Universidad Carlos III de Madrid
www.uc3m.es

Grado en Ingeniería en Tecnologías de Telecomunicación

2014-2015

Trabajo Fin de Grado

HERRAMIENTA WEB PARA CREAR MINI-CURSOS ON-LINE

Sergio Velázquez Vozmediano

Tutor

Mario Muñoz Organero

Leganés, Septiembre de 2015

Resumen

La aparición de dispositivos móviles en los últimos años, con las mismas funcionalidades que un ordenador de sobremesa, ha revolucionado de igual manera la forma de desarrollar aplicaciones y páginas webs. Ahora es necesario que estas sean compatibles con distintos tamaños de pantalla y provean al usuario una experiencia igualmente satisfactoria.

Como respuesta a ello, se han popularizado dos herramientas que parecen que van a, en cierto modo, monopolizar el desarrollo web: Bootstrap y AngularJS. La primera es un framework que permite agilizar y amoldar el diseño estético de la página a varios dispositivos. La segunda es una librería SPA JavaScript que permite optimizar el ancho de banda consumido, obteniendo solo aquella información necesaria.

El trabajo consta acerca de la creación de una aplicación web que permita a los usuarios cursar asignaturas online, haciendo uso de las herramientas anteriormente nombradas.

Para ello, se deberá configurar, por un lado, una base de datos MySQL que permita almacenar información sobre, usuarios, cursos, tests de autoevaluación, etc. Por otro lado, se desarrollará un sistema de ficheros donde se guardará el contenido que los profesores suban a los alumnos, tales como PDFs, DOCX, etc.

Todos estos datos serán accesibles a través de Java Servlets, que además se hacen cargo de la correcta autenticación del usuario en el sistema. Por último, los alumnos interactuarán a través de los controladores de AngularJS que actualizarán dinámicamente la información mostrada.

Una vez acabado el desarrollo, se configurará la PaaS gratuita Openshift, de modo que se despliegue la aplicación y sea accesible al público.

Abstract

In the last years, the appearance of mobile devices, that support the same functionalities as a common desktop computer, has revolutionized also the way to develop web pages and apps. Nowadays, they need to be suitable with different screen sizes so that they provide the same satisfying experience to the user.

In response, two tools have become very popular, and they seem to monopolize the market in the near future: Bootstrap and AngularJS. The first one is a framework that allows the developer to speed up the design and make it mobile-friendly. The second one is a JavaScript SPA Library that optimizes the bandwith so only necessary info is retrieved.

This Bachelor's thesis is about creating a web application that enables users to access to different courses by using the two tools mentioned above.

For that, first, a MySQL database will be configured to store data about users, courses, tests. Then, a file system will be created so that the content uploaded by the teachers to the students can be saved, such as PDFs, DOCX, etc.

All this data will be accessible through Java Servlets, which also will manage user identification. Finally, the students will interact trough AngularJS controllers so the info presented to them is dynamically updated.

Once the development is finished, Openshift PaaS will be configured so the application is deployed and open to the public.

Índice de tablas

Índice de figuras

Índice de códigos

Índice

Abreviaciones

SPA Single Page Application

PaaS Platform as a Service

AJAX Asynchronous JavaScript And XML

XML eXtensible Markup Language

JSON JavaScript Object Notation

DOM Document Object Model

JSP JavaServer Page

1. Introducción, motivación y objetivos

a. Historia del desarrollo web

Cuando se empezaron a dar los primeros pasos en materia de desarrollo web, el principal objetivo marcado por el creador del lenguaje HTML, Tim Berners-Lee, era publicar la información que almacenaba el CERN, la Organización Europea para la Investigación Nuclear, de modo que fuese más accesible entre los científicos que allí trabajaban, sin importar el ordenador que estuviesen utilizando.

Tras unos años de popularización del nuevo lenguaje, en 1994 Rasmus Lerdorf comenzó a desarrollar PHP, de modo que empezaba a introducir las primeras interacciones posibles entre las páginas webs y los usuarios. Después se lanzarían distintas plataformas y scripts, tales como JavaScript, Flash, que ahondan en la mentalidad de una mayor participación del usuario a la hora de visualizar el contenido.

b. Actualidad del desarrollo web

En el día de hoy, tras una gran proliferación de smartphones, tablets, y demás dispositivos móviles a partir del año 2007, el concepto de web como plataforma está muy extendido. Es decir, las aplicaciones tradicionales de escritorio ahora usan las ventajas de la conexión a internet para personalizar el contenido mostrado a cada usuario concreto. Además, todo parece indicar que se seguirá profundizando en este sentido en años futuros, como demuestra el desarrollo de múltiples aplicaciones por parte de los gigantes del sector: Google y Facebook.

No obstante, estos dispositivos poseen unas características muy distintas entre sí, tales como diferentes tamaños de pantalla, accesibilidad, ancho de banda disponible, etc. También, hay numerosos navegadores web en el mercado que interpretan con distintos matices el lenguaje HTML.

Esto hace que la complejidad aumente exponencialmente para un desarrollador web, que debería de emplear tiempo extra en optimizar para cada una de las combinaciones posibles.

Al calor de esta demanda, han surgido numerosos proyectos de código libre a lo largo de los últimos años, que intentan paliar alguno de estos problemas.

c. Motivación

La motivación de este proyecto es, en primer lugar, echar un primer vistazo sobre las herramientas que son populares en el desarrollo web y que son fundamentales en la industria, puesto que es un mercado en alza. Para luego entrar en materia con las más importantes obteniendo una base de conocimiento.

Además, es interesante abordar desde cero un proyecto relativamente grande porque de esta manera se perfecciona el flujo de trabajo, acercándonos así a un desarrollo más óptimo.

d. Objetivo

El objetivo consiste en crear una aplicación web sobre cursos online que haga uso de algunas de las herramientas punteras en este campo.

El usuario podrá registrarse en el sitio web y una vez dentro podrá apuntarse a cursos de doce semanas que aún no hayan comenzado. Por otro lado, los profesores serán los encargados de subir contenido a dichos cursos, que incluirá documentos PDF, DOCx, imágenes, exámenes de autoevaluación. Además, habrá un chat que comunicará a profesores y alumnos y un sistema de notificaciones que les avisará de cualquier contenido nuevo.

Por último, también se prevé la creación de un panel de administrador que facilite manejar el sitio web.

2. Estado del arte

Como hemos comentado en el capítulo anterior, el desarrollo web en la actualidad está explorando herramientas que faciliten la compatibilidad con distintos dispositivos y que optimicen dichos recursos: **Front-End Frameworks** y **librerías SPA**, respectivamente. Además, también se tratarán las distintas opciones para desplegar la aplicación en un servidor y abrirla al público.

a. Front-End Frameworks

Un front-end framework es una herramienta que engloba una colección de contenido tanto HTML, como CSS y JavaScript, listo para ser usado. No es difícil, para el desarrollador medio, crear su propio contenido en alguno de estos lenguajes, no obstante, por las razones que se han dado antes en temas de compatibilidad, hoy en día, el uso de frameworks está extendido como si de una necesidad se tratase.

Un framework nos permite crear páginas webs adaptables y flexibles a cualquier tamaño, así como acelerar y simplificar la creación de contenido CSS, con altos niveles de personalización. Todo esto con gran robustez, pues el mismo código ha sido ya probado por miles de personas anteriormente.

No obstante, se ha producido desde 2010 una masificación de frameworks, que poseen distintos tipos de complejidad en base a la personalización que permiten. Además, algunos incluso poseen librerías derivadas de JQuery para la realización de peticiones HTTP asíncronas.

Algunos frameworks famosos son:

- BootMetro
- Kickstrap
- Foundation
- Semantic UI
- Pure
- Uikit

El problema que surge con tal cantidad para elegir es que no se produce una estandarización clara, y por ello, la comunidad de desarrolladores se divide o invierte su tiempo en proyectos que, en ocasiones, son abandonados o no son actualizados regularmente. Con el paso del tiempo, algunos han adquirido más popularidad que otros, es el caso de **Bootstrap.**

Para elegir finalmente un framework que esté ampliamente documentado, cuyos niveles de popularidad sean mayores, y que siga en desarrollo, de modo que el conocimiento adquirido en la realización del proyecto continúe siendo válido en el futuro, podemos echar mano de una herramienta muy útil de Google, **Google Trends.**

Si comparamos el número de búsqueda de algunos de los principales frameworks durante los últimos años, obtenemos:

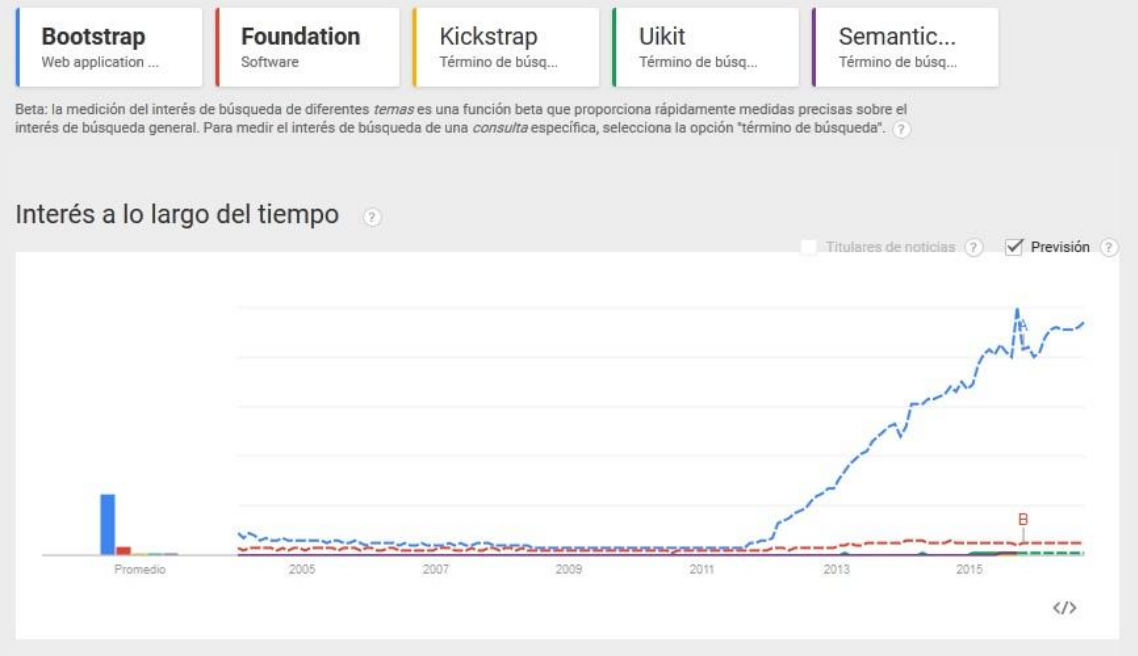


Figura 2.1 *Comparación de popularidad de Frameworks en Google Trends*

Por tanto, parece claro que el elegido es **Bootstrap**, puesto que goza de amplia y creciente popularidad y además está previsto que siga así en el futuro.

1. Bootstrap

Su funcionalidad más atractiva es la de proporcionar una rejilla de doce columnas que divide el espacio de la página web y que se redistribuirán de manera

ordenada tras cambiar el tamaño de pantalla. Estas, además, pueden combinarse entre ellas dando lugar a diferentes configuraciones:

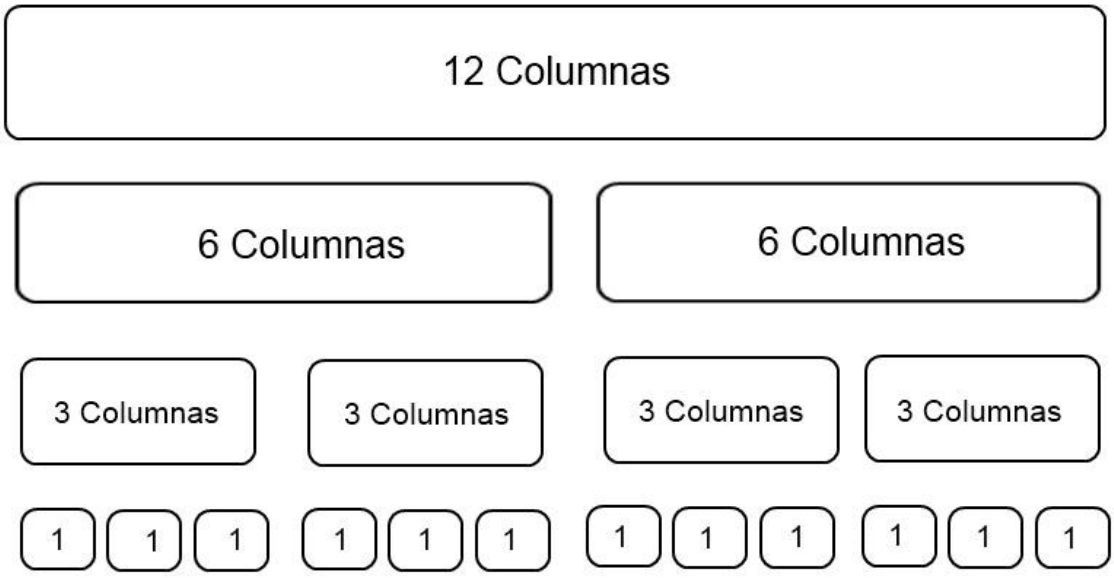


Figura 2.2 *Sistema de rejilla de Bootstrap*

No obstante, también es posible especificar de manera concreta la distribución que seguirán las columnas según el tamaño del dispositivo mediante las clases de CSS:

- Col-xs: Para dispositivos de anchura menor o igual a 767px.
- Col-sm: De anchura mayor o igual a 768 px.
- Col-md: De anchura mayor o igual a 992 px.
- Col-lg: De anchura mayor o igual a 1200 px.

Estas clases mantendrán su posición mientras que la anchura cumpla sus parámetros, y solo en caso de no hacerlo las columnas quedarán apiladas, cambiando la distribución de la página web pero de manera ordenada.

Con la última versión de Bootstrap, se añadió la posibilidad de ocultar y mostrar elementos según la resolución, con las clases de CSS .visible y .hidden.

A modo de ejemplo sobre el funcionamiento del sistema de rejilla, el siguiente código HTML define, en una fila, varios bloques que ocupan distintas columnas según la resolución de pantalla:

```
<div class="row">
            <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2">
                  Bloque 1
            </div>
            <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2">
                  Bloque 2
            </div>
            <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2">
                  Bloque 3
            </div>
            <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2">
                  Bloque 4
            </div>
</div>
```

Código 2.1 *Ejemplo del sistema de rejilla de Bootstrap*

El código funciona de la siguiente manera:

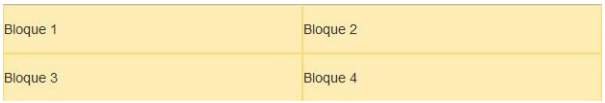
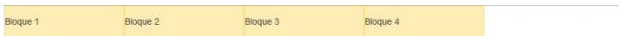
Resolución	Modo	Explicación	Visualización
< 767 px	Extra small	Cada bloque ocupa 6 columnas, y por tanto, se apilan de dos en dos.	
> 768 px	Small	Cada bloque ocupa 4 columnas, por tanto, se apila el bloque 4.	
> 992 px	Medium	Cada bloque ocupa 3 columnas, rellenando toda la fila.	
> 1200 px	Large	Cada uno ocupa 2 columnas, 4 columnas quedan libres.	

Tabla 2.1 *Explicación del código de ejemplo 2.1*

A parte del sistema de rejilla explicado con anterioridad, Bootstrap también proporciona, por un lado, clases CSS que otorgan un diseño estético a grupos de botones, campos de input, iconos, barras de progreso, etc. Por otro lado, también hay disponibles componentes de JavaScript, que permiten condensar mayor información en una única página html haciéndola más dinámica, como pueden ser modals, popovers, dropdowns, etc.

b. Librerías SPA

Una SPA es una página web que carga dinámicamente los recursos requeridos por el usuario sin realizar nunca una recarga completa de la web, como ocurría tradicionalmente. Esto proporciona una experiencia mucho más fluida al cliente puesto que su uso se asemeja al de una aplicación de escritorio clásica. Además, se produce una optimización del ancho de banda usado, puesto que solo se piden aquellos elementos que necesitan una actualización y no la página entera.

Estas SPA pueden ser desarrolladas mediante frameworks o librerías de JavaScript, que utilizan peticiones asíncronas (AJAX) para refrescar la información demandada por el usuario. Por otro lado, el transporte de datos se realiza en XML o JSON, que deben ser tratados debidamente antes de actualizar el DOM.

No obstante, hay algunas características que aún chocan con herramientas pensadas para el modelo de página web clásico, como puede ser el historial del navegador o la indexación para los motores de búsqueda.

Como en el caso de los Front-End Frameworks, también proliferaron varias de estas librerías, siendo especialmente importantes:

- AngularJS
- Ember.js
- Meteor.js
- Backbone.js
- Knockout.js

Se diferencian básicamente en priorizar y centrarse en unas características más que en otras. Como en el caso anterior, nos gustaría poder utilizar una herramienta ampliamente documentada y con gran soporte, siguiendo el procedimiento anterior:

Figura 2.3 *Comparación de popularidad de SPA Frameworks en Google Trends*

Por tanto, utilizaremos **AngularJS**.

1. AngularJS

La gran baza de esta librería es que fue desarrollado por miembros de Google, con lo que en cuanto a soporte y publicidad barre a sus competidores. Tiene como base la arquitectura Model-View-Controller.

La arquitectura MVC, que ha sido ampliamente utilizada en el desarrollo web con anterioridad, delimita tres áreas con distintas funciones:

- Model: Son los datos que sustentan la aplicación. Está formado por el objeto scope.
- View: Es la información que se muestra al usuario.
- Controller: Es la parte con la que el usuario interactúa, de modo que modifica el Model, y actualiza la View.

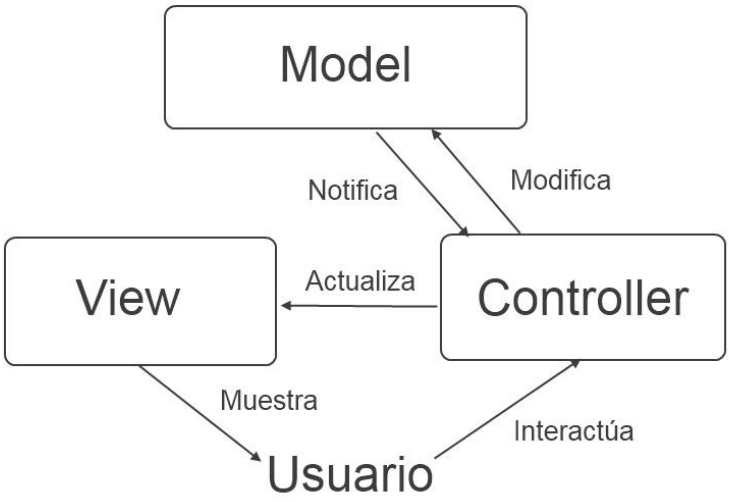


Figura 2.4 *Arquitectura MVC*

Sin embargo, AngularJS añade el 2-way data binding, es decir, ligar los datos en doble sentido. Básicamente funciona como MVC solo que ahora si algo cambia en View se verá reflejado automáticamente en Model, y viceversa. Y Controller pasa a llamarse ViewModel, siendo la nomenclatura de la arquitectura **MVVM**. Esto hace, por ejemplo, que si un usuario introduce un dato en campo de input, inmediatamente el model responde a ello, facilitando funcionalidades como la de filtrar datos dinámicamente, autocompletar, etc.

Por otro lado, existen las directivas que son etiquetas que se añaden a elementos HTML, pasando así de ser estáticos a dinámicos. Algunas de ellas son:

- Ng-app : Debe ser declarado en la raíz del documento HTML de manera que englobe a todo el código que va a utilizar AngularJS.
- Ng-model: Automáticamente introduce el elemento en el scope. Por ejemplo, suele usarse en campos de inputs.
- Ng-if: Verifica una expresión y en caso de ser verdadera, muestra el elemento. Si es falsa, dicho elemento no será mostrado.

- Ng-view: Es la directiva que maneja las distintas vistas posibles de la aplicación.
- Ng-click: Al hacer click sobre el elemento se ejecutará la función especificada.

No obstante, estas son solo algunas de las que ha desarrollado Google, la amplia comunidad crea constantemente módulos que pueden ser añadidos fácilmente a cualquier proyecto. Por ejemplo, sistemas de notificaciones, subida de archivos, traducción de textos, etc.

Otra característica importante es el de las rutas, es decir, se puede definir una parte del documento en el cual se insertará otro en el momento en el que cambie la URL, acercándonos de esta manera al comportamiento clásico de las páginas web. A estas rutas se les puede asignar un controlador y scope específico y distinto.

c. Despliegue de la aplicación web

Una vez que la aplicación ha sido desarrollada en local en el IDE de Eclipse, el siguiente paso de cara a abrirla al público es subir la aplicación a un web host. La oferta de web hosting es muy amplia, no obstante, hay que buscar uno que se ajuste a las características de nuestra aplicación, lo que la limita bastante. Para que soporte a nuestra aplicación necesitamos:

- Soporte para lenguaje Java.
- Servidor de aplicación Tomcat.
- Base de datos MySQL.
- Posibilidad de crear el sistema de ficheros.

Además, puesto que el tema del proyecto es la creación de la aplicación web, buscaremos un hosting de tipo PaaS, es decir, Platform as a Service, de manera que el proveedor suministre herramientas y un entorno favorable y sencillo para desplegar la aplicación. Otra posible solución sería ir un paso más allá y utilizar un hosting de tipo

IaaS, es decir, Infrastructure as a Service, donde el proveedor suministra hardware virtualizado, con lo que las posibilidades de personalización aumentan.

Una vez discurrido esto, la oferta de PaaS que soporten el lenguaje Java son:

Nombre	Pros	Contras
Cloudbees	• Interfaz muy simple. • MySQL	• La cuenta de prueba solo dura dos semanas.
Google App Engine	• Soporte de Google • MySQL • Google APIS	• Complicado de desplegar • No es posible crear el sistema de ficheros • Imposibilidad de trasladar la aplicación en el futuro.
Openshift	• Posibilidad de trasladar la aplicación • MySQL • Sistema de ficheros	• Interfaz algo compleja.

Tabla 2.2 *Oferta de PaaS disponibles*

Estas son algunas de las más conocidas, también existen otras interesantes como Jelastic. En nuestro caso, puesto que se trata de desplegar la aplicación a modo de prueba y con el simple fin de investigar, elegiremos la plataforma que nos permite poner online la aplicación a coste cero y mantenerla en el tiempo: **Openshift**.

1. RedHat Openshift

Cuenta con el apoyo de Red Hat, conocida por ser una empresa que dió nombre a una popular distribución de Linux. Es necesario crear un repositorio local que está sincronizado con la aplicación y la actualización se realiza mediante línea de comandos, lo cual aumenta ligeramente la complejidad frente a otras opciones.

Es posible hostear páginas de aplicaciones ya desarrolladas como WordPress o Drupal. Las opciones de servidor varían entre distintas versiones de Tomcat, PHP, Ruby, Python, etc. Además es posible añadir diferentes bases de datos como MySQL, MongoDB, PostgreSQL.

3. Diseño y arquitectura

La aplicación, que estará disponible en dos idiomas, Español e Inglés, y que recibirá el nombre de **Teach Us!**, comenzará con una vista inicial que servirá de bienvenida a los usuarios. En ella incluiremos datos sobre los profesores docentes y sus titulaciones y sobre algunos de los cursos que están disponibles en ese momento. Estéticamente utilizará la herramienta comentada anteriormente, Bootstrap, no obstante no utilizará AngularJS, la cual reservaremos para la propia aplicación, es decir, una vez que el usuario entre en el sistema.

El nivel más bajo de la aplicación serán, por un lado, la base de datos MySQL, que almacenarán los datos de registro de los usuarios, información sobre los cursos, test de autoevaluación, mensajes de los chats, y por otro, el sistema de ficheros, que contendrá el contenido que los profesores suban a cada curso. Estos solo serán accesibles a través de Java Servlets.

El sistema de ficheros esquematizado es:

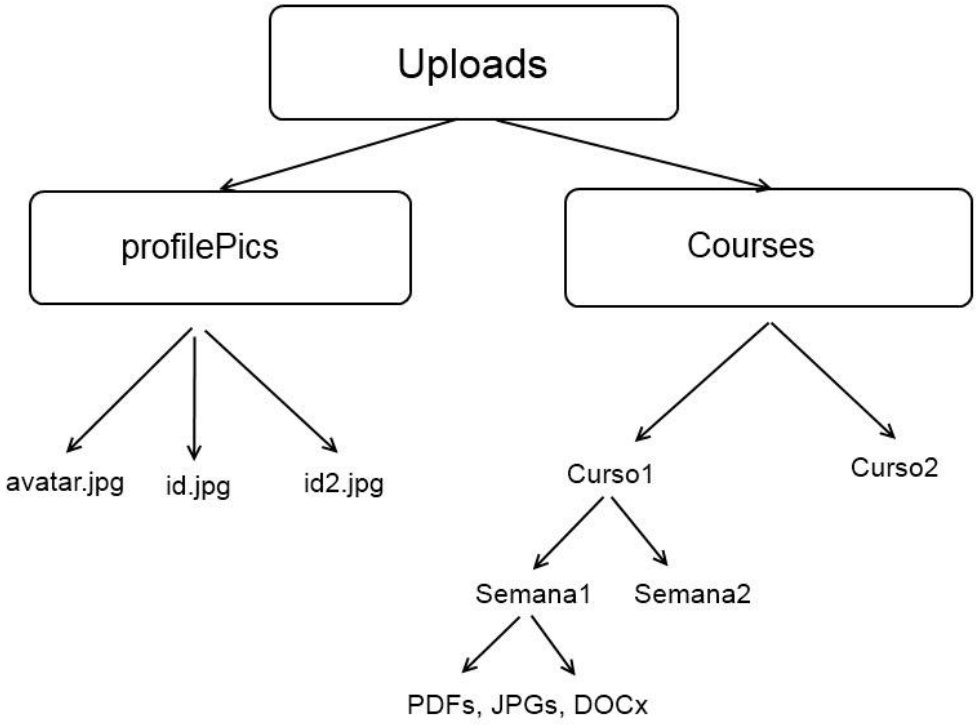


Figura 3.1 *Esquema sistema de ficheros*

a. Index

Para las peticiones que se deben realizar tanto para el registro como para el login, utilizaremos AJAX, mediante la librería JQuery. Por otro lado, este fichero index será del tipo JSP, que nos servirá para manejar el idioma de la página según las preferencias del usuario, así como para obtener los datos sobre los cursos disponibles indicados con anterioridad.

1. Registro

El registro se realizará tras recibir en el servlet Register los datos de un formulario en la página principal. En dicho servlet, se realizarán las operaciones necesarias para trasladar dichos datos a la base de datos, aplicando una función de hash basado en SHA256 junto a un salt generado automáticamente, de manera que la contraseña nunca se almacena pero es posible comprobar si es correcta. Además se enviará un correo con las credenciales al email indicado por el usuario a través de un servidor smtp gratuito proporcionado por Google.

2. Login

En el login, en un primer momento en el servlet SignIn se comprobará que los datos introducidos en el formulario son correctos. Después se hace sendRedirect hacia el servlet Overview, el cual cargará los datos necesarios para que funcione correctamente la aplicación.

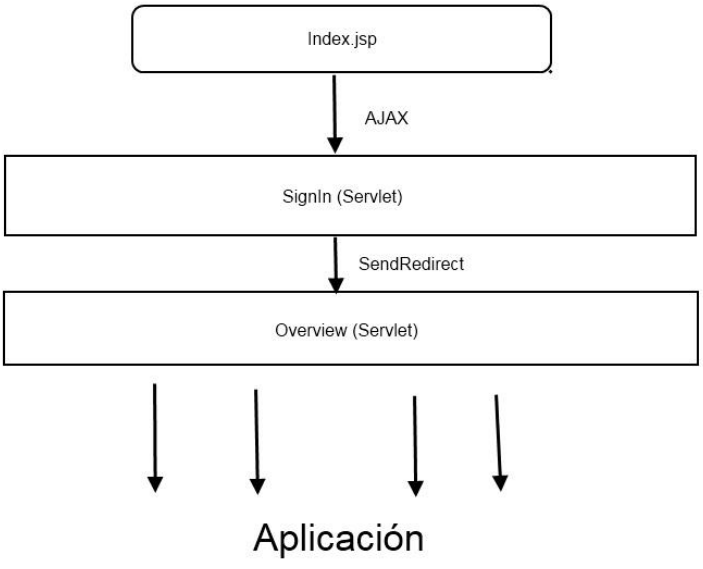


Figura 3.2 *Login*

3. ResetPassword

Se permitirá al usuario el reseteo de su contraseña con indicar su email, este será buscado en la base de datos, y en caso de ser encontrado, se generará una contraseña aleatoria, que será enviada a dicho email. Además se actualizará el hash y el salt almacenados.

b. Aplicación

Una vez dentro de la aplicación seguiremos el esquema general de la **Figura 3.2** (siguiente página), que deriva de la anteriormente contemplada en la **sección 2.b.i**, es decir la arquitectura MVVM.

Además, siempre habrá dos Controllers superpuestos, uno el general, llamado TeachUsController, que se hará cargo de aquellos elementos que siempre estarán presentes en la aplicación, como el perfil de usuario, barra de navegación, etc. Y otro el específico de la ruta. Además, estas rutas/vistas serán cuatro.

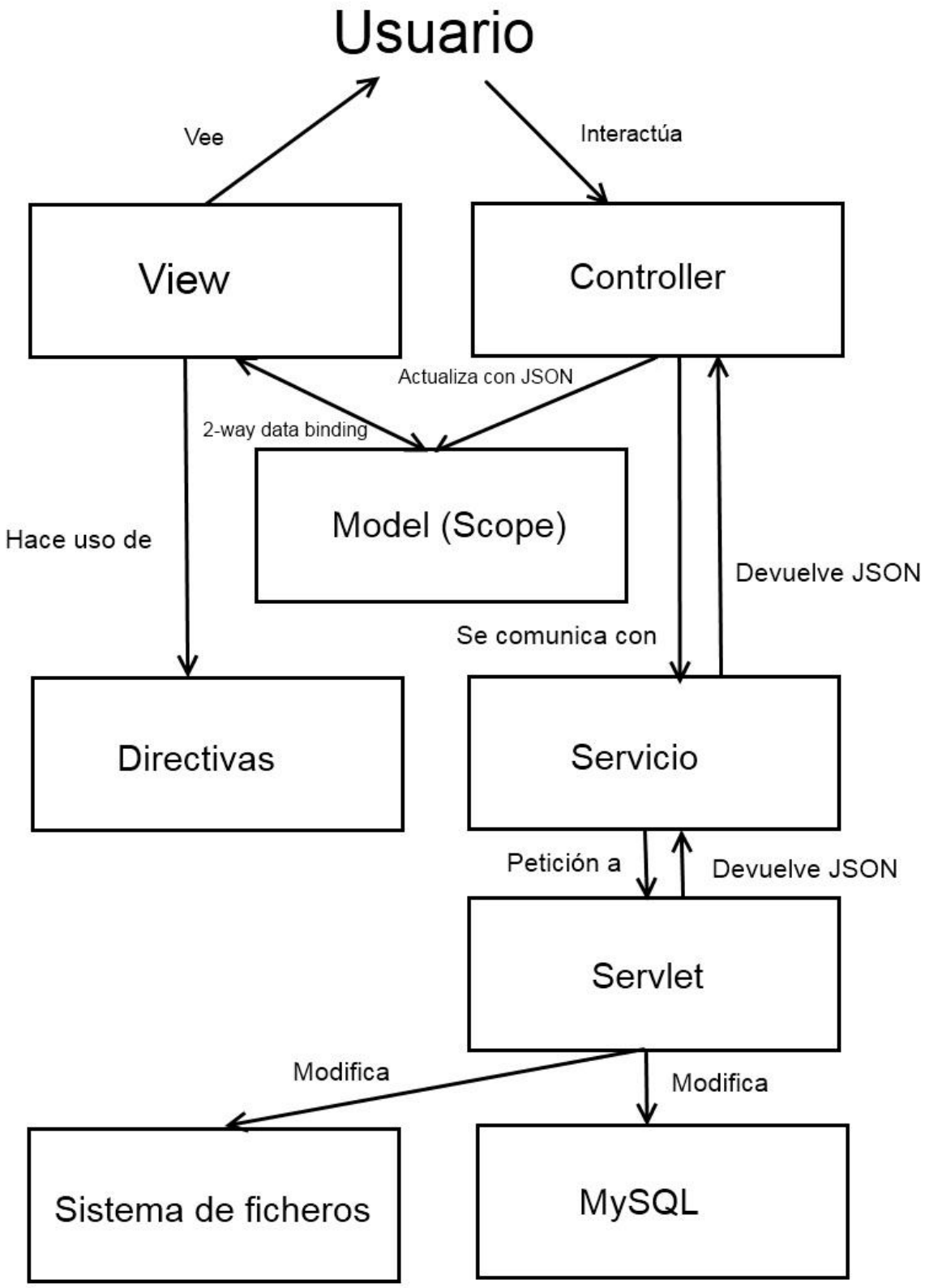


Figura 3.3 *Arquitectura general de la aplicación*

1. Overview

Tendrá el TeachUsController como controlador, es decir, no es una ruta sino que contendrá las funciones para cambiar la foto de perfil, salir de la aplicación y cambiar el idioma, que estarán presentes en la barra de navegación. Además, se encargará del sistema de notificaciones.

1. Servlets

Los servlets que serán usados para llevar a cabo las funciones indicadas anteriormente son: ChangePicture, ChangeLanguage, SignOut y GetNotifications.

2. Home

Esta vista dará la bienvenida al usuario a la aplicación, mostrando algunos avisos sobre actualizaciones que han ocurrido en los cursos a los que está apuntado. Además, podrá comprobar las materias en las que está apuntado, así como acceder a ellas directamente. También, con fines publicitarios se mostrarán algunos cursos que están disponibles para el idioma que haya sido seleccionado.

1. Controlador y servlets

Recibirá el nombre de homeController y será bastante simple pues su única función será obtener del servlet Home, a través del servicio homeService, los datos sobre los cursos, tanto los propios como los publicitarios.

3. AllCourses

Esta vista proporcionará un vistazo a todos las asignaturas que hay disponibles en la aplicación, pudiendo acceder a ellas desde allí. Se incluirá un campo de input de tipo texto, que filtrará los nombres de las materias, mostrando solo aquellas que coinciden con lo allí escrito.

1. Controlador y servlets

Tendrá el nombre de allCoursesController, y, de igual manera que el anterior, solo deberá recibir del servlet AllCourses todos los cursos e incluirlos en el scope para su posterior visualización.

4. View

Es la vista más compleja puesto que contendrá la lógica para mostrar a los usuarios unos datos u otros según su papel en el curso, es decir, los profesores podrán:

- Subir fichero al sistema de ficheros
- Crear test
- Visualizar contenido del curso
- Editar o eliminar el contenido del curso

Por otro lado, los usuarios solo serán capaces de:

- Visualizar contenido del curso

Además, aquellas cuentas que no estén apuntadas al curso deberán también poder hacerlo.

1. Controlador y servlets

Será el viewController y, a través del viewService, generará peticiones a los servlets UploadFile, CreateTest, JoinCourse, DeleteFile, RenameFile, View. Además, aquellos servlets que generan contenido nuevo (UploadFile y CreateTest) manejarán que los alumnos reciban las notificaciones correspondientes.

5. Chat

Es una ruta interna a View, es decir, está relacionada con el curso. Permitirá al alumno mandar y recibir mensajes con otros alumnos o el profesor, así como recibir el historial de los que se han mandado.

1. Controlador y servlets

Será el chatController y utilizará los Servlets SendToChat y ReceiveFromChat, que almacenarán y leerán, respectivamente, de la base de datos MySQL.

6. Admin

Solo será redireccionado a esta vista si al entrar en la aplicación se identifica la cuenta como la de administrador. Entonces, se podrán a llevar a cabo las acciones de crear y borrar cursos, dar permisos de profesor a una cuenta de usuario en un curso y eliminar usuarios.

1. Controlador y Servlet

En realidad, se escribirá una aplicación distinta y totalmente separada que manejará las funciones de Admin, y será el servlet Overview el que decida a cual lleva al usuario. De esta manera, la seguridad aumenta considerablemente. El controlador será teachUsAdminController y los nuevos servlets utilizados serán: BanUser, DeleteCourse, CreateCourse y GrantRights.

4. Desarrollo

Para explicar el desarrollo, se seguirá el esquema anteriormente utilizado en el capítulo 3. Es decir, se explicará el código CSS, controladores, servicios, servlets, tablas MySQL, librerías, etc, que cada una de las vistas utiliza.

Comenzaremos con algunas generalidades del proyecto: el software utilizado para la creación del proyecto será Eclipse Luna, ampliamente conocido. Este entorno de desarrollo nos permite crear automáticamente, seleccionando **Dynamic Web Project**, la estructura que debe tener la aplicación para funcionar bajo el servidor de aplicación Tomcat, el cual se usará en su última versión, la 8.0. El proyecto inicial será:

Figura 4.1 *Estructura del proyecto*

Además, desde la vista Servers en Eclipse Luna es posible crear un nuevo servidor Tomcat 8.0 que funcionará en local y que nos servirá para realizar pruebas de la aplicación. Por otro lado, también deberemos lanzar un servidor MySQL en local, donde almacenaremos la base de datos. Se hace de manera directa siguiendo las instrucciones del primer enlace de la bibliografía.

La conexión se realizará usando un pool de conexiones, es decir, abrimos varias conexiones a la base de datos y las mantenemos abiertas, reutilizándolas para varios usuarios. Esta manera es más eficaz que abrir y cerrar la conexión, no obstante, se debe tener especial cuidado en liberarlas porque si no, la aplicación quedará bloqueada. La manera de crear este pool de conexiones es:

- Crear un archivo context.xml en WebContent > META-INF, que contenga:

```
<Context reloadable="true">
    <Resource name="jdbc/TeachUs"
              auth="Container"
              type="javax.sql.DataSource"
              username=USER
              password=CONTRASEÑA
              driverClassName="com.mysql.jdbc.Driver"
              url="jdbc:mysql://localhost/tfg" //Localización del
servidor MYSQL
              maxActive="8"
              maxIdle="4"/>
</Context>
```

Código 4.1 *context.xml*

- Por otro lado, la clase Manager, que funcionará de intermediario entre los servlets y la base de datos, en su constructor debe realizar:

```
    private Connection connection;

    public Manager() throws SQLException,NamingException {
        connect();
    }
    private void connect() throws SQLException,NamingException {
      Context initCtx = new InitialContext();
      Context envCtx = (Context) initCtx.lookup("java:comp/env");
      DataSource ds = (DataSource) envCtx.lookup("jdbc/TeachUs");
      connection = ds.getConnection();
    }
```

Código 4.2 *Constructor de la clase Manager*

Cuyo funcionamiento es el de cargar el pool de conexiones que hemos definido en context.xml e inicializar el objeto connection que representa la conexión de la base de datos. Es muy importante que después de interactuar con la base MySQL se cierre la conexión mediante el método:

```
        public void close() {
          if (connection != null) {
              try {
                  connection.close();
              } catch (SQLException ex) {
                  ex.printStackTrace();
              }
              connection = null;
          }
      }
```

Código 4.3 *Método close() de la clase Manager*

Además es necesario añadir la librería **mysql-connector-java-5.1.33-bin.jar**, en el lugar indicado en la **Figura 4.1**.

a. Index

Para el archivo index.jsp que funciona como página de bienvenida a la aplicación, hemos utilizado de base el template: **Just-one-page de Codeply**. La manera en que funciona es que define varias páginas apiladas dentro del mismo archivo dentro de las etiquetas html section, y luego navega de una a otra mediante un animación en JavaScript. Por ejemplo

```
<section class="container-fluid" id="section1">
    <div class="v-center">
        <h1 class="text-center">Teach Us!</h1>
        <h2 class="text-center lato animate
slideInDown"><%=label_index_txt1%></h2>
        <p class="text-center">
            <br>
            <a href="#section4" class="btn btn-danger btn-lg btn-
huge lato btn-green" data-toggle="modal" data-
target="#myModal"><%=label_register_ref%></a>
        </p>
        <p class="text-center">
            <br>
            <a href="#" data-toggle="modal" class="white" data-
target="#loginModal"><%=label_index_join%></a>
        </p>
    </div>
    <a href="#section2">
            <div class="scroll-down bounceInDown animated">
            <span>
                <i class="fa fa-angle-down fa-2x"></i>
            </span>
            </div>
        </a>
</section>
```

Código 4.4 *Sección 1 de index.jsp*

Como se puede observar, hay varias etiquetas html a otras secciones, en este caso la sección 4 y 2, que añaden a la URL la cadena: "#section4" o "#section2", cuando este cambio es percibido por el navegador, se ejecuta la función en JavaScript:

```
$('html,body').animate({
          scrollTop: target.offset().top - 50
        }, 800);
```

Código 4.5 *JavaScript de index.jsp*

Cuyo efecto es variar el scroll del navegador hasta dicha sección. Por otro lado, la manera de determinar el idioma de la página se hará utilizando la posibilidad de ejecutar código Java en el archivo index.jsp.

En un primer lugar, comprobaremos si hay un parámetro en la URL que indique un idioma distinto al castellano, esto es debido a que este parámetro se añade siempre que en la página se pulsa el botón para cambiar de lenguaje. Por otro lado, todo el texto a traducir son en realidad objetos String que tomarán un valor u otro según el idioma, y serán estos objetos Java los mostrados como texto en la página. Este sistema tiene la ventaja de agrupar todos los idiomas en el mismo fichero y poder añadir nuevos lenguajes sin necesidad de cambiar nada, y sin tener archivos paralelos con distintos lenguajes, los cuales son más dificultosos de actualizar. No obstante, el archivo aumenta considerablemente en longitud. El código queda de la siguiente manera:

```
<%
String lang = request.getParameter("lang"); //Obtenemos el parámetro
if(lang==null) lang="ES"; //Si no hay, elegimos el castellano
String label_index_* //Aquí definimos todo lo que se va a traducir
;
if(lang.compareTo("ENG")==0){ //Comprobamos el idioma
label_index_* = "Texto en inglés"; //Inglés
}else{
label_index_* = "Texto en español"; //Español
}
%>
```

Código 4.6 *Idioma en index.jsp*

1. Registro

1. Código HTML

El primer paso para desarrollar la función de registro en la aplicación es crear un formulario en HTML. Los datos que se pedirán del usuario son:

- Nombre
- Apellido
- Email
- Contraseña
- País (España o Estados Unidos/Reino Unido)
- Fecha de nacimiento

El email y la contraseña los utilizaremos para la autenticación del usuario en la página. Por otro lado, el país determinará el idioma en el que se mostrará la aplicación, y la fecha de nacimiento será interesante a modo de estudio sobre los posibles intereses que tienen los alumnos. Además añadiremos la etiqueta data-loading-text, con la cual cambiaremos el valor del botón mientras la petición se está procesando. El código en este caso no se mostrará por su simpleza.

2. *Petición AJAX*

No obstante, el formulario creado con anterioridad no será enviado automáticamente tras pulsar el botón de tipo submit, sino que en JavaScript capturaremos este evento y procesaremos la petición asíncrona al servlet para producir el registro.

```
$('#button').on('click', function () {
	$("#register").submit(function() {
	var $btn = $('#button').button('loading')//Cuando se va a
procesar la petición se cambia el valor del botón
	var url = "register";//Url del servlet
	$.ajax({
		type: "POST",
		url: url,
		cache:false,
		data: $("#register").serialize(),
		success: function(data)
		{
			if(data==-2){
			$('#errorModal').modal('show')//Mostramos un modal
si hay error
			}else if(data==-1){
			$('#errorModal').modal('show')
			}else{
			$('#registerModal').modal('show') )//Mostramos
otro modal si todo está correcto
			}
			$btn.button('reset');
		}
	});
	return false; //Así el formulario no se envía dos veces
	});
$btn.unbind('click');
	})
```

Código 4.7 *Petición AJAX para el registro*

3. *Servlet*

La función del servlet, llamado Register, será abrir una conexión con la base de datos y enviar los datos del formulario para su posterior almacenamiento.

Después de realizarse el registro, se utilizarán esos mismos datos para enviar un mail al correo indicado con sus credenciales. Finalmente, se enviará el resultado del método de la clase Manager de vuelta a la función de JavaScript, para valorar si ha ocurrido algún error.

```
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
{
	try{
		Manager m = new Manager();
		short output = m.register(
					request.getParameter("firstname"),
					request.getParameter("lastname"),
					request.getParameter("password"),
					request.getParameter("email"),
					request.getParameter("dob"),
					request.getParameter("country"));
		m.close();
		if(output == 1)
		Email.SendEmail(
			request.getParameter("email"),
			request.getParameter("password")); //Welcome email
		PrintWriter out=response.getWriter();
		out.println(output);
		}catch(Exception e){
			e.printStackTrace();
			return;
	}
}
```

Código 4.8 *Servlet Register*

4. *Clase Email*

Para el envío del correo, vamos a crear la clase Email, que va a utilizar la librería, desarrollada por Oracle, JavaMail. Se enviará desde una cuenta Gmail, creada para la ocasión, y se aprovechará un servidor smtp que proporciona Google.

Necesitamos crear un objeto Transport que inicializaremos gracias a la clase Session, incluida en la librería y que representa la sesión del mail, indicándole el protocolo. Por último, necesitamos los mails de origen y destino, la contraseña del de origen, y el cuerpo y asunto del mensaje. Con esos elementos, se construye el mensaje que será enviado.

```
    public static void SendEmail(String to, String password) throws
AddressException,    MessagingException
    {   final String PASSWORD="*******";
        final String EMAIL = "teachustfg@gmail.com";
        Properties emailProperties = System.getProperties();
        emailProperties.put("mail.smtp.port", "25");
        emailProperties.put("mail.smtp.auth", "true");
        emailProperties.put("mail.smtp.starttls.enable", "true");
        Session mailSession = Session.getInstance(emailProperties,
            new GMailAuthenticator(EMAIL, PASSWORD));
        String toEmails = to;
        String emailSubject = "Bienvenido a TeachUs!";
        String emailBody = "Bienvenido a TeachUs! \n \n"
                  +"\t Tus credenciales son: \n"+
                  "\n \t \t email: "+ to + "\n \t \t contraseña:" +
password;
        MimeMessage emailMessage = new MimeMessage(mailSession);
        emailMessage.addRecipient(Message.RecipientType.TO, new
InternetAddress(toEmails));
        emailMessage.setSubject(emailSubject);
        emailMessage.setContent(emailBody, "text/html");
        String fromUser = EMAIL;
        String fromUserEmailPassword = PASSWORD;
        String emailHost = "smtp.gmail.com";
        Transport transport =
mailSession.getTransport("smtp"); //inicializar el objeto Transport
        transport.connect(emailHost, fromUser,
fromUserEmailPassword);
        transport.sendMessage(emailMessage, //Send email
emailMessage.getAllRecipients());
        transport.close();
        System.out.println("Email sent successfully.");
    }
```

Código 4.9 *Método SendEmail de la clase Email*

5. *Tabla MySQL*

Antes de explicar el método de la clase Manager en el cual se almacena en la base de datos la información concerniente al registro del usuario, se debe crear la tabla en el esquema MySQL. Los atributos que debe tener la tabla son:

- Id: de tipo entero y auto incrementable, para identificar al usuario de manera rápida.
- Nombre: de tipo varchar, con longitud máxima de cincuenta.
- Apellido: de tipo varchar, con longitud máxima de cien.
- Email: de tipo varchar, con longitud máxima de cien.
- Fecha de nacimiento: de tipo date, para almacenar únicamente días, meses y años.

- Hash: de tipo varchar, con longitud de 64. Es el resultado de la función de hash con el algoritmo SHA256 que toma como parámetros de entrada un Salt aleatorio y la contraseña.
- Salt: de tipo varchar, mencionado en el punto anterior.
- Country: de tipo varchar, un pequeño identificador de tres letras del país.

Una vez identificados los atributos necesarios, creamos la tabla:

```
create table  users(
            id int not null auto_increment,
            firstname varchar(50) not null,
            lastname varchar(100) not null,
            email varchar(100),
            date_of_birth date not null,
            hash varchar(64) not null,
            salt varchar(64) not null,
            country varchar(3),
            primary key(id)
)Engine=innodb;
```

Código 4.10 *Sentencia MySQL para crear la tabla de usuarios*

6. *Método en Manager*

Por último, se almacena la información en la tabla creada con anterioridad. No obstante, primero echemos un vistazo a la manera en la cual encriptaremos las contraseñas, de modo que no sea posible acceder a ellas aunque la base de datos reciba un ataque.

Utilizaremos el Secure Hash Algorithm en su versión de 256 bits, esta función de hash la aplicaremos a un salt, 64 bytes generados aleatoriamente, combinado con la contraseña introducida por el usuario. Posteriormente, solo se almacena el resultado de dicha función, que no puede ser revertida, y el salt, a fin de que, en el login, se combine la contraseña y el salt guardado y se vuelva a ejecutar la función sobre ellos para comprobar si coincide con el hash almacenado.

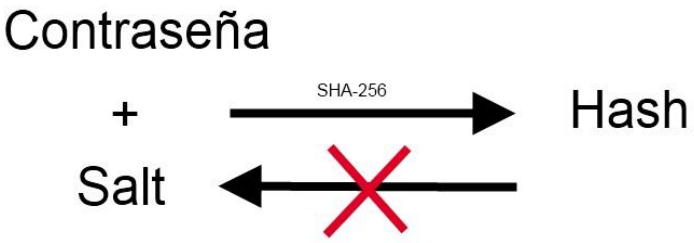


Figura 4.2 *Esquema de seguridad de contraseñas*

```
public short register (String firstname, String lastname, String
pw, String email, String DOB, String country) throws SQLException {

      connection.setTransactionIsolation(Connection.TRANSACTION_REPE
ATABLE_READ);
      connection.setAutoCommit(false);
            String consult = "Select email from users where
email=?";
            PreparedStatement stmtConsult =
connection.prepareStatement(consult);
            stmtConsult.setString(1, email);
            ResultSet rs = stmtConsult.executeQuery();
            if(rs.next()){
               return -2;
            }
            String query = "INSERT INTO Users (firstname, lastname,
email,date_of_birth, hash, salt, country)VALUES (?,?,?,?,?,?,?)";
      PreparedStatement stmt1 = connection.prepareStatement(query);
            stmt1.setString(1, firstname);
            stmt1.setString(2, lastname);
            stmt1.setString(3, email);
            try {
                  SimpleDateFormat dateFormat = new
SimpleDateFormat("dd-MM-yyyy");
                  stmt1.setDate(4, new
java.sql.Date(dateFormat.parse(DOB).getTime()));
                  byte[] salt = genSalt(); //Generamos el salt
                  byte[] hash = code(pw,salt); //Generamos el hash
                  String hashString =
DatatypeConverter.printBase64Binary(hash); // Bytes a String
                  String saltString =
DatatypeConverter.printBase64Binary(salt); // Bytes a String
            stmt1.setString(5, hashString);
            stmt1.setString(6, saltString);
            } catch (ParseException | NullPointerException |
NoSuchAlgorithmException | UnsupportedEncodingException e) {

                        e.printStackTrace();
            }
            stmt1.setString(7, country);
            int num = stmt1.executeUpdate();
                if (num > 0) {
                      connection.commit();
                      connection.setAutoCommit(true);
                      return 1;
                  } else {
                      connection.rollback();
                      connection.setAutoCommit(true);
                      return -1;
                                    }
}
```

Código 4.11 *Método register de la clase Manager*

En primer lugar, y dado que vamos a realizar dos interacciones seguidas con la base de datos, hacemos que el AutoCommit sea falso para asegurar que ambas se realizan

sobre los mismos datos. Seguidamente, se prepara la consulta con PreparedStatement, que aumenta considerablemente la seguridad y nos protege frente a ataques de inyección a la base de datos. Después, indicamos todos los parámetros, generando el salt y el hash, y se vuelve al estado anterior si todo ha sido correcto con AutoCommit a verdadero. El código para generar el hash y el salt es, también, muy simple:

```
    public byte[] genSalt() throws
NullPointerException,NoSuchAlgorithmException{
        SecureRandom r = SecureRandom.getInstance("SHA1PRNG");
        byte[] salt = new byte[16];
        r.nextBytes(salt); //General el salt
        return salt;
    }
   public byte[] code(String password, byte[] salt) throws
NoSuchAlgorithmException,UnsupportedEncodingException{
        MessageDigest dig = MessageDigest.getInstance("SHA-256");
        dig.reset();
        dig.update(salt);
        byte[] hash = dig.digest(password.getBytes("UTF-8"));
        return hash;
    }
```

Código 4.12 *Métodos para generar el salt y el hash de la clase Manager*

2. Login

1. Código HTML

El formulario, que debe rellenar el usuario para entrar a la aplicación, consta de email y contraseña y será escrito como un modal. Por otro lado, no será necesario realizar una petición asíncrona pues se debe redirigir al alumno al interior de la aplicación.

2. Servlet

En el servlet, llamado Signin, que recibirá los datos del formulario de entrada, se deben realizar cuatro funciones:

- Realizar la llamada a la base de datos para comprobar si los datos introducidos coinciden con algún usuario existente.
- Guardar el objeto Java que representa a un usuario, clase bean que contiene como atributos los campos de la tabla MySQL explicada en la **sección 4.a.1.5.**
- Identificar al usuario como una cuenta corriente o de administrador.
- En base a lo anterior, realizar el redireccionamiento correcto.

```
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException
{
	try{
		Manager m = new Manager();
		User u = m.logIn(request.getParameter("email"),
		request.getParameter("password"));
		if(u != null){
			HttpSession session = request.getSession();
			session.setAttribute("user", u);
			if(m.isAdmin(u.getId())==true){ //Es Admin
					m.close();
					response.sendRedirect("admin.jsp");
			}else{ //No es admin
					m.close();
					response.sendRedirect("overview");
			}
		}else{ //Hay error en los datos de entrada
			m.close();
			request.setAttribute("signinError", "true");
			RequestDispatcher rd =
			request.getRequestDispatcher("index.jsp");
			rd.forward(request, response); //redirección
				}
	}catch(Exception e){
		e.printStackTrace();
		return;
		}
```

Código 4.13 *Servlet signin para el login*

3. Tabla MySQL

Por un lado, la manera de utilizar la base de datos será muy parecida al apartado anterior cuando se realizaba el registro, simplemente ahora se requerirán esos datos que fueron introducidos. No obstante, la manera de comprobar si el usuario es administrador será usando una tabla distinta llamada roles.

Esta tabla deberá almacenar qué relación tiene cada usuario con los cursos de la aplicación, y también si tienen el rol de administrador. Los atributos son:

- Id_user: entero que referencia a un id de la tabla de usuarios.
- Role: enum que puede tomar tres valores: ‘student’, ‘teacher’ o ‘admin’.
- Id_course : entero que referencia a un id de la tabla de cursos.

Que crearemos con la siguiente sentencia MySQL:

```
create table roles(id_user int not null, role
enum('student','teacher','admin') not null, id_course int,
constraint foreign key(id_user) references users(id),constraint
foreign key(id_course) references courses(id))Engine=innodb;
```

Código 4.14 *Sentencia MySQL para crear la tabla roles*

El parámetro id_course puede ser null precisamente porque el rol de administrador es general para toda la aplicación y no válido solo para un único curso.

4. Método en Manager

El método isAdmin en la clase Manager se basa en comprobar si el id del usuario tiene asociado un rol de administrador, y solo en ese caso devuelve verdadero.

3. ResetPassword

Esta funcionalidad recoge mucho código de las anteriormente explicadas. En primer lugar, necesita que el usuario introduzca el email de la cuenta de la que ha perdido la contraseña. Tras esto, se genera una petición AJAX, como la de la **sección 4.a.1.2,** y se busca en la base de datos si dicho email existe, y si es así se genera una contraseña, de una manera similar a la que se genera el salt en la **sección 4.a.1.6**. De otro modo, se muestra al usuario un mensaje de error.

Por último, la nueva contraseña se manda al email indicado, reutilizando el método de la **sección 4.a.1.4.**

b. Aplicación

El primer componente de la aplicación es el servlet Overview, al que redirecciona el servlet Login, y cuya única función es trasladar al usuario al archivo overview.jsp.

1. Overview

1. Código HTML

Como se trató en la **sección 3.b.1,** Overview engloba a toda la aplicación y siempre está presente, sin importar la vista en la que se encuentre el alumno. Para entender mejor las partes que compondrán la aplicación, se muestra el código de overview.jsp de manera muy simplificada:

```
<html ng-app="teachUsApp" ng-controller="teachUsController">
      <body>
            <div id="wrapper">
                  <nav class="navbar navbar-inverse navbar-fixed-top"
            role="navigation">
                  //Aquí se definen las barras de navegación, con
                  //funcionalidades como mostrar notificaciones, el
                  //perfil del usuario, ajustes, los cursos en los que
                  //está apuntado, etc
                  </nav>
                  <div id="page-wrapper">
                        <div ng-view>
                        //Esta es la vista, que cargará un
                        //archivo parcial y será manejada por otro
                        //controlador
                        </div>
                  </div>
            </div>
      </body>
</html>
```

Overview

Vista

Código 4.15 *Overview.jsp simplificado*

De una manera más gráfica, con la versión final de **Teach Us!**:

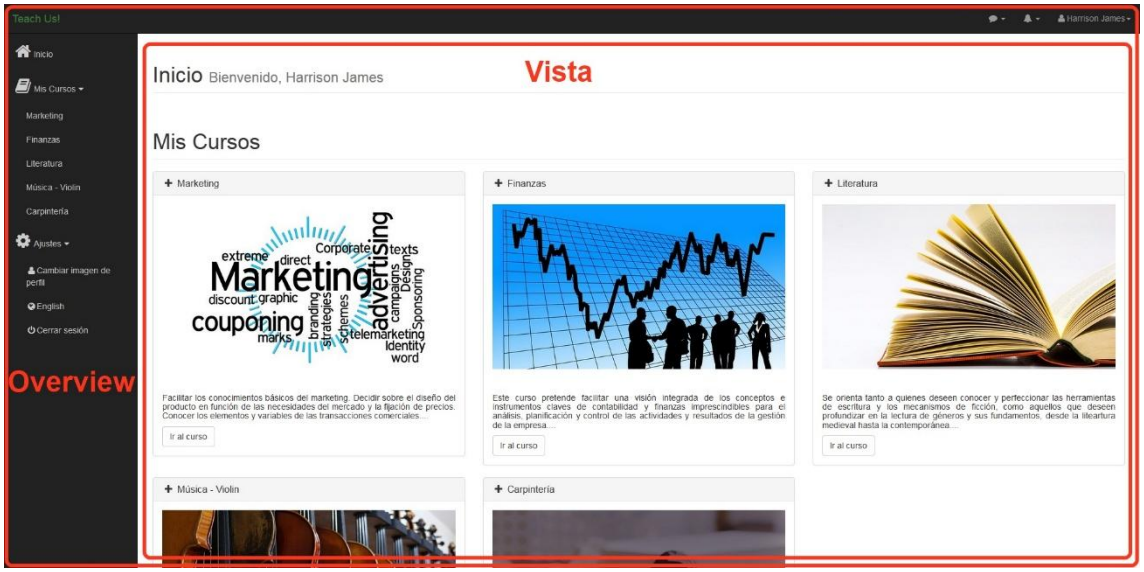


Figura 4.3 *Esquema de overview.jsp*

Todos los elementos que quedan fuera de la vista, por tanto, serán manejados por el controlador TeachUsController. Además, las vistas serán cuatro: Home, AllCourses, View y Chat.

Por otro lado, el idioma que toman los elementos en la aplicación funciona de igual manera que en la **sección 4.a**, con la excepción de que en este caso el objeto String lang toma su valor en base al país seleccionado por el usuario al registrarse.

Las funciones que engloba overview son:

- Recoger las notificaciones, tanto de contenido como de chat.
- Cambiar la foto de perfil.

También se podrá salir de la aplicación o cambiar el idioma, no obstante, se realizarán con peticiones síncronas, que necesitarán la recarga de la página.

2. Sistema de Notificaciones

a. Controlador

En primer lugar, habrá dos tipos de notificaciones: de contenido y de chat. Las primeras ocurren cuando un profesor sube un archivo y las segundas cuando un profesor escribe en un chat, ambas cuando el usuario esté apuntado a esos cursos, evidentemente. El sistema de notificaciones en el controlador consta de dos pasos.

El primero debe ejecutarse en cuanto se entra a la aplicación y tiene como objetivo recoger las notificaciones que se han producido mientras el usuario estaba offline. El segundo debe realizarse cada cierto intervalo de tiempo y trata de obtener notificaciones que puedan producirse mientras el usuario está conectado a la aplicación, además se debe avisar al usuario del nuevo evento con una notificación al estilo Growl. Este segundo paso será ejecutado en la promesa del primero, es decir, cuando se haya recibido la respuesta de la petición asíncrona.

Primero definimos el controlador:

```
teachUsApp.controller('teachUsController', function($scope,
                                          $rootScope, $interval,
                                          teachUsService, growl){
//Aquí definimos funciones
}
```

Código 4.16 *Definición del controlador teachUsController*

Ahora se debe hacer la llamada a teachUsService, que es el que realizará la petición al servlet. Esta función del servicio, que luego se visualizará en profundidad, tiene dos argumentos: las fechas de las últimas notificaciones de contenido y chat, de modo que, en primer momento al enviar "0" se nos permita obtener todas las notificaciones, y posteriormente solo las nuevas.

```
teachUsService.getNotifications("0", "0").then(function(data){
                    if(data !=null){
                        for(var i=0; i<data.length; i++){
                            var data_aux = [data[i].id_course,
data[i].course_name, data[i].sentDate];
                            if(data[i].kind=="content"){

$rootScope.contentNotifications.push(data_aux);
                            }else{

$rootScope.chatNotifications.push(data_aux);
                            }
                        }
                    }
```

Código 4.17 *Primer paso del sistema de notificaciones*

En la promesa, distinguimos entre los dos tipos de notificaciones y los almacenamos en sus respectivos arrays, en este caso en el rootScope, scope global accesible para todos los controladores, para que se pueda interaccionar con ellos desde otro punto de la aplicación.

```
 $scope.refreshNotifications=$interval(function (){
      var contentLength="0", chatLength="0";
      if($rootScope.contentNotifications.length>0)
            contentLength=$rootScope.contentNotifications
                  [$rootScope.contentNotifications.length-1][2];
      if($rootScope.chatNotifications.length>0)
            chatLength=$rootScope.chatNotifications
                  [$rootScope.chatNotifications.length-1][2];
      teachUsService.getNotifications(chatLength,
      contentLength).then(data){
      //Aquí se distingue la notificación y se active el growl.
      growl.warning(msg,{title: title, ttl: 10000});
      }
},30000);
```

Código 4.18 *Segundo paso del sistema de notificaciones*

$interval nos permite llevar a cabo la función cada treinta segundos. Comprobamos cual es la fecha de la última notificación almacenada, y se realiza la llamada al servicio, repitiéndose después lo realizado en el **código 4.17.** Por último, para mostrar la notificación al estilo Growl, utilizaremos la librería **angular-growl-2** desarrollada por **Jan Stevens**, que para su correcta visualización debe añadirse la etiqueta div growl en el page-wrapper del **código 4.16.**

b. Servicio

La función en el servicio teachUsService constará únicamente de una petición

asíncrona de tipo get, con los dos parámetros siendo las últimas fechas de los arrays de notificaciones.

```
teachUsApp.service('teachUsService', function($http){ //Definimos
                                                      //el servicio
 return{
 getNotifications: function(lastSentDateChat, lastSentDateContent){
            return $http.get('getnotifications', {
            params: {
            lastSentDateContent : lastSentDateContent,
            lastSentDateChat : lastSentDateChat
            }}).then(function(result){
                  return result.data; //promesa
            });
```

Código 4.19 *Función getNotifications del servicio teachUsService*

c. Servlet

El servlet que recoge la petición, llamado GetNotifications, tiene como objetivo realizar la llamada a la base de datos estableciendo tres parámetros:

- Id del usuario: Lo obtendrá del objeto Usuario de la sesión guardada.
- Últimas fechas de las notificaciones: Lo obtendrá de la petición HTTP.

Por último, devolverá cualquier notificación, con sus correspondientes datos, nueva al servicio en texto JSON.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
 try {
      User u = (User)request.getSession().getAttribute("user");
      if(u==null){
            response.sendRedirect("index.jsp");
            return;
      }
      Manager m = new Manager();
      JsonArray container = m.getNotifications(u.getId(),
                        request.getParameter("lastSentDateContent"),
                        request.getParameter("lastSentDateChat"));
      m.close();
      response.setContentType("application/json");
      response.getWriter().write(container.toString());
      } catch (SQLException | NamingException | ParseException e) {
      e.printStackTrace();
}}
```

Código 4.20 *Servlet GetNotifications*

d. Tabla MySQL

Las notificaciones serán almacenadas en una nueva tabla dentro de la base de datos MySQL. Se definirán cuatro campos:

- Id_User: El usuario al que se le debe mostrar la notificación. De tipo int.
- Kind: El tipo de notificación que es. Será de tipo enum, con dos valores posibles: "content" o "chat".
- Id_course: El curso al que hace referencia la notificación.
- sentDate: La fecha a la que ha sido enviada la notificación. De tipo timestamp para almacenar incluso los segundos.

```
create table notifications(
      id_user int not null,
      kind enum('content','chat') not null,
      id_course int,
      sentDate datetime not null,
      constraint foreign key(id_user) references users(id),
      constraint foreign key(id_course) references courses(id)
)Engine=innodb;
```

Código 4.21 *Sentencia de creación de la tabla de notificaciones*

e. Método en Manager

El método getNotifications de la clase Mánager primero debe determinar cuál de las dos fechas obtenidas por parámetro es posterior.

```
Timestamp timestampContent= new
                        java.sql.Timestamp(dateContent.getTime());
Timestamp timestampChat = new
                        java.sql.Timestamp(dateChat.getTime());
if(timestampChat.after(timestampContent)) lastSentDate =
lastSentDateChat;
          else  lastSentDate = lastSentDateContent;
}
```

Código 4.22 *Obtención de la fecha posterior*

Tras esto, se realiza la petición a la base de datos y se recogen las notificaciones siguientes a dicho momento, que serán almacenadas en un objeto JsonArray para ser devueltas al servlet.

```
String query = "select notifications.id_course, notifications.kind,
notifications.sentDate, courses.course from notifications "
+ "left join courses on courses.id = notifications.id_course where
id_user=? and notifications.sentDate > ? order by
notifications.sentDate asc;";
      JsonArray notifications = new JsonArray();
      PreparedStatement stmt = connection.prepareStatement(query);
      stmt.setInt(1, idUser);
      stmt.setString(2, lastSentDate);
      ResultSet rs = stmt.executeQuery();
  while(rs.next()){
      JsonObject aux = new JsonObject();
      aux.addProperty("course_name",rs.getString("courses.course"));
      aux.addProperty("id_course",rs.getInt("notifications.id_course
"));
      aux.addProperty("kind", rs.getString("notifications.kind"));
      aux.addProperty("sentDate",
rs.getTimestamp("notifications.sentDate").toString());
      notifications.add(aux);
      }
      return notifications;
}
```

Código 4.23 *Método getNotifications en la clase Mánager*

3. Cambiar foto de perfil

a. Controlador

Simplemente el usuario debe poder cambiar la foto que se mostrará, por ejemplo, en el chat. Para ello, hay que incluir en el campo de input de tipo file donde se subirá el archivo jpg:

```
<input type="file" name="file"
onchange="angular.element(this).scope().uploadedFile(this)"/>
```

Código 4.24 *Evento onchange al cambiar de foto de perfil*

La función uploadedFile tiene como objetivo cargar en el scope el archivo allí indicado, posteriormente, se realiza la función changepicture, que lleva a cabo la llamada al servicio, cuando se pulsa el botón de enviar.

```
$scope.uploadedFile = function(element) {
                        $scope.$apply(function($scope) {
                          $scope.files = element.files;
                        });
};
$scope.changepicture = function(){
      teachUsService.changePicture($scope.files).then(function(){
            $('#changePictureModal').modal('hide');
        });
```

Código 4.25 *Funciones changepicture y uploadedFile*

b. Servicio

Para el caso de ChangePicture, deberemos enviar, como es lógico, la imagen de perfil que el usuario desea mostrar. Por tanto, será una petición de tipo Post.

```
changePicture : function(files){
	var fd = new FormData();
	var url = 'changepicture';
	angular.forEach(files,function(file){
		fd.append('file',file);
	});
	return $http.post(url, fd, {
		withCredentials : false,
		headers : {
			'Content-Type' : undefined
		},
		transformRequest : angular.identity
	}).then(function(){
		return;
	});
},
```

Código 4.26 *Petición changePicture*

En el caso de la petición post, cuando se envían archivos, hay que crear manualmente la petición y establecer el Content-Type a undefined. Lo lógico sería fijarla como multipart/form-data, no obstante se trata de un error de AngularJS.

c. Servlet

El archivo enviado se debe tratar correctamente y guardarlo en el directorio correcto en el que están todas las fotos de perfil. El nombre que recibirá será el id del usuario y si ya existe una foto de perfil debe ser borrada.

```
protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
	User u = (User) request.getSession().getAttribute("user");
	if(u==null){
		response.sendRedirect("index.jsp");
		return;
	}
	Part filePart = request.getPart("file");
	String fileName = u.getId() + ".jpg";
	String relativeDir = PROFILEPICS_DIRECTORY + File.separator;
  File f = new File(MULTIPARTCONFIG_LOCATION+relativeDir+fileName);
	if(f.isFile()) f.delete();
     filePart.write(MULTIPARTCONFIG_LOCATION+relativeDir+fileName);
	}
}
```

Código 4.27 *Servlet ChangePicture*

4. Cerrar sesión

Se debe llevar a cabo la invalidación del objeto HttpSession guardado y redireccionar al usuario a la página index. El servlet será llamado a través de un elemento a en HTML, que tenga en el campo href el valor del servlet. La petición será, por tanto, síncrona.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
      try{
            HttpSession sesion = request.getSession();
            sesion.invalidate();
            response.sendRedirect("index.jsp");
      }catch(Exception e){
            e.printStackTrace();
            return;
      }}
```

Código 4.28 *Servlet SignOut*

5. Cambiar idioma

Con el objeto Usuario guardado en la sesión, hacemos una llamada a la base de datos indicando el lenguaje actual, de modo que establezca el idioma alternativo, en este caso el inglés. Después, se actualiza el usuario en la sesión.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
 try {
  User u = (User) request.getSession().getAttribute("user");
  if(u==null){
      response.sendRedirect("index.jsp");
      return;
  }
  Manager m = new Manager();
  m.changeLanguage(u.getId(),u.getCountry());
  HttpSession session = request.getSession();
  session.setAttribute("user", m.getUser(u.getId()));
  m.close();
  respse.sendRedirect("overview");
 } catch (SQLException | NamingException e) {
                  e.printStackTrace();
            }
```

Código 4.29 *Servlet ChangeLanguage*

2. Home

Es la vista que se carga por defecto, y la que dará la bienvenida al usuario a la aplicación. Tiene como objetivo mostrar un resumen de la actividad de la cuenta del alumno.

Es decir, se mostrarán los cursos a los que está apuntado el usuario, un aviso si hay notificaciones pendientes y algunos cursos que están disponibles, a modo de publicidad.

El primer paso consiste en definir la ruta y su correspondiente controlador.

```
teachUsApp.config(function($routeProvider){
            $routeProvider
                  .when('/home',
                        {
                              controller: 'homeController',
                              templateUrl: 'partials/home.jsp'
                        })
                  .otherwise({ redirectTo:'/home'});
})
```

Código 4.30 *Definición de la ruta /home*

1. Notificaciones pendientes

Debido a que las notificaciones están almacenadas en el rootScope, estas son también accesibles desde el model de Home. Por tanto, basta con mostrar el aviso si la longitud de alguno de los dos arrays de notificaciones es mayor que cero.

```
<div ng-if="$root.chatNotifications.length>0"
      class="alert alert-info alert-dismissible" role="alert">
      //Aquí definimos el texto a mostrar en el aviso
</div>

<div ng-if="$root.contentNotifications.length>0"
      class="alert alert-warning alert-dismissible" role="alert">
      //Aquí definimos el texto a mostrar en el aviso
</div>
```

Código 4.31 *Ng-if para notificaciones pendientes*

2. Cursos

Los cursos, tanto a los que está apuntado el usuario como los que son de publicidad, van a ser obtenidos del mismo servlet.

a. Controlador

Estos cursos deben ser cargados inmediatamente después de entrar en la vista, por tanto, la llamada al homeService no estará dentro de una función que espera a ser llamada, sino que se ejecutará siempre al inicio.

```
teachUsApp.controller('homeController', function($scope,
$rootScope, homeService){
	homeService.getCourses().then(function(courses){
		$rootScope.idUser = courses.idUser;
		$rootScope.myCourses = courses.myCourses;
		$scope.courses=courses;
	});
})
```

Código 4.32 *Controlador homeController*

Es decir, se realiza la llamada a la función getCourses de homeService, y cuando obtenemos la respuesta, almacenamos los cursos a los que está apuntado y el id del usuario en el rootScope. Esto es debido a que en overview son utilizados, aunque solo con fines estéticos.

b. Servicio

En este caso, la petición get al servlet Home es bastante sencilla

```
teachUsApp.service('homeService', function($http){
	return{
		getCourses: function(){
		return $http.get('home')
			.then(function(result){
				return result.data;
			});
		}}
})
```

Código 4.33 *Servicio homeService*

c. Servlet

Este servlet, llamado Home, deberá comprobar primero que el usuario ha realizado correctamente la autenticación. Tras esto, realiza dos llamadas a la base de datos para obtener, por un lado, los cursos a los que está apuntado, y por otro, los de publicidad. Finalmente, puesto que toda la información debe ser transmitida en JSON, se crea un JsonObject que contiene los dos arrays de cursos y el id de usuario.

```
	protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
	try{
		Manager m = new Manager();
		User u = (User)
			request.getSession().getAttribute("user");
		if(u==null){
			response.sendRedirect("index.jsp");
			m.close();
			return;
		}
		List<Course> myCourses =  m.getMyCourses(u.getId());
		List<Course> adsCourses =
				m.getAdsCourses(u.getCountry(), myCourses);
		m.close();
		String coursesjson =
			serializationCoursesAndId(myCourses,adsCourses,
								u.getId());
		response.setContentType("application/json");
		response.getWriter().write(coursesjson);
	}catch(Exception e){
	e.printStackTrace();
	return;
	}
}
```

Código 4.34 *Servlet Home*

d. Tabla MySQL

Toda la información relativa a los cursos, será guardada en una nueva tabla MySQL, con los siguientes campos:

- Id: de tipo int y auto_increment.
- Course: Varchar de cien caracteres, almacenará el nombre
- Description: Varchar de quinientos caracteres, supone una pequeña descripción del contenido del curso.
- Language: Varchar de dos caracteres, se refiere al idioma del curso.
- Deadline: de tipo date, es la fecha en la cual da comienzo el curso y hasta la que se pueden apuntar los alumnos.

```
create table courses(
	id int not null auto_increment,
	course varchar(100) not null,
	description varchar(500),
	language varchar(2),
	deadline date, primary key(id)
)Engine=innodb;
```

Código 4.35 *Sentencia MySQL para crear la tabla courses*

e. Método en Manager

En el **código 4.34,** hemos usado dos funciones distintas de la clase Manager, no obstante, son muy parecidas.

```
public List<Course> getMyCourses(int id) throws SQLException{
      String query = "select courses.id, courses.course,
                  courses.description, "+ "courses.language,
                  courses.deadline from courses left join roles "
                  + "on courses.id = roles.id_course where
                  id_user=?;";
      PreparedStatement stmt = connection.prepareStatement(query);
      stmt.setInt(1, id);
      ResultSet rs = stmt.executeQuery();
      List<Course> myCourses = new ArrayList<Course>();
      while(rs.next()){
            Course cr = new Course();
            cr.setId(rs.getInt("courses.id"));
            cr.setName(rs.getString("courses.course"));
            cr.setDescription(rs.getString("courses.description"));
            cr.setLanguage(rs.getString("courses.language"));
            cr.setDeadline(rs.getString("courses.deadline"));
            myCourses.add(cr);
      }
      stmt.close();
      return myCourses;
}
```

Código 4.36 *Método getMyCourses de la clase Manager*

En primer lugar, se crea la petición que solo obtendrá los cursos en los que en la tabla roles (ver **sección 4.a.2.3**) haya una entrada que le relacione con el usuario. A partir de esto, se crea la lista que será devuelta y se rellena con cada uno de los cursos obtenidos. Lo único que cambia en getAdsCourses es que se obtienen todos los cursos del idioma indicado y se cruzan con los cursos almacenados en la lista myCourses, aquellos que coinciden se eliminan.

3. AllCourses

En esta vista, el usuario podrá acceder y tener una vista general de todos los cursos disponibles en la aplicación. Además, se aplicará un buscador que filtre los nombres. La ruta será /allcourses y el controlador allCoursesController.

1. Código HTML

En primer lugar, se configura el campo de entrada de tipo texto que funcionará como filtro y la manera de mostrar todos los cursos.

```
<input type="text" class="form-control" ng-model="nameCourse.name"
                    placeholder='<%= label_search_placeholder %>'/>

<div class="col-lg-3" ng-repeat="course in allcourses |
                filter:nameCourse:strict | orderBy:'course.name' ">
//Aquí se acceden a los atributos de course mediante
//{{course.atributo}}
</div>
```

Código 4.37 *Código HTML de allcourses.jsp*

Es decir, el atributo de la variable almacenada en el scope, y que se establece en el campo de entrada, debe coincidir con el atributo de la variable auxiliar que recorre el array allcourses para que se filtre de manera correcta.

2. Controlador

Como el caso anterior, también nos encontramos con un controlador muy sencillo, con una única función: obtener todos los cursos del servlet AllCourses, para ello utiliza el allCoursesService.

```
teachUsApp.controller('allCoursesController', function($scope,
                                    $rootScope, allCoursesService){
      allCoursesService.getAllCourses().then(function(allcourses){
            $scope.allcourses=allcourses;
      });
})
```

Código 4.38 *Controlador AllCourses*

Por tanto, se realiza la llamada a getAllCourses del servicio, y cuando se recibe la respuesta se dispone a almacenar dicho JSON en el scope.

3. Servicio

En el servicio, allCoursesService, se realiza una petición de método get al servlet AllCourses.

```
teachUsApp.service('allCoursesService', function($http){
      return{
            getAllCourses: function(){
                  return $http.get('allcourses')
                              .then(function(result){
                                    return result.data;
                                    });
                  }
      }
})
```

Código 4.39 *Servicio AllCoursesService*

4. Servlet

El servlet, AllCourses, funciona de manera análoga al servlet Home, **sección 4.b.2.2.c,** es decir, en primer lugar, se realiza la autenticación del usuario, obteniendo el objeto guardado en la sesión. Tras esto, se abre la conexión con la base de datos y se realiza la petición. Finalmente, dicha conexión se cierra y el objeto Java que representa una lista de cursos es pasado a lenguaje JSON.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
	try{
		User u = (User)
				request.getSession().getAttribute("user");

		if(u==null){
			response.sendRedirect("index.jsp");
			return;
		}
		Manager m = new Manager();
		List<Course> allCourses= (List<Course>)
					m.getAllCoursesByLang(u.getCountry());
		m.close();
		String allCoursesjson = new Gson().toJson(allCourses);
		response.setContentType("application/json");
		response.getWriter().write(allCoursesjson);
	}catch(Exception e){
		e.printStackTrace();
		return;
	}
}
```

Código 4.40 *Servlet AllCourses*

5. Método en Manager

El método getAllCoursesByLang pretende obtener todos los cursos de la base de datos ordenados por idioma, mostrando en primer lugar los que coinciden con el del usuario.

```
public List<Course> getAllCoursesByLang(String language) throws
SQLException{
	String query = "select * from courses order by language=?
							desc,deadline asc;";
	PreparedStatement stmt = connection.prepareStatement(query);
	stmt.setString(1, language);
	ResultSet rs = stmt.executeQuery();
	List<Course> allCourses = new ArrayList<Course>();
	while(rs.next()){
		//Almacenamos los cursos en allCourses(ver código 4.36)
	 }stmt.close();return allCourses;
```

Código 4.41 *Método getAllCoursesByLang en Manager*

4. View

Esta es la vista más importante de la aplicación, pues es donde el usuario podrá obtener el conocimiento de las asignaturas, que es el objetivo de la aplicación. Esta vista se mostrará de manera diferente a profesores, alumnos inscritos en el curso y no inscritos. La manera de diferenciarlos será a través de la tabla roles (ver **sección 4.a.2.3**), que indica el permiso de la cuenta en dicho curso.

Por otro lado, la ruta de view será /view, y el curso que queremos cargar es indicado en el scope global, $rootScope.courseId. Esta variable toma el valor del identificador del curso cuando se clickea en ver curso, y posteriormente, en la vista view se obtiene toda la información de la base de datos sobre el curso a partir del id. Otra solución posible hubiese sido hacer que la ruta fuese /view/:courseId, no obstante, al estar trabajando con dos controladores simultáneamente, la comunicación entre ambos se hacía algo más compleja que el hecho de utilizar una variable global.

1. Apuntarse al curso

La primera acción que podrá realizar un alumno en un curso al que no está apuntado será inscribirse. Para ello, el curso no debe haber comenzado, es decir, que la fecha límite aún no se haya cumplido.

a. Controlador

```
$scope.joinCourse = function(){
       viewService.joinCourse($rootScope.courseId).then(function(){
            viewService.getCourse($rootScope.courseId).then(
                  function(course){
                        $scope.course=course;
                        $rootScope.courseName=course.name;
                  });
      });
};
```

Código 4.42 *Función joinCourse del controlador viewController*

Esta función, que es ejecutada tras pulsar el botón de unirse al curso, en primer lugar, realiza una llamada al servicio viewService para comprobar si es posible que el usuario, que está identificado en la sesión, se una al curso, que está identificado en $rootScope.courseId. Tras esto, vuelve a recargar el curso para mostrar las opciones de alumno.

b. Servicio

```
joinCourse: function(courseId){
    return $http.get('joincourse', {
      params: {
            idCourse : courseId
               }
      }).then(function(){
                  return;
      });
},
getCourse: function(courseId){
    return $http.get('view', {
      params: {
            id: courseId
              }
      }).then(function(result){
            return result.data;
       });
}
```

Código 4.43 *Funciones joinCourse y getCourse del servicio viewService*

Ambas funciones son simples peticiones get. La diferencia de la función getCourse respecto a otras que hemos visto anteriormente y que también obtenían información sobre cursos, es que esta recoge los datos sobre el contenido del curso.

c. Servlet

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
 try{
      Manager m = new Manager();
      int idCourse = Integer.parseInt(request.getParameter("id"));
      Course cr =  m.getCourse(idCourse);
      User u = (User) request.getSession().getAttribute("user");
      m.deleteNotifications(u.getId(), "content" ,cr.getId());
      short output = 0;
      if(u!=null) output = m.getRole(u.getId(), cr.getId());
      String Coursejson = serializationCourseAndRole(cr,output,
                                          m.getTeacher(cr.getId()));
      m.close();
      response.setContentType("application/json");
      response.getWriter().write(Coursejson);
}catch(Exception e){
      e.printStackTrace();
      return;
}}
```

Código 4.44 *Servlet View*

En primer lugar, a partir del parámetro idCourse en la petición se obtiene la información del curso de la base de datos. Después, identificamos al usuario en la sesión, y borramos las notificaciones de contenido que tenga dicho usuario en ese curso, puesto que lo está visualizando. Por último, identificamos el rol que tiene el usuario en ese curso y el profesor para mostrarlo. Finalmente, preparamos toda esa información en un objeto JSON para devolverlo al servicio.

En este objeto JSON debe también estar presente todos los archivos que constituyen el contenido de la asignatura. La ruta a dichos archivos la obtenemos mediantes la funciones getFiles y getTest.

```
public JsonArray getFiles(int id)throws SQLException,NamingException{
 JsonArray files = new JsonArray();
 File f = new File(COURSES_DIRECTORY+id);
 for(int i=1; i<13; i++){
  File f_aux= new File(f.getPath()+File.separator+String.valueOf(i));
  if(f_aux.isDirectory() && f_aux.list().length > 0){
      JsonObject week = new JsonObject();
      week.addProperty("week", String.valueOf(i));
      JsonArray files_aux = new JsonArray(); //Array de archivos para
                                                una semana
      for (File fileEntry : f_aux.listFiles()){
            JsonObject file_aux = new JsonObject();
            file_aux.addProperty("file", fileEntry.getName());
            files_aux.add(file_aux);
      }
      week.add("test", this.getTest(id,i));
      week.add("files", files_aux);
      files.add(week);
  }}
  return files;
}
public JsonElement getTest(int idCourse, int week) throws
SQLException, NamingException {
      Manager m = new Manager();
      List<Question> test =  m.getTest(idCourse,week);
      m.close();
      return new Gson().toJsonTree(test); //Objeto a JSON
}
```

Código 4.45 *Funciones getFiles y getTest del servlet View*

En getFiles, la constante String COURSES_DIRECTORY almacena la ruta absoluta al directorio donde se van a guardar los archivos. En este directorio, se creará una carpeta por cada curso, que recibirá como nombre el identificador. Por último en la carpeta del curso, se creará una más con el número de la semana (doce máximo). Por

tanto, la función bucea en todo el directorio buscando el contenido del curso y pasándolo a un JsonArray que luego devuelve. Por otro lado, el test se obtiene directamente de la clase Manager, pudiendo existir uno por semana.

Del servlet JoinCourse, lo único reseñable es la manera de obtener las fechas en Java y compararlas, de modo que se pueda conocer si la fecha límite se ha cumplido.

```
java.util.Date aux =Calendar.getInstance().getTime();
Date now =  new java.sql.Date(aux.getTime());
Course cr =m.getCourse(idCourse);
if(Date.valueOf(cr.getDeadline().substring(0, 10)).after(now)){
      //La fecha límite no se ha cumplido
      m.joinCourse(idCourse,u.getId());
}
```

Código 4.46 *Comprobación de fecha límite*

En primer lugar, hay que diferenciar entre las dos clases útil.Date y sql.Date. La segunda nos permite comparar fechas con los métodos before y after. Por tanto, hay que obtener la fecha límite del objeto Course y compararla con now. Aplicarle substring(0,10) se debe a que se añadían caracteres al final que impedían usar el método valueOf para pasar de String a Date.

d. Tabla MySQL

Los tests serán almacenados en la base de datos, con los siguientes campos:

- Id_course: relacionado con el identificador del curso que corresponda.
- Week: de tipo int, indica la semana a la que pertenece el test.
- Id_question: de tipo int, es un número que señala el número de pregunta.
- Q: varchar de quinientos caracteres, almacena la pregunta del test.
- Aa: varchar de trescientos caracteres, almacena la respuesta A a la pregunta.
- Ab: varchar de trescientos caracteres, almacena la respuesta B a la pregunta.
- Ac: varchar de trescientos caracteres, almacena la respuesta C a la pregunta.
- Ad: varchar de trescientos caracteres, almacena la respuesta D a la pregunta.

- Right_answer: de tipo enum, pudiendo tomar los valores “a”, “b”, “c”, “d”. Indica la respuesta correcta a la pregunta.
- Correction: varchar de trescientos caracteres, es un texto corto que se mostrará a modo de corrección si el usuario falla.

```
create table tests(
      id_course int not null,
      week int not null,
      id_question int not null,
      q varchar(500) not null,
      aa varchar(300) not null,
      ab varchar(300) not null,
      ac varchar(300) not null,
      ad varchar(300) not null,
      right_answer enum("a","b","c","d") not null,
      correction varchar(300),
      constraint foreign key(id_course) references courses(id)
)Engine=innodb;
```

Código 4.47 *Sentencia MySQL para crear la tabla tests*

e. Método en Manager

Ha habido varias funciones de la clase Manager que han aparecido en la **sección 4.b.4.1.c.**

- getRole: Es una simple sentencia select a la tabla roles, para obtener el valor del campo role, devolviendo uno si el usuario no está apuntado al curso, dos si el usuario está inscrito como alumno o tres si se trata del profesor.
- getTeacher: Junta las tablas roles y users mediante la sentencia left join, de esta manera se puede conocer que cuenta está inscrita como profesor en el curso, y además obtener el nombre del profesor, almacenado en la tabla users.

```
select users.id, users.firstname, users.lastname from users left
join roles on roles.id_user = users.id where roles.id_course = ?
and roles.role = 'teacher';
```

Código 4.48 *Sentencia MySQL de la función getTeacher*

- getTest: Se creará una clase bean Question, que tenga como atributos los campos discutidos en el apartado anterior. Cada pregunta se almacena en uno de estos objetos, que finalmente formarán una lista, de la misma manera que el **código 4.36**.

- getCourse: Es una simple sentencia select a la tabla courses, todo la información de la respuesta se almacena en una variable de la clase bean Course que es devuelta.

2. Subida de archivos

Esta funcionalidad solo estará disponible para los profesores, y el desarrollo de esta es básicamente el mismo que el de cambio de foto de perfil (**sección 4.b.3**). Con la diferencia de que, además, el profesor debe indicar la semana a la que sube el contenido, el nombre y la extensión del fichero. El código de la función en el controlador y en el servicio no se muestra por dicha similitud.

En el servlet, UploadFile, se debe incluir una función que cree una notificación a todas las cuentas inscritas en dicho curso, será la función sendNotifications(String kind, int idCourse).

```
public void sendNotifications(String kind, int idCourse) throws
SQLException{
 List<Integer> listIdUsers = this.getAllUsersFromCourse(idCourse);
 String query = "REPLACE INTO notifications SET id_user = ?, kind
                              =?, id_course = ?, sentDate = now();";
 PreparedStatement stmt = connection.prepareStatement(query);
 stmt.setString(2, kind);
 stmt.setInt(3, idCourse);
 for(int i = 0; i< listIdUsers.size(); i++){
      stmt.setInt(1, listIdUsers.get(i));
      stmt.executeUpdate();
 }
 stmt.close();
}

public List<Integer>  getAllUsersFromCourse(int idCourse) throws
SQLException{
 String query = "select roles.id_user from roles where id_course =
                                    ? and roles.role != 'teacher'";
 PreparedStatement stmt = connection.prepareStatement(query);
 stmt.setInt(1, idCourse);
 ResultSet rs = stmt.executeQuery();
 List<Integer> listIdUsers =  new ArrayList<Integer>();
 while(rs.next()){
      Integer aux = rs.getInt("roles.id_user");
      listIdUsers.add(aux);
 }
 stmt.close();
 return listIdUsers;
}
```

Código 4.49 *Función sendNotifications en Manager*

En primer lugar, se deben obtener todos los ids de los usuarios a los que se enviará la notificación. Esta acción la realiza la función getAllUsersFromCourse. Tras esto, se utiliza la sentencia Replace de MySQL para que, solo cree la notificación en el caso de que no exista ya una del mismo tipo.

3. *Visualización de contenido*

Los archivos subidos del apartado anterior son después encontrados por el servlet View (sección **4.b.4.1.c**). No obstante, como hemos visto, dicho servlet devuelve en un objeto JSON el nombre del fichero y la semana a la que pertenece. El fichero en sí será devuelto por el servlet Uploads.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
      String type = request.getParameter("type");
      String filename = request.getPathInfo().substring(1);
      String RelativeDir;
      if(type.compareTo("file")==0){ //Está intentando acceder a un
                  //archivo de tipo file (del contenido del curso)
        User u = (User) request.getSession().getAttribute("user");
        if(u!=null){ //Si el usuario existe en la sesión se crea el
                        RelativeDir del tipo file
            String idCourse = request.getParameter("idCourse");
            String week =  request.getParameter("week");
            RelativeDir = COURSES_DIRECTORY + File.separator +
             idCourse  + File.separator + week + File.separator;
            }else{ //Está intentando acceder a un archivo sin
                  //Estar guardado en la sesión
                  response.sendRedirect("../index.jsp");
                  return;
            }
      } else if(type.compareTo("profilePic")==0){ //Tipo foto de
                                                  //perfil
            RelativeDir = PROFILEPICS_DIRECTORY + File.separator;
      } else if(type.compareTo("pic")==0){ //Tipo foto de
                                          //introducción del curso
            String idCourse = request.getParameter("idCourse");
            RelativeDir = COURSES_DIRECTORY + File.separator +
                                                  idCourse;
       }else{ //El tipo no coincide con ninguno
            response.sendRedirect("../index.jsp");
            return;
      }
```

Código 4.50 *Servlet Uploads (Parte I)*

En primer lugar, se debe identificar qué tipo de archivo se está tratando de mostrar al usuario, pues hay de tres tipos:

- File: Los archivos subidos por el profesor.
- profilePic: Fotos de perfil.
- Pic: Fotos de introducción.

Esto es importante pues cada uno de estos tipos está en una ruta distinta del sistema de ficheros (ver **Figura 3.3**). Una vez se ha dilucidado este hecho, hay que crear la ruta relativa a dicho tipo de contenido, el objeto String RelativeDir.

```
if(RelativeDir.contains("{{") || filename.contains("{{")){
      return;
}else{
      File file = new File(UPLOADS_DIRECTORY+RelativeDir,
                                                        filename);
      if(type.compareTo("profilePic")==0){
            if(file.isFile()){
              response.setHeader("Content-Type",
                        getServletContext().getMimeType(filename));
              response.setHeader("Content-Length",
                                    String.valueOf(file.length()));
              response.setHeader("Content-Disposition", "inline;
                              filename=\"" + filename + "\"");
              Files.copy(file.toPath(),
                                    response.getOutputStream());
            }else{
                  filename = "avatar.jpg";
                  File avatarFile = new
                      File(UPLOADS_DIRECTORY+RelativeDir, filename);
                  response.setHeader("Content-Type",
                        getServletContext().getMimeType(filename));
              response.setHeader("Content-Length", String.valueOf(
                                              avatarFile.length()));
              response.setHeader("Content-Disposition", "inline;
                                    filename=\"" + filename + "\"");
              Files.copy(avatarFile.toPath(),
                                       response.getOutputStream());
            }
        }else{
            if(file.isFile()){
              response.setHeader("Content-Type",
                        getServletContext().getMimeType(filename));
              response.setHeader("Content-Length",
                                    String.valueOf(file.length()));
              response.setHeader("Content-Disposition", "inline;
                                    filename=\"" + filename + "\"");
              Files.copy(file.toPath(),
                                       response.getOutputStream());
            }
        }
```

Código 4.51 *Servlet Uploads (Parte II)*

Lo que realiza la segunda parte del código es establecer las cabeceras de la respuesta HTTP y copiar el archivo a su OutputStream, de manera que el archivo es transferido al usuario. No obstante, en el caso de ser de tipo profilePic, es posible que el usuario no haya subido ninguna foto y por tanto deba ser cargada avatar.jpg. Por último, la primera condición (RelativeDir.contains('{{')) es debida a que algunos parámetros se toman de AngularJS, y resultaba que se realizaban peticiones a este servlet antes de que AngularJS cambie el scope por el valor real, resultando en excepciones al no encontrar dicho archivo.

Ejemplo de utilización del servlet Uploads para descargar un archivo del sistema de ficheros: /uploads/nombre.extension?type=file&idCourse=1&week=1.

4. Modificación y eliminación de archivos

Estas funcionalidades son muy parecidas a la de subida y visualización de archivos. La manera de encontrar dichos archivos es la misma, simplemente se deben ejecutar dos funciones distintas de la clase File.

```
File oldF = new File(MULTIPARTCONFIG_LOCATION+relativeDir+new
String(request.getParameter("oldname").getBytes("iso-8859-1"),
"UTF-8"));
if(oldF.exists()){
 File newF = new File(MULTIPARTCONFIG_LOCATION+relativeDir+new
 String(request.getParameter("newname").getBytes("iso-8859-1"),
 "UTF-8")+"."+request.getParameter("extension"));
 oldF.renameTo(newF);
}

File f = new File(MULTIPARTCONFIG_LOCATION+relativeDir+fileName);
f.delete();
```

Código 4.52 *Modificación y eliminación de archivos*

En este caso, MULTIPARTCONFIG_LOCATION fija la ruta raíz del sistema de ficheros. Por otro lado, la función getBytes pasa la codificación a UTF-8 para la correcta visualización de acentos, etc.

5. Creación y visualización de tests

La aplicación también permite a los profesores subir exámenes de autoevaluación.

El profesor debe, en primer lugar, crear el examen tipo test. Consiste en unos campos de entrada en html que se enlazan con el scope de AngularJS mediante la etiqueta ng-model, haciendo uso de la funcionalidad 2-way data binding.

```
input ng-model="question.q" class="form-control" type="text"
            placeholder='<%= label_view_test_question%>'/> </h5>
<form action="">
      <input type="radio" name ="answer" ng-model=
"question.rightAnswer" value ="a"><input ng-model="question.aa"
class="form-control" type="text" placeholder='<%=
label_view_test_answer%>'/><br>
      <input type="radio" name ="answer" ng-model=
"question.rightAnswer" value ="b"><input ng-model="question.ab"
class="form-control" type="text" placeholder='<%=
label_view_test_answer%>'/><br>
      <input type="radio" name ="answer" ng-model=
"question.rightAnswer" value ="c"><input ng-model="question.ac"
class="form-control" type="text" placeholder='<%=
label_view_test_answer%>'/><br>
      <input type="radio" name ="answer" ng-model=
"question.rightAnswer" value ="d"><input ng-model="question.ad"
class="form-control" type="text" placeholder='<%=
label_view_test_answer%>'/>
</form>
<input ng-model="question.correction" class="form-control"
type="text" placeholder='<%= label_view_test_correction%>'/>
```

Código 4.53 *Código HTML del formulario para crear test*

Al servlet CreateTest se enviará un objeto JSON en la petición get que será un array de varias cuestiones, que pasarán a ser almacenadas en la base de datos.

```
if(m.getRole(u.getId(),Integer.valueOf(request.getParameter("idCour
se")))==3){
      String newTest = new
      String(request.getParameter("newTest").getBytes(
                                          "iso-8859-1"), "UTF-8");
      JsonArray newTestjson = new
                  JsonParser().parse(newTest).getAsJsonArray();
      m.createTest(newTestjson,
                  Integer.valueOf(request.getParameter("idCourse")),
                  Integer.valueOf(request.getParameter("week")));
      m.close();
}
```

Código 4.54 *Servlet CreateTest*

Como se puede ver, se debe utilizar la librería JsonParser para pasar del parámetro newTest, que viaja en la petición get y que es recibida como un objeto String, a un objeto JsonArray que será manejado en la función en manager.

La función en manager, createTest, simplemente introduce una fila en la base de datos por cada pregunta del test (ver **sección 4.b.4.1.d**), de manera similar a la ya explicada en numerosas ocasiones.

5. Chat

1. *Recibir mensajes*

La función de recibir mensajes en el chat tiene mucho que ver con la de recibir las notificaciones en overview (ver **sección 4.b.1.2**), puesto que las dos realizan peticiones al servlet en intervalos de tiempo mandando la fecha de lo último recibido. En este caso es incluso más sencillo porque solo hay un array, a diferencia del sistema de notificaciones que había dos distintos.

a. Controlador

En el controlador, en primer lugar, se realiza una petición que va recoger al menos los diez últimos mensajes enviados al chat. Y será en la promesa de esta petición donde se escriba el código para realizar la petición en intervalos, usando $interval.

```
chatService.getMessages($rootScope.courseId,"0").then(
 function(datamsgs){
 $rootScope.remove($rootScope.chatNotifications,$rootScope.courseId);
 if(datamsgs!=null) $scope.messages=datamsgs;
 $scope.refreshMessages=$interval(function () {
      //Este código se ejecuta cada 500 ms
      if($scope.messages!=null){
      chatService.getMessages($rootScope.courseId,
$scope.messages[$scope.messages.length-1].date).then(function(data2){
            if(data2 !=null){
                  var msg = {
                  //Aquí se crea el mensaje a partir de data2
                  };
                  $scope.messages.push(msg);
                  if($scope.messages.length>10)
                        $scope.messages.shift(); //Eliminamos el
                                                 //primer mensaje del array
            }
      })
      }else{
      chatService.getMessages($rootScope.courseId,"0").then(
      function(data2){
            $scope.messages=data2;
      })}},500);
```

Código 4.55 *Función getMessages del controlador chatController*

b. Servicio

El servicio, chatService, es simplemente una petición get al servlet ReceiveFromChat, con dos parámetros: courseId, almacenado en el rootscope, y lastSentDate, que es la fecha de creación del último mensaje almacenado, y por tanto, a partir de la cual debemos obtenerlos.

c. Servlet

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
 try{
      Manager m = new Manager();
      User u = (User) request.getSession().getAttribute("user");
      if(u==null){
            response.sendRedirect("index.jsp");
            m.close();
            return;
      }
      String lastSentDate = request.getParameter("lastSentDate");
      int idCourse =
                  Integer.parseInt(request.getParameter("idCourse"));
      List<Message> messages = m.getMessages(idCourse, lastSentDate);
      m.deleteNotifications(u.getId(), "chat", idCourse);
      m.close();
      response.setContentType("application/json");
      if(messages!=null) response.getWriter().write(
                                    new Gson().toJson(messages));
 }catch(Exception e){
      e.printStackTrace();
      return;
}}
```

Código 4.56 *Servlet ReceiveFromChat*

En el servlet, primero se obtienen los datos que viajan en los parámetros de la petición. Después se realiza la llamada a la base de datos para recoger los mensajes a partir de la fecha almacenada en el objeto lastSentDate. Por último, se borran las notificaciones de tipo chat y se devuelve la respuesta en lenguaje JSON.

d. Tabla en MySQL

Los mensajes del chat se guardan en una tabla MySQL con los siguientes campos:

- Id_user: identifica al usuario que envía el mensaje.
- Id_msg: identifica el mensaje.

- Msg: de tipo varchar de mil caracteres. Es el cuerpo del mensaje.
- Id_course: identifica el curso del chat.
- sentDate: fecha a la que se envía el mensaje. De tipo datetime, para almacenar días, meses, años, horas, minutos y segundos.

La sentencia MySQL para crear la tabla es:

```
create table chats(
      id_user int not null,
      id_msg int not null auto_increment,
      msg varchar(1000) not null,
      id_course int,
      sentDate datetime not null,
      primary key (id_msg),
      constraint foreign key(id_user) references users(id),
      constraint foreign key(id_course) references courses(id)
)Engine=innodb;
```

Código 4.57 *Sentencia MySQL para crear la tabla de chat*

e. Método en Manager

En la clase manager, se realiza la petición utilizando el modificador limit 10, para solo obtener diez mensajes. Además, se debe unir con la tablas users, para que se devuelvan todos los datos que van a mostrarse en el chat, como el nombre, id, etc.

```
select * from (
      select users.firstname, users.lastname, chats.id_msg,
       chats.id_user, chats.msg, chats.sentDate
      from chats left join users on chats.id_user = users.id
      where
       id_course=? and chats.sentDate > ?
      order by sentDate
      desc limit 10
) as auxtable
      order by auxtable.sentDate asc
```

Código 4.58 *Petición en el método getMessages de Manager*

2. Enviar Mensajes

La manera de enviar mensajes al chat va a ser desarrollada de manera muy parecida a la de recibirlos.

a. Controlador

La función sendMessages en el controlador realiza simplemente la llamada a la función en el servicio. No obstante, haciendo uso del 2-way data binding de AngularJS y puesto que el campo de texto, donde el usuario escribe el mensaje a enviar, y la variable msg en el scope están enlazados, se puede hacer que una vez enviado el texto se vacíe el campo, simplemente poniendo la variable a una cadena de texto vacía.

```
$scope.sendMessage = function(){

      chatService.sendMessage($rootScope.courseId,$scope.msg).then(
            function(){
                  $scope.msg =""; //2-way data binding
            }
      )
}
```

Código 4.59 *Función sendMessage del controlador chatController*

b. Servicio

```
sendMessage: function(courseId, msg){
      return $http.get('sendtochat', {
                  params: {
                        idCourse: courseId,
                        msg : msg
                  } })
                  .then(function(){
                        return ;
                  });
},
```

Código 4.60 *Función sendMessage del servicio chatService*

En este caso, mandamos al servlet SendToChat, el id del curso, que está almacenado en el rootScope y el cuerpo del mensaje. El usuario será identificado en el servlet mediante el objeto guardado en la sesión.

c. Servlet

En el servlet, SendToChat debemos manejar, por un lado, la codificación del cuerpo del mensaje: al ser una aplicación en castellano y en inglés, hay que soportar todos los caracteres de dichos lenguajes. Por otro lado, hay que comprobar si el usuario que ha enviado el mensaje es profesor en dicho curso y en ese caso mandar una notificación a todos los alumnos, pues así ha sido diseñado el sistema de notificaciones.

```
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
	try{
		Manager m = new Manager();
		User u = (User)request.getSession().getAttribute("user");
		if(u==null){
			response.sendRedirect("index.jsp");
			m.close();
			return;
		}
		int idCourse =
Integer.parseInt(request.getParameter("idCourse"));
		if(m.getRole(u.getId(), idCourse)==3)
		 m.sendNotifications("chat", Integer.valueOf(idCourse));
		String msg = new
		String(request.getParameter("msg").getBytes(
							"iso-8859-1"), "UTF-8");
		m.writeMessage(u.getId(),idCourse, msg);
		m.close();
		response.setContentType("application/json");
		}catch(Exception e){
			e.printStackTrace();
			return;
		}
	}
```

Código 4.61 *Servlet SendToChat*

d. Método en Manager

En este caso, el método es muy simple pues solo consta de una sentencia de tipo insert, que hace que todos los mensajes se almacenen. Por su simpleza no se muestra.

6. Admin

Como se ha indicado con anterioridad, el panel de administrador corresponde a una nueva aplicación que no tiene ningún tipo de conexión con la hasta ahora desarrollada. El diseño estético y la arquitectura utilizada será la misma, no obstante.

En el panel de administrador se podrán realizar las siguientes funciones: crear y eliminar cursos, dar permisos de profesor a una cuenta en un curso y eliminar usuarios. Para utilizar todas las funcionalidades de AngularJS, es interesante en nuestro caso contar con todos los cursos y usuarios cargados en el scope, de manera que el administrador pueda usar los filtros disponibles y así tener una visión más intuitiva a la hora de llevar a cabo las acciones anteriormente comentadas.

1. Carga de todos los cursos y usuarios

Se ejecutan en el controlador, AdminController, dos llamadas al servicio para obtener de la base de datos la información correspondiente a cursos y usuarios.

```
teachUsAdminService.getAllCourses().then(function(allcourses){
            $scope.allcourses = allcourses;
      })
teachUsAdminService.getAllUsers().then(function(allusers){
            $scope.allusers = allusers;
      })
```

Código 4.62 *Funciones getAllCourses y getAllUsers de AdminController*

En el servicio, AdminService, se realizan las peticiones get a los servlets. En este caso son los servlets AllCourses y GetAllUsers. El primero es el mismo utilizado en la **sección 4.b.3.4**, cuya respuesta en lenguaje JSON ahora nos permite mostrar la información de manera distinta. El segundo, GetAllUsers, es un servlet al uso cuya única funcionalidad es ejecutar el método getAllUsers de la clase mánager, que simplemente realiza una sentencia select MySQL que devuelve todos los campos de la tabla users.

2. Crear curso

Para crear el curso, el administrador debe rellenar un formulario en HTML en el cual se recogen los datos sobre: nombre, fecha límite de inscripción, descripción, idioma e imagen del curso. Estos datos, a excepción de la imagen, son los que se almacenan en la base de datos en la tabla courses (ver **sección 4.b.2.2.d**). Por otro lado, la imagen debe ser guardada bajo el nombre intromini.jpg en un nuevo directorio que tome el nombre del id del curso en la carpeta de cursos del sistema de ficheros (ver **figura 3.3**).

La petición al servlet en este caso será post, enviando todos los datos del formulario.

```
createCourse : function(files, name, language, deadline,
description){
	var fd = new FormData();
	var url = 'createcourse';
	angular.forEach(files,function(file){
	fd.append('file',file);
	});
	fd.append("name", name);
	fd.append("language", language);
	fd.append("deadline", deadline);
	fd.append("description", description);
	return $http.post(url, fd, {
		withCredentials : false,
		headers : {
		'Content-Type' : undefined
		},
		transformRequest : angular.identity
		}).then(function(){
			return;
		});
}
```

Código 4.63 *Función createCourse en el servicio adminService*

Por otro lado, el servlet se encargará de manejar dicha información y de guardarla en la base de datos y de crear el nuevo directorio.

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
 User u = (User) request.getSession().getAttribute("user");
 Manager m;
 try { m = new Manager();
 if(u==null){
	m.close();
	response.sendRedirect("index.jsp");
	return;
 }else if(m.isAdmin(u.getId())==false){//Se comprueba si el usuario
es admin
	m.close();
	response.sendRedirect("index.jsp");
	return;
 }else{
	Part filePart = request.getPart("file");
	String fileName = "intromini.jpg";
	int idCourse = m.createCourse (request.getParameter("name")
,request.getParameter("description"),request.getParameter("language")
,request.getParameter("deadline"));
	String RelativeDir =COURSES_DIRECTORY + File.separator +
idCourse + File.separator;
	File f = new File(MULTIPARTCONFIG_LOCATION+RelativeDir);
	if(f.exists()==false) f.mkdirs(); //Se crea el directorio
	filePart.write(MULTIPARTCONFIG_LOCATION+RelativeDir+fileName);
	}//Se guarda la imagen en el directorio
	m.close();
	} catch (SQLException | NamingException e){e.printStackTrace();
```

Código 4.64 *Servlet CreateCourse*

El método en la clase manager simplemente consta de una sentencia insert en MySQL almacenando todos los datos enviados en el formulario.

3. *Eliminar curso*

En primer lugar, el administrador debe indicar en un campo de texto el nombre del curso a borrar, a partir de la tercera letra se mostrará todos los cursos que coinciden con dicho nombre, pudiendo ser seleccionado uno de ellos.

```
<input ng-model="courseName.name" class="form-control" type="text"
placeholder='<%= label_admin_deletecourse_txt1%>'/>
      <ul class="list-group">
          <li class="list-group-item" ng-repeat="course in allcourses
| filter:courseName:strict | orderBy:'course.name'" ng-show=
                                    "courseName.name.length > 2" >
       <a href="" ng-click=
                        "setDeleteCourse(course)">{{course.name}}</a>
          </li>
       </ul>
```

Código 4.65 *Filtrado de nombres de cursos*

Cuando es seleccionado, se muestra el nombre del curso y la imagen, de este modo el administrador puede identificarlo correctamente antes de eliminarlo.

El servlet ahora será el reflejo del anterior, no obstante es algo más complejo pues para el borrar el directorio del sistema de ficheros hay que recorrer el directorio entero e ir vaciándolo progresivamente.

```
File f = new File(MULTIPARTCONFIG_LOCATION+relativeDir);
              if(f.isDirectory()){
                   File[] files = f.listFiles();
                   for(int i = 0; i < files.length; i++){
                         if(files[i].isDirectory()){
                               File[] files2 = files[i].listFiles();
                               for(int j = 0; i < files.length; j++){
                                     files2[j].delete();
                               }
                         }
                        files[i].delete();
                   }
              }
f.delete();
```

Código 4.66 *Borrado del contenido del curso*

Por supuesto, el servlet también incluye una llamada a la base de datos para eliminar todas las referencias al curso en las tablas.

4. Eliminar Usuario

Este apartado es muy parecido al anterior. El filtro se hará ahora sobre los emails de los usuarios, y en el servlet BanUser se borrarán las referencias en la base de datos y la posible foto de perfile subida en el directorio profilePics del sistema de ficheros.

5. Dar permisos de profesor en un curso

Esta funcionalidad usa parte de las dos anteriores. Ahora los campos de entrada que debe rellenar el administrador son dos: email del usuario y nombre del curso, utilizando el filtrado explicado con anterioridad.

En la base de datos se actualizará la tabla roles insertando una fila con los ids del usuario y del curso y el campo role pasará a ser “teacher”.

5. Despliegue de la aplicación web

Lo primero es dirigirse a la página de openshift, openshift.redhat.com, donde nos permite crear una cuenta gratuitamente. Tras esto hay que crear una aplicación utilizando como servidor Tomcat 7, a la que después añadiremos el módulo de la base de datos MySQL.

Tras esto, se pasa a la fase de configuración, en la cual es necesario modificar algunos fragmentos de códigos del capítulo anterior para su correcta compatibilidad. En primer lugar, es necesario crear una copia del repositorio de la aplicación, siguiendo los pasos que nos da Openshift, desde la línea de comandos:

```
//Crear repositorio
$ git clone <git_url> <directory_to_create>
//Enviar cambios
$ git commit -a -m 'Some commit message'
$ git push
```

Código 5.1 *Crear repositorio y enviar cambios*

Y será en la carpeta webapps donde se debe colocar el archivo .WAR, es decir la aplicación comprimida que, por ejemplo, se puede obtener desde el IDE Eclipse exportando el proyecto a dicho formato.

Además, habrá que realizar cambios en el archivo context.xml (**Código 4.1**), de manera que se actualicen los datos de url, username, password, etc. Para conocer dicha información nueva, hay que conectarse remotamente con rhc, programa creado por Openshift, al directorio de la aplicación y en el directorio Mysql/env tomar los valores que tienen los ficheros OPENSHIFT_MYSQL_DB_URL/PASSWORD/USERNAME.

Por último, para configurar el sistema de ficheros es necesario conectarse con algún cliente FTP, por ejemplo FileZilla, que permita crear la jerarquía de carpetas(**Figura 3.3**) y que nos muestre la ruta, pues esta debe ser utilizada por todos los servlets que manipulan en algún momento el sistema de ficheros.

Tras esto, es corriente seguir encontrando pequeños problemas que atienden a diferencias entre las versiones que se han utilizado para desarrollar en local y las finalmente usadas en Openshift.

La aplicación **TeachUs!** puede ser visitada en la dirección:

http://teachus-sergiotfg.rhcloud.com/TFG/

6. Pruebas

Durante el desarrollo, se han utilizado dos herramientas muy útiles para la corrección de errores: AngularJS Batarang y la consola de desarrollador del navegador Chrome.

AngularJS Batarang es un plugin para el navegador Chrome que permite visualizar los objetos almacenados en el scope y llevar a cabo tareas de debug.

La consola de desarrollador del navegador Chrome es muy útil porque, por un lado, captura todas las peticiones realizadas al servlet y su estado, lo cual es muy beneficioso cuando se está trabajando con peticiones asíncronas que no se muestran explícitamente. Por otro lado, permite el diseño estético de la aplicación, mostrando cada elemento y sus propiedades y de donde hereda.

Tras el desarrollo, una herramienta interesante es PageSpeed de Google, que da una puntuación a la velocidad de carga y puede corregir ciertos fallos como la carga de scripts en momentos no propicios para ello.

Se obtiene una puntuación de 65/100 en velocidad para móviles, un 92/100 en la experiencia para móviles, y 70/100 en velocidad para ordenadores. Por contextualizar dichas puntuaciones, si se realiza la prueba a la página de la universidad, www.uc3m.es, obtiene notas similares.

7. Presupuesto y normativa

a. Presupuesto

La manera en que se monetizaría el trabajo realizado debería ser la venta completa de la aplicación web a alguna compañía que necesite de una virtualización de algunas clases. Por ejemplo, universidades, tanto públicas como privadas, academias, institutos, colegios, etc.

En este caso, se valoraría el precio de venta alrededor de los **7000€ + IVA**. Aunque puede parecer un precio bajo si tenemos en cuenta el tiempo invertido en este proyecto, hay que valorar que gran parte se ha invertido en el aprendizaje y búsqueda de las herramientas utilizadas. Por este hecho, este mismo desarrollo, incluso con más funcionalidades, podría haberse realizado en 4-6 semanas si se manejan las herramientas con soltura.

Otra opción, que se considera inviable, sería poner en funcionamiento la web con publicidad como única fuente de ingreso. No obstante, simplemente para cubrir el coste salarial de cinco o seis profesores, los alumnos inscritos en cada curso superarían ampliamente lo manejable. Además habría que cubrir costes de hosting, cuyos precios vienen marcados en la página de Openshift. En este caso el presupuesto sería:

Concepto	Precio (mensual)
Coste salarial de 5 profesores	5000 €
Hosting	15€ + 15€ de 15GB extras de almacenamiento
Desarrollo	1800 € (Solo un mes)
Administrador	800 €
Total	7630 €

Tabla 7.1 *Presupuesto de poner en marcha la aplicación*

Los cinco profesores podrían manejar seis o siete cursos, pertenecientes a la misma materia, lo cual llevaría a un total de unos treintaicinco cursos en la aplicación. El hosting encarece su precio debido a que es necesario contratar almacenamiento extra que pueda contener el sistema de ficheros creado.

b. Normativa

Puesto que TeachUs! recoge datos personales de los usuarios a la hora de registrarse, hay que cumplir con la Ley Orgánica de Protección de Datos de Carácter Personal. Para ello a la hora del registro hay que:

- Mostrar el aviso de que el registro en la aplicación conlleva la aceptación de nuestra política de privacidad.
- Garantizar la seguridad de dichos datos, realizar copias de seguridad semanales.
- Registrar el fichero de datos en cuestión, en nuestro caso la tabla users de MySQL, en la Agencia Española de Protección de Datos.

En el caso además, de que la web funcionase con publicidad, esto conllevaría el uso de cookies y, por ello, habría que cumplir:

- Mensaje informando al usuario de que da su consentimiento.
- Apartado en el que se explique para qué son las cookies y con qué finalidad se utilizan, como se pueden configurar, etc.
- Enlace siempre disponible a dicho apartado.

8. Conclusiones y proyectos futuros

Una vez desarrollada la aplicación, podemos concluir que los objetivos planteados en el primer capítulo han sido cumplidos.

- ✓ Subida de contenido
- ✓ Chat
- ✓ Exámenes de autoevaluación
- ✓ Sistema de notificaciones
- ✓ Panel de administrador

Además, se han obtenido los conocimientos básicos sobre dos herramientas punteras como son AngularJS y Bootstrap. No obstante, sobre la primera aún se podría profundizar mucho en el uso de directivas, que apenas ha sido tratado en el proyecto. Por otro lado, también es cierto que una web de estas características siempre está abierta a ampliaciones de contenido, entre las que podrían figurar:

- Sistema drag & drop para la subida de archivos
- Barra de carga durante la subida de archivos
- Sistema de reporte de mensajes inapropiados en el chat que lleguen al administrador.
- Interacción con redes sociales mediante el uso de las APIs proporcionadas por Youtube, Twitter y Facebook.
- Nuevos idiomas
- Posibilidad de editar los datos de registro
- Mayor número de iteraciones en la codificación de la contraseña para hacer más costoso un posible ataque.

En otro orden de cosas, después de llevar a cabo un proyecto de longitud considerable, la necesidad de realizar un diseño concreto y específico de la arquitectura de la aplicación se ha hecho notar. Si, como ha sido el caso, algunas funcionalidades se añaden posteriormente al comienzo del desarrollo es corriente encontrar dificultades a la hora de encajarlas en la aplicación.

Además, ha habido varios problemas a la hora de compatibilizar versiones entre el desarrollo en local y la posterior subida a la PaaS. Por ejemplo, en local se utilizaba Java 8 y Tomcat 8, que son las últimas versiones lanzadas, no obstante Openshift y la mayoría de las plataformas consultadas soportaba Java 7 y Tomcat 7, con lo cual se hacía necesario rehacer el diseño y buscar alternativas. Esta quizás haya sido de las enseñanzas más valiosas: hay que conocer para qué soporte se va a diseñar, y ceñirse a dichas características.

También, por otro lado, lo útil de las comunidades de desarrolladores: en ocasiones, no es necesario reinventar la rueda. En ese sentido, se agradece que herramientas como AngularJS, que cuentan con multitud de gente usándola, favorezcan la cooperación facilitando la compartimentación de módulos que son implementables en otros proyectos. Esto no solo es favorable desde el punto de vista de ahorrar tiempo y trabajo, sino que además, el código ha sido probado en múltiples ocasiones y perfeccionado por numerosos desarrolladores.

Summary

Introduction, motivations and goals

The history of website development

When the first steps of website development were taken, the main goal set by the creator of HTML language, Tim Berners-Lee, was to publish information stored by the CERN, the European Organization for Nuclear Research, in a way that it would be easily accessed by the scientist that were working there, despite the computer they were using.

After some years were the new language raised its popularity, in 1994 Rasmus Lerdorf started to develop PHP, which means that the first interactions between websites and users started to appear. Then, new platforms and scripts arrived, such as JavaScript, Flash, that dig deeper for the interactions that users can make to the content of the website.

Current website development

Nowadays, after a big proliferation of smartphones, tablets and other mobile devices since 2007, the website concept as a platform is very widespread. In other words, the traditional desktop applications now use the advantage of internet to customize the content that is showed to each user. Besides, it all seems to point out that this is only the beginning, as the development of multiple apps by the industy giants, Google and Facebook, proves.

However, this devices have very different features compared to each other, such as distinct screen sizes, accessibility, available bandwidth, etc. Also, there are numerous browsers that read the HTML slightly different.

This means that the complexity increases exponentially for a website developer, who should spend some extra time optimizing for each possible mixture.

In light of this need, multiple open source projects have surged lately in order to help these situations.

Motivations

The motivation of this project is, first of all, to take a look at the tools that are very popular in website development and essential in the industry, because it is a growing market. Then, the most important tools will be studied and used, getting the knowledge base.

Besides, it is interesting to address a fairly big project from scratch because the workflow is improved, getting close to an optimized development.

Goals

The goal consists in creating a web application about online courses that make use of cutting edge tools in this sector.

The user will be able to register in the website and, once inside, he may join twelve week courses that have not started yet. Furthermore, the teachers will upload content to such courses, which includes PDF, DOCx, pictures and tests. In addition, a chat will be implemented to communicate users and teachers and a notification system will be created so that users are aware of any change in the course.

Finally, an administration panel will be made so that the management of the website is eased.

Extended summary

State of the art

In order to make new websites compatible with different screen sizes and to optimize the bandwith, two tools have emerged lately: **Front-End Frameworks** and **SPA libraries**.

Front-End Frameworks

A Front-End Framework is an utility that encloses a collection of HMTL, CSS and JavaScript content, ready to be used. They are prepared to adapt to every configuration the devices may have, so, even if coding your own content is not very hard, the use of frameworks has been extended as if it was a need.

However, since 2010 a lot of frameworks have appeared in the market, with different levels of complexity due to the customization they allow. In addition, some of these even use JQuery-based libraries to make asynchronous HTTP petitions.

With these many frameworks the problem is that there was no clear standardization, and so the developer's community was divided and sometimes invested its time in projects that occasionally were abandoned o or not up to date. Over the years, **Bootstrap** is, by far, the most popular and therefore it will be used in this project.

Bootstrap

It provides a twelve columns grid that divides the space of the web page and will redistribute in orderly way after the screen size changes. For example this code:

```
<div class="row">
            <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2">
                  Block 1
            </div>
            <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2">
                  Block 2
            </div>
            <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2">
                  Block 3
            </div>
            <div class="col-xs-6 col-sm-4 col-md-3 col-lg-2">
                  Block 4
            </div>
</div>
```

Code 2.1 *Example of the grid in Bootstrap*

Works this way:

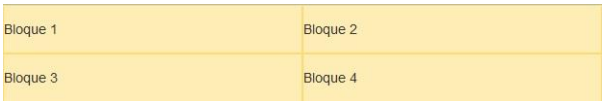
Resolution	Mode	Explanation	Visualization
< 767 px	Extra small	Each block occupies 6 columns, y therefore, the last two stack up	
> 768 px	Small	Each block occupies 4 columns, so the last block stacks up	
> 992 px	Medium	Each block occupies 3 columns, so they fill the entire row	
> 1200 px	Large	Each block occupies 2 columns, so 4 columns are left blank	

Table 2.1 *Explanation of code 2.1*

SPA libraries

A Single Page Application is a web that dynamically loads the requested resources by the user without loading the entire page. This provides the user a more fluid experience because it resembles a classic desktop app and saves bandwidth.

These libraries are written in JavaScript and use asynchronous petitions (AJAX). Also, the information is transported in XML or JSON that need to be treated before updating the DOM in HTML.

In this sector, **AngularJS** is the most popular.

AngularJS

The biggest advantage is that it was developed by employees of Google, so it has major support compared to its competitors. It is based on the Model-View-Controller architecture:

- Model: It is the data, in AngularJS is represented by the object Scope.
- View: It is the information that it is showed to the user.
- Controller: The user interacts with it, so the model is modified and the view is updated.

Besides that, AngularJS adds the 2-way data binding, where de view and the model are linked so if one gets modified the other will know.

Finally, the routes are a very important feature, because a part of the HTML document can be defined to contain views. These views changes depending on the url and have specific a controller and a different scope.

Architecture and design

The app, called **Teach Us!,** will be available in two languages: Spanish and English. The initial view of the app is a landing page, which will feature information about disposable courses and the teachers hired.

The lowest level of the application is, first, a MySQL database that stores data about courses, messages in chats, tests, users, such as name, password, etc. Second, a file system will be created so every content uploaded by teachers, such as PDF, DOCx, etc, is saved there. These two will be accessed through Java Servlets.

1. Index

In here, only Bootstrap will be used, while AngularJS comes in once the user has logged in. To make the asynchronous petitions we are using AJAX, that comes along the JQuery library.

1. Register

The Java Servlet Register is control here. After it has received the data from a form, it will transport it to the data base, where the codification of sensitive fields will take place. This codification will be based in the SHA256 algorithm and it will apply a

hash function to the password plus some randomly generated bytes, usually called salt. The result of this is stored in the data base, so the password can only be checked but it is not actually stored in the data base. Also, an email will be sent to the email provided by the user using a free smtp server by Google.

2. Login

In the login, first, the SignIn servlet will check the data introduced by the user. After that, it will SendRedirect to the servlet Overview, which is responsible for loading the data need in the application.

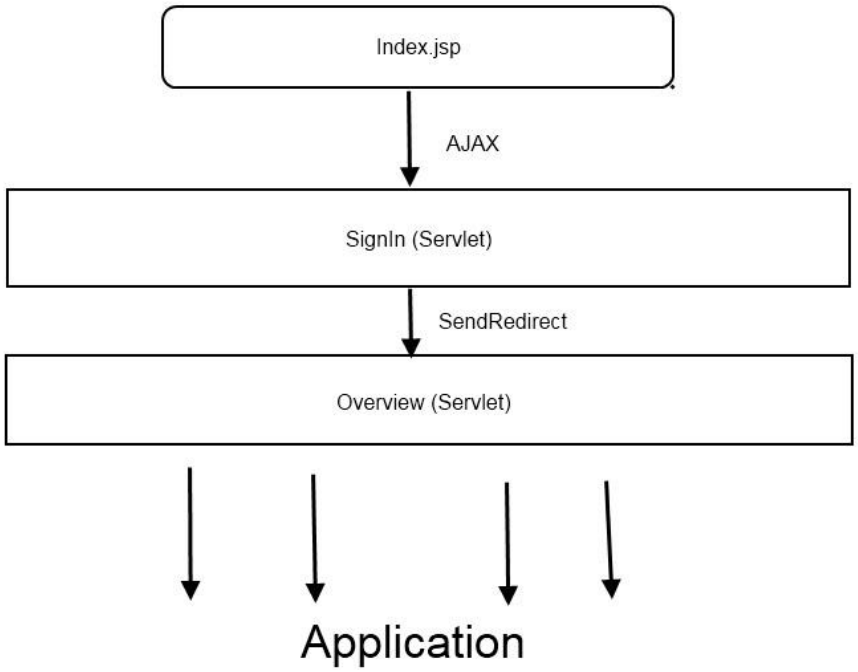


Figure 3.1 *Login*

3. Reset Password

The user will be able to reset his password just by writing his email, which will be searched in the data base, if it exists, a new password will be randomly generated and sent to the email. Of course, then it will be codified just as explain before in the register section.

2. Application

Once the user is logged in the application, the general scheme is represented in the **figure 3.2.** Besides, there are two controllers overlapped, one is called

TeachUsController and will always be functioning because it will manage the elements that are always present, such as the navigation bar or the notification system. The other one is specific to the route, also called view. These views are four.

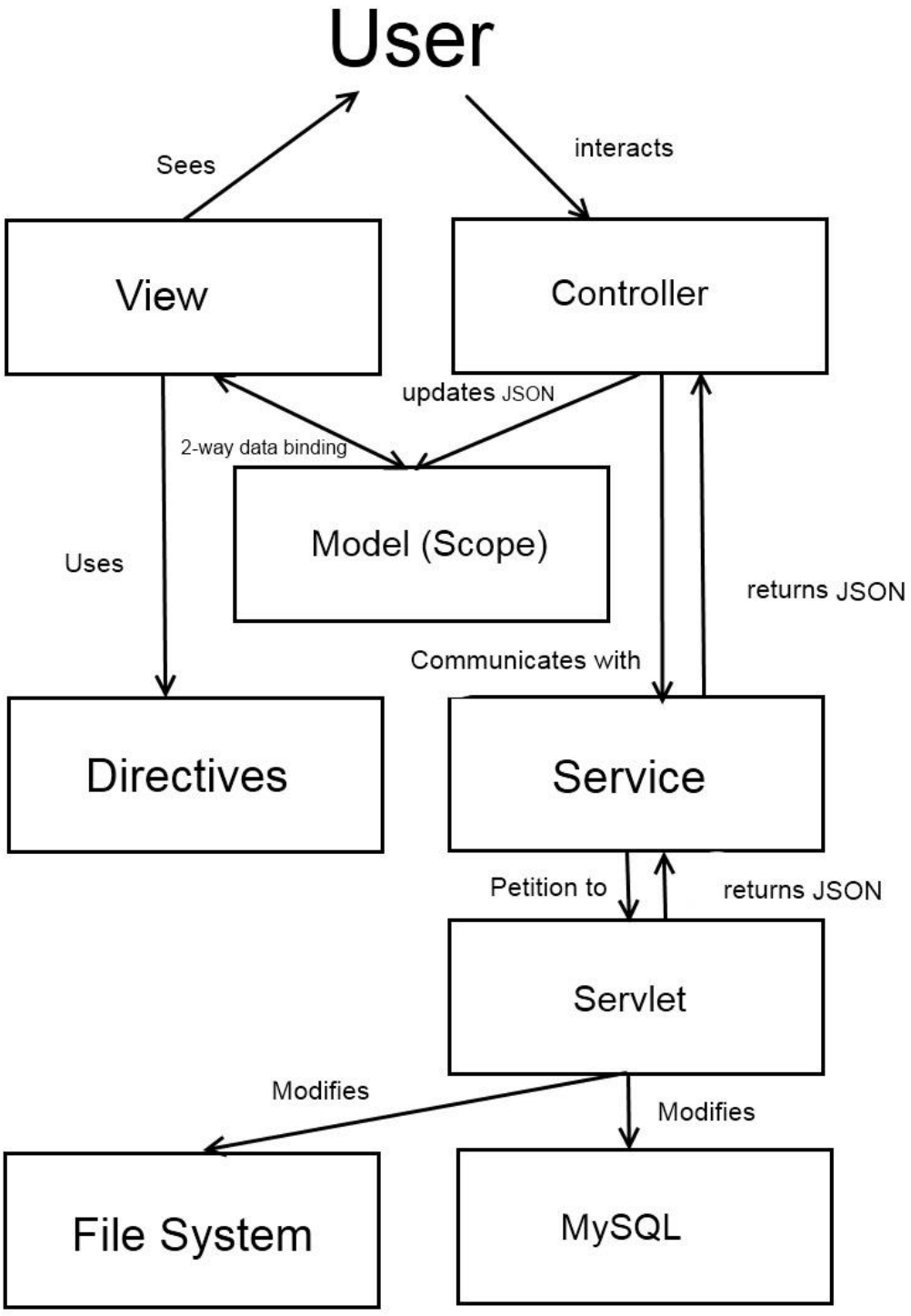


Figure 3.2 *General scheme of the application*

1. Overview

Its controller will be the teachUsController. It is not a route, but contains functionalities available in every view, such as change profile picture, signout, change language and notification system.

2. Home

This view is the landing page inside the application, it is meant to show the courses that the user has signed for, alerts about updates in courses, and some available courses for advertising purposes.

3. AllCourses

In this view, every course in the app will be shown. The user is going to be able to write in an input text field that will filter the courses, so only the ones that matches are shown.

4. View

This is the most complex view because it contains the logic to show the users data depending on their role. This means that teacher can:

- Upload file to the file system.
- Create test.
- Access to the content uploaded.
- Rename or remove files.

However, students can:

- Access to the content uploaded.

And those accounts that have not signed up for the course will only be able to do it if the deadline has not slipped past.

5. Chat

This is an internal route to view, so it is related to the course. It will allow the student to send and receive messages to other students or the teacher.

6. Admin

An account can only be redirected to this route if it is identified as the administrator of the site. Then, it will allow to create or delete courses, grant teacher rights to an user in a course and ban users.

Development

This is a chapter that goes step by step explaining every functionality in the Spanish version, so it cannot be summarize. However, the basics are:

Defining routes

```
teachUsApp.config(function($routeProvider){
            $routeProvider
                  .when('/home',
                        {
                              controller: 'homeController',
                              templateUrl: 'partials/home.jsp'
                        })
                  .otherwise({ redirectTo:'/home'});
})
```

Code 4.30 *Defining home route*

Defining controllers and calling its service

```
teachUsApp.controller('homeController', function($scope,
$rootScope, homeService){
      homeService.getCourses().then(function(courses){
            $rootScope.idUser = courses.idUser;
            $rootScope.myCourses = courses.myCourses;
            $scope.courses=courses;
      });
})
```

Code 4.32 *Controller homeController*

Function in a service that calls a servlet

```
teachUsApp.service('homeService', function($http){
      return{
            getCourses: function(){
            return $http.get('home')
                  .then(function(result){
                        return result.data;
                  });
            }}
```

Code 4.33 *Service homeService*

Java Servlet

```
	protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException{
	try{
		Manager m = new Manager();
		User u = (User)
			request.getSession().getAttribute("user");
		if(u==null){
			response.sendRedirect("index.jsp");
			m.close();
			return;
		}
		List<Course> myCourses =  m.getMyCourses(u.getId());
		List<Course> adsCourses =
				m.getAdsCourses(u.getCountry(), myCourses);
		m.close();
		String coursesjson =
			serializationCoursesAndId(myCourses,adsCourses,
								u.getId());
		response.setContentType("application/json");
		response.getWriter().write(coursesjson);
	}catch(Exception e){
	e.printStackTrace();
	return;
	}
}
```

Code 4.34 *Home Servlet*

Method from Manager Clase

```
public List<Course> getMyCourses(int id) throws SQLException{
	String query = "select courses.id, courses.course,
			courses.description, "+ "courses.language,
			courses.deadline from courses left join roles "
			+ "on courses.id = roles.id_course where
			id_user=?;";
	PreparedStatement stmt = connection.prepareStatement(query);
	stmt.setInt(1, id);
	ResultSet rs = stmt.executeQuery();
	List<Course> myCourses = new ArrayList<Course>();
	while(rs.next()){
		Course cr = new Course();
		cr.setId(rs.getInt("courses.id"));
		cr.setName(rs.getString("courses.course"));
		cr.setDescription(rs.getString("courses.description"));
		cr.setLanguage(rs.getString("courses.language"));
		cr.setDeadline(rs.getString("courses.deadline"));
		myCourses.add(cr);
	}
	stmt.close();
	return myCourses;
```

Code 4.36 *getMyCourses method from Manager clase*

Conclusions and future projects

Once the app has been developed, we may conclude that the goals proposed in the first chapter have been accomplished.

- ✓ Content uploading
- ✓ Chat
- ✓ Test
- ✓ Notification system
- ✓ Administration panel

In addition, it has been gained the knowledge base about two cutting edge tools, such as AngularJS and Bootstrap. However, it could be dug deeper about the first one in the use of directives, which are barely treated in this project. On the other hand, this kind of website is always open to updates, which may feature:

- Drag & drop system to upload files.
- Loading bar while the content is uploaded.
- System to report inappropriate messages in the chat and deliver to the administrator.
- Incorporating social networking through public APIs by Youtube, Facebook, and Twitter.
- New languages.
- User being able to change account info.
- Increase number of iterations when coding the password, in order to make more expensive an attack.

On a separate issue, after accomplishing a project with such length, the need to do a specific design of the app architecture has surfaced. If, as it has happened, some functionalities are added after the beginning of the development, it is common to find difficulties when they are put together.

Besides, there has been complications when deploying the app in the PaaS, as some versions where not the same in local. For instance, in local Java 8 and Tomcat 8 were used, which are the latest versions released, however, Openshift and most of the

platforms consulted supported only Java 7 and Tomcat 7, which lead to redesign and look for alternatives. This perhaps has been one of the most valuable lesson: You have to know what you are designing for and keep to those features.

Also, it has been learnt how useful developers communities are: Sometimes, you do not need to reinvent the wheel. In this regard, I am grateful that tools like AngularJS, which has lots of people using it, favor the cooperation easing the partitioning of modules that can be implemented later in other projects. This does not only save time and work but also the code has been tested multiple times and has been perfected by the feedback of numerous developers.

Bibliografía

1. *Netbeans. Setting Up the MySQL Database Server in the Windows Operating System.* https://netbeans.org/kb/docs/ide/install-and-configure-mysql-server.html
2. *CodePly. Just-One-Page Bootstrap Template.* http://www.bootstrapzero.com/bootstrap-template/just-one-page
3. *Página web oficial de Bootstrap.* http://getbootstrap.com/
4. *Página web oficial de AngularJS.* https://angularjs.org/
5. *Página web oficial de Openshift.* https://www.openshift.com/
6. *Lerner A.* (2013). Ng-book: The complete book on AngularJS. Fullstack.
7. *Spurlock J.* (2013). Bootstrap. O'Reilly.
8. *Sheshadri S. & Green B.* (2014). AngularJS: Up and Running: Enhanced Productivity with Structured Web Apps.
9. *BOE. Ley de orgánica de protección de datos de carácter personal.* http://www.boe.es/buscar/act.php?id=BOE-A-1999-23750
10. *BOE. Ley de servicios de la sociedad de la información y de comercio electrónico. http://www.agpd.es/portalwebAGPD/canaldocumentacion/legislacion/normativa_estatal/common/pdfs/Ley_34_2002.pdf*