/3.0/en/intro/conventions.html):

- Las tablas deben nombrase en plural y en caso de estar formado por varias palabras, deben ser separadas por guiones bajos. Algunos ejemplos serían: *usuarios, areas, actividad_deportivas, horarios_reservas, grupo_usuarios.*
-

- Las tablas con relaciones “muchos a muchos” se nombran usando el nombre de cada tabla, en orden alfabético y en plural.

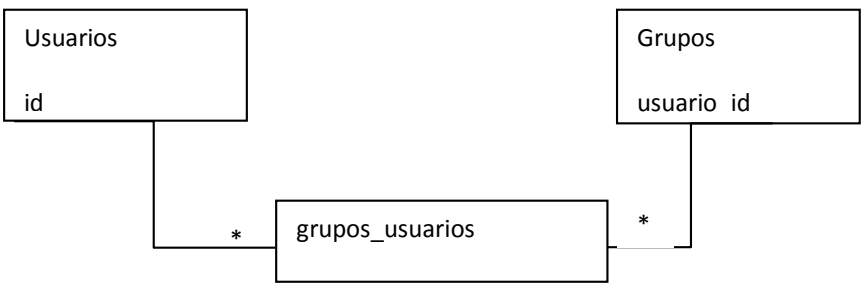


Imagen 42: Relación "muchoas a muchoes"

[9] Al leer los plurales en español, no tienen el sentido correcto. Hay que tener en cuenta que las convenciones de CakePHP y, en general, cualquier convención en el mundo de la programación, proviene del inglés. Si escribimos los nombres en inglés todo tiene bastante más sentido.

Campos de las Tablas

- Los campos que contengan más de una palabra serán escritos separados por un guión bajo, p. ej: *nombre_usuario.*

- Las claves foráneas o ajenas (foreign keys) en las relaciones “a muchos” (hasMany), “pertenece a” (belongsTo) y “a uno” (hasOne) se reconocen por defecto si el nombre del campo se escribe usando el singular de la tabla con la que se relaciona y terminando en **_id.**

Por ejemplo el modelo Usuario tiene una relación “a muchos” con el modelo Mensaje. En la tabla mensajes escribiremos un campo con el nombre usuario_id. En caso de que el nombre de la tabla de la clave foránea tenga varias palabras como sería, por ejemplo, en *tipo_usuarios*, la clave sería *tipo_usuario_id*.

Imagen 43: Relación "a muchos"

Todas las tablas que use CakePHP deben tener una clave primaria (id), excepto las tablas de las relaciones “muchos a muchos”, que no tienen por qué tener una clave principal id. Se puede usar, en lugar de un tipo auto incrementable, un tipo carácter de longitud 36 (char(36)). En ese caso CakePHP usará un UUID único de 36 caracteres al salvar cada nuevo registro mediante el método *Table::save()*.

Naturalmente se puede seguir el criterio que a uno más le convenga, pero si nos acostumbramos a usar las convenciones del framework, nuestro trabajo será mucho más sencillo y aprovecharemos realmente las ventajas y la potencia de CakePHP.

Aunque las tablas se crean usando phpMyAdmin, como ya se ha indicado, se adjunta aquí el código SQL necesario para crearlas. Se puede usar este código desde la pestaña SQL de phpMyAmin, una vez hemos seleccionado la base de datos.

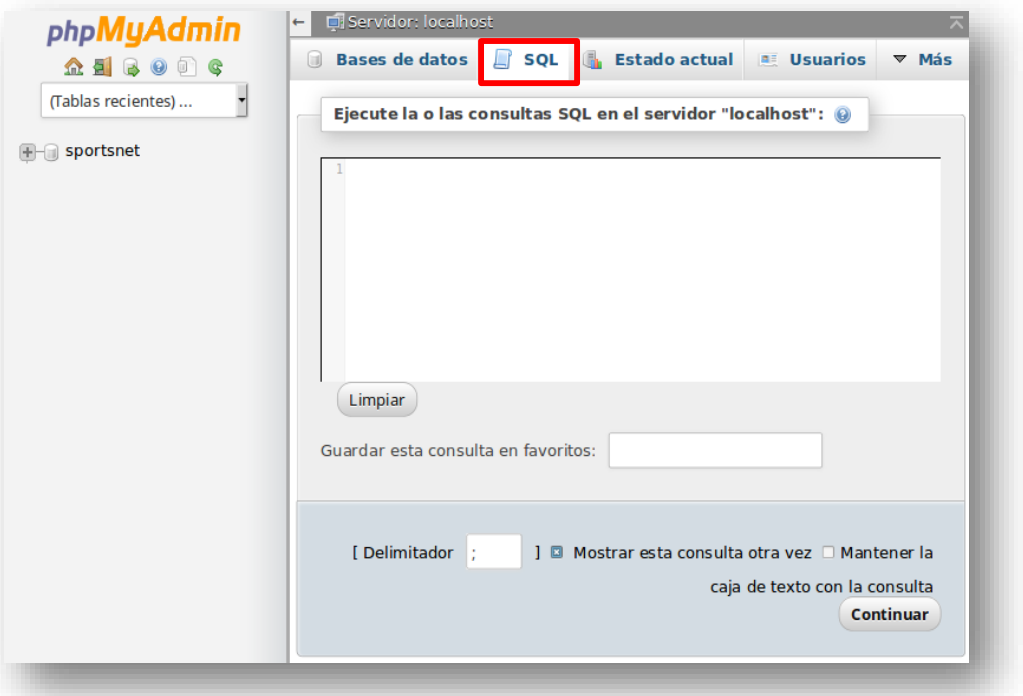


Imagen 44: Ventana SQL en phpMyAdmin

```
REATE TABLE IF NOT EXISTS `usuarios` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(50) NOT NULL,
  `apellidos` varchar(100) NOT NULL,
  `email` varchar(50) NOT NULL,
  `password` varchar(255) NOT NULL,
  `imagen` varchar (255) NULL
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `modified` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `email` (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS `roles` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(255) CHARACTER SET utf8 NOT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `modified` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS `espacios` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nombre` varchar(255) CHARACTER SET utf8 NOT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `modified` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS `espacios_usuarios` (
 `usuario_id` int(11) NOT NULL,
 `espacio_id` int(11) NOT NULL,
 `role_id` int(11)  NOT NULL,
 `admitido` boolean  NOT NULL DEFAULT FALSE,
 `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
 `modified` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`usuario_id`, `espacio_id`, `role_id`),
 FOREIGN KEY usuario_key(usuario_id) REFERENCES usuarios(id),
 FOREIGN KEY espacio_key(espacio_id) REFERENCES espacios(id),
 FOREIGN KEY role_key(role_id) REFERENCES roles(id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS `comentarios` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `titulo` varchar(255) CHARACTER SET utf8 NOT NULL,
 `texto` text CHARACTER SET utf8 NOT NULL,
 `espacio_id` int(11) NOT NULL,
 `usuario_id` int(11) NOT NULL,
 `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
 `modified` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1;
```

```
CREATE TABLE IF NOT EXISTS `respuestas` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `titulo` varchar(255) CHARACTER SET utf8 NOT NULL,
 `texto` text CHARACTER SET utf8 NOT NULL,
 `comentario_id` int(11) NOT NULL,
 `usuario_id` int(11) NOT NULL,
 `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
 `modified` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS `anuncios` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `titulo` varchar(255) CHARACTER SET utf8 NOT NULL,
 `texto` text CHARACTER SET utf8 NOT NULL,
 `precio`  decimal( 10, 2 ) NOT NULL,
 `espacio_id` int(11) NOT NULL,
 `usuario_id` int(11) NOT NULL,
 `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
 `modified` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS `mensajes` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `titulo` varchar(255) CHARACTER SET utf8 NOT NULL,
 `texto` text CHARACTER SET utf8 NOT NULL,
 `precio`  decimal( 10, 2 ) NOT NULL,
 `anuncio_id` int(11) NOT NULL,
 `usuario_id` int(11) NOT NULL,
 `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
 `modified` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```
CREATE TABLE IF NOT EXISTS `eventos` (
 `id` int(11) NOT NULL AUTO_INCREMENT,
 `titulo` varchar(255) CHARACTER SET utf8 NOT NULL,
 `descripion` text CHARACTER SET utf8 NOT NULL,
 `fecha`  timestamp NULL,
 `espacio_id` int(11) NOT NULL,
  `usuario_id` int(11) NOT NULL,
 `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,
 `modified` timestamp NULL DEFAULT NULL,
 PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Consideraciones sobre la base de datos:

- La tabla espacios_usuarios contiene una clave primaria compuesta. A diferencia de CakePHP 2.x, dónde las claves compuestas no eran soportadas, CakePHP 3.x soporta las claves compuestas casi en cualquier situación.

- Los campos created y modified se han añadido para poder llevar un mínimo control de los cambios que se realizan en los registros. Estos campos, como veremos durante el desarrollo, los manejará CakePHP automáticamente.

- En algunas tablas hay un campo fecha que se inicializa a NULL. El campo se habría inicializado a CURRENT_TIMESTAMP que permitiría asignar automáticamente la fecha actual (en formato *timestamp)* cuando se crea un registro nuevo. Lamentablemente sólo se puede tener en cada tabla una columna *timestamp* inicializada a CURRENT_TIMESTAMP y en este caso, se ha preferido que esa columna sea created por motivos de administración y gestión de dichas tablas.

5.8 Creación de los Modelos, Controladores y Vistas para SPORTSNET

La creación de Modelos, Controladores y Vistas, en adelante MVC, puede hacer en CakePHP de dos formas. La forma manual o usando la consola Bake.

Por comodidad usaremos la consola Bake que nos permitirá crear nuestros objetos a través de sencillas preguntas. Como su nombre indica, los comandos se tienen que ejecutar en la consola.

Pero antes de empezar deberemos atender a las convenciones, que al igual que para los nombres de las tablas de la base de datos, existen para nombrar los MVC.

Nombres de los ficheros

En general, los nombres de los ficheros coinciden con el nombre de las clases y siguen los standard PSR-4[10] para la autocarga de clases. El nombre de los ficheros, al igual que el de las clases, como veremos a continuación, se escribe en plural con formato *CamelCase*[11], es decir con la primera letra de cada palabra en mayúscula y sin separar las palabras. El nombre termina con el tipo de objeto al que se refiere el fichero (MVC). Algunos ejemplos de nombres de ficheros serían:

- TablasHelper.php => contendría la clase TablasHelper
- UsuariosController.php => contendría la clase UsuariosController

Los ficheros se almacenan en un directorio cuya ruta coincide con su espacio de nombres (*Namespace*[12]).

Por ejemplo, la clase *UsuariosController*, tendrá un espacio de nombres en la siguiente ruta: ***src/Controller/Usuarios/***

A continuación veremos, por separado, las especificaciones de nombres para Modelos, Vistas y Controladores. Cuando nos referimos al "nombre" de una Vista, Modelo o Controlador, nos referimos al nombre de la clase que lo representa.

Controladores

Los nombres de los controladores se escriben plural con *CamelCase* y sin espacios. Y terminan con la palabra *Controller*.
Algunos ejemplos de nombres de controladores para nuestra aplicación serían:

- UsuariosController
- EspaciosController
- EspaciosUsuariosController

La ruta habitual para los Controladores es s***rc/Controller<NombreDelControlador>***

Por ejemplo:

```
src/Controller/UsuariosController.php
```

Modelos

Los nombre de los modelos, siguen el mismo principio de nomenclatura que los controladores, pero teniendo en cuenta que el nombre del modelo debe coincidir con el nombre de la tabla a la que representan. Así por ejemplo, el modelo que corresponde a la tabla de espacios_usuarios, se llamará EspaciosUsuarios y se almacenan en un fichero con nombre *EspaciosUsuarios.php*

La ruta habitual para los Modelos es src/Model/<TipoDeModelo>/<NombreDelModelo>

Por ejemplo:

```
src/Model/Table/EspaciosUsuarios.php
```

[10] http://www.php-fig.org/psr/psr-4/
[11] *CamelCase*: La primera letra de cada palabra se escribe en mayúscula.
[12] http://php.net/manual/es/language.namespaces.rationale.php

Nota: El uso de un subdirectorio *<TipoDeModelo>* no es obligatorio pero ayuda a tener una mejor organización de los ficheros y las clases.

Vistas

Los nombres de las plantillas que actuaran como vistas, se corresponden con los nombres de las funciones del controlador de las que reciben los datos que van a mostrar. El nombre del fichero se escribe en minúscula y, si la función tiene más de una palabra, se separarán con un guión bajo.

El patrón básico para nombrar las vistas es
src/Template/<Controller>/nombre_de_la_función_con_guiones_bajos.ctp.

Supongamos que tenemos una función en el controlador "Usuarios" que se llama listaUsuarios().

En este caso, la vista sería:

```
src/Template/Usuarios/lista_usuarios.ctp.
```

Entidades

Mientras que los modelos de tipo Table representan a conjuntos de objetos (p.ej: tablas completas de la base de datos), los modelos del tipo Entity representan registros individuales de esas tablas. Es decir, lo que llamaríamos instancias de una clase.

Estas entidades contienen las propiedades y métodos necesarios para acceder y gestionar a los datos que contienen.

Aunque no es necesario el uso de entidades para manejar la base de datos,

Si usamos todas estas convenciones nos evitaremos muchos problemas y facilitaremos la gestión y el mantenimiento de nuestra aplicación.

Veamos ahora, a modo de ejemplo, un resumen de todas estas convenciones.

- Tabla en base de datos: "usuarios"
- Clase Table: "UsuariosTable" => src/Model/Table/UsuariosTable.php
- Clase Entidad: "Usuario" => **src/Model/Entity/Usuario.php**
- Controlador: "UsuariosController" => src/Controller/UsuariosController.php
- Vista => src/Template/Usuarios/index.ctp

Nótese que sin configurar nada más, CakePHP sabe que cuando recibe la URL http://sportsnet.local/usuarios/ tiene que realizar una llamada a la función index() del **Controlador** UsuariosController, donde el **Modelo** Usuario está automáticamente disponible (y, por supuesto, enlazado directamente con la tabla 'usuarios'), y la salida se enviará a la **Vista** index.ctp.

La consola Bake

La consola *bake* se ejecuta sobre la línea de comandos y para poder usarla hay que asegurarse de que el cliente PHP está instalado.

A través de *bake* podemos generar todos los objetos necesarios de nuestra aplicación. La ejecución del comando sin ningún parámetro devuelve el listado de opciones que podemos usar.

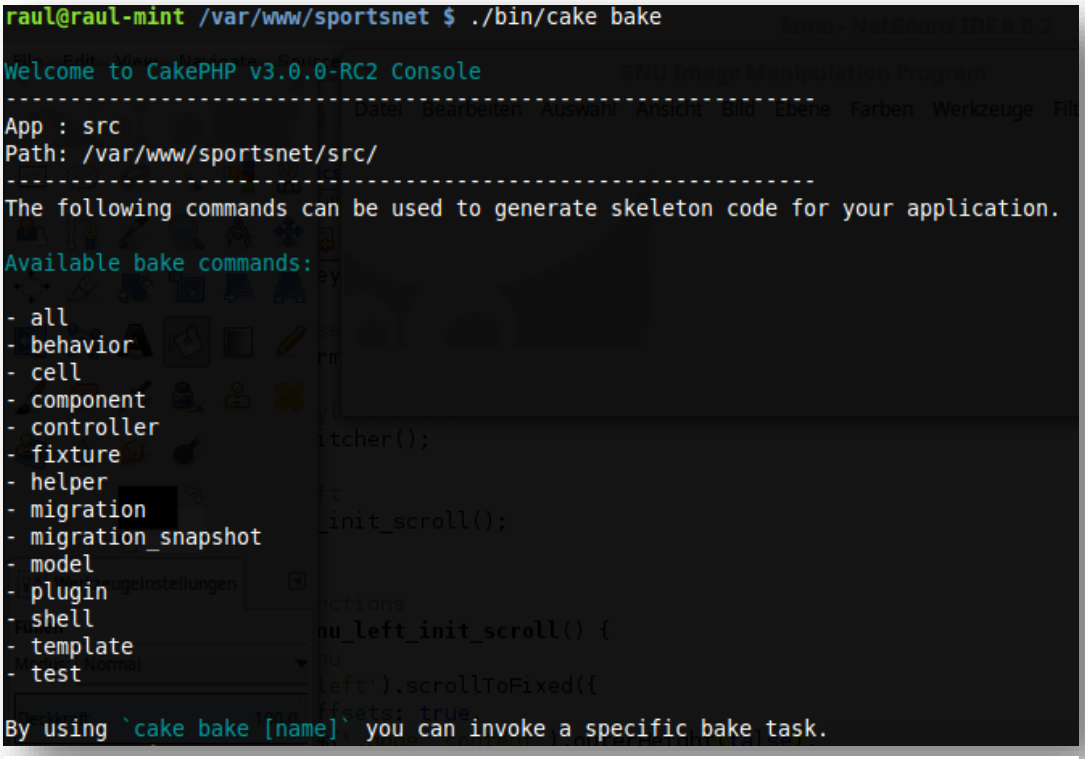


Imagen 45: Consola bake

Es recomendable que en este punto, CakePHp tenga configurada ya la bases de datos y que pueda conectarse con ella. Si no lo ha hecho todavía consulte el apartado "5.1.3 ***Detalles de implementación***".

Crear los Modelos

Usando la opción "model" de bake nos mostrará una lista de modelos que podemos crear, basándose en los objetos que existan en la base de datos.

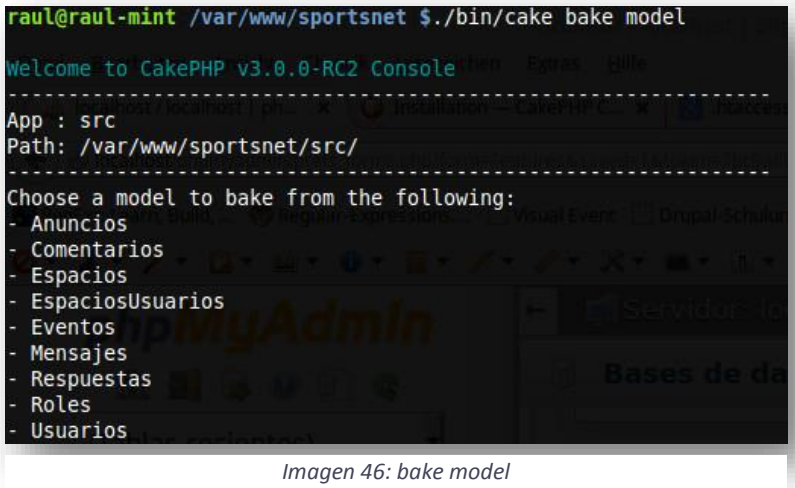


Imagen 46: bake model

Vamos a crear el modelo para “Anuncios” con el siguiente comando.

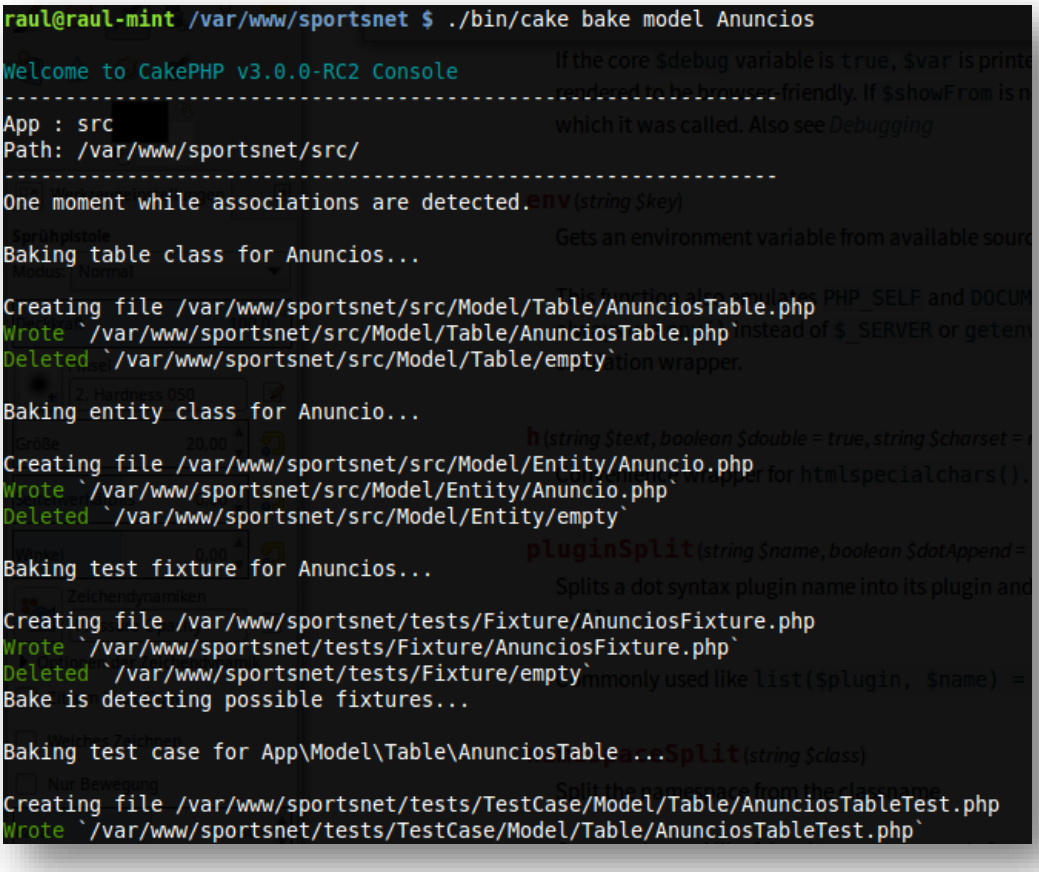


Imagen 47: Creación del modelo para Anuncios

Como podemos ver, al crear el modelo, CakePHP ha generado por nosotros la Entidad, y los archivos de Test para la realización de Tests Unitarios de la aplicación.
Además ha generado las clases Fixture correspondientes que nos permitirán insertar datos de prueba.
En la imagen vemos las rutas dónde estos ficheros han sido guardados.

Crear los Controladores

La creación de controladores se realiza de la misma forma que para los modelos, pero usando en este caso "*bake controller*"

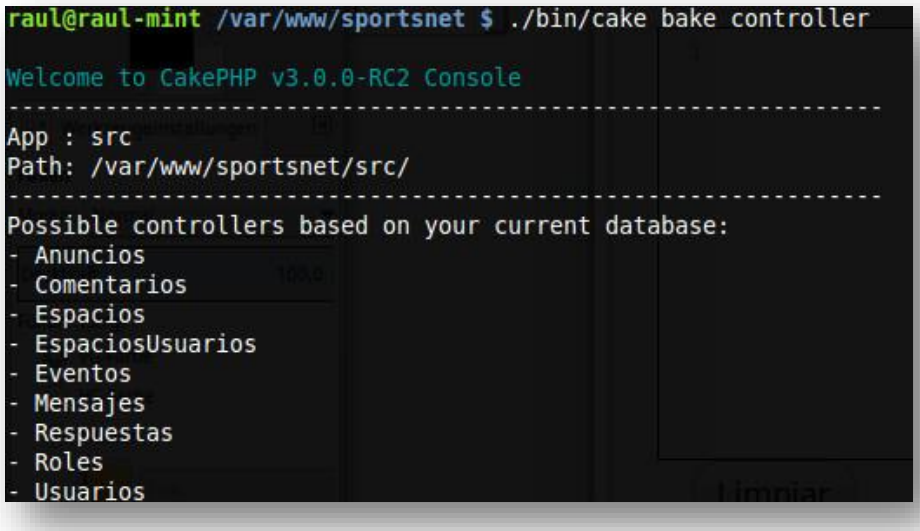


Imagen 48: Creación de controladores

Crear las Plantillas (Vistas)

Del mismo modo procedemos con las plantillas, pero esta vez el comando es "*bake template*"

Crear todos los objetos a la vez

La consola de bake permite realizar todos estos pasos de una sola vez. Poe ejemplo, vamos a crear los MVC de “Eventos”.

```
raul@raul-mint /var/www/sportsnet $ ./bin/cake bake all eventos
Welcome to CakePHP v3.0.0-RC2 Console
---------------------------------------------------------------
App : src
Path: /var/www/sportsnet/src/
---------------------------------------------------------------
Bake All
---------------------------------------------------------------
One moment while associations are detected.

Baking table class for Eventos...

Creating file /var/www/sportsnet/src/Model/Table/EventosTable.php
Wrote `/var/www/sportsnet/src/Model/Table/EventosTable.php`

Baking entity class for Evento...

Creating file /var/www/sportsnet/src/Model/Entity/Evento.php
Wrote `/var/www/sportsnet/src/Model/Entity/Evento.php`

Baking test fixture for Eventos...

Creating file /var/www/sportsnet/tests/Fixture/EventosFixture.php
Wrote `/var/www/sportsnet/tests/Fixture/EventosFixture.php`
Bake is detecting possible fixtures...

Baking test case for App\Model\Table\EventosTable ...

Creating file /var/www/sportsnet/tests/TestCase/Model/Table/EventosTableTest.php
Wrote `/var/www/sportsnet/tests/TestCase/Model/Table/EventosTableTest.php`

Baking controller class for Eventos...

Creating file /var/www/sportsnet/src/Controller/EventosController.php
Wrote `/var/www/sportsnet/src/Controller/EventosController.php`
Bake is detecting possible fixtures...

Baking test case for App\Controller\EventosController ...
```

Imagen 49: Crear todos los objetos a la vez

Una vez todos los objetos ha sido generados podemos ver como CakePHP ha guardado cada fichero de clase dónde le corresponde.

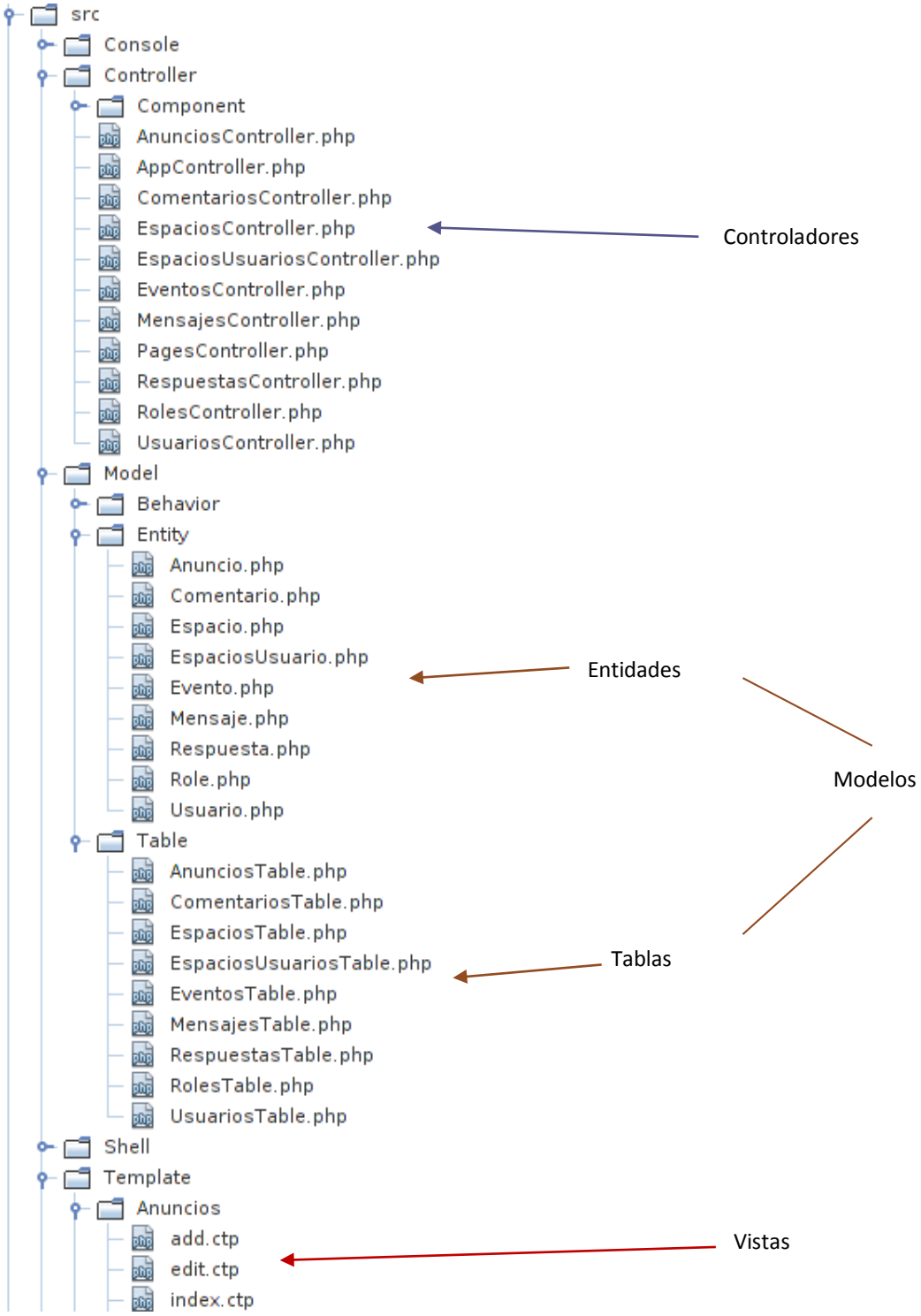


Imagen 50: Ficheros creados por bake

Llegados a este punto, sólo nos queda probar que tal funciona nuestra aplicación. Así que nos dirigimos a la URL: http://sportsnet.local/espacios

Tal como hemos visto, CakePHP reconocerá esta dirección y ejecutará el método index() del controlador EspaciosController.

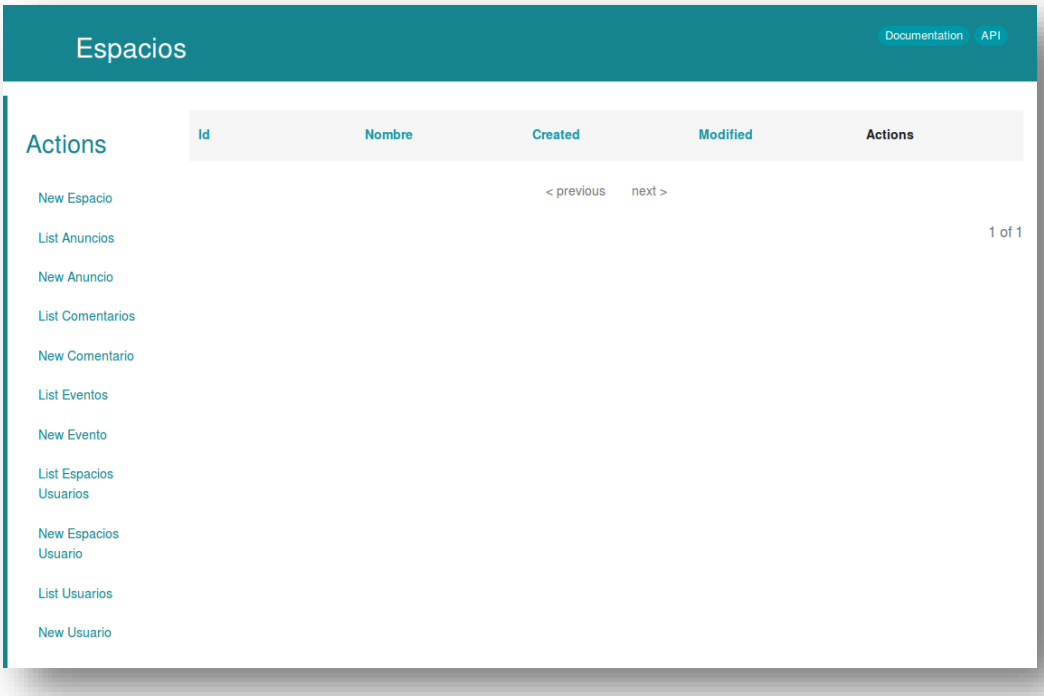


Imagen 51: Gestion de Espacios

Como vemos, la aplicación permite realizar todas las operaciones básicas con los espacios. Es cierto que la aplicación es muy rudimentaria, pero, para no haber escrito nada de código, no está nada mal, ¿no?

El lector puede probar con otros controladores de la aplicación como eventos, mensajes, usuarios, etc.

Creamos un nuevo usuario y vemos dos cosas que nos llaman la atención:

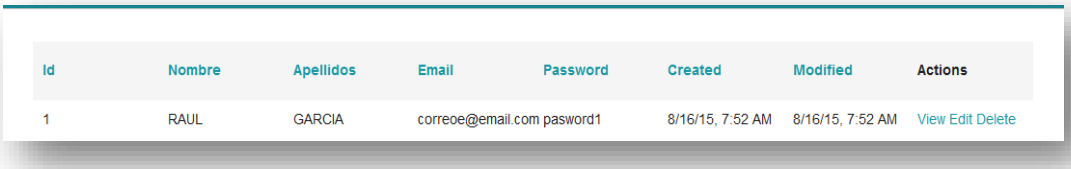


Imagen 52: Nuevo usuario.

1.- El password se almacena en texto claro.
2.- El formato de las fechas no es el español.

De momento dejamos este tema pendiente y más tomaremos las medidas adecuadas para corregir estas dos situaciones.

5.9 Un sistema de login para nuestra aplicación

Por el momento nuestra aplicación, aunque muy rudimentaria, es capaz de trabajar con datos personales, así que vamos a crear un sistema para que sólo los usuarios registrados puedan acceder.

En nuestra aplicación usaremos el *email* como nombre de usuario. Si usáramos "*username*", CakePHP será capaz de auto-configurar muchas cosas por nosotros. No obstante podremos decirle a CakePHP que campo tiene que usar como "username" y así seguir aprovechando las ventajas que el framework nos ofrece.

En CakePHP la autenticación es manejada por el controlador AuthComponent que se encuentra en el espacio de nombres Cake\Controller\Component\AuthComponent
Lo primero que haremos pues, será decirle a nuestra aplicación que use este controlador y eso lo hacemos editando el fichero var/www/src/Controller/AppController.php y añadiendo las siguientes líneas:

```php
/**
 * Initialization hook method.
 *
 * Use this method to add common initialization code like loading components.
 *
 * @return void
 */
public function initialize() {
  $this->loadComponent('Flash');
  $this->loadComponent('Auth', [
    'loginAction' => [
      'controller' => 'Usuarios',
      'action' => 'login',
      'plugin' => null
    ],
    'loginRedirect' => [
      'controller' => 'Espacios',
      'action' => 'index'
    ],
    'logoutRedirect' => [
      'controller' => 'Pages',
      'action' => 'display',
      'home'
    ],
    'authenticate' => [
      'Form' => [
        'fields' => ['username' => 'email']
      ]
    ],
  ]);
}

public function beforeFilter(Event $event) {
  $this->Auth->allow(['index', 'view', 'display']);
}
```

Aquí le decimos a CakePHP que controlador debe usar y que función (action) debe ejecutarse cuando se efectúe un *login* un cuáles cuando se efectúe un *logout.* Además le indicamos que en el formulario el campo *email* se usará como *username*.

Con esta función le decimos a CakePHP que no se necesita autenticación para las acciones index(), view().

Imagen 53: Activando y configurando controlador de autenticación

```php
'loginAction' => [
    'controller' => 'Users',
    'action' => 'login',
    'plugin' => null
],
```

Imagen 55: Configuración AuthComponent por defecto

```php
'loginAction' => [
  'controller' => 'Usuarios',
  'action' => 'login',
  'plugin' => null
],
```

Imagen 55: Nuestra configuración AuthComponent

Nótese que internamente, la acción "*loginAction*" está configurada por defecto para cargar el controlador "*Users*" y ejecutar la acción "*login*" del mismo. Al igual que nos ha ocurrido con el campo de usuario, tenemos que indicarle al controlador Auth, qué controlador debe cargar en lugar de "*Users*" y qué acción del mismo debe ejecutar. En este caso la acción a ejecutar coincide con la nuestra.

Los usuarios deben de ser capaces de registrarse en la aplicación, por lo que deberemos permitir que la acción add() pueda ser ejecutada sin hacer login previamente y por supuesto, los usuarios deben de ser capaces de desloguearse, hacer logout().

Al igual que hemos hecho en AppController implementaremos la función **beforeFilter()** pero en el controlador de usuarios.

En /var/www/sportsnet/src/Controller/UsuariosController.php añadiremos las siguientes funciones:

```
public function beforeFilter(Event $event) {
    // Permite a los usuarios registrarse y desloguearse.
    // Aquí no debería añadirse la acción de "login", ya que podría causar
    // problemas con el normal funcionamiento de AuthComponent
    $this->Auth->allow(['add', 'logout']);
}
```

Imagen 56: Permisos sobre acciones

Ahora implementemos las acciones propias que permitirán el login/logout

```
public function login() {
    if ($this->request->is('post')) {
        $user = $this->Auth->identify();
        if ($user) {
            $this->Auth->setUser($user);
            return $this->redirect($this->Auth->redirectUrl());
        }
        $this->Flash->error(__('Invalid username or password, try again'));
    }
}

public function logout() {
    return $this->redirect($this->Auth->logout());
}
```

Imagen 57: Login/logout

Ahora los usuarios pueden registrarse y autentificarse contra la aplicación. Pero antes de permitir el registro retomemos un tema que habíamos dejado pendiente al final del apartado anterior

Guardar los passwords cifrados (Password hashing)

Para evitar problemas de seguridad es imprescindible que los passwords de los usuarios se guarden siempre cifrados. Para eso usaremos el controlador "DefaultPasswordHasher" (Cake\Auth\DefaultPasswordHasher), que por defecto usa el algoritmo ***blowfish*** [13] que, actualmente, se considera el cifrado más fuerte usado en la mayoría de empresas.

Esta acción se tiene que hacer en cada usuario, es decir en cada instancia de la clase Usuario. Recordemos que tenemos un modelo para manejar este tipo de objetos, nos referimos, a los modelo del tipo Entity.

Por tanto las modificaciones las haremos en
/var/www/sportsnet/src/Model/Entity/Usuario.php

```php
namespace App\Model\Entity;

use Cake\ORM\Entity;
use Cake\Auth\DefaultPasswordHasher;

/**
 * Usuario Entity.
 */
class Usuario extends Entity {

  /**
   * Fields that can be mass assigned using newEntity() or patchEntity().
   *
   * @var array
   */
  protected $_accessible = [
    'nombre' => true,
    'apellidos' => true,
    'email' => true,
    'password' => true,
    'espacios' => true,
  ];

  protected function _setPassword($password) {
    return (new DefaultPasswordHasher)->hash($password);
  }
}
```

Imagen 58: Hashing passwords

[13] https://es.wikipedia.org/wiki/Blowfish

En CakePHP 2 se recomendaba el uso de *BlowfishPasswordHasher*, ya que el método que se usaba hasta esa versión " SimplePasswordHasher" se eliminaría de la versión 3.x como así ha sido. BlowfishPasswordHasher es el precursor del actual método por defecto "DefaultPasswordHasher" ya que usa el mismo algoritmo de cifrado. Además vemos que se hacía en el modelo directamente ya que CakePHP2.x no tenía soporte para Entidades (Entities).

```php
// app/Model/User.php

App::uses('AppModel', 'Model');
App::uses('BlowfishPasswordHasher', 'Controller/Component/Auth');

class User extends AppModel {

// ...

public function beforeSave($options = array()) {
    if (isset($this->data[$this->alias]['password'])) {
        $passwordHasher = new BlowfishPasswordHasher();
        $this->data[$this->alias]['password'] = $passwordHasher->hash(
            $this->data[$this->alias]['password']
        );
    }
    return true;
}

// ...
```

Imagen 59: Hashing passwords CakePHP 2.x

Ahora ya tenemos nuestros modelos y nuestros controladores, sólo nos falta la vista, que en este caso corresponde con el típico formulario de acceso con un campo de texto para el nombre de usuario, que en nuestro caso será el E-Mail, y un campo de texto para el password.

Creamos entonces nuestra plantilla con el siguiente contenido en:
/var/www/sportsnet/src/Template/Usuarios/login.ctp

```php
<!-- File: src/Template/Usuarios/login.ctp -->

<div class="users form">
  <?= $this->Flash->render('auth') ?>
  <?= $this->Form->create() ?>
  <fieldset>
    <legend><?= __('Por favor indique su nombre de usuarios y su password') ?></legend>
    <?= $this->Form->input('email') ?>
    <?= $this->Form->input('password') ?>
  </fieldset>
  <?= $this->Form->button(__('Login')); ?>
  <?= $this->Form->end() ?>
</div>
```

Imagen 60: Formulario de login

Ahora si intentamos cargar nuestra lista de usuarios, al igual que hacíamos antes, mediante la URL: http://sportsnet.local/usuarios seremos redirigidos automáticamente al formulario de Login, indicándonos que no estamos autorizados para acceder a esa página.

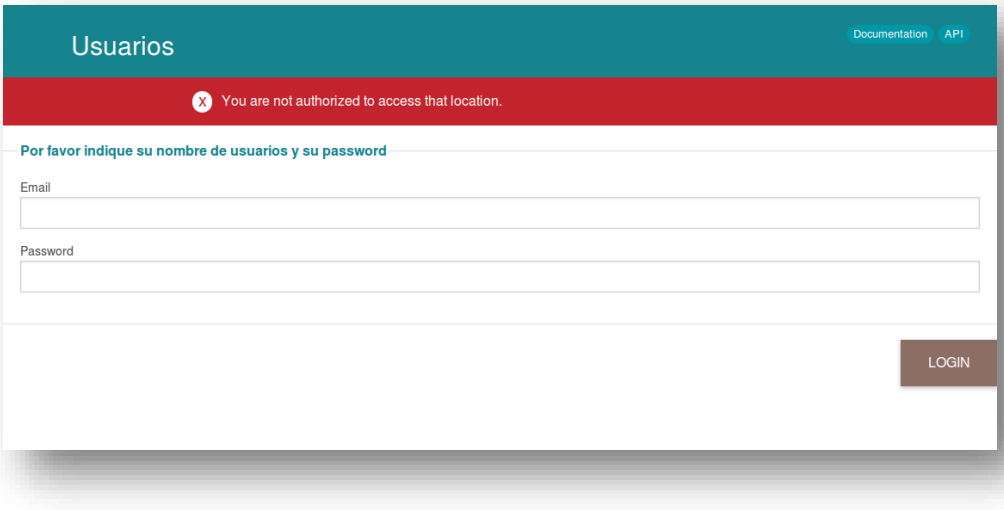


Imagen 61: Formulario de Login

Al intentar autenticarnos, sin embargo, recibimos de nuevo un error.

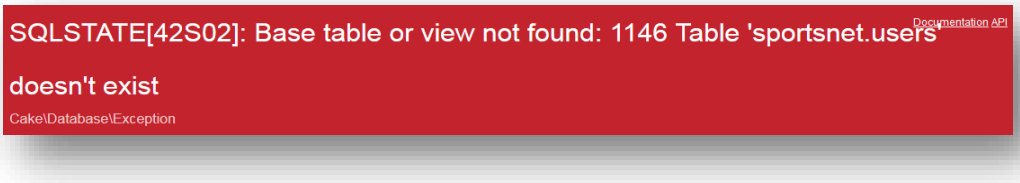


Imagen 62: Error tabla no encontrada

Esto es debido a que el módulo de autentificación busca, por defecto, la tabla "*users*" para autentificar a los usuarios. Nuestra tabla, por el contrario, se llama "*usuarios*".

Llegados a este punto, deberá reconocer el lector, que el uso de la lengua de Shakespeare supone una gran ventaja en el mundo de la programación.

Para solucionar nuestro problema debemos indicarle al módulo de autentificación qué tabla debe usar para buscar a los usuarios. El cambio lo haremos en el fichero **AppControler.php**, modificando el método **initialize**() y añadiendo o siguiente:

```php
/**
 * Initialization hook method.
 *
 * Use this method to add common initialization code like loading components.
 *
 * @return void
 */
public function initialize() {
  $this->loadComponent('Flash');
  $this->loadComponent('Auth', [
    'loginAction' => [
      'controller' => 'Usuarios',
      'action' => 'login',
      'plugin' => null
    ],
    'loginRedirect' => [
      'controller' => 'Espacios',
      'action' => 'index'
    ],
    'logoutRedirect' => [
      'controller' => 'Pages',
      'action' => 'display',
      'home'
    ],
    'authenticate' => [
      'Form' => [
        'fields' => ['username' => 'email'],
        'userModel' => 'Usuarios',
      ]
    ],
  ]);
}
```

Imagen 63: Indicar a Auth que tabla usar para la auteticación

Ahora ya podemos crear usuarios a través de la URL: http://sportsnet.local/usuarios/add

Una vez creado el usuario, nos logueamos con él y veremos ya la lista de usuarios.

5.10 La página de entrada de SPORTSNET

Nuestra aplicación comienza ya a tomar forma. La parte lógica, es decir, la parte funcional, la tenemos más o menos desarrollada. Somos capaces de registrar usuarios, editar sus datos, eliminarlos, etc.
Además, hemos limitado al acceso para que sólo los usuarios registrados puedan acceder a la aplicación.
Evidentemente aún falta mucho por hacer, ya que el objetivo es que cada usuario según sus permisos, pueda realizar diferentes acciones. Eso lo veremos en capítulos posteriores.
Ahora es el momento de preocuparnos por el aspecto visual de nuestra aplicación. En primer lugar vamos a crear nuestra página de inicio. Para ello podemos partir de la que CakePHP ha configurado por nosotros y que se muestra cuando accedemos a http://sportsnet.local.

La página principal a la que nos referimos se encuentra en

```
/var/www/sportsnet/src/Template/Pages/home.ctp
```

Hacemos una copia de esta página en el mismo lugar pero la llamaremos **index.ctp.** Ahora para que CakePHP cargue nuestra página en lugar de home.ctp deberemos cambiar la ruta que CakePHP usa para encontrar las páginas.

5.11 El Sistema de enrutado (routing) de CakePHP.

El sistema de enrutado permite que se puedan asociar direcciones URL a controladores y acciones. Las rutas se definen en el fichero **routes.php** ***(/var/www/sportsnet/config/routes.php***)

```
Router::scope('/', function ($routes) {
    /**
     * Here, we are connecting '/' (base path) to a controller called 'Pages',
     * its action called 'display', and we pass a param to select the view file
     * to use (in this case, src/Template/Pages/home.ctp)...
     */
    $routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'home']);
```

Imagen 64: Routing por defecto

Como vemos aquí, CakePHP tiene definida para una URL "/" (o lo que es lo mismo http://sportsnet.local) que debe ejecutar el controlador "PagesController", ejecutar la acción "display" y usar la plantilla "home".

Cómo ya hemos comentado, CakePHP interpreta lo siguiente:

- El controlador se encontrará en el directorio Controller y se llamará PageController.php,
- La vista estará en el directorio Templates, bajo un subdirectorio llamado Pages, y el fichero se llamará home.ctp.

Si nosotros queremos usar nuestra propia página **index.ctp** deberemos modificar el fichero **routes.php** como sigue:

```
Router::scope('/', function ($routes) {
    /**
     * Here, we are connecting '/' (base path) to a controller called 'Pages',
     * its action called 'display', and we pass a param to select the view file
     * to use (in this case, src/Template/Pages/home.ctp)...
     */
    $routes->connect('/', ['controller' => 'Pages', 'action' => 'display', 'index']);
```

Imagen 65: Definir nuestra propia página de inicio.

Ahora al cargar de nuevo la aplicación, la página principal será **index.ctp** y no **home.ctp**. Ya podemos empezar a definir nuestra página principal.
Para continuar el desarrollo nos centraremos en resolver los casos de uso definidos en el apartado 3. Análisis.

5.12 Implementación de los casos de uso

Una vez tenemos el prototipo, vamos a implementar en CakePHP la funcionalidad. Para realizar la implementación vamos a guiarnos por los casos de uso que identificamos en el apartado de diseño.
Cada caso de uso lo dividiremos en tareas más pequeñas y manejables, de forma que al término de la realización de las tareas tendremos el caso de uso terminado y con ello, habremos implementado una parte de la aplicación que será totalmente funcional.
Al final de la implementación de los casos de uso tendremos la aplicación completamente funcional. En realidad, cuando usamos la consola “bake”, CakePHP ya realizó la mayor parte del trabajo por nosotros, pero nosotros vamos a realizar los ajustes necesarios para adecuarlos a nuestras necesidades.

CU-1: Como usuario no registrado quiero ver una página principal desde la que poder loguearme, acceder a la página de registro a la página de recordatorio de contraseña.

Tarea 1: Implementación del Controlador

Para implementar esta tarea deberemos implementar en el controlador de usuarios (UsuariosController), la funcionalidad de loguear/desloguear usuarios, esto es las funciones login y logout.

Archivo: sportsnet/src/Controller/UsuariosController.php

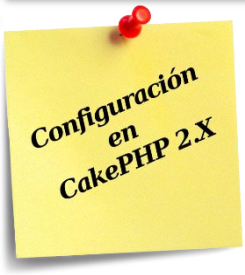
```
class UsuariosController extends AppController {

public function login() {
   if ($this->request->is('post')) {
     $user = $this->Auth->identify();
     if ($user) {
       $this->Auth->setUser($user);
       return $this->redirect($this->Auth->redirectUrl());
     }
     else {
       $this->Flash->error(__('Datos de acceso incorrectos. Inténtelo de nuevo.'),
     ['key' => 'auth']);
       return $this->redirect($this->Auth->logout());
     }
   }
  }

  public function logout() {
    return $this->redirect($this->Auth->logout());
  }
}
```

Imagen 66: CU-1 - Controlador de login/logout

Comentarios:

$user = $this->Auth->identify();

Esta línea identifica al usuario mediante las credenciales facilitadas en el formulario de acceso. Si la autentificación es correcta, se devuelve un objeto con los datos usuarios y se registra en la información de la sesión: $this->Auth->setUser($user);



En CakePHP 2 autenticación de usuarios se realizaba de la siguiente forma:

```
public function login() {
   if ($this->request->is('post')) {
    if ($this->Auth->login()) {
        return $this->redirect($this->Auth->redirectUrl());
        }
         $this->Session->setFlash (
                                __('Datos de acceso incorrectos. Inténtelo de
                                nuevo.',
                                'default',
                                 array(),
                                'auth'
                    );
         }
         }
```

Una vez autenticado el usuario se le redirige a la página configurada en el apartado "Un sistema de login para nuestra aplicación "

```
return $this->redirect($this->Auth->redirectUrl());
```

Si el usuario no es identificado con écito, se le devuelve a la página configurada para logout en el apartado ""Un sistema de login para nuestra aplicación " y se manda el mensaje de error mediante el helper "Flash

```
$this->Flash->error(
__('Datos de acceso incorrectos. Inténtelo de nuevo.'),
      ['key' => 'auth']
);
```

La llamada $this->Flash almacena los mensajes en variables de sesión[14], por lo que estarán disponibles mientras dure las sesión o mientas las variables no sean eliminadas de la misma.
En la llamada podemos asignar una clave a cada tipo de mensaje, de forma que podamos identificarlos cuando queramos mostrarlos.

Por ejemplo:

['key' => 'auth'] en la plantilla usaremos **<?=** $this->Flash->render('auth'); **?>** para mostrar el mensaje correspondiente.

Tarea 2: Implementación de la Vista

Una vez implementado el controlador, vamos a crear la vista que, a diferencia de CakePHP2, en esta versión, se define como plantilla o template (ficheros .ctp) con el mismo nombre que la acción a la que

[14] Las variables de sesión se almacenan en $_SESSION

representa y que se almacena en el directorio "Template" en un subdirectorio llamado como el controlador al que hacen referencia En este caso, nuestro objetivo es presentar la página principal. En el fichero de routing *sportsnet/config/routes.php* configuramos ya el controlador, la acción y la plantilla que se usará para la página principal.

Archivo: /sportsnet/src/Template/Pages/index.ctp

```
<div id="container" class="home clearfix">
 <?= $this->element("Headers/header_home"); ?>
 <!-- Page Content -->
 <div id="main-content" class="content-home">
  <!-- Slider -->
  <div id="myCarousel" class="carousel slide" data-ride="carousel">
   <!-- Indicators -->
   <ol class="carousel-indicators">
    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
    <li data-target="#myCarousel" data-slide-to="1"></li>
    <li data-target="#myCarousel" data-slide-to="2"></li>
   </ol>
   <div class="carousel-inner">
    <div class="item active">
     <?= $this->Html->image('sports1.jpg',
                                        array('alt' => 'First slide', 'border' => '0')
                                       ); ?>
     <div class="container">
      <div class="carousel-caption">
       <h1>SPORTSNET</h1>
       <p>SportsNet. Su red social enfocada al deporte.</p>
      </div>
     </div>
    </div>
    <div class="item">
     <?= $this->Html->image('social1.jpg',  array('alt' => 'Second slide', 'border' => '0'));?>
     <div class="container">
      <div class="carousel-caption">
       <h1>SOCIAL</h1>
                <p>Practique su deporte favorito y/o comparta sus experiencias con otra gente que
                comparta sus aficiones. Participe en eventos o incluso organice usted eventos.</p>
      </div>
     </div>
    </div>
    <div class="item">
     <?= $this->Html->image('sports2.jpg', array('alt' => 'Third slide', 'border' => '0')); ?>
     <div class="container">
      <div class="carousel-caption">
       <h1>INTERCAMBIE, COMPRE, VENDA...</h1>
                <p>Quiere empezar a practicar un deporte pero no quiere gastar mucho dinero
                  en materiales?</p>
                <p>Ha dejado de practicar un deporte y no sabe qué hacer con los materiales que
                utilizaba?</p>
        <p>Pregunte a la comunidad...</p>
       </div>
```

Imagen 67: CU-1 - Vista de la página principal - Parte 1

```
        </div>
      </div>
      <a class="left carousel-control" href="#myCarousel" data-slide="prev">
        <span class="glyphicon glyphicon-chevron-left"></span>
      </a>
      <a class="right carousel-control" href="#myCarousel" data-slide="next">
        <span class="glyphicon glyphicon-chevron-right"></span>
      </a>
    </div>
    <!-- /#slider -->
    <!-- Content-->
    <div id="content" class="row">
      <h1>SPORTSNET</h1>
      <div class="col col-lg-8 col-md-8 col-sm-6 col-xs-6">
        <?= $this->element("Teasers/home_teasers"); ?>
      </div>
      <div class="col col-lg-4 col-md-4 col-sm-6 col-xs-6">
        <?= $this->element("Forms/login"); ?>
      </div>
    </div><!-- /#content -->
  </div><!-- /#main-content -->
  <?= $this->element("Footers/footer_homepage"); ?>
```

Imagen 68: CU-1 - Vista de la página principal - Parte 2

Comentarios:

En esta versión de la aplicación se ha optado por definir un slider estáticos, es decir, en código HTML exclusivamente. En hipotéticas futuras versiones puede cambiarse este comportamiento y se puede dotar a la aplicación de la capacidad para personalizar las imágenes y sus textos o las cajas de texto que aparecen en la página principal.
También puede optar el lector por tomarse esta tarea como ejercicio adicional para mejorar sus capacidades en el uso de CakePHP 3.

<?= … ?>

Estos tags de php equivalen a **<?php echo …?>** y, evidentemente se pueden usar sólo con elementos que generen una salida. Es decir no podríamos usarlos con una variable de tipo "array".

<?= $this->element("Headers/header_home"); ?>

Con esta línea incluimos un elemento[15] en la plantilla, es algo así como los "include" de PHP, pero permite que se le puedan pasar variables de la siguiente forma:

<?= $this->element("Headers/header_home", ['variable' => $valor, 'variable2 => $valor2]);?>

Que en la plantilla serían referenciadas como $variable y $variable2

[15] Los elementos (Elements) son piezas de código (habitualmente HTML) que representas elementos completos que suelen ser reutilizados en diferentes partes de la aplicación. Ejemplos de ellos, serían

Los Elementos se almacenan habitualmente en la carpeta **src/Template/Element/[directorio]/<elemento>.ctp**

Como podemos ver aquí hemos creado multitud de elementos que podrán ser utilizados en las diferentes plantillas a lo largo del desarrollo. La que ahora nos ocupa es el elemento login: **<?php** echo $this->element("Forms/login"); **?>**

```
<!-- File: src/Template/Element/Forms/login.ctp -->
<div class="form login">
 <?= $this->Flash->render('auth') ?>
 <?= $this->Flash->render('register') ?>
 <?= $this->Form->create(null, ['url' => ['controller' => 'Usuarios', 'action' => 'login']]); ?>
 <fieldset>
  <legend><?= __('Por favor indique su E-Mail y su Password') ?></legend>
            <?= $this->Form->input('email') ?>
            <?= $this->Form->input('password') ?>
            <?= $this->Html->link('Ha olvidado su password?',
                                  '/usuarios/recover_password'); ?>
            <?= $this->Html->link('Regístrese', '/usuarios/add'); ?>
 </fieldset>
 <?= $this->Form->button(
        __('Login'), ['name' => "btnLogin", 'class' => "btn", 'type' => 'submit']
 ); ?>
 <?= $this->Form->end() ?>
</div>
```

Imagen 69: CU-1 - Elemento "login"

Comentarios:

<?= $this->Flash->render('auth'); ?>

Esta línea mostrará los mensajes identificados con las clave “auth” (ver Tarea 1).

<?= $this->Form->create(null, ['url' => ['controller' => 'Usuarios', 'action' => 'login']]);?>

Esta línea hace uso del Helper[16] ‘Form’ que creará el código HTML del formulario. A este helper se le pueden pasar variables, la primera contendrá los datos recogidos en el formulario y la segunda es un array de opciones que definen los atributos del formulario, como action, title, etc.

Como vemos en nuestro caso le pasamos como primer parámetro “null”, ya que no queremos guardar los datos de login introducidos en ningún objeto, simplemente queremos validarlos, en el segundo parámetro pasamos un conjunto de opciones, entre ellas, el controlador y el action que deberán ejecutarse. Estos parámetros fomarán la url relativa que ocupa el parámetro action típico de cualquier formulario.

[16] Helpers: http://book.cakephp.org/3.0/en/views/helpers.html

<?= $this->Form->input(….); ?>

Este tipo de líneas crean los campos de introducción de datos del formulario

<?= $this->Html->link(…);**?>**

Con están líneas creamos los enlaces a otras páginas. El primer parámetro es el título que se mostrará para el enlace y en el segundo parámetro pasamos la URL con la que queremos enlazar. En este caso también se le pueden pasar otras opciones en forma de array, como tercer parámetro[17].

<?= $this->Form->button(..) ;?>

Así creamos el botón de acción del formulario y, de la misma forma que en otros helpers, usamos un array para indicar que opciones serán usadas al construir el código html del botón.
Para cerrar el formulario (o sea, añadir el tag </form>, usamos $this->Form->end();

Tanto en el controlador como en la vista hemos visto que las cadenas de texto se encuentran en lo que pare una función sin nombre,

__(“Texto a mostrar”);

Esta función permite que esta cadena sea tratada como un texto traducible y permite a las funciones de internacionalización de CakePHP encontrarlas y, en su caso, devolver el texto traducido correspondiente[18]. Si no se encuentra traducción se devuelve el mismo texto. Siempre es una buena idea usar esta función, aunque inicialmente la aplicación no vaya a tener soporte multi-idioma.
De esa forma, si la aplicación alguna vez tuviera que soportar múltiples idiomas, el trabajo estaría ya medio hecho y solo habría que añadir los ficheros con los textos y las traducciones correspondientes.

El lector puede ahora echar un vistazo al resto de elementos que conforman la página principal.
Dichos elementos son introducidos por las líneas:

<?= $this->element("Headers/header_home"); ?>
<?= $this->element("Footers/footer_homepage"); ?>
<?= $this->element("Teasers/home_teasers"); ?>

Tarea 3: Diseño - CSS y JS

Una vez hemos preparado la parte funcional, vamos a ocuparnos ahora de la parte estética. El diseño. Como podemos ver en nuestra plantilla hemos usado algunas clases en el código HTML que usaremos ahora en nuestros ficheros SCSS. (recordemos que hemos decidido usar scss). Crearemos pues el siguiente fichero:

[17] Helper Link: http://book.cakephp.org/3.0/en/views/helpers/html.html#creating-links
[18] Localizacion e internacionalización: http://book.cakephp.org/3.0/en/core-libraries/internationalization-and-localization.html

sportsnet/compass/scss/pages/_home.scss

```
.home #content{
  padding:15px;
}
.home .footer{
  padding-left: 0px;
}

.carousel-inner > .item > img {
  width:100%;
}
```

Imagen 70: CU-1 - Fichero SCSS página principal

En realidad vemos poco código en la página principal, porque la mayoría del mismo está repartido en los ficheros correspondientes a los elementos que conforman la página principal.

Para ver el resto del código, puede el lector inspeccionar los ficheros:

sportsnet/compass/scss/components/_top.scss
sportsnet/compass/scss/components/_footer.scss

El CSS y el Javascript correspondiente al slider se encuentran compilados dentro del propio framework bootstrap[19], ya que se ha optado esta vez por usar el componente standard en lugar de usar alguno de los existentes en internet.

Ahora podemos visitar la página principal de nuestra aplicación y comprobar el resultado: http://sportsnet.local

[19] Consulte el subapartado “El Layout” del apartado “La página de entrada de Sportsnet” para ver como se ha integrado bootstrap en nuestra aplicación.

CU-2: Como usuario no registrado quiero tener la opción de poder registrarme en la página web con un proceso sencillo.

Tarea 1: Implementación del Controlador

Fichero: sportsnet/src/Controller/UsuariosController.php

```
class UsuariosController extends AppController {
/**
  * Add method
 * @return void Redirects on successful add, renders view otherwise.
  */
 public function add() {
  $this->set('title', Configure::read("App.title"));
  $this->set('description', Configure::read("App.description"));
  $this->set('page_title', __("Registro de nuevo usuario"));
  $usuario = $this->Usuarios->newEntity();
  if  ($this->request->is('post')) {
   $usuario = $this->Usuarios->patchEntity($usuario, $this->request->data);
   $usuario ->imagen = 'no_profile.jpg';
   if  ($this->Usuarios->save($usuario)) {
          $this->Flash->success(__('El usuario ha sido guardado.'), ['key' => 'register']);
          return $this->redirect(['controller' => 'pages', 'action' => 'index']);
   }
   else {
          $this->Flash->error(__('El usuario no ha podido ser guardado. Inténtelo de nuevo, por
          favor.'), ['key' => 'register']);
   }
  }
  $espacios = $this->Usuarios->Espacios->find('list', ['limit' => 200]);
  $this->set(compact('usuario', 'espacios'));
  $this->set('_serialize', ['usuario']);
 }
```

Imagen 71: CU-2 - Controlador añadir usuario

Comentarios:

$this->set('page_title', __("Registro de nuevo usuario"));

Esta línea crea una variable $page_title con el valor de "Registro de nuevo usuario") que podemos usar en la plantilla con el mismo nombre de la acción, que veremos en la siguiente tarea.

$this->Usuarios->newEntity();
$this->Usuarios->patchEntity($usuario, $this->request->data);

Como ya comentamos en esta versión de CakePHP se ha introducido el concepto de Entidad, que vendría ser, una instancia de una clase. Antes de poder guardar un objeto, en este caso un "usuario", creamos una instancia vacía, a las que asignaremos los datos del formulario mediante la siguiente línea:

A continuación llamaremos al método "save": **$this->Usuarios->save($usuario)**
Y asignaremos el tipo de error correspondiente dependiendo del resultado devuelto.

```
$this->Flash->success( __('El usuario ha sido guardado.'), ['key' => 'register']);
$this->Flash->error( __('El usuario no ha podido ser guardado. Inténtelo de nuevo, por favor.'),
['key' => 'register']);
```

En caso de éxito redirigiremos al usuario al controlador principal “Pages” y le diremos que use la plantilla “Pages/index.ctp” para mostrar el resultado de su función principla, que en este caso es “display” (ver apartado “*El Sistema de enrutado (routing) de CakePHP*”)

```
return $this->redirect(['controller' => 'pages', 'action' => ’index’]);
```

Las siguientes dos líneas, no nos hacen falta para nuestra acción de registro. No obstante las vamos a explicar y usted, como lector, decida si las necesita para su implementación.

En esta línea se obtienen todos los espacios relacionados con el usuario actual.

```
$espacios = $this->Usuarios->Espacios->find('list', ['limit' => 200]);
```

En la siguiente línea, la función “compact” se encarga de construir un array asociativo usando como nombre de clave, los valores “usuarios” y “espacios” y como valores, las variables con los con mismos nombres;

```
$this->set(compact('usuario', 'espacios'));
```

Esta línea daría como resultado lo siguiente:

```
array(‘usuario’ => $usuarios, ‘espacios’ => $espacios);
```

Y por último, la siguiente línea crea una versión serializada de los datos de usuario, un objeto JSON para ser utilizado directamente si no se desea usar una plantilla para la vista sino que directamente se desean los datos.

```
$this->set('_serialize', ['usuario']);
```

Tarea 2: Implementación de la Vista

Fichero: sportsnet/src/Template/Usuarios/add.ctp

```
<div id="container" class="register clearfix">
 <?= $this->element("Headers/header_home"); ?>
 <div class="form_register">
  <?= $this->Flash->render('register'); ?>
  <?= $this->Form->create($usuario, array('action' => 'add')); ?>
  <fieldset>
   <legend><?= $page_title; ?></legend>
   <?php
            echo $this->Form->input('nombre', ['placeholder' => __('Nombre')]);
            echo $this->Form->input('apellidos', ['placeholder' => __('Apellidos')]);
            echo $this->Form->input('email', ['placeholder' => __('E-Mail')]);
            echo $this->Form->input('password', ['placeholder' => __('Contraseña')]);
   ?>
  </fieldset>
  <?= $this->Form->button(__('Submit'), ['class' => 'btn']); ?>
  <?= $this->Form->end(); ?>
 </div>
 <?= $this->element("Footers/footer_homepage"); ?>
```

Imagen 72: CU-2 - Vista añadir Usuario

Comentarios:

<?= $this->Flash->render('register'); ?>

Con esta línea se mostraran los mensajes identificados con las clave “register” (ver apartado anterior.

<?= $this->Form->create($usuario, array('action' => 'add')); ?>

Esta línea hace uso del Helper[20] ‘Form’ que creará el código HTML del formulario. A este helper se le pueden pasar variables, la primera contendrá los datos recogidos en el formulario y la segunda es un array de opciones que definen los atributos del formulario, como action, title, etc. Como vemos en nuestro caso le pasamos el “action” del formulario que corresponde con la función de controlador actual que recogerá los datos que se envíen a través del formulario.

En la función add, las líneas concretas que recogen los datos son:

```
$usuario = $this->Usuarios->newEntity();
$usuario = $this->Usuarios->patchEntity($usuario, $this->request->data);
```

Realmente se podría haber usado una sola línea, pero de esta forma hacemos el código más legible.

```
$usuario = $this->Usuarios->patchEntity(
                                $this->Usuarios->newEntity(), $this->request->data
);
```

[20] Información sobre los Helpers: http://book.cakephp.org/3.0/en/views/helpers.html

Tarea 3: Diseño - CSS y JS.

En este caso crearemos un fichero para el formulario de registro:

sportsnet/compass/scss/forms/_register.scss

```
.register{
 .form_register{
  padding:20px 10px;
  font-size:1.2em;
  width: 50%;
  margin-top: 60px !important;
  margin: 0 auto;
  fieldset{
   margin-bottom: 20px;
  }

  label{
   float:left;
   width:20%;
   padding: 5px 0;
  }
  input{
   width:80%;
  }

  div{
   margin-bottom:20px;
  }
  #mensaje label,
  #mensaje textarea{
   width:100%;
  }
 }
```

Imagen 73: CU-2 - SCSS formulario Registro

Ahora podemos probar a registrarnos en: http://sportsnet.local/usuarios/add[21]

[21] Es posible que no veamos la aplicación con el diseño que hemos planteado en el prototipo. Eso puede ser debido a que no se haya completado el diseño de los elementos que conforman la página principla o que se haya configurado el Layout de la aplicación. Ver apartado anterior y el subapartado “El Layout” del apartado “La página de entrada de Spostsnet”.

CU-3: Como usuario registrado quiero poder gestionar mis datos personales a través de una interfaz web amigable.

Vamos a ocuparnos a hora de permitir al usuario registrado que pueda cambiar sus datos personales. Aprovecharemos también ahora para definir lo que serán las páginas internas de la aplicación, que todo usuario registrado verá.
Es decir, definiremos los "elementos" que contendrán las páginas para los usuarios registrados.

Tarea 1: Implementación del Controlador

Al igual que en los casos de uso anteriores, este afecta también al controlador de usuarios, *UsuariosController*, por tanto vamos a nadir las funcionalidad de editar los datos de usuario.

```
class UsuariosController extends AppController {
/**
  * Edit method
  * @param string|null $id Usuario id.
  * @return void Redirects on successful edit, renders view otherwise.
  * @throws \Cake\Network\Exception\NotFoundException When record not found.
  */
public function edit($id=null) {
$usuario = $this->Usuarios->get($id);
   $this->set("page_title", __("Perfil de {0} {1}", [$usuario->nombre, $usuario->apellidos]));

   if ($this->request->is(['patch', 'post', 'put'])) {
    $usuario = $this->Usuarios->patchEntity($usuario, $this->request->data);

    if ($this->request->data['image_profile']['name']) {
     $usuario->imagen = 'users' . DS . $usuario->id . DS . $this->request->data['image_profile']['name'];
    }

    if ($this->Usuarios->save($usuario)) {
     if ($this->request->data['image_profile']['name']) {
      $allowedTypes = ['image/png', 'image/jpg'];
      if (in_array($this->request->data['image_profile']['type'], $allowedTypes)) {
       $imagesDir = WWW_ROOT . 'img' . DS;
       $srcImage = $this->request->data['image_profile']['tmp_name'];
       move_uploaded_file($srcImage, $imagesDir . $usuario->imagen);
      }
      else {
       $this->Flash->error(__('Solo archivos con formato jpg o png.'), ['key' => 'upload_error']);
      }
     }

     $this->Flash->success(__('Los datos han sido actualizados.'), ['key' => 'save_data']);
    }
    else {
     $this->Flash->error(__('Los datos no han podido ser actualizados.'), ['key' => 'save_data']);
    }
   }

   $this->set("user_image", $usuario->imagen);
   $this->set(compact('usuario'));
   $this->set('_serialize', ['usuario']);}
}
```

Imagen 74: CU-3 - Controlador de edición del perfil de usuario

Comentarios:

Se destacaran únicamente aquellas líneas que sean nuevas o que, por su especial uso constituyan nueva fuente de información que no haya sido explicada hasta ahora.

```
$this->set("page_title", __("Perfil de {0} {1}", [$usuario->nombre, $usuario->apellidos]));
```

Con esta línea establecemos el título de la página que se mostraré en la parte superior derecha de la página. En el texto observamos que aparecen dos elementos {0} {1} y vemos que a la función ___() se le pasa como parámetro un array con dos elementos. Estos dos elementos se sustituirán en el texto por los elementos {0} y {1} respectivamente.

```
$imagesDir = WWW_ROOT . 'img' . DS;
$srcImage = $this->request->data['image_profile']['tmp_name'];
move_uploaded_file($srcImage, $imagesDir . $usuario->imagen);
```

Al editar el perfil permitiremos que los usuarios puedan establecer su propia foto de perfil.

$this->request->data['image_profile'] contiene todos los datos referentes al archivo, entro otros cabe destacar:

['name']: Nombre del fichero subido.
['tmp_name']: Fichero temporal.
['type']: Mimetype del fichero.
['size']: Tamaño en bytes del fichero.

move_uploaded_file($srcImage, $dstImage);

Por último, esta función de PHP mueve el archive desde su ubicación temporal, usada en la subida, hasta el destino definitivo del archivo.

Tarea 2: Implementación de la Vista

Fichero: sportsnet/src/Template/Usuarios/edit.ctp

```
<div id="container" class="profile clearfix">
 <?= $this->element("Headers/header_home", ["page_title" => $page_title,
 "user_name" => $usuario->nombre . " " . $usuario->apellido]
); ?>

  <!-- Page Content -->
 <div id='content' class='content-profile'>
  <div class='col col-lg-10 col-md-9 col-content'>
   <div class="username clearfix bg-info">
    <label><?= __("Nombre de Usuario");?></label><span><?= $usuario->email; ?>
    </span>
   </div>
   <?= $this->Flash->render('upload_error'); ?>
   <?= $this->Flash->render('save_data'); ?>
   <?= $this->Form->create($usuario,
['action' => 'edit', 'name' => 'profile', 'id' => 'frmProfile', 'enctype' => 'multipart/form-data']); ?>
          <fieldset class="col col-lg-6 col-md-6">
              <legend><?= __("Datos personales"); ?></legend>
               <?php
               echo $this->Form->input('nombre', ['placeholder' => __('Nombre')]);
               echo $this->Form->input('apellidos', ['placeholder' => __('Apellidos')]);
               echo $this->Form->input('email', ['placeholder' => __('E-Mail')]);
               echo $this->Form->input('password', ['placeholder' => __('Contraseña')]);
               echo $this->Form->input('repeat_password', ['placeholder' => __('Repetir Contraseña')]);
               ?>
          </fieldset>
          <fieldset class="col col-lg-6 col-md-6">
                  <legend><?= __("Foto del perfil"); ?></legend>
              <div id="imgProfile">
               <?= $this->Html->image($user_image, ['alt' => $usuario->nombre . " " . $usuario-
          >apellidos]); ?>
                <?= $this->Form->file('image_profile', ['label' => false]); ?>
              </div>
          </fieldset>
          <?= $this->Form->button(__('Guardar') . '<i class="fa fa-floppy-o"></i>', ['type' => 'submit',
          'class' => 'btn', 'escape' => false]); ?>
          <?= $this->Form->end(); ?>
  </div>
  <div class = 'col col-lg-2 col-md-3 col-toolbar'>
   <?= $this->element("Toolbar/toolbar"); ?>
  </div>
 </div><!--/#content -->
</div><!-- /#wrapper -->
```

Imagen 75: CU-3 - Vista de edición del Perfil

Tarea 3: Diseño - CSS y JS

En este caso crearemos un fichero para la página del perfil:

sportsnet/compass/scss/forms/_profile.scss

```
.profile{
 .username{
  padding:10px;
  margin-bottom:10px;
  label{
   float:left;
   width:220px;
  }
 }
 #frmProfile{
  padding:20px 10px;
  fieldset{
   margin-bottom: 20px;
   label{
    float:left;
    width:160px;
    padding: 5px 0;
   }
   div{
    margin-bottom:20px;
   }
   #imgProfile{
    img{
     width:220px;
    }
   }
   input{
    font-size:1.2em;
   }
   input[type="file"]{
    margin-top:7px;
   }
  }
  fieldset.col{
   padding-left:0px;
   padding-right:0px;
  }
 }
 .btn{
  float: left;
  i{
   margin-left:10px;
  }
```

Imagen 76: CU-3 - SCSS edición del perfil del usuario

CU-4: Como Usuario registrado quiero poder gestionar mi/s espacio/s web de una forma sencilla e intuitiva.

Tarea 1: Implementación del Controlador

Fichero: sportsnet/src/Controller/EspaciosController.php

```
class EspaciosController extends AppController {

 /**
  * Index method
  *
  * @return void
  */
 public function index() {

  $usuario = $this->Auth->user();

  $espacios = $this->Espacios->find()
   ->where(['Espacios.id >' => 0])
   ->matching('Usuarios', function ($q) {
   return $q->where(['Usuarios.id' => $this->Auth->user('id')]);
  });

  $this->set('title', __("ESPACIOS"));
  $this->set('page_title', __("Espacios de {0}", $this->Auth->user('nombre') . " " . $this->Auth->user('apellidos')));
  $this->set('usuario', $usuario);
  $this->set('espacios', $this->paginate($espacios));
  $this->set('_serialize', ['espacios']);
 }
```

Imagen 77: CU-4 - Controlador de espacios

Comentarios:

```
$espacios = $this->Espacios->find()
 ->where(['Espacios.id >' => 0])
 ->matching('Usuarios', function ($q) {
  return $q->where(['Usuarios.id' => $this->Auth->user('id')]);
});
```

Aquí se hace uso de una de las nuevas características de CakePHP 3 que no existían en CakePHP2 y que permiten filtrar los datos maestros dependiendo de una condición de los datos asociados. Lo primero que hacemos es obtener los espacios con un id mayor de 0, puesto que 0 será considerado siempre como la propia aplicación.

```
$espacios = $this->Espacios->find()
    ->where(['Espacios.id >' => 0])
```

A continuación aplicamos el filtrado sobre los datos:

```
->matching('Usuarios', function ($q) {
return $q->where(['Usuarios.id' => $this->Auth->user('id')]);
```

Le indicamos que solo queremos conservar los datos que coincidan con los relacionados que resulten de buscar en la tabla de usuarios y en la tabla de usuarios_espacios los que coincidan con el id de usuario actual, obtenido mediante $this->Auth->user('id')

La SQL generada sería, concretamente:

```
SELECT
  Espacios.id AS `Espacios__id`,
  Espacios.nombre AS `Espacios__nombre`,
  Espacios.descripcion AS `Espacios__descripcion`,
  Espacios.created AS `Espacios__created`,
  Espacios.modified AS `Espacios__modified`,
  Usuarios.id AS `Usuarios__id`,
  Usuarios.nombre AS `Usuarios__nombre`,
  Usuarios.apellidos AS `Usuarios__apellidos`,
  Usuarios.email AS `Usuarios__email`,
  Usuarios.password AS `Usuarios__password`,
  Usuarios.imagen AS `Usuarios__imagen`,
  Usuarios.created AS `Usuarios__created`,
  Usuarios.modified AS `Usuarios__modified`,
  EspaciosUsuarios.usuario_id AS `EspaciosUsuarios__usuario_id`,
  EspaciosUsuarios.espacio_id AS `EspaciosUsuarios__espacio_id`,
  EspaciosUsuarios.role_id AS `EspaciosUsuarios__role_id`,
  EspaciosUsuarios.admitido AS `EspaciosUsuarios__admitido`,
  EspaciosUsuarios.created AS `EspaciosUsuarios__created`,
  EspaciosUsuarios.modified AS `EspaciosUsuarios__modified`
FROM
  espacios Espacios
  INNER JOIN usuarios Usuarios ON Usuarios.id = 1
  INNER JOIN espacios_usuarios EspaciosUsuarios ON (
    Espacios.id = (EspaciosUsuarios.espacio_id)
    AND Usuarios.id = (EspaciosUsuarios.usuario_id)
  )
WHERE
```

Imagen 78: CU-4 - Controlador de espacios - SQL

Tarea 2: Implementación de la Vista

Fichero: sportsnet/src/Template/Espacios/index.ctp

```
<div id="container" class="spaces clearfix">
 <?= $this->element("Headers/header", ["page_title" => $page_title, 'usuario' => $usuario]);  ?>
 <!-- Page Content -->
 <div id='content' class='content-spaces'>
  <div class='col col-lg-10 col-md-9 col-content'>
   <?= $this->Flash->render('space_action'); ?>
   <div class="row">
    <?= $this->Html->link('<i class = "fa fa-plus-circle"></i>' . __('Crear nuevo Espacio'), ['controller' => 'Espacios', 'action' => 'add'], ['class' => ['btn', 'btnCrear'], 'title' => 'Crear espacio', 'escape' => false]); ?>
   </div>
   <?php foreach ($espacios as $espacio): ?>
    <!-- Row -->
    <div class="space row">
     <h3>
      <?= $this->Html->link(__('Ver {0}', h($espacio->nombre)), ['action' => 'view', $espacio->id], ['title' => 'Ir a espacio ' . h($espacio->nombre)]); ?>
     </h3>
     <div class="img_space">
      <?php
      $imgSpace = "spaces" . DS . $this->Number->format($espacio->id) . DS . $this->Number->format($espacio->id) . ".jpg";
      echo $this->Html->image($imgSpace, ['alt' => h($espacio->nombre)]);
      ?>
     </div>
     <p class="description"><?= h($espacio->descripcion) ?></p>
     <ul class="actions">
      <li> <?= $this->Html->link('<i class = "fa fa-eye"></i>' . __('Ver'), ['action' => 'view', $espacio->id], ['escape' => false]) ?></li>
      <li> <?= $this->Html->link('<i class = "fa fa-pencil"></i>' . __('Editar'), ['action' => 'edit', $espacio->id], ['escape' => false]) ?></li>
      <li> <?= $this->Form->postLink('<i class = "fa fa-trash-o"></i>' . __('Eliminar'), ['action' => 'delete', $espacio->id], ['escape' => false, 'confirm' => __('Está seguro de que quiere  eliminar el espacio "{0}"?', h($espacio->name))]) ?></li>
     </ul>
    </div><!-- /#row-->
   <?php endforeach; ?>
   <div class="paginator">
    <ul class="pagination">
     <?= $this->Paginator->prev('< ' . __('previous')) ?>
     <?= $this->Paginator->numbers() ?>
     <?= $this->Paginator->next(__('next') . ' >') ?>
    </ul>
    <p><?= $this->Paginator->counter() ?></p>
   </div>
  </div>
  <div class='col col-lg-2 col-md-3 col-toolbar'>
   <?= $this->element("Toolbar/toolbar"); ?>
  </div>
 </div><!--/#content -->
</div><!-- /#wrapper -->
```

Imagen 79: CU-4 - Vista de espacios

Tarea 3: Diseño - HTML, CSS y JS

```
.spaces{
 #content {
  .row{
   border-bottom: 1px solid #000;
   padding-bottom:10px;
   margin-bottom:10px;
   .img_space{
    float:left;
    margin-right:10px;
    img{
     width: 350px;
    }
   }
  }
 }
}
```

Imagen 80: CU-4 - Ficheo SCSS

CU-15: Como Administrador de mi espacio quiero poder ver el espacio con todos los datos relacionados

Es probable que éste sea el caso de uso más completo, puesto que implica la consulta de todos los tipos de datos que existen en el sistema, así como las relaciones entre ellos.

Tarea 1: Implementación del Controlador

Ficher: sportsnet/src/Controller/ComentariosController.php

```
/**
 * Espacios Controller
 *
 * @property \App\Model\Table\EspaciosTable $Espacios
 */
class EspaciosController extends AppController {
/**
  * View method
  *
  * @param string|null $id Espacio id.
  * @return void
  * @throws \Cake\Network\Exception\NotFoundException When record not found.
  */
 public function view($id = null) {
  $usuario = $this->Auth->user();

  $espacio = $this->Espacios->get($id, [
   'contain' => ['Usuarios',
    'Anuncios' => ['Usuarios',
     'Mensajes' => ['Usuarios']
    ],
    'Comentarios' => ['Usuarios',
     'Respuestas' => ['Usuarios']
    ],
    'Eventos' => ['Usuarios'],
   ]
  ]);
  $imagesDir = WWW_ROOT . 'img' . DS;
  $spaceDir = 'spaces' . DS . $id;
  $dir = new Folder($imagesDir . $spaceDir);
  $files = $dir->find();

  if (count($files) > 0) {
   $spaceImage = $spaceDir . DS . $files[0];
  }
  else {
   $spaceImage = "no_space_image.jpg";
  }

  $this->set("page_title", __("Espacio {0}", [$espacio->nombre]));
  $this->set('space_image', $spaceImage);
  $this->set('espacio', $espacio);
  $this->set('usuario', $usuario);
  $this->set('_serialize', ['espacio']);
```

Imagen 81: CU-16 - Controlador de Vista espacio

Comentarios:

```
$espacio = $this->Espacios->get($id, [
    'contain' => ['Usuarios',
     'Anuncios' => ['Usuarios',
      'Mensajes' => ['Usuarios']
     ],
     'Comentarios' => ['Usuarios',
      'Respuestas' => ['Usuarios']
     ],
     'Eventos' => ['Usuarios'],
    ]
  ]);
```

Esta es la línea más importante a comentar. Y aquí es dónde podemos ver el verdadero potencial del nuevo CakePHP3, mediante la sentencia “contains” podemos referirnos a entidades asociadas y además, encadenar con las entidades asociadas de esas mismas entidades.

CakePHP se encarga entonces de obtener los registros relacionados, construyendo un array asociativo que contendrá como elementos, a su vez, las diferentes entidades asociadas. Veamos la vista para entenderlo:

Tarea 2: Implementación de la Vista

Ficher: sportsnet/src/Template/Espacios/view.ctp

```
<div id="container" class="space clearfix">
 <?= $this->element("Headers/header", ["espacio" => $espacio, "page_title" => $page_title, "user_name" => $usuario['nombre'] . " " . $usuario['apellidos']]); ?>
 <!-- Page Content -->
 <div id='content' class='content-space'>
  <div class='col col-lg-10 col-md-9 col-content'>
<!-- Espacio -->
    <div id ="espacio" class="espacio active">
     <div class="row espacio">
      <?= $this->Html->image($space_image, ['alt' => $espacio->nombre, 'class' => 'clearfix']); ?>
      <div class="description"><?= $espacio->descripcion; ?></div>
      <ul class="actions">
       <li><?= $this->Html->link('<i class="fa fa-plus-circle"></i>' . __('Nuevo comentario'), ['controller' => 'Comentarios', 'action' => 'add', $espacio->id], ['class' => 'btn', 'title' => "Añadir comentario", 'escape' => false]) ?></li>
       <li><?= $this->Html->link('<i class="fa fa-pencil"></i>' . __('Editar'), ['controller' => 'Espacios', 'action' => 'edit', $espacio->id], ['class' => 'btn', 'title' => "Editar espacio", 'escape' => false]) ?></li>
       <li><?= $this->Form->postLink('<i class="fa fa-trash-o"></i>' . __('Eliminar'), ['controller' => 'Espacios', 'action' => 'delete', $espacio->id], ['class' => 'btn', 'title' => "Eliminar espacio", 'escape' => false, 'confirm' => __('Desea eliminar el espacio {0}?', $espacio->nombre)]) ?></li>
      </ul>
     </div>
```

Imagen 82: CU-16 Vista Espacio - Datos del Espacio

Comentarios:

```
<li><?= $this->Html->link('<i class="fa fa-plus-circle"></i>' . __('Nuevo comentario'), ['controller' => 'Comentarios', 'action' => 'add', $espacio->id], ['class' => 'btn', 'title' => "Añadir comentario", 'escape' => false]) ?></li>
```

Con esa línea crearemos un botón que nos permitirá crear comentarios en el espacio actual. Fijémonos que le pasamos el id del espacio al controlador add. En el CU-5 podremos ver para qué y cómo manejaremos ese id.

```
$espacio
```

Esta es nuestra variable Estrella que contiene todos los datosrelacionados con el espacio actual.

```
'content-space', active
```

Prestemos atención a estas clases y las que aparecerán en sucesivas partes del archivo. Las necesitaremos luego.

```
<?php if (count($espacio->anuncios) === 0): ?>
        <div class="clearfix bg-info">
          <?= __("No hay Comentarios activos"); ?>
        </div>
      <?php else: ?>
        <?php foreach ($espacio->comentarios as $comentarios): ?>
          <div class="row comentario">
            <h3><?= h($comentarios ->titulo) ?></h3>
            <div class="user-info">
              <div class="date">Publicado el <?= h($comentarios->created) ?></div>
              <?= $this->Html->image($comentarios->usuario->imagen, ['alt' => h($comentarios-
>usuario->nombre . " " . $comentarios->usuario->apellidos)]) ?>
              <div><?= $this->Html->link(h($comentarios->usuario->nombre . " " . $comentarios-
>usuario->apellidos), ['controller' => 'Usuarios', 'action' => 'view', $comentarios->usuario->id],
['title' => "Ver usuario"]) ?></div>
            </div>
            <div class="msg-info">
              <p class="texto">
                <?= h($comentarios->texto) ?>
              </p>
            </div>
            <ul class="actions">
              <li><?= $this->Html->link(__('Respuestas ({0})', count($comentarios->respuestas)),
['controller' => 'Comentarios', 'action' => 'view', $comentarios ->id], ['title' => "Ver respuestas",
'class' => "show-responses", 'data-open' => "respuestas-$comentarios->id", 'escape' => false])
?></li>
              <li><?= $this->Html->link('<i class="fa fa-pencil"></i>' . __('Editar'), ['controller' =>
'Comentarios', 'action' => 'edit', $comentarios->id], ['title' => "Editar comentario", 'escape' =>
false]) ?></li>
              <li><?= $this->Form->link('<i class="fa fa-comment-o"></i>' . __('Responder'), ['controller'
=> 'Respuestas', 'action' => 'add'], ['title' => "Responder comentario", 'escape' => false]) ?></li>
              <li><?= $this->Form->postLink('<i class="fa fa-trash-o"></i>' . __('Eliminar'), ['controller' =>
'Comentarios', 'action' => 'delete', $comentarios->id], ['title' => "Eliminar comentario", 'escape' =>
false, 'confirm' => __('Desea eliminar el comentario {0}?', $comentarios->id)]) ?></li>
            </ul>
```

Comentarios:

Imagen 83: CU-16 Vista Espacio - Datos de los comentarios

`<?php foreach ($espacio->comentarios as $comentarios): ?>`

Con esta línea accedemos a los objetos “comentario” asociados a este espacio, tal y como habíamos dicho cuando vimos el controlador, los objetos relacionados se representas como arrays de objetos asociativos.

`$comentarios->usuario->apellidos`

De la misma forma accedemos a los datos del usuario relacionado con ese mismo comentario.

'class' => "show-responses", 'data-open' => "respuestas-$comentarios->id"

Para finalizar prestemos atención y recordemos estos datos que se crearan como atributos del enlace para mostrar las respuestas a cada comentario

```
<div id="respuestas-<?= $comentarios->id ?>" class="respuestas hide">
          <?php foreach ($comentarios->respuestas as $respuesta): ?>
            <div id="respuesta_<?= $respuesta->id ?>" class="respuesta row">
             <h3><?= $respuesta->titulo ?><span><?= "#" . $respuesta->id ?></span></h3>
             <div class="user-info">
              <div class="date">Publicado el <?= h($respuesta->created); ?></div>
              <?= $this->Html->image($respuesta->usuario->imagen, ['alt' => $respuesta->usuario-
>nombre . " " . $respuesta->usuario->apellidos]) ?>
             <div><?= $respuesta->usuario->nombre . " " . $respuesta->usuario->apellidos
?></div>
            </div>
            <div class="msg-info">
             <p class="texto">
              <?= h($respuesta->texto) ?>
             </p>
            </div>
            <ul class="actions">
             <li><?= $this->Html->link('<i class="fa fa-pencil"></i>' . __('Editar'), ['controller' =>
'Respuestas', 'action' => 'edit', $respuesta->id], ['title' => 'Editar respuesta', 'escape' => false])
?></li>
             <li><?= $this->Form->postLink('<i class="fa fa-trash-o"></i>' . __('Eliminar'),
['controller' => 'Respuestas', 'action' => 'delete', $respuesta->id], ['title' => 'Eliminar respuesta',
'escape' => false, 'confirm' => __('Está seguro de que quiere eliminar esta respuesta?')]) ?></li>
            </ul>
           </div><!-- /#respuesta-->
          <?php endforeach; ?>
         </div><!-- /#respuestas-->
        </div><!-- /#comentario-->
       <?php endforeach; ?>
      <?php endif; ?>
    </div><!-- /#espacio-->
```

Imagen 84: CU-16 Vista Espacio - Respuestas a los comentarios

Comentarios:

<?php foreach ($comentarios->respuestas as $respuesta): ?>

Al igual que antes podemos acceder a las respuestas de cada comentario directamente iterando sobre el array que CakePHP ha preparado por nosotros en el controlador. También al igual que antes podemos acceder a las propiedades del objeto respuesta, mediante la sintaxis: $respuesta->titulo

respuestas-<?= $comentarios->id ?>, hide

Como es habitual, deberemos prestar atención a los atributos y class destacados

```
<!-- Anuncios -->
    <div id ="anuncios" class="anuncios hide">
     <h3><?= __("Anuncios"); ?></h3>
     <?php if (count($espacio->anuncios) === 0): ?>
      <div class="clearfix bg-info">
       <?= __("No hay Anuncios activos"); ?>
      </div>
     <?php else: ?>
      <?php foreach ($espacio->anuncios as $anuncio): ?>
       <div class="row anuncio">
        <h3><?= h($anuncio->titulo) ?></h3>
        <div class="user-info">
         <div class="date">Publicado el <?= h($anuncio->created) ?></div>
         <?= $this->Html->image($anuncio->usuario->imagen, ['alt' => h($anuncio->usuario->nombre
. " " . $anuncio->usuario->apellidos)]) ?>
         <div><?= $this->Html->link(h($anuncio->usuario->nombre . " " . $anuncio->usuario-
>apellidos), ['controller' => 'usuarios', 'action' => 'view', $anuncio->usuario->id], ['title' => "Ver
usuario"]) ?></div>
        </div>
        <div class="msg-info">
         <p class="texto">
          <?= h($anuncio->texto) ?>
         </p>
        </div>
        <ul class="actions">
         <li><?= $this->Html->link(__('Respuestas'), ['controller' => 'Anuncios', 'action' => 'view',
$anuncio->id], ['title' => "Ver respuestas", 'class' => "show-responses", 'data-open' => "mensajes-
$anuncio->id", 'escape' => false]) ?></li>
         <li><?= $this->Html->link('<i class="fa fa-pencil"></i>' . __('Editar'), ['controller' => 'Anuncios',
'action' => 'edit', $anuncio->id], ['title' => "Editar anuncio", 'escape' => false]) ?></li>
         <li><?= $this->Form->postLink('<i class="fa fa-trash-o"></i>' . __('Eliminar'), ['controller' =>
'Anuncios', 'action' => 'delete', $anuncio->id], ['title' => "Editar anuncio", 'escape' => false, 'confirm' =>
__('desea eliminar el anuncio #{0}?', $anuncio->id)]) ?></li>
         <li><?= $this->Form->postLink('<i class="fa fa-comment-o"></i>' . __('Responder'),
['controller' => 'Mensajes', 'action' => 'add'], ['title' => "Responder anuncio", 'escape' => false]) ?></li>
        </ul>
```

Imagen 85: CU-16 Vista espacio - Datos Anuncios

```
<!-- Mensajes-->
          <div id="mensajes-<?= $anuncio->id ?>" class="mensajes hide">
           <?php foreach ($anuncio->mensajes as $mensaje): ?>
            <div id="mensaje<?= $mensaje->id ?>" class="mensaje row">
             <h3><?= $mensaje->titulo ?><span><?= "#" . $mensaje->id ?></span></h3>
             <div class="user-info">
              <div class="date">Publicado el <?= h($mensaje->created); ?></div>
              <?=  $this->Html->image($mensaje->usuario->imagen,  ['alt'  =>  $mensaje->usuario-
>nombre . " " . $mensaje->usuario->apellidos]) ?>
              <div><?= $mensaje->usuario->nombre . " " . $mensaje->usuario->apellidos ?></div>
             </div>
             <div class="msg-info">
              <p class="texto">
               <?= h($mensaje->texto) ?>
              </p>
             </div>
             <ul class="actions">
              <li><?=  $this->Html->link('<i  class="fa  fa-pencil"></i>'  .  __('Editar'),  ['controller'  =>
'Mensajes', 'action' => 'edit', $mensaje->id], ['title' => 'Editar respuesta', 'escape' => false]) ?></li>
              <li><?=  $this->Form->postLink('<i  class="fa  fa-trash-o"></i>'  .  __('Eliminar'),
['controller'  =>  'Mensajes',  'action'  =>  'delete',  $mensaje->id],  ['title'  =>  'Eliminar  respuesta',
'escape' => false, 'confirm' => __('Está seguro de que quiere eliminar esta respuesta?')]) ?></li>
             </ul>
            </div><!-- /#mensaje-->
           <?php endforeach; ?>
          </div><!-- /#mensajes-->
         </div>
        <?php endforeach; ?>
       <?php endif; ?>
      </div>
```

Imagen 86: CU-16 Vista espacio - Respuestas a los anuncios

```
<!-- Eventos -->
    <div id ="eventos" class="eventos hide">
     <h3><?= __("Eventos"); ?></h3>
     <?php if (count($espacio->eventos) === 0): ?>
      <div class="clearfix bg-info">
       <?= __("No hay eventos actualmente"); ?>
      </div>
     <?php else: ?>
      <?php foreach ($espacio->eventos as $eventos): ?>
       <div class="row evento">
        <div><?= h($eventos->fecha) ?></div>
        <div><?= h($eventos->titulo) ?></div>
        <div><?= h($eventos->descripion) ?></div>
        <ul class="actions">
         <li><?= $this->Html->link('<i class="fa fa-pencil"></i>' . __('Editar'), ['controller' => 'Eventos', 'action' => 'edit', $eventos->id], ['title' => 'Editar evento', 'escape' => false]) ?></li>
         <li><?= $this->Form->postLink('<i class="fa fa-trash-o"></i>' . __('Eliminar'), ['controller' => 'Eventos', 'action' => 'delete', $eventos->id], ['title' => 'Eliminar Evento', 'escape' => false, 'confirm' => __('Está seguro de que quiere eliminar el evento {0}?', $eventos->titulo)]) ?></li>
        </ul>
       </div>
      <?php endforeach; ?>
     <?php endif; ?>
    </div>
```

Imagen 87: CU-16 Vista espacio - Eventos

```
<!-- Miembros -->
    <div id ="miembros" class="miembros hide">
     <h3><?= __("Miembros"); ?></h3>
     <?php if (count($espacio->usuarios) === 0): ?>
      <div class="clearfix bg-info">
       <?= __("No hay miembros activos"); ?>
      </div>
     <?php else: ?>
      <?php foreach ($espacio->usuarios as $usuarios): ?>
       <div class="row miebro">
        <span  class="user-image"><?=  $this->Html->image($usuarios->imagen,  ['alt'  =>  $usuarios-
>nombre, 'class' => 'clearfix']); ?></span>
         <span class="user-name"><?= h($usuarios->nombre) . " " . h($usuarios->apellidos) ?></span>
         <span class="user-email"><?= h($usuarios->email) ?></span>
         <ul class="actions">
          <li><?= $this->Html->link('<i class="fa fa-pencil"></i>' . __('Ver'), ['controller' => 'Usuarios',
'action' => 'view', $usuarios->id], ['class' => 'btn', 'title' => 'Ver usuario', 'escape' => false]); ?></li>
           <li><?= $this->Form->postLink('<i class="fa fa-trash-o"></i>' . __('Expulsar'), ['controller' =>
'Espacios', 'action' => 'expel', $usuarios->id], ['class' => 'btn btnExpel', 'title' => 'Expulsar usuario',
'escape' => false, 'confirm' => __('Está seguro de que quiere expulsar al usuario {0} {1}?', [$usuarios-
>nombre, $usuarios->apellidos])]) ?></li>
         </ul>
        </div>
       <?php endforeach; ?>
     <?php endif; ?>
    </div>
   </div><!--/.col-content -->
   <div class = 'col col-lg-2 col-md-3 col-toolbar'>
    <?= $this->element("Toolbar/toolbar"); ?>
   </div>
  </div><!--/#content -->
</div><!-- /#wrapper -->
</div><!-- /#container -->
```

Imagen 88: CU-16 Vista espacio - Miembros

Además de estas vistas, tendremos que modificar nuestra vista de la barra lateral:

Fichero: sportsnet/src/Template/Element/Sidebar/sidebar.ctp

```
<!-- Sidebar -->
<aside id="sidebar-wrapper">
 <div id="logo">
  <h3> <?= $this->Html->link("SPORTSNET", "/", ['title' => "A la página principal"]); ?></h3>
 </div>
 <?= $this->element("Components/search"); ?>
 <nav class="sidebar-nav">
  <?= $this->Html->link('<i class="fa fa-home"></i>' . __('Espacios'), ['controller' => 'espacios', 'action'
=> 'index'], ['escape' => false]); ?>
  <?php if (isset($espacio->id)) : ?>
   <?= $this->Html->link('<i class="fa fa-users"></i>' . __('Comentarios'), ['controller' => null, 'action'
=> '#comentarios'], ['title' => "Ver Comentarios", 'class' => "show-comments", 'data-open' => "espacio",
'escape' => false]); ?>
   <?= $this->Html->link('<i class="fa fa-users"></i>' . __('Miembros'), ['controller' => null, 'action' =>
'#miembros'], ['title' => "Ver Miebros", 'class' => "show-members", 'data-open' => "miembros", 'escape'
=> false]); ?>
   <?= $this->Html->link('<i class="fa fa-bullhorn"></i>' . __('Anuncios'), ['controller' => null, 'action'
=> '#anuncios'], ['title' => "Ver Anuncios", 'class' => "show-anounces", 'data-open' => "anuncios",
'escape' => false]); ?>
   <?= $this->Html->link('<i class="fa fa-calendar"></i>' . __('Eventos'), ['controller' => null, 'action' =>
'#eventos'], ['title' => "Ver Eventos", 'class' => "show-events", 'data-open' => "eventos", 'escape' =>
false]); ?>
  <?php endif; ?>
  <?= $this->Html->link('<i class="fa fa-sign-out"></i>' . __('Salir'), ['controller' => 'users', 'action' =>
'logout'], ['escape' => false]); ?>
 </nav>
 <!-- Footer -->
 <?= $this->element("Components/social_links"); ?>
 <?= $this->element("Components/static_pages"); ?>
</aside><!-- /#sidebar-wrapper -->
```

Imagen 89: CU-16 Sidebar

Comentarios:

<?php if (isset($espacio->id)) : ?>

Lo primero que observamos en los cambios es que ahora los elementos de la misma sólo semostrarán si existe algun espacio definido, es decir, que sólo podremos ver los elementos si nos encontramos dentro de un espacio.

<?= $this->Html->link('' . __('Comentarios'), ['controller' => null, 'action' => '#comentarios'], ['title' => "Ver Comentarios", 'class' => "show-comments", 'data-open' => "espacio", 'escape' => false]); ?>

Los enlaces de los elementos también se han modificado, de forma que ahora no apuntan a una acción de un controlador, recordemos que ya tenemos todos los obejtos y que no necesitamos ningún controlador para mostrar datos.

['controller' => null , 'action' => '#comentarios'] : Esto construirá enlaces del tipo href="#comentarios" que inicialmente no ejecutan ninguna acción.

'class' => "show-comments", 'data-open' => "espacio" : Esto asignará la clase "show-comments" al enlace y añadirá un Nuevo atributo "data-open" que apunta a la sección correspondiente (ver apartado anterior)

Tarea 3: Diseño - HTML, CSS y JS

Ficher: sportsnet/compass/scss/pages/_space.scss

```
.space{
 #content{
  .row{
   padding-bottom:10px;
   margin-bottom:10px;
   clear: both;
  }
  .actions .btn{
   margin-right: 15px;
   float:left;
  }

  ul.actions{
   float:right;
  }
  #espacio{
   img{
    width: 100%;
   }
   .description{
    padding:10px 0;
   }
   .comentario{
    text-align: justify;
    .msg-info,
    .user-info{
     float:left;
     padding:10px;
     img{
      width:80px;
      margin:10px 0px;
     }
    }
    .user-info{
     width:20%;
     text-align: center;
    }
    .msg-info{
     width:80%;
    }
    .texto{
     margin-top:10px;
    }
   }
```

```
.respuestas {
  margin-top: 50px;
  h3{
    span{
      float:right;
    }
  }
  .respuesta{
    background-color: rgba(255, 209,146, 0.4);
    padding: 20px;
    margin: 20px 0px;
    position: relative;
  }
  .row.respuesta{
    border-bottom:0px;
  }
  .msg-info,
  .user-info{
    float:left;
    padding:10px;
    img{
      width:80px;
      margin:10px 0px;
    }
  }
  .user-info{
    width:20%;
    text-align: center;
  }

  .msg-info{
    width:80%;
  }
  .texto{
    margin-top:10px;
  }
  ul.actions{
    position:absolute;
    right:20px;
    bottom: 0px;
  }
}
```

Imagen 91: CU-16 SCSS - Sección Respuestas a los comentarios

```
#anuncios{
 > h3{
   margin-bottom:10px;
   border-bottom: 1px solid #000;
   text-align: center;
  }
  .anuncio{
   text-align: justify;
   .msg-info,
   .user-info{
     float:left;
     padding:10px;
     img{
       width:80px;
       margin:10px 0px;
     }
   }
   .user-info{
     width:20%;
     text-align: center;
   }
   .msg-info{
     width:80%;
   }
   .texto{
     margin-top:10px;
   }
  }
  .mensajes {
   margin-top: 50px;
   .mensaje{
     background-color: rgba(255, 209,146, 0.4);
     padding: 20px;
     margin: 20px 0px;
   }
   .row.mensaje{
     border-bottom:0px;
   }
   .msg-info,
   .user-info{
     float:left;
     padding:10px;
     img{
       width:80px;
       margin:10px 0px;
     }
   }
   .user-info{
     width:20%;
     text-align: center;
   }

   .msg-info{
     width:80%;
   }
   .texto{
     margir
```

Imagen 92: CU-16 SCSS - Sección Anuncios y respuestas

```
  #eventos{
    > h3{
      margin-bottom:10px;
      border-bottom: 1px solid #000;
      text-align: center;
    }
  }
  #miembros{
    > h3{
      margin-bottom:10px;
      border-bottom: 1px solid #000;
      text-align: center;
    }
    span{
      margin:15px;
      width:33%;
      font-size:1.2em;
    }
    .btnExpel{
      background-color: rgba(204, 0, 0, 0.5);
      &:hover{
        background-color:rgba(204, 0, 0, 1);
      }
    }
    .user-image{
      img{
        width:80px;
        margin:10px 0px;
      }
    }
    .user-name{
      text-transform: uppercase;
    }
  }
`
```

Imagen 93: CU-16 SCSS - Sección Eventos y miembros

Ficher: /var/www/sportsnet/webroot/js/main.js

Llega el momento de nuestro primer contacto con Javascript y aprovecharemos ahora para definir el comportamiento de la barra lateral sidebar, así como de los enlaces repartidos por la página.

```
$(document).ready(function() {
  $("#menu-toggle").on("click", function(e) {
    e.preventDefault();
    $("#wrapper").toggleClass("toggled");
    $(".footer").toggleClass("toggled");
  });
  $(".show-responses").on("click", function(e) {
    e.preventDefault();
    var to_open = $(this).attr("data-open");
    $("#" + to_open).toggleClass("hide");
  });
  $(".show-comments").on("click", function(e) {
    e.preventDefault();
    hideActive();
    var to_open = $(this).attr("data-open");
    $("#" + to_open).toggleClass("hide");
    $("#" + to_open).addClass("active");
  });
  $(".show-members").on("click", function(e) {
    e.preventDefault();
    hideActive();
    var to_open = $(this).attr("data-open");
    $("#" + to_open).toggleClass("hide");
    $("#" + to_open).addClass("active");
  });
  $(".show-anounces").on("click", function(e) {
    e.preventDefault();
    hideActive();
    var to_open = $(this).attr("data-open");
    $("#" + to_open).toggleClass("hide");
    $("#" + to_open).addClass("active");
  });
  $(".show-events").on("click", function(e) {
    e.preventDefault();
    hideActive();
    var to_open = $(this).attr("data-open");
    $("#" + to_open).toggleClass("hide");
   $("#" + to_open).addClass("active");
  });
  $("a.link_opener").on("click", function(e) {
    e.preventDefault();
    $("#" + to_open).addClass("active");
  });
});

function hideActive() {
```

Imagen 94: CU-16 Fichero JS

Comentarios:

A buen seguro, el lector, tiene conocimientos de jQuery y entiende perfectamente lo que hace el código mostrado. Tan sólo se quiere aquí hacer hincapié en las clases que se utilizan, y se insta al lector a volver a revisar las visitas dónde se habían remarcado estas clases indicando que nos servirían más adelante.

CU-5: Como Usuario registrado quiero poder publicar comentarios.

Tarea 1: Implementación del Controlador

Ficher: sportsnet/src/Controller/ComentariosController.php

```
/**
 * Comentarios Controller
 *
 * @property \App\Model\Table\ComentariosTable $Comentarios
 */
class ComentariosController extends AppController {
/**
  * Add method
  * @param int $id espacio dónde será creado el comentario
  * @return void Redirects on successful add, renders view otherwise.
  */
 public function add($id = null) {
  $comentario = $this->Comentarios->newEntity();
  $usuario = $this->Auth->user();
  if ($id) {
   $espacio = $this->Comentarios->Espacios->get($id);
  }
  if ($this->request->is('post')) {
   $comentario = $this->Comentarios->patchEntity($comentario, $this->request->data());
   $espacio = $this->Comentarios->Espacios->get($this->request->data['espacio_id']);
   $comentario->usuario_id = $this->Auth->user('id');
   $comentario->espacio_id = $espacio->id;

   if ($this->Comentarios->save($comentario)) {
    $this->Flash->success(__('El comentario ha sido guardado'), ['key' => 'save_data']);
    return $this->redirect(['controller' => 'Espacios', 'action' => 'view', $espacio->id]);
   }
   else {
    $this->Flash->error(__('El comentario no ha podio guardarse. inténtelo de nuevo.'), ['key' =>
'save_data']);
   }
  }
  $this->set("page_title", __("Espacio {0}", [$espacio->nombre]));
  $espacios = $this->Comentarios->Espacios->find('list', ['limit' => 200]);
  $this->set(compact('comentario', 'espacios', 'usuario', 'espacio'));
  $this->set('_serialize', ['comentario']);
 }
}
```

Imagen 95: CU-5 - Controlador nuevo comentario

Comentarios:

```
if ($id) {
   $espacio = $this->Comentarios->Espacios->get($id);
}
```

Como hemos indicado cuando definíamos la vista para mostrar los objetos dentro de un espacio, creamos un botón para poder crear nuevos comentarios y dijimos que aunque no era habitual, pasábamos al controlador de adición un parámetro para poder identificar el espacio donde debía crearse el comentario.
Ahora es el momento de ver que hacemos con ese parámetro y en primer lugar lo encerramos en un "if" ya que cuando llamamos por primera vez al controlador este valor se trasmite a través de la URL y el formulario de creación se muestra correctamente, sin embargo, al intentar guardar el nuevo comentario, este parámetro está vacío y el intento de búsqueda de un registro usando una clave primaria nula produce una excepción.
De esta forma, sólo buscaremos el espacio cuando llamemos por primera vez al formulario de creación del comentario y no al guardarlo.

```
$espacio = $this->Comentarios->Espacios->get($this->request->data['espacio_id']);
$comentario->espacio_id = $espacio->id;
```

Como veremos a continuación, en nuestra vista tenemos un campo que identifica al espacio al cuál va a ser asignado el comentario. En este punto lo recuperamos y lo asignamos al campo correspondinte antes d eguardar el comentario. De otro modo nos daría error al guardar el campo "espacio_id" vacío[22].

[22] Revise el archivo sportsnet/src/Model/Table/ComentariosTable.php para ver las normas de validación establecidad para los campos de la tabla Comentarios.

Tarea 2: Implementación de la Vista

Fichero: sportsnet/src/Template/Comentarios/add.ctp

```
<div id="container" class="comment clearfix">
 <?= $this->element("Headers/header", ["espacio" => $espacio, "page_title" => $page_title,
"user_name" => $usuario['nombre'] . " " . $usuario['apellidos']]); ?>
 <!-- Page Content -->
 <div id='content' class='content-profile'>
  <div class='col col-lg-10 col-md-9 col-content'>
   <div class="form_register">
    <?= $this->Flash->render('save_data'); ?>
    <?= $this->Form->create($comentario, array('action' => 'add')); ?>
    <fieldset>
     <legend><?= __("Crear Comentario"); ?></legend>
     <?php
     echo $this->Form->input('titulo', ['placeholder' => __('Nombre')]);
     echo $this->Form->input('texto', ['placeholder' => __('Descripción')]);
     echo $this->Form->input('espacio_id', ['type' => "hidden", 'value' => $espacio['id']]);
     ?>
    </fieldset>
    <?= $this->Form->button(__('Guardar') . ' <i class = "fa fa-floppy-o"></i> ', ['type' =>
'submit', 'class' => 'btn', 'escape' => false]); ?>
    <?= $this->Form->end(); ?>
   </div>
  </div>
  <div class = 'col col-lg-2 col-md-3 col-toolbar '>
   <?= $this->element("Toolbar/toolbar"); ?>
  </div>
 </div><!--/#content -->
</div><!-- /#wrapper -->
</div><!-- /#container -->
```

Comentarios:

```
echo $this->Form->input('espacio_id', ['type' => "hidden", 'value' => $espacio['id']]);
```

Como ya hemos comentado, esta campo nos sirve para identificar el espacio al que pertence el comentario y que hemos obtenido en primera instancia usando el „id“ pasado a través de la ULR y posteriormente en el controlador obteniendo el espacio asociado

```
if ($id) {
   $espacio = $this->Comentarios->Espacios->get($id);
}
```

A continuación lo hemos pasado a la vista a través de

```
$this->set(compact('comentario', 'espacios', 'usuario', 'espacio'));
```

Tarea 3: Diseño - HTML, CSS y JS

En principio, todos los estilos han sido definidos durante el desarrollo de los anteriores casos de uso. NO obstante el diseño es mejorable y se deja al lector como ejercicio, el ajustar a su necesidades el estilo básico de este caso de uso.

6 Evaluación y Pruebas

En este apartado describiremos algunas de las pruebas que se han hecho para comprobar la funcionalidad de la aplicación.

Las pruebas para cualquier aplicación web se realizan a través de los Test Unitarios. En la actualidad, prácticamente todas las herramientas de desarrollo permiten la integración y ejecución de este tipo de tests de una forma prácticamente automática. Todos los grandes Framewoks poseen herramientas para crear tests unitarios y CakePHP no iba a ser una excepción [23]. A través de los tests unitarios comprobaremos el correcto de todas las funcionalidades de la aplicación. Lamentablemente estos tests sólo comprueban el correcto funcionamiento de cada "unidad", es decir módulos y funciones, a nivel individual y no de forma global, para lo que se requieren los llamados tests de integración. CakePHP3 también soporta este tipo de tests aunque su creación no está tan automatizada como en el caso de los test unitarios.

6.1 Test Unitarios

Los tests unitarios funcionan sobre el Framework PHPUnit [24]que debe ser instalado previamente antes de poder comenzar a crear los tests.
La instalación de PHPUnit a nivel del sistema comprende los siguientes pasos:

```
wget https://phar.phpunit.de/phpunit.phar
chmod +x phpunit.phar
sudo mv phpunit.phar /usr/local/bin/phpunit
phpunit --version
```

Nota: La última versión de PHPUnit requiere PHP 5.6

Podríamos también haber instalado PHPUnit usando "composer", pero sólo nos habría servido para CakePHP, de esta forma lo tendremos disponible para cualquier aplicación sobre la que queramos usar tests unitarios.

Desde cualquier directorio del sistema podemos ejecutar phpuniti simplemente escribiendo

```
phpunit
```

[23] Testing CakePHP 3: http://book.cakephp.org/3.0/en/development/testing.html
[24] PHPUnit: https://phpunit.de/

Para que CakePHP pueda almacenar los datos de las pruebas deberemos de configurar una base de datos de prueba, para ello en el fichero **config/app.php** descomentaremos y configuraremos el siguiente bloque de código:

```
/**
 * The test connection is used during the test suite.
 */
'test' => [
  'className' => 'Cake\Database\Connection',
  'driver' => 'Cake\Database\Driver\Mysql',
  'persistent' => false,
  'host' => 'localhost',
  //'port' => 'nonstandard_port_number',
  'username' => 'sportsnet',
  'password' => 'sportsnet1234',
  'database' => 'sportsnet_tests',
  'encoding' => 'utf8',
  'timezone' => 'UTC',
  'cacheMetadata' => true,
  'quoteIdentifiers' => false,
//'init' => ['SET GLOBAL innodb_stats_on_metadata = 0'],
],
```

Imagen 96 configuración BD testing:

Para ejecutar los test simplemente escribiremos una línea como:

```
phpunit  tests/TestCase/Model/Table/UsuariosTableTest
```

Para crear los Tests en CakePHP hay ue seguir las siguientes convenciones:

1. Los archivos que contienen los Tests deben residir en el directorio tests/TestCase/[Tipo]/[Subtipo]
   Dónde [Tipo] podría ser “Controller”, “Model”, “View” y subtipo sería “Table”, “Helper” o “Component”.
   El subtipo debe ir con el tipo adecuado. Por ejemplo, “Model” iría con “Tabla” y “Controller con “Component”.

2. Los nombres de los ficheros debe terminet con “Test.php”. Por ejemplo: *UsuariosTableTest.php* o *UsuariosControllerTest.php.*

3. Las clases deben extender *Cake\TestSuite\TestCase*, *Cake\TestSuite\IntegrationTestCase* or *\PHPUnit_Framework_TestCase.*

4. Al igual que el resto de clases en CakePHP, los nombres de las clases deben coincidir con el nombre del fichero. Por ejemplo, el fichero *UsuariosTableTest.php* contendrá la clase *Class UsuariosTableTest extends TestCase.*

5. El nombre de cualquier método que contenga un test (es decir, sentencias “assertion”) debería empezar con “test”, por ejemplo *testIndex()*, *testView()*, *testDelete()*. También se puede usar la anotación *@test* para marcar los métodos como tests.

En cualquier test unitario existe una función básica que se encarga de inicializar los objetos necesarios para el test. Se trata de la función ***setUp().***

Fichero: sportsnet/tests/TestCase/Model/Table/UsuariosTableTest.php

```
/**
   * setUp method
   * @return void
   */
 public function setUp() {
    parent::setUp();
    $config = TableRegistry::exists('Usuarios') ? [] : ['className' => 'App\Model\Table\UsuariosTable'];
     $this->Usuarios = TableRegistry::get('Usuarios', $config);
 }
```

Imagen 97: TestCase - función básica setUp()

6.2 Fixtures

Cuando realizamos tests sobre el código de los modelos y la base de datos, necesitamos usar las "fixtures", de forma que podamos generar datos ficticios de forma automática. De esta forma, podemos realizar tests con datos sin interferir en los datos reales de la aplicación. Otro beneficio adicional es que podemos trabajar y probar la aplicación con datos, de forma que los errores son descubiertos durante el propio desarrollo y además podemos ir comprobando el aspecto de la propia aplicación junto con los datos.

Durante el transcurso de una fixture basada en un test, CakepHp lleva a cabo los siguientes pasos:

1. Crea las tablas necesarias para cada fixture.
2. Llena las tablas con datos, si éstos están disponibles en la fixture.
3. Ejecuta los métodos de test.
4. Vacía las tablas de fixture.
5. Elimina las tablas de fixture de la base de datos.

6.3 Generando Tests

La mejor forma de empezar con los test es dejar que CakePHP hage el trabajo por nostros, al menos, al principio. Vamos a usar la "consola bake" para generar nuestros tests y nuestro datos d eprueba o fixtures.

Para no extendernos demasiado y dado que la generación autom´tica de tests unitarios se hace siempre igual, crearemos únicamente el test para la tabla de usuarios.

Desde el directorio del proyecto, en la consola escribimos:

```
bin/cake bake test Table Usuarios
```

Cake empezará a preguntarnos por los diferentes objetos a crear:

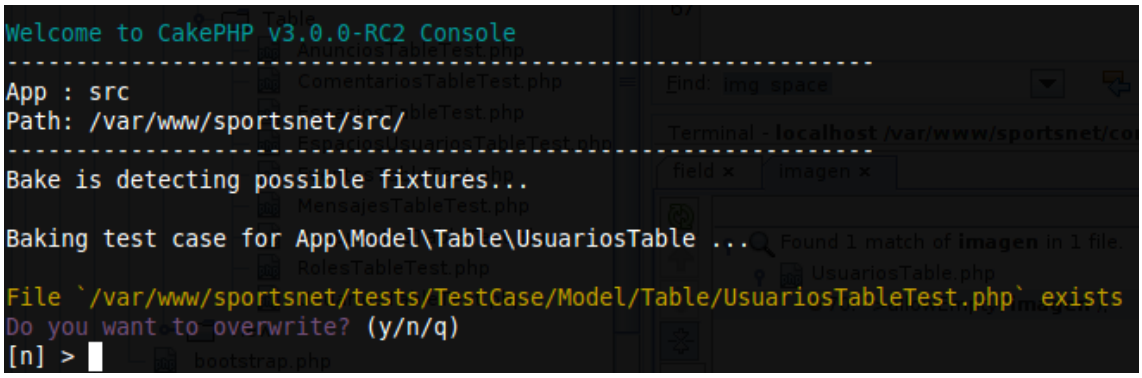


Imagen 98: Generación de tests unitarios

Hay que prestar atención, porque este proceso podría sobreescribir alguno de los ficheros que podríamos haber modificado. Hay que prestar atención, especialmente cuando generamos tests para controladores o para entidades, ya que son los más susceptibles de ser modificados, como por ejemplo la entidad Usuarios (***sportsnet/src/Model/Entity/Usuario.php*** dónde escribimos la función que nos permitía cifrar los passwords antes de guardarlos.

A continuación haremos lo mismo pero esta vez generaremos la fixture o datos datos de prueba

```
bin/cake bake fixture Usuarios
```

Al igual que antes CakePHP nos guiará para crear los objetos necesarios.

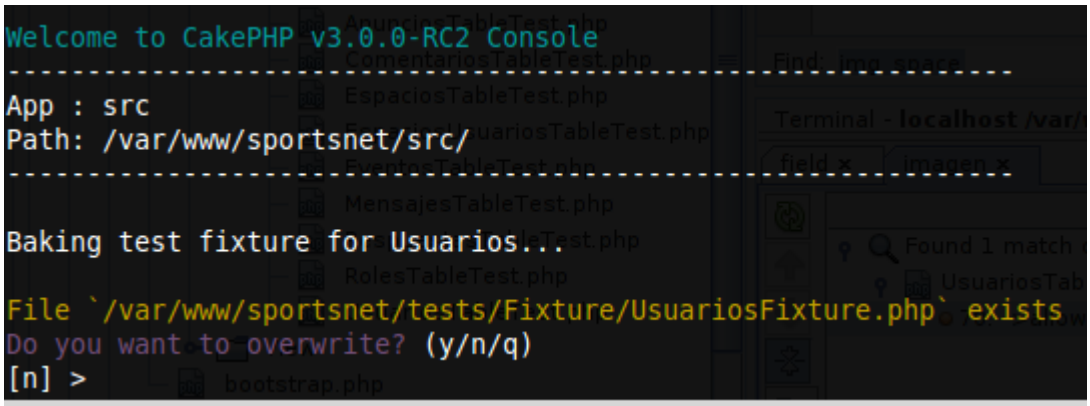


Imagen 99: Generación de fixtures

Una vez hecho esto podemos ya empezar a probar nuestro primer test unitario. Para ello ejecutamos desde la consola

```
phpunit tests/TestCase/Model/Table/UsuariosTableTest
```

La salida debería ser algo como:

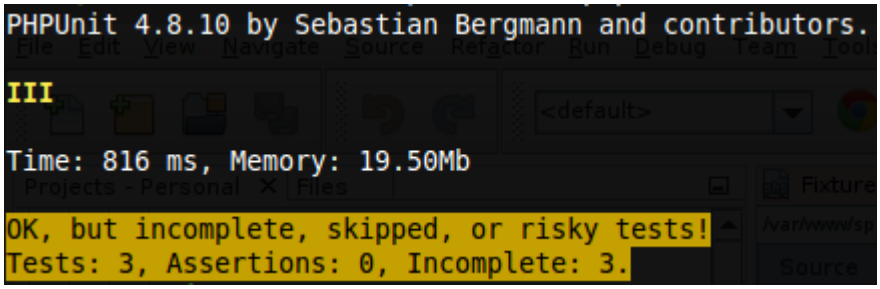


Imagen 100: resultado test unitario

El test nos dice que ha ido todo bien, bueno casi todo bien.
Nos dice que se han realizado 3 test con éxito, eso significa que, al menos se ha intentado hacer insert en la tabla con tres registros. Pero que ha habido también 3 tests más (o acciones) que no se han realizado.

Observemos que ha creado CakePHP por nosotros.

Fichero: tests/TestCase/Model/Table/UsuariosTableTest.php

```
class UsuariosTableTest extends TestCase {

    public $fixtures = [
        'Usuarios' => 'app.usuarios',
        'Anuncios' => 'app.anuncios',
        'Espacios' => 'app.espacios',
        'Comentarios' => 'app.comentarios',
        'Respuestas' => 'app.respuestas',
        'Eventos' => 'app.eventos',
        'EspaciosUsuarios' => 'app.espacios_usuarios',
        'Roles' => 'app.roles',
        'Mensajes' => 'app.mensajes'
    ];

    public function setUp()    {
        parent::setUp();
        $config = TableRegistry::exists('Usuarios') ? [] : ['className' => 'App\Model\Table\UsuariosTable'];
        $this->Usuarios = TableRegistry::get('Usuarios', $config);
    }

    public function tearDown(){
        unset($this->Usuarios);
        parent::tearDown();
    }

    public function testInitialize(){
        $this->markTestIncomplete('Not implemented yet.');
    }

    public function testValidationDefault(){
        $this->markTestIncomplete('Not implemented yet.');
    }

    public function testBuildRules()
    {
        $this->markTestIncomplete('Not implemented yet.');
    }
}
```

Imagen 101: Testcase generado por CakePHP

Ahora podremos entender que significa „Incomplete: 3“

Al crear test unitarios, muchas veces necesitamos crear métodos vacíos porque aún no sabemos qué se va a validar exactamente. En ese caso PHPUnit evalúa estos métodos como ciertos, devolviendo falsos positivos. Tampoco podemos llamar a $this->fail() porque entonces se evaluaría siempre a falso y daría, en este caso, falsos negativos. Este tipo de acciones haría que la utilidad de los test Unitarios desapareciera. Por ese motivo se introducen en PHPUnite diferentes funciones que permiten simplemente dejar marcados los métodos con un cierto estado.

void markTestIncomplete(string $message): Establece el test como incompleto

void markTestSkipped(string $message): Indica que el método no debe evaluarse y debe omitirse su evaluación..

Más información en: https://phpunit.de/manual/current/en/incomplete-and-skipped-tests.html

parent::tearDown();

Este método debe de llamarse en todos los ficheros de test y es el encargado de limpiar todos los datos que se han usado en ese test. En este caso se encarga de limpiar todos los datos usados para la validación de los usuarios. Esto incluye, por supuesto, las tablas que se han creado para los test.

6.4 Errores durante los tests

Unable to insert fixtures for "App\Test\TestCase\Model\Table\ComentariosTableTest" test case. SQLSTATE….

Tendremos que revisar la Base de Datos. CakePHP debe eliminar todas las tablas que cree con los dtaos de prueba. Si esto no ocurriera por alun motivo, las siguientes ejecuciones de tests se ecnontrarían con datos en las tablas, lo que podría ocsaionar conflictos con los nuevos inserts que producen los tests.
Del mismo modo ocurriría si hemos añadido una nueva columna en alguna de las tablas y hemos actualizado los modelos, vistas, controladores y test para adecuarlos. En el caso que hubiera tablas o datos residuales del algún test anterior que no se hubieran eliminado. La herramienta de test podría encontrar que la tabla sobre la que pretende ejecutarse no contiene la nueva columna y eso produciría un error del tipo ***"columna no encontrada"***

Unable to insert fixtures for "App\Test\TestCase\Model\Table\UsuariosTableTest" test case. SQLSTATE[23000]: Integrity constraint violation: 1062 Duplicate entry 'Lorem ipsum dolor sit amet' for key 'email' in [/sportsnet/vendor/cakephp/cakephp/src/TestSuite/Fixture/FixtureManager.php, line 254]

Esto ocurre porque la columna email se ha declarado como UNIQUE, pero al automatizar la creación de las fixtures, se crean datos duplicados para las pruebas. Para corregirlo, editaremos el fichero afectado y cambiaremos el valor del campo. En este caso se trata del fichero de fixture correspondiente a la tabla de usuarios: ***sportsnet/tests/Fixture/UsuariosFixture.php***

Toda la información sobre testing en CakePHP3 puede encontrarse en:
http://book.cakephp.org/3.0/en/development/testing.html

6.5 Consideraciones finales

Los test, tanto unitarios como de integración, se van generando a medida que las funcionalidades van requiriéndolo. La creación de tests para todas y cada una de las funciones de una aplicación es una tarea enormemente costosa y escapa al propósito de este proyecto. A partir de lo descrito se deja como trabajo de una futura ampliación, a generación de tests detallados para cada funcionalidad.

Por otro lado, la validación a través de herramientas web como validator.w3.org para comprobar el ajusta a estándares web no es posible en este momento, puesto que la web no se encuentra alojada en ningún

servidor accesible por estas herramientas. No obstante, es muy importante hacer estos tests en cuanto la web esté alojada en un servidor al que estas herramientas puedan acceder.

7 Conclusiones

Las nuevas funcionalidades que nos ofrece CakePHP en comparación con la versión anterior nos muestran que el desarrollo de aplicaciónes no se ha facilitado en gran medida, pero por el contrario se ha dotado al framework de mayor control sobre el manejo de datos y se ha dotado el mismo con nuevas herramientas que, si bien, no son triviales de manejar, sí ofrecen la potencia necesaria para poder desarrollar cualquier requisito necesario para la aplicación.

Entre las características más importantes podemos destacar:

- **Nuevo ORM:** Uno de los cambios más importantes es el nuevo ORM[25], que es más flexible y potente. Ha sido desarrollado desde cero y se han solucionado algunos problemas y deficiencias respecto a las versiones anteriores del framework. Este nuevo ORM utiliza el patron DataMapper[26] y mejora la generación de consultas flexibilizando el retorno de resultados en formato de objetos o de arrays.

- **Enrutamiento más rápido y flexible:** Nos ayuda a estructurar mucho mejor nuestras rutas donde podemos transformar una cadena de URL a partir de una matriz de parámetros.

- **Migraciones mejoradas:** Basado en PHINX nos provee un plugin para desplegar nuestras migraciones de una manera más sencilla.

- **Mejor Internacionalización:** El subsistema I18n ha sido totalmente reescrito y ahora aprovecha la extensión intl de PHP para ofrecernos mensajes de traducción y librerías más completas.

- **Mejora de la barra de herramientas de depuración:** DebugKit es una gran herramienta que nos ayuda a depurar nuestras aplicaciones CakePHP, esta herramienta también ha sido reescrita para ofrecer más funcionalidades y aunque no viene instalado por defeto, no resulta complicado ponerlo a funcionar.

- **Uso de composer:** Composer es una herramienta de manejo de dependencias de nuestras aplicaciones en PHP. En versiones anteriores, el único uso de composer era para instalar paquetes o plugins independientes. En esta nueva versión, todo, incluyendo el propio CakePHP se puede instalar con composer. Esto convierte a esta maravillosa herramienta en el método oficial de instalación y gestión.

- **Bibliotecas independientes:** Una parte de las librerías de CakePHP se han dividido para que puedan ser usadas fuera de las aplicaciones en CakePHP o en aplicaciones que utilizan la versión 2.x de forma que éstas puedan aprovechar las nuevas características de CakePHP3.0.

[25] Mapeo Objeto-Relacional: https://es.wikipedia.org/wiki/Mapeo_objeto-relacional
[26] DataMapper pattern: https://olagato.wordpress.com/2009/11/03/datamapper-pattern-una-mirada-hacia-el-futuro/

En resumen, tenemos un nuevo y más robusto framework que nos ofrece nuevas y potentes herramientas con las que trabajar. En su contra tiene, como ya hemos comentado, que la complejidad de uso no disminuye en la misma proporción que sus mejoras. Esperemos que los desarrolladores tengan este tipo de cosas en cuenta para futuras versiones.

Conclusiones finales

Si bien es cierto que no hemos desarrollado el 100% de la aplicación, sí que pensamos que hemos cumplido con los objetivos planteados en el apartado de introducción

- Obtener una visión general de las diferencias entre las versiones 2 y 3 del Framework de desarrollo CakePHP.

- Establecer un precedente de desarrollo para futuros proyectos que utilicen el mismo framework que se usa en este proyecto. En este caso CakePHP 3.

En cuanto al primer objetivo, queda claro que hemos constatado que el sistema de manipulación de datos de CakePHP3 es mucho más potente que el de CakePHP 2 y que, por tanto no es una adaptación trivial la que hay que hacer para poder migrar proyectos de la versión anterior a esta nueva versión.
Esto mismo viene a hacernos cumplir el segundo objetivo, que consiste en servir de referencia para posibles migraciones de proyectos antiguos a versiones más modernas de CakePHP3.

Debido a la naturaleza propia del trabajo y a la duración limitada del mismo, es evidente que no se puede profundizar más en estos objetivos y de deja propuesta esta profundización en los cambios como futuras mejoras.

8 Bibliografía y enlaces

Ertel A., Laborenz K. (2015). Responsive Webdesign 2a Ed. Bonn: Rheinwerk Verlag GmbH (ant. Galileo Computing). ISBN: 978-3-8362-3200-5

Krug S. (2015). Don't Make Me Think – Web & Mobile Usability 3a Ed. Frrechen: mitp Verlags GmbH & Co. KG. ISBN: 978-3-8266-9705-0

Martínez T. (2012). Aplicación Web para la gestión de entrenos de deportistas. Proyecto Final de carrera. Valencia: Universidad Politécnica de Valencia.

Moret J. M. (2013). Aplicación Web de bases de datos usando el Framework CakePHP. Proyecto Final de carrera. Valencia: Universidad Politécnica de Valencia

*Apache***.** <http://apache.org> [Consulta: 01 de Agosto de 2015]

HTLM5 **.** <http://www.w3schools.com/html/html5_intro.asp> [Consulta: 01 de Agosto de 2015]

CakePHP v3 – CookBook. <http://book.cakephp.org/3.0/en/index.html> [Consulta: 01 de Agosto de 2015]

CakePHP v2 – CookBook. <http://book.cakephp.org/2.0/en/index.html> [Consulta: 01 de Agosto de 2015]

Compass. <http://compass-style.org/> [Consulta: 01 de Agosto de 2015]

CSS3. <http://www.w3schools.com/css/css3_intro.asp> [Consulta: 01 de Agosto de 2015]

CSS-Tricks. Code Snippets. <https://css-tricks.com/snippets/> [Consulta: 01 de Agosto de 2015]

MySQL. <https://www.mysql.com/> [Consulta: 01 de Agosto de 2015]

Netbeans. <http://netbeans.org> [Consulta: 01 de Agosto de 2015]

phpMyAdmin. <http://www.phpmyadmin.net> [Consulta: 01 de Agosto de 2015]

PHPUnit Manual <https://phpunit.de/manual/5.0/en/index.html> [Consulta: 07 de Octubre de 2015

Portal de CakePHP. <http://cakephp.org> [Consulta: 01 de Agosto de 2015]

Portal de PHP. < http://php.net> [Consulta: 01 de Agosto de 2015]

Sass. <http://sass-lang.com/> [Consulta: 01 de Agosto de 2015]