



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN

GRADO EN INGENIERÍA EN TECNOLOGÍAS DE LA
TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

SISTEMA DE REGISTRO DE ASISTENCIA DE ALUMNOS Y
PROFESORES

Autor : Jorge Bazaco Caloto

Tutor : Gregorio Robles Martínez

Curso académico 2013/2014

Resumen

Este trabajo desarrolla una herramienta para controlar la asistencia e impartición de las lecciones, tanto del personal docente como de los alumnos, en un ámbito universitario. Pretende, por tanto, aportar una solución para la recogida de dicha información, simplificar el proceso y añadir funciones más allá del simple control de asistencia; pudiendo analizar el desarrollo de una asignatura y, a su vez, permitir realizar un *feedback* para conseguir mejorar la formación académica.

Desde la implantación del plan Bolonia en el ámbito universitario, la asistencia a clase se ha impuesto de forma obligatoria, surgiendo la necesidad de controlar la misma.

Abstract

This project develops a tool to control the attendance and teaching of lessons, related not just with the students but also with the teaching staff, in an University scope. It claims, therefore, to be a solution to collect that information, to simplify the process and to add new features that go further than the single attendance control; being able to analyse the subjects' development and, at the same time, to make a feedback in order to improve the academic background.

Since the establishment of Bolonia's plan in the University background, the class attendance has been made mandatory, appearing the need of controlling it.

Índice general

1. Introducción	1
1.1. Explicación del trabajo	1
1.2. Situación tecnológica actual	1
1.3. Comparación con otros sistemas	2
1.3.1. Recogida de firmas en papel	2
1.3.2. Pasar lista	3
1.3.3. Control de asistencia del profesorado de la URJC	3
1.3.4. Otros sistemas de geolocalización en interiores	3
1.4. Estructura de la memoria	5
2. Objetivos	7
3. Tecnologías utilizadas	9
3.1. Códigos <i>QR</i>	9
3.2. <i>HTML5</i>	10
3.3. <i>JavaScript</i>	12
3.3.1. <i>AJAX (Asynchronous JavaScript And XML)</i>	13
3.3.2. <i>jQuery</i>	13
3.3.3. <i>JSON</i>	14
3.4. <i>CSS3</i>	15
3.5. <i>Bootstrap</i>	16
3.6. <i>Python y Django</i>	18
3.6.1. <i>Python</i>	18
3.6.2. <i>Django</i>	19

3.7. <i>Pylint</i>	21
3.8. <i>Git</i>	21
4. Implementación	23
4.1. Interacción entre los diferentes subsistemas	23
4.1.1. Estructura de directorios y ficheros	25
4.2. Diseño de la base de datos	27
4.2.1. <i>Models</i> o modelos de <i>Django</i>	28
4.3. Tipos de usuarios	32
4.4. Diseño de las vistas y las <i>URLs</i>	33
4.5. Ficheros <i>JavaScript</i>	60
4.6. Diseño de la aplicación y diferencias entre dispositivos	61
4.6.1. Distribución y tamaño del contenido	61
4.6.2. Simulación de una aplicación nativa	62
4.6.3. Otras diferencias	64
5. Conclusiones	67
5.1. Evaluación del resultado	67
5.2. Conocimientos aplicados	68
5.3. Conocimientos adquiridos	69
5.4. Trabajos futuros	69
A. Instalación y uso	71
B. Recurso para descargar la aplicación	73
C. Licencia	75
D. Manual de usuario	77
Bibliografía	93

Capítulo 1

Introducción

1.1. Explicación del trabajo

El presente trabajo desarrolla una aplicación web para entornos universitarios, que permite registrar y comprobar la asistencia de los profesores y alumnos a clase. Esta aplicación permite, además, realizar una gestión de las asignaturas, obteniendo estadísticas de las mismas, no solo de la asistencia, sino también de la valoración de los alumnos, permitiendo realizar una retro-alimentación para que los profesores puedan mejorar y permitiendo detectar anomalías en el desarrollo de alguna asignatura.

La aplicación también permite a los profesores crear nuevas clases, organizar seminarios (a los cuales pueden apuntarse los alumnos) y utilizarla como una vía de comunicación con los alumnos.

Además, la página web presenta un foro donde todos los miembros de la universidad pueden comunicarse de forma pública para facilitar su colaboración y mejorar el entorno universitario.

1.2. Situación tecnológica actual

Actualmente se está produciendo una gran evolución en el ámbito móvil, con la aparición de los denominados *smartphones* o teléfonos inteligentes, aportando una herramienta para conectarse a internet desde cualquier sitio. El hecho de que una gran parte de la población tenga uno de estos dispositivos y lo lleve consigo la mayor parte del tiempo, situó a los móviles en el centro de este proyecto, afectando a la mayoría de decisiones, anteponiendo el móvil ante otros

dispositivos, pero sin descuidar a los ordenadores.

El principal problema es la gran diversidad de *smartphones*, referido tanto a la diversidad hardware —principalmente los diferentes tamaños de pantalla— como a la variedad de sistemas operativos, e incluso entre ellos, diferentes versiones. Una fuente de inspiración en este trabajo ha sido el sistema operativo móvil de *Mozilla*, *FirefoxOS*, compañía creadora de uno de los navegadores de internet más utilizados en la actualidad; y en menor medida *Ubuntu Phone*, con una idea parecida, pero siendo un *SO* que ha pasado más desapercibido. La base de este sistema operativo es el desarrollo de aplicaciones utilizando tecnologías libres y abiertas, en concreto *CSS*, *HTML* y *JavaScript*, que se ejecutan dentro de un navegador. La principal ventaja es que desarrollando una aplicación web con estas tres tecnologías, se obtiene un programa compatible con casi cualquier dispositivo, teniendo que preocuparse exclusivamente de pequeñas variaciones debidas a las diferencias entre navegadores de internet (ya que se cambia la idea de adaptarse a diferentes sistemas operativos por diferentes navegadores de internet) y a los diferentes tamaños de pantalla y métodos de control —pantalla táctil en móviles y *tablets* o ratón y teclado en ordenadores, por ejemplo—.

1.3. Comparación con otros sistemas

A continuación se realiza un breve análisis de algunos sistemas de control de asistencia, tanto tradicionales como informáticos, y tanto reales como posibles alternativas que quedaron descartadas en la fase inicial de este trabajo.

1.3.1. Recogida de firmas en papel

Uno de los sistemas más comunes consiste en una hoja con firmas de los asistentes para probar su asistencia.

Para el caso de la asistencia de alumnos, algunos profesores facilitan una hoja a sus alumnos en la que todos los asistentes deben firmar para demostrar que estaban en clase.

La principal desventaja de este método es que el procesamiento de toda esta información es complejo, por ejemplo, en el caso de los profesores comprobando la asistencia de sus alumnos tienen que ver hoja por hoja si un alumno asistió o no a cada lección para extraer una media. Además, aunque este sistema parezca imposible de falsificar, en realidad cualquier alumno

podría firmar por otro compañero. Otro inconveniente es que es fácil traspapelar o perder alguna hoja, frente a la organización que ofrece un sistema informatizado.

1.3.2. Pasar lista

Otro método consiste en un profesor nombrando a cada alumno y apuntando si ha asistido.

Presenta mejoras respecto a la recogida de firmas, como son una mayor organización al poder el profesor recoger la información de todas las clases en el mismo documento y una mayor fiabilidad al obligar al alumno a demostrar su asistencia directamente de forma presencial al interesado de la misma.

La mayor desventaja es el tiempo que se pierde al pasar lista y no disponer de las funcionalidades extras que presenta la aplicación informática.

1.3.3. Control de asistencia del profesorado de la URJC

La Universidad Rey Juan Carlos ha implantado un sistema informático para controlar que sus profesores imparten sus clases.

Para ello, se les facilita una tarjeta personal a los profesores que deben pasarla por un lector para demostrar su asistencia. El inconveniente de este sistema es la necesidad de instalar lectores, con su correspondiente coste. Además, el sistema no contempla la asistencia de los alumnos y carece de funciones presentes en la aplicación desarrollada, como por ejemplo la retroalimentación de los alumnos.

Una ventaja con respecto a la aplicación desarrollada en este trabajo es la posibilidad de justificar una ausencia, presente en el sistema de la Universidad Rey Juan Carlos. La aplicación de este trabajo permite la creación sencilla de clases extraoficiales para recuperar aquellas que no se han podido impartir.

1.3.4. Otros sistemas de geolocalización en interiores

Para demostrar la asistencia a un evento es necesario poder demostrar que se estuvo físicamente en el mismo. Actualmente la localización en interiores es un campo poco evolucionado, en parte por la falta de un estándar, lo cual puede ser debido a la ausencia de un sistema que suponga una solución real, barata, precisa y fácil de implementar y extenderse.

Por ello, para la realización de este proyecto se han tenido en cuenta varias tecnologías:

- **GPS (Sistema de Posicionamiento Global) o localización por redes:** el *GPS* permite determinar la posición de un objeto mediante el empleo de satélites. En la mayoría de dispositivos modernos hay implantado un módulo *GPS* o pueden ser localizados mediante la red a través de la cual se conectan a internet. El principal problema es la poca precisión que se obtiene en interiores, y sus ventajas son que está muy extendido y que a través de *JavaScript* —tecnología que utiliza la aplicación— es posible obtener la información recogida por estos dos métodos.
- **NFC (Comunicación de Campo Cercano):** es una tecnología abierta que permite la comunicación a una distancia muy corta —unos pocos centímetros— entre un dispositivo de pequeño tamaño sin alimentación (pasivo) y un dispositivo que genera un campo electromagnético (activo); también es posible la comunicación entre dos dispositivos activos pero para este caso solo interesa la conexión activo-pasivo. Para utilizarlo en este sistema, se colocarían etiquetas *NFC* pasivas en cada aula (las cuales tienen un coste muy reducido y ocupan muy poco espacio), que tendrían un código que identifica a ese aula —que por seguridad podría modificarse diariamente—, y el usuario al pasar su móvil cerca de la etiqueta obtendría el código para demostrar su asistencia. Este sistema actualmente tiene muchos problemas pero podría ser interesante en el futuro si se solucionan. Los principales problemas se deben principalmente a ser una tecnología relativamente nueva, y que, a pesar de ser un estándar, a día de hoy son pocos los dispositivos que lo integran y no está estandarizado su acceso desde *JavaScript*, aunque *Mozilla* tiene desarrollada y propuesta una implementación propia para ser estandarizada.
- **Códigos QR (Código de Respuesta Rápida o código bidimensional):** son códigos de barras bidimensionales —imágenes cuadradas formadas por píxeles blancos y negros—, cuya popularidad se debe a su sencillez y a su estandarización abierta. Pueden almacenar un código alfanumérico e imprimirse o mostrarse en una pantalla y ponerse en cada aula, funcionando de manera similar a la solución propuesta con el *NFC*, pero sin sus inconvenientes, al poder ser escaneados con una cámara, la cual está presente en la mayoría de dispositivos modernos, y a la cual se tiene acceso a través de *JavaScript*.

- **Dispositivos de bajo consumo con *Wi-Fi* o *Bluetooth*:** hay multitud de sistemas de localización en interiores basados en la distribución de dispositivos con antenas *Wi-Fi* o *Bluetooth* por un edificio, cada uno de los cuales transmite un mensaje diferente que identifica a cada localización. El problema principal es el coste de los dispositivos y la necesidad de diseñar la distribución de los dispositivos para un edificio concreto, dificultando la implantación de la aplicación en cualquier universidad al requerir este diseño específico.

Tras estudiar las ventajas e inconvenientes de cada sistema de localización, se ha optado por combinar el empleo de la localización integrada en *JavaScript* mediante el *GPS* o el uso de redes, junto con los códigos *QR*. Esta decisión se debe a su facilidad, tanto en la integración en la aplicación —al ser tecnologías muy extendidas hay mucha información y software libre para su empleo—, como en su utilización por parte del usuario. Además, como el código *QR* es realmente una cadena alfanumérica, es posible dar otra alternativa a usuarios que tengan dispositivos sin soporte *JavaScript* o sin cámara —aunque sea un bajo porcentaje, pero se persigue la universalidad— pudiendo introducir el código manualmente.

1.4. Estructura de la memoria

- Capítulo 1 “Introducción”: este capítulo realiza una descripción concisa del trabajo realizado, situándolo en su contexto tecnológico y comparándolo con otros sistemas similares. Finalmente se describe la estructura de la memoria.
- Capítulo 2 “Objetivos”: este capítulo describe el objetivo principal y los subobjetivos que han guiado el desarrollo del trabajo.
- Capítulo 3 “Tecnologías utilizadas”: este apartado explica las principales tecnologías empleadas para la creación de la aplicación, para una mejor comprensión de la propia implementación.
- Capítulo 4 “Implementación”: este apartado detalla el funcionamiento de la aplicación resultante. Se explica la estructura del proyecto y la interacción entre los distintos subsistemas, la estructura de la base de datos y los distintos tipos de usuarios que utilizan

la aplicación y sus funciones o permisos. También se enumeran y explican las distintas páginas o recursos existentes y el diseño de las mismas para obtener una compatibilidad universal.

- Capítulo 5 “Conclusiones”: este apartado realiza un análisis del resultado, evaluando el resultado de los objetivos propuestos inicialmente, detallando los conocimientos aplicados en el desarrollo del trabajo y los conocimientos adquiridos durante la realización del mismo. Finalmente plantea posibles futuras líneas de trabajo sobre el resultado obtenido.
- Apéndice A “Instalación y uso”: este apéndice explica como arrancar el servidor para poder utilizar la aplicación.
- Apéndice D “Manual de usuario”: este apéndice contiene un manual de usuario para utilizar la aplicación.

Capítulo 2

Objetivos

El objetivo de este trabajo se basa en el desarrollo de una herramienta capaz de registrar la asistencia a clase y de procesar dicha información.

Los principales subobjetivos que han guiado el desarrollo del presente programa son:

- Aplicación universal: éste es uno de los objetivos más importantes; desarrollar un programa que pueda utilizarse desde prácticamente cualquier plataforma, para que todos los usuarios puedan acceder a la aplicación de registro de asistencia desde sus propios dispositivos, ya que todos los alumnos y profesores deben poder indicar que han asistido a una clase.
- Registrar la asistencia a clase: debe poder permitir a un usuario demostrar que ha asistido a una lección y esta información debe quedar guardada y ser accesible para el usuario que deba visualizarla.
- Introducir mejoras en el ámbito académico y no limitarse exclusivamente a ofrecer una alternativa a métodos más tradicionales. Es importante que aporte algo que los métodos tradicionales no pueden ofrecer, ya sean funcionalidades añadidas o una mayor facilidad para realizar la tarea, para que exista un interés en cambiar al sistema informatizado.
- Sencillez en su uso: la aplicación será utilizada por una gran variedad de personas, algunas de las cuales pueden no estar muy familiarizadas con las nuevas tecnologías, por lo que debe ser fácil de utilizar. Además de la interfaz para los alumnos y profesores, también debe ser fácil de utilizar para el resto de perfiles, como es el caso de los encargados de

controlar la asistencia de los profesores y observar el buen desarrollo de las asignaturas.

- Aportar un elemento social: debido al gran interés que despiertan las redes sociales, es una buena idea integrar alguna función social para atraer a los usuarios, ya que, aunque sea un sistema que podría imponérseles, es importante agradecerles para conseguir un mejor despliegue; muchos proyectos de este tipo acaban fracasando por no tener un buen apoyo inicial, produciéndose un reclamo por parte de los usuarios para volver al sistema tradicional. Sin embargo, este aspecto terminó ocupando un segundo plano a favor de otras funcionalidades, pero originó el resultado de una de las funcionalidades añadidas a la simple recolección de datos sobre la asistencia, como se describirá en más detalle posteriormente.
- Demostrar la capacidad de *HTML5*: aunque el objetivo de desarrollar una aplicación universal y la situación actual han sido algunas de las causas que han llevado al empleo de *HTML5*, esta decisión se ha convertido además en el objetivo de demostrar sus posibilidades de futuro. Más allá de la simple creación de un programa para controlar la asistencia, este trabajo pretende también concienciar sobre las posibilidades de *HTML5* —junto a *CSS3* y *JavaScript*— y las ventajas que supone crear una única aplicación, frente a la mayor complejidad de tener que crear un programa diferente para cada sistema operativo.

Capítulo 3

Tecnologías utilizadas

En esta sección se describen las tecnologías empleadas para la realización de la presente aplicación.

3.1. Códigos *QR*

Los códigos *QR* (*Quick Response*), también denominados códigos bidimensionales, son imágenes en blanco y negro que cifran una cadena de caracteres alfanuméricos, de forma similar a los códigos de barras pero de dos dimensiones —vertical y horizontal—, permitiendo almacenar más información.

Fueron inventados en 1994 por un empleado de la empresa automovilística *Toyota Group* para etiquetar las partes de un vehículo, pero se extendieron rápidamente a muchos otros campos debido a la facilidad de empleo, su rapidez, su gran versatilidad y su licencia libre que provocaron su estandarización, siendo actualmente una tecnología muy utilizada.

En comparación con los códigos de barras formados por líneas negras verticales que se escanean con un haz de luz, los códigos *QR* son una matriz cuadrada con fondo blanco sobre la que se representan pequeños cuadrados negros que se leen a través de una cámara. La imagen consta de tres grandes cuadros negros fijos en tres de sus esquinas y un último cuadro de menor tamaño cercano a la cuarta esquina, utilizados para normalizar el tamaño de la imagen, su orientación y su ángulo de visión. El resto del espacio queda reservado para codificar la información y para el código de corrección de errores. Los códigos *QR* utilizan cuatro modos estandarizados de codificación: numérica, alfanumérica, binaria y kanji (caracteres japoneses).

Dentro de la aplicación de control de asistencia, tiene como función aportar una alternativa o complemento a la localización en interiores facilitada por el terminal desde el que se registre el usuario (gracias al *GPS* o a la red), facilitando la introducción de un código asignado a una determinada lección de forma rápida, sin necesidad de escribir manualmente la clave.

3.2. *HTML5*

HTML (HyperText Markup Language) [1] es un lenguaje de marcado, utilizado para la creación de páginas web.

Los documentos *HTML* comienzan con una *declaración* para indicar la versión de *HTML* en la que se ha escrito la página. Este lenguaje está formado por *elementos* que indican el tipo de contenido que presentan entre su apertura y su cierre, pudiendo anidarse unos elementos dentro de otros. El primer elemento, escrito después de la declaración, es la etiqueta *html* a la que se anidan directamente una etiqueta *head* y una etiqueta *body*.

- El elemento *head* contiene, principalmente, información general de la página como puede ser el título o el icono, además de otros elementos, incluyendo generalmente referencias a los *scripts* o código *JavaScript* utilizado en la página y a las *hojas de estilo* para definir la apariencia de la misma —aunque estos dos elementos pueden estar directamente incluidos en el documento *HTML*, se suelen escribir en documentos externos y limitarse a indicar su referencia en el *HTML*—.
- El elemento *body* tiene el contenido propio de la página, lo que se muestra en la ventana del navegador. En ocasiones los *scripts* son incluidos dentro del *body*, aunque la tendencia actual es escribirlos dentro del elemento *head* para una mejor organización.

Generalmente, cada elemento se indica con una etiqueta de inicio y una etiqueta de fin —aunque algunos elementos se pueden abrir y cerrar en una sola etiqueta—, pudiendo tener contenido entre ambas etiquetas y *atributos* contenidos en la etiqueta de apertura. Los atributos están formados por un par nombre-valor, aunque algunos atributos pueden carecer de valor, y se utilizan para definir características de ese elemento, como puede ser especificar un subtipo dentro del tipo de elemento, un enlace o la forma en la que se muestra —o en la que no se muestra, en caso de ocultarse— aunque el estilo suele definirse en un fichero a parte. Además los atributos

se pueden utilizar para identificar un elemento mediante un *id* o identificador único o una *class* o clase que pueden compartir varios elementos.

HTML5 es la quinta y última versión de *HTML*, todavía en desarrollo —actualmente no es una versión cerrada—, y es compatible con las versiones anteriores. La principal causa de su aparición es la gran evolución de la web en los últimos años y el cambio en la forma de interactuar con el usuario, pasando de páginas estáticas a páginas dinámicas.

Algunas de sus mejoras incluyen la introducción de nuevos elementos y atributos. La mayoría de los elementos nuevos se comportan igual que otros más antiguos (los elementos *span* y *div*), pero aportan un significado semántico, como es el caso de *header* referente a la cabecera de la página o *footer* que contiene el pie de la página; pero también se han creado elementos completamente nuevos como es el caso de la inclusión nativa de vídeo y audio sin la necesidad de aplicaciones externas como *flash* y el elemento *canvas* que permite pintar sobre un recuadro del tamaño que se le asigne. Entre los atributos nuevos destaca la mejora de los formularios aumentando los diferentes tipos de *inputs* o campos de entrada, añadiendo nuevos como fechas, horas o números, o el atributo *data-* que permite asignar datos a un elemento, que serán útiles para algún *script* que actúe sobre ellos.

Sin embargo, han aparecido otras mejoras muy importantes como son la posibilidad de acceder de forma directa al hardware del dispositivo sin necesidad de aplicaciones externas, por ejemplo el acceso a la cámara (permitiendo a la aplicación de este trabajo escanear el código *QR*) o al micrófono; las nuevas formas de interactuar con el servidor —los *WebSockets* que permiten al servidor comunicarse con el cliente sin que éste tenga que realizar una petición— y la posibilidad de utilizar una página sin conexión —permitiendo crear con *HTML5* aplicaciones que no requieran acceso a internet o utilizar una web con el contenido previamente descargado la última vez que se accedió con conexión—.

Para aprovechar todas las capacidades que *HTML5* aporta, se suele combinar con *CSS* y *JavaScript*, creando un ecosistema que permite desarrollar aplicaciones web complejas, variadas y dinámicas.

3.3. *JavaScript*

JavaScript [1] es un lenguaje de programación orientado a objetos utilizado principalmente en el desarrollo de aplicaciones web en el lado del cliente, aunque también es utilizado en algunos casos en el lado del servidor y en otros entornos.

A pesar del nombre, *JavaScript* no es un lenguaje similar a *Java* ni con un propósito similar, pero adoptó este nombre debido a la popularidad de *Java* cuando se desarrolló. Surgieron diferentes versiones similares a *JavaScript* hasta que finalmente se propuso como estándar *ECMAScript*, para evitar incompatibilidades según el navegador utilizado.

Centrándose en el lado del cliente en páginas web, que es el ámbito que afecta a este trabajo, la función principal de estos *scripts* es la modificación de lo mostrado en la ventana del navegador —ocultar elementos, mostrar elementos que estén ocultos o mostrar nuevos elementos generados o solicitados—. Para poder modificarlo se desarrolló y estandarizó una interfaz denominada *árbol DOM* (*Document Object Model*), que permite representar y manipular tanto documentos *HTML* como documentos *XML* (otro lenguaje de marcado). El conjunto de elementos del documento *HTML* forman el *árbol DOM*, con una estructura similar a un árbol genealógico, en el cual la raíz es el documento *HTML* del que cuelga el elemento *html* que generalmente tiene como hijos al elemento *head* y al elemento *body* y se va extendiendo con los descendientes (o elementos anidados) de ambos.

JavaScript es un lenguaje bastante orientado a la gestión de eventos. Sobre algunos elementos o variables, por ejemplo elementos *HTML*, se crea un *listener* o manejador de eventos, por ejemplo se programa el código *JavaScript* de tal manera que si se realiza un *click* de ratón sobre cierto elemento se dispara el evento *click* sobre dicho elemento —el lanzamiento del evento está integrado de forma nativa por el navegador— y el manejador captura el evento y realiza la función que se le hubiese encomendado o programado. Hay ciertos elementos *HTML*, como es el caso del elemento *a*, que tienen una función por defecto nativa en el navegador; en el caso del elemento *a* es la función de solicitar la página a la que enlaza dicho elemento, con un manejador de eventos que se programe con *JavaScript* se puede evitar ese comportamiento por defecto y/o añadir un nuevo comportamiento.

3.3.1. *AJAX (Asynchronous JavaScript And XML)*

Inicialmente la modificación del *árbol DOM* con *JavaScript* no añadía información que no existiese en el documento *HTML* o en el *script* de *JavaScript* en el momento de su descarga o información introducida por el usuario, por ejemplo a través de un formulario. Sin embargo, posteriormente la necesidad de una web más dinámica dio lugar a la aparición de la tecnología *AJAX (Asynchronous JavaScript And XML)* que permite realizar peticiones al servidor y recibir una respuesta y mostrarla sin necesidad de recargar la página entera, añadiendo con *JavaScript* en el *árbol DOM* la información recibida, la cual puede ser previamente procesada.

AJAX no es una tecnología basada exclusivamente en *JavaScript*, sino que también se basa en *HTML* y *CSS* para mostrar el contenido, para servir de interfaz al usuario para solicitar las interacciones *AJAX* o simplemente para mostrar al usuario que se está procesando una solicitud.

Esta tecnología permite el diseño de páginas web más eficientes al poder solicitar solo la información requerida, aprovechando los datos previamente descargados, y también permite crear aplicaciones más fluidas y dinámicas de cara al usuario final, logrando simular el funcionamiento de programas instalados en un dispositivo a través de páginas web cargadas en un navegador, evitando parpadeos en la información mostrada (al cambiar innecesariamente todo el contenido de una página por el de otra prácticamente igual) y eliminando la sensación de estar esperando a que la página se descargue, ya que, mientras se descargan y se procesan los datos solicitados, la página sigue mostrándose y según el caso puede seguir siendo funcional —ya que las peticiones al servidor se realizan en segundo plano y de forma transparente para el usuario—.

3.3.2. *jQuery*

Una de las bibliotecas de *JavaScript* más utilizadas es *jQuery* [2]. Esta biblioteca facilita en gran medida el acceso y la manipulación del *árbol DOM*, el manejo de eventos y la modificación de estilos o del *CSS*, así como la implementación de funciones muy útiles y facilitar la realización y procesamiento de peticiones *AJAX*. Una de las causas de su popularidad es la licencia de software libre que permite utilizarla tanto para proyectos libres como proyectos privados.

Además de las ventajas debidas a la mayor abstracción y facilidad, ahorrando tiempo y espacio (menos líneas de código que implican menos datos que almacenar y menos información que transmitir por la red cuando se solicite el *script*), es muy importante el gran soporte y apoyo

por parte de la comunidad, encontrándose mucha información y ayuda para resolver problemas relacionados con el desarrollo de páginas web basadas en *jQuery*, y la gran cantidad de *plugins* o funcionalidades añadidas de software libre que hay desarrolladas, pudiendo reciclar el trabajo de otros usuarios para ahorrar tiempo y realizar funciones complejas que de otra forma es muy probable que no se llegasen a incorporar en un proyecto en el que no sea esencial dicha funcionalidad o se acabarían implementando de una manera más rudimentaria.

La función principal de *jQuery* es `$()`, un alias de `jQuery()`, a la que se le pasa como parámetro una regla *CSS* o el nombre de una etiqueta *HTML*, seleccionando todos los elementos que concuerdan con la expresión. Sobre estos elementos seleccionados se puede aplicar una función de *jQuery*, como puede ser eliminarlos, modificar su estilo o programar un manejador de eventos entre otras.

Tradicionalmente, los *scripts* se añadían al final de un documento *HTML* para que no se ejecutasen antes de que el documento *HTML* se hubiese cargado por completo, sin embargo, posteriormente se añadió un evento *ready* que se dispara cuando el elemento *HTML* termina de cargar. En el caso de *jQuery* se puede manejar este evento mediante la función *ready* sobre el elemento *document* —`$(document).ready(nombreDeLaFuncion)`— o de una forma más abreviada, pasándole la función que se debe ejecutar cuando se cargue la página a la función `$()`.

3.3.3. JSON

JSON o *JavaScript Object Notation* [3] es un formato ligero para el intercambio de datos. Es fácil de leer y de escribir por humanos, y fácil de generar y *parsear* (leer) para un ordenador.

Un documento *JSON* contiene datos que pueden ser convertidos a objetos o variables en distintos lenguajes de programación, al igual que existen multitud de bibliotecas para realizar la operación inversa.

Es un formato muy utilizado en la actualidad para enviar información desde un servidor a un cliente ante una petición *AJAX*, siendo dicha información interpretada por código *JavaScript* del cliente, permitiéndole trabajar con esos datos como si se tratase de cualquier otra variable.

3.4. CSS3

CSS (Cascading Style Sheets) u hojas de estilo en cascada [4] es el lenguaje utilizado para describir la presentación de documentos *HTML* o *XML*, aunque también es utilizado para modificar la interfaz de usuario de algunos programas.

El *CSS* para un documento *HTML* puede ser incluido dentro de la etiqueta de apertura de un elemento, mediante el atributo *style* indicando como valor el estilo de dicho elemento. Otra opción mejor, consiste en poner el estilo de todo el documento entre las etiquetas de apertura y cierre de un elemento *style* dentro del propio *HTML*, para una mejor lectura y que el contenido del *HTML* sea más claro, reduciendo el tamaño de las etiquetas de los elementos y evitando repetirlo para cada uno de los elementos que compartan características. Sin embargo, la mejor opción es poner esa información en un documento externo al *HTML*, en un documento *CSS*, y referenciarlo desde la cabecera del documento *HTML* con un elemento *link*, con el atributo *rel* con valor *stylesheet* y el atributo *href* con el valor de la *URL* del fichero *CSS*.

Un documento *CSS* consta de una lista de reglas que definen las propiedades de estilo de la página. Cada regla o conjunto de reglas consiste en uno o más *selectores* y un *bloque de declaración* que contiene los estilos a aplicar para los elementos del documento que cumplan con el selector que les precede. Cada bloque de estilos se define entre llaves, y está formado por una o varias declaraciones de estilo con el formato `propiedad:valor;`. Los selectores permiten especificar los elementos según su etiqueta *HTML*, su *id* precedido del carácter '#', su *clase* precedido de un punto, o mediante pseudo-classes precedidas del carácter ':' —como por ejemplo su posición dentro del documento (primero, segundo, impar...) o su estado (seleccionado, con el ratón encima del elemento...)—; además, es posible especificar que el elemento debe estar dentro de algún otro elemento. Algunas de las propiedades que se pueden aplicar a un elemento son su tamaño, su borde, su fondo, su posición en la pantalla...

Además de las propiedades especificadas por el diseñador de la página web, se deben tener en cuenta las hojas de estilo que tiene el navegador y las especificadas por el usuario. Se debe tener en cuenta que un elemento hereda las propiedades de estilo de elementos superiores, elementos dentro de los cuales está anidado, a no ser que se especifiquen otras que las sobrescriban. El nombre de cascada de *CSS* tiene que ver con como se interpretan las propiedades de estilo de un elemento en caso de que haya varias reglas que se superpongan, teniendo preferencia según

su importancia indicada explícitamente, su especificidad (cómo de concreto es el selector) y en caso de igualdad por el orden (prevaleciendo la última que aparezca).

CSS3 [1] es la tercera versión de *CSS*, la cual se encuentra todavía en desarrollo, al igual que el estándar *HTML5*, permitiendo que evolucionen juntas para una mejor integración y conseguir un buen desarrollo de las novedades que se desean integrar en ambos estándares, complementándose entre ambas junto con *JavaScript*. *CSS3* está dividido en varios documentos separados, denominados *módulos*. Cada módulo añade nuevas funcionalidades a las definidas por *CSS2*, preservándose las anteriores para mantener la compatibilidad. Un inconveniente de trabajar con *CSS3* es que al no ser un estándar cerrado y tener muchos módulos en desarrollo, los distintos navegadores van adaptando sus funcionalidades a ritmos diferentes y se debe revisar la compatibilidad de las nuevas funciones e incluso añadir propiedades distintas para cada navegador para una misma funcionalidad.

3.5. *Bootstrap*

Bootstrap [5] es un *framework* o conjunto de herramientas de software libre para desarrollar aplicaciones web, aportando, principalmente, funcionalidades para diseñar el estilo de una página web haciendo uso de *HTML*, *CSS* y *JavaScript*. Fue creado por trabajadores de *Twitter* (una famosa red social) para lograr una homogeneidad en el estilo de las páginas creadas por la empresa, aunque fue publicada de forma libre y su uso se ha popularizado, siendo actualmente una herramienta muy utilizada y el proyecto más popular de *GitHub*, página para alojar proyectos utilizando el sistema de control de versiones *Git*.

Bootstrap sigue una filosofía *mobile first*, dando prioridad a la navegación y visualización de las páginas web, desarrolladas con este *framework*, en dispositivos móviles. Sin embargo, esta herramienta permite desarrollar simultáneamente aplicaciones web con una interfaz adecuada para dispositivos de diversos tamaños de forma sencilla.

Una de las principales características de *Bootstrap* es la existencia de clases *HTML* que asignadas a un elemento permiten indicar el ancho de la pantalla que ocuparán, pudiendo asignar diferentes anchuras según el tamaño de la pantalla en la que se visualiza, añadiendo varias clases al elemento. Para ello, *Bootstrap* divide la pantalla en 12 columnas, anchura que se puede repartir entre distintos elementos según se desee, sin ser necesario ocupar una fila entera. De

esta forma, por ejemplo, si se tienen cuatro párrafos, se puede decidir que en pantallas grandes cada uno ocupe 3 columnas y estén en la misma fila, uno al lado de otro, en pantallas medianas como puede ser la de un *tablet* que haya dos párrafos por fila (una anchura de 6 columnas por párrafo) y en un móvil que cada párrafo ocupe todo el ancho colocándose uno debajo de otro (una anchura de 12 columnas), lo cual se puede realizar de una manera tan sencilla como añadir a cada párrafo la clase `col-xs-12 col-md-6 col-lg-3`, siendo *xs* pantallas muy pequeñas, *md* medianas y *lg* grandes, existiendo también *sm* pequeñas; y en este caso para las pequeñas se aplicaría la misma anchura que para las *xs*, ya que se aplica la regla de la clase definida con mayor tamaño que sea menor que el tamaño del dispositivo (ver Figura 3.1).

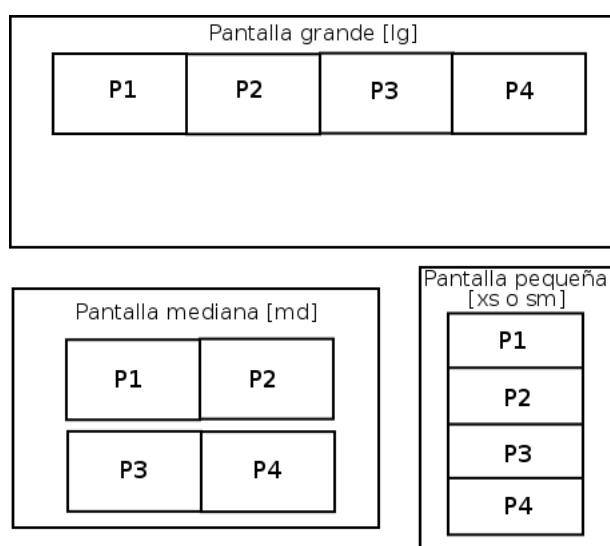


Figura 3.1: Ejemplo de distribución de 4 párrafos con la clase `col-xs-12 col-md-6 col-lg-3` con *Bootstrap* según el tamaño de la pantalla

Otras características muy utilizadas son sus iconos, el diseño de los botones y el de los formularios y las barras de navegación (las cuales pueden plegarse en dispositivos pequeños y tener menús desplegables gracias al empleo de *JavaScript*).

Una ventaja, como en la mayoría de tecnologías previamente citadas, es que su popularidad es la causa de una gran comunidad apoyando la plataforma y realizando nuevas funcionalidades de software libre basadas en *Bootstrap*. Un buen ejemplo es la página <http://bootsnipp.com/> donde los usuarios diseñan diferentes estructuras —como pueden ser elementos tan dispares como un perfil de usuario, un carro de la compra de una tienda por internet o una lupa para ampliar secciones de una imagen— ofreciéndolas en su mayoría de forma gratuita y siendo muy fáciles

de integrar en cualquier proyecto que utilice *Bootstrap*.

3.6. *Python y Django*

3.6.1. *Python*

Python [6] es un lenguaje de programación que, según sus desarrolladores, destaca por ser potente, rápido, comportarse bien con otros lenguajes, ser compatible con los sistemas operativos más importantes, fácil de aprender y abierto.

Es el lenguaje de programación que se ha utilizado en el desarrollo de este trabajo, en el lado del servidor, para procesar la información y las peticiones de los usuarios y enviarles dichos datos con una determinada presentación.

Es un lenguaje de propósito general (para el desarrollo de programas de diversos ámbitos), en gran medida debido a que es de programación multiparadigma, es decir, permite programación orientada a objetos, programación imperativa y programación funcional, otorgándole gran versatilidad.

Está diseñado para tener una sintaxis muy limpia, que desemboque en un código legible. Para ello, emplea una sintaxis muy estricta (es un lenguaje fuertemente tipado), con distinción de mayúsculas y minúsculas y una indentación —empleo de tabuladores o espacios al principio de una línea, aunque tiene por norma de estilo el uso de espacios— obligatoria en la construcción de los diferentes bloques o estructuras. Las normas de indentación, además de una mejor lectura, permiten potenciar la filosofía de simpleza que defiende el lenguaje, ya que a medida que se va complicando el código, van aumentando las estructuras anidadas, implicando una mayor tabulación y, por tanto, la disminución de espacio en la línea para escribir el código, obligando al programador a hacerlo de una forma más sencilla; característica que ha recibido buenas y malas críticas.

Python es además un lenguaje de alto nivel, muy abstracto, lo cual lo hace fácil de aprender y utilizar y permite generar *scripts* rápidamente y con menos líneas de código. Integra un sistema de tipado dinámico (capaz de deducir el tipo de un dato sin tener que especificarlo explícitamente) y un control automático de memoria (evitando al programador tener que preocuparse por la asignación y liberación de memoria). Otro añadido importante es la resolución dinámica

de nombres, realizada en tiempo de ejecución.

3.6.2. *Django*

Django [7] es un *framework* o conjunto de herramientas para desarrollo web, de código abierto y escrito en su totalidad en *Python*. Fue desarrollado inicialmente para gestionar páginas de noticias de la *World Company* de Lawrence, pero en 2005 fue liberada al público bajo una *licencia BSD*.

Se basa en el *paradigma Modelo-Vista-Controlador*, consistente en separar los datos y la *lógica de negocio* de una aplicación (modelo), de la interfaz de usuario (vista) y del módulo encargado de gestionar los eventos y las comunicaciones (controlador).

Su filosofía general consiste en la creación de sitios web complejos de forma rápida, facilitar la reutilización de elementos —realizando módulos independientes que no necesiten conocer la existencia del resto, en la medida de lo posible—, disminuir la cantidad de código, dar prioridad a lo explícito frente a lo implícito (cualidad heredada de *Python*), ser consistente en todos los niveles y evitar repetir código o información, siendo una de sus principales citas ‘*Don’t repeat yourself (DRY)*’ o ‘no te repitas’ en español.

Además de los objetivos globales, *Django* puede dividirse en cinco grandes bloques para una mejor comprensión:

- Los *models* o modelos: es la definición de los elementos que se almacenan en la base de datos; generalmente, cada modelo se mapea o equivale a una tabla en la base de datos. De nuevo, lo explícito prima sobre lo implícito, sin asumir comportamientos en función de los nombres de los campos de un modelo, siendo necesario indicar el tipo de dato que almacenarán. Los modelos deberían incluir toda la lógica sobre ellos mismos, desde el tipo de datos que almacenará hasta los campos para ser interpretados por los humanos, para una mejor comprensión de un determinado modelo u objeto y una mayor organización.
- La *API* de la base de datos: *Django* aporta abstracciones para interactuar con la base de datos de una forma sencilla. Sus objetivos principales consisten en lograr eficiencia en el acceso a la base de datos, intentando reducir el número de accesos a la misma, una sintaxis rica y concisa, que permita declaraciones expresivas, y permitir el acceso desde un objeto a otro relacionado en ambos sentidos, pero permitiendo además la posibilidad

de realizar declaraciones de más bajo nivel para acceder a la base de datos, ofreciendo así una gran versatilidad.

- Las *URLs*: es una parte muy importante en el desarrollo de una aplicación en *Django*, es la interfaz entre el cliente y el servidor, son las referencias que permiten al cliente solicitar, crear o modificar información. Su filosofía defiende desemparejar las *URLs* de las funciones para una mayor reutilización, ya que diferentes sitios web podrían tener diferentes *URLs* para la misma función. Además deben ser flexibles, permitiendo cualquier *URLs* concebible y permitir que tengan una estética cuidada, sin obligar al empleo de caracteres o características específicas.
- Las *templates* o plantillas: *Django* presenta un sistema para *renderizar* o generar dinámicamente la respuesta a una petición a partir de unas plantillas previamente creadas por el desarrollador y un contexto, que puede variar según determinadas circunstancias como pueden ser la *URL* de la petición, la información disponible en la base de datos o el usuario que realiza la petición. Este sistema pretende separar la lógica de la presentación (introduciendo esta última en la propia plantilla). También busca evitar redundancia, basándose en la premisa de que la mayoría de páginas web dinámicas comparten una gran parte del diseño, como puede ser la cabecera, la barra de navegación o el pie de página. Las plantillas no pretenden inventarse un nuevo lenguaje de programación por lo que solo tienen funcionalidades básicas, pensando además de la separación con la lógica, en la posibilidad de ser creadas por diseñadores en lugar de programadores; y otra razón es la de evitar la inclusión de código malicioso, el cual es a su vez otro objetivo de los desarrolladores. Este sistema presenta además otras características, pero son menos relevantes.
- Las *views* o vistas: son funciones *Python* que reciben una petición web y devuelven una respuesta web. Esta respuesta puede ser un *HTML*, una redirección, una imagen... La vista contiene la lógica para devolver esa respuesta. Se busca la simplicidad, que la creación de una vista sea igual de fácil que escribir una función de *Python*. Las vistas deben tener acceso a un *objeto petición* que contenga la información de la solicitud actual; para ello siempre reciben este objeto como parámetro, en lugar de tener que acceder a dicha información desde una variable global. Además, debe ser capaz de poder diferenciar pe-

cciones *GET* y *POST*, ya que son diferentes y, por tanto, deben ser gestionadas de forma diferente —generalmente una petición *GET* es realizada para solicitar un recurso y una petición *POST* para crearlo o modificarlo—.

3.7. *Pylint*

Pylint [8] es una herramienta para la depuración de código escrito con el lenguaje de programación *Python*, para mejorar tanto su presentación como su rendimiento.

Las comprobaciones que realiza *Pylint* son las listadas a continuación:

- El cumplimiento de las normas de estilo definidas en el *PEP 8* [9], documento que recoge las reglas de estilo del lenguaje *Python*, entre las que se encuentran la longitud de una línea de código, el nombre de las variables o el empleo de cuatro espacios para cada nivel de indentación.
- Detección de errores. Comprueba que se realice la importación de los distintos módulos utilizados o que se esté accediendo a un miembro dentro de una variable que no contiene dicho elemento, entre otros errores.
- Mejorar la eficiencia y reducir la complejidad del código. *Pylint* alerta de situaciones como son, por ejemplo, el paso de demasiados parámetros a una función o el empleo de muchas ramificaciones dentro de una misma función, que dificultan la interpretación del código tanto por un humano como por una máquina.
- Código redundante. Detecta variables a las que se les asigna un valor pero nunca se las referencia, código repetido, la importación de módulos no utilizados....

La siguiente *URL*, <http://pylint-messages.wikidot.com/all-messages>, dirige a una página con todos los posibles mensajes que puede mostrar *Pylint*.

3.8. *Git*

Git [10] es un *sistema de control de versiones* libre y abierto, creado por Linus Torvalds. Está diseñado para controlar desde pequeños hasta grandes proyectos de forma rápida y eficiente.

Un sistema de control de versiones es un programa para la gestión de los cambios en un proyecto. *Git* permite registrar los cambios que se van realizando, asignándoles una breve descripción para saber que incluye cada cambio, para poder llevar un seguimiento de la evolución del programa pero también para saber el estado de cada versión por si se quiere retroceder a una versión anterior, ya que esta herramienta permite además cambiar de una versión a otra —tener guardadas por separado cada una de las versiones implicaría mayor tiempo al tener que copiar todo el proyecto y un mayor espacio ocupado, ya que *Git* lo optimiza guardando solo las diferencias entre una versión y otra—.

Otra característica importante es la posibilidad de crear ramificaciones y trabajar en cada una de ellas sin afectar al resto, permitiendo estar trabajando con varios aspectos complejos de un programa de forma simultánea y completamente aislada, o dando la posibilidad de dividir un proyecto para desarrollar partes específicas para un determinado servicio o un determinado cliente, e incluso facilitar el desarrollo de proyectos en los que trabajen varias personas.

Capítulo 4

Implementación

En esta sección se realiza una descripción más específica de la aplicación resultante de este trabajo.

4.1. Interacción entre los diferentes subsistemas

El funcionamiento del sistema se basa en la interacción de varias tecnologías y subsistemas. Es importante distinguir en primer lugar entre el cliente y el servidor.

El cliente funciona gracias a un navegador de internet, el cual muestra la información e interfaz al usuario final, para que pueda utilizar la aplicación. Para ello, el navegador procesa ficheros *CSS*, *JavaScript* y *HTML*, además de otros elementos como imágenes, todos ellos enviados por el servidor. Para obtener los documentos citados, el navegador del cliente realiza peticiones al servidor provocadas por la interacción del usuario —por ejemplo al pulsar sobre un enlace o al escribir una *URL* en la barra del navegador— o debidas a referencias dentro de un documento *HTML* recibido en una petición anterior (ver Figura 4.1).

Las peticiones del cliente son procesadas por *Django* en el lado del servidor. *Django* comprueba primero la *URL* sobre la que se realiza la petición y la localiza en un fichero (`urls.py`) que le indica qué función debe procesar la petición, conociendo la función cómo se debe procesar dicha petición y qué respuesta se debe enviar al cliente (ver Figura 4.2). Las peticiones del cliente pueden ser para solicitar algún cambio en la base de datos —mediante una petición de tipo *POST*— o simplemente para obtener información (enviada al cliente mediante un documento *HTML* o fichero *JSON*) o algún fichero —mediante una petición de tipo *GET*—; siendo

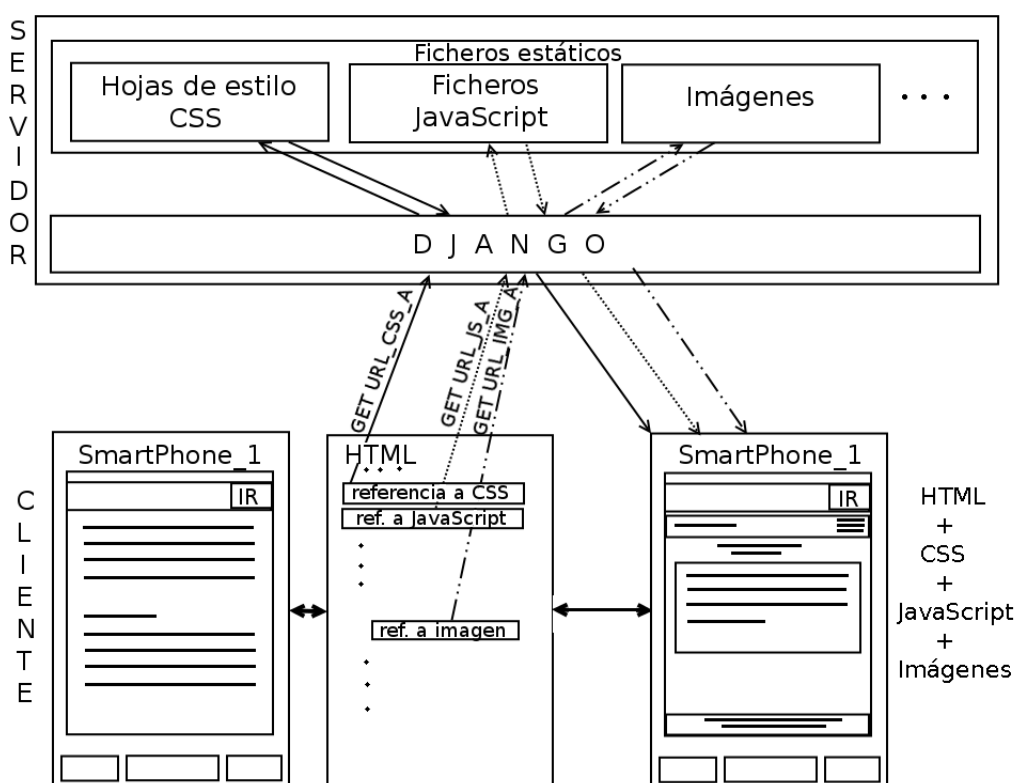


Figura 4.1: Proceso posterior a la recepción de un documento HTML con elementos referenciados, mostrando el resultado en la pantalla de un móvil

Django es el encargado de manejar la base de datos, tanto para leer como para escribir en ella.

Entre los documentos servidos se distinguen los estáticos de los generados dinámicamente. Para el caso de los documentos dinámicos, ficheros *HTML* y *JSON*, *Django* utiliza la información de la petición y la información de la base de datos para *renderizar* o rellenar unas plantillas previamente creadas (excepto para algunas respuestas con *JSON* que no requieren el uso de plantillas).

Todos los documentos están almacenados en el servidor o son generados en el mismo, y al ser enviados al cliente son procesados por el navegador de forma diferente según el tipo de respuesta, o más concretamente según el tipo de los datos recibidos. De esta forma, en lo más básico se encuentran los documentos *HTML* que contienen la información a mostrar con una estructura básica — generados con plantillas, algunas creadas específicamente para la aplicación y otras genéricas de *Django*—; después se encuentran las hojas de estilo *CSS* que indican cómo se debe mostrar la información, aportando una mejor apariencia y una interfaz más fácil de uti-

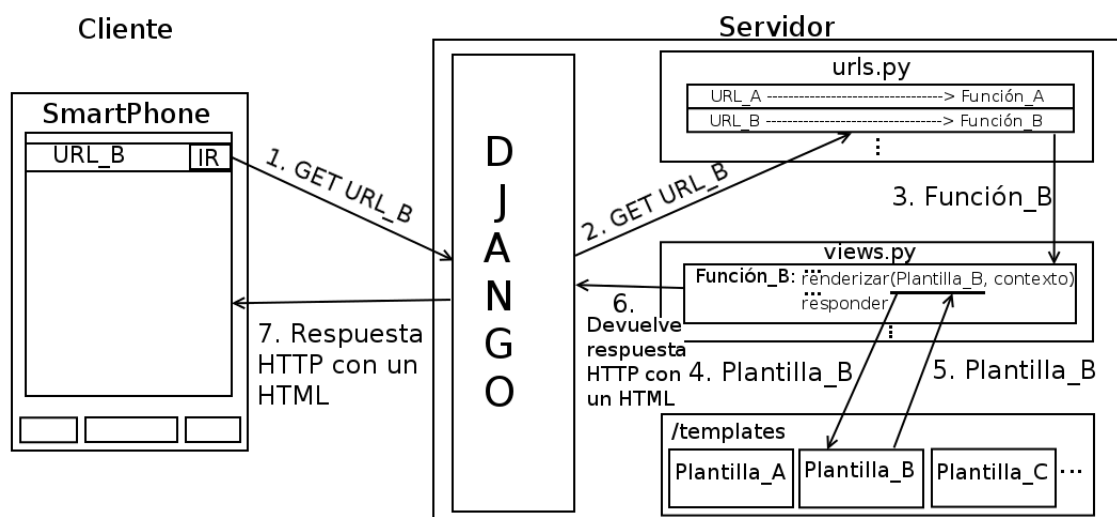


Figura 4.2: Proceso de solicitud de una página sin *AJAX* (mostrado de forma simplificada)

lizar; los ficheros *JavaScript* enriqueciendo la interacción con el usuario, permitiendo páginas más dinámicas; y los ficheros *JSON* que contienen instrucciones e información para ser manejados por los *scripts* de *JavaScript*, implicando generalmente cambios en el *árbol DOM* —o lo que es lo mismo, cambios en la página generada por el *HTML* y *CSS* recibidos en peticiones anteriores—.

4.1.1. Estructura de directorios y ficheros

En la raíz del proyecto *Django* se encuentran los documentos mostrados en la Figura 4.3 y descritos a continuación:

- `app`: contiene la funcionalidad de la aplicación *Django*. Entre los documentos se encuentran `admin.py`, para modificar el panel de administración que genera *Django* por defecto; `admin_csv.py`, encargado de procesar los archivos *CSV* para la creación de usuarios y la vinculación de usuarios con asignaturas; `forms.py`, que define los formularios que se utilizan para la creación o edición de los distintos elementos de la base de datos desde la página principal (es decir, estos formularios no se utilizan en el panel de administración); `models.py`, que define los distintos elementos que se almacenan en la base de datos, específicos de la aplicación; y `views.py`, que contiene la lógica para procesar las peticiones de los usuarios.

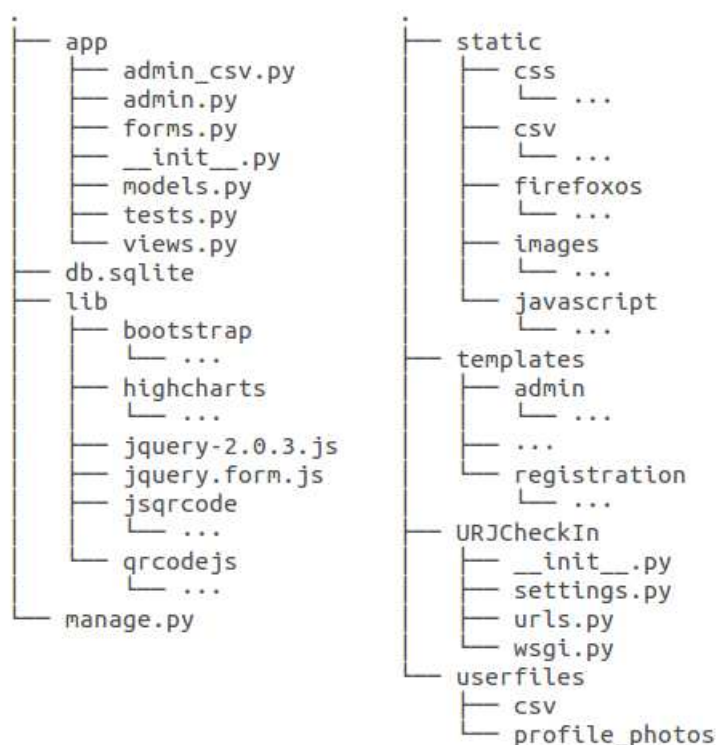


Figura 4.3: Esquema de ficheros dentro del proyecto *Django*, realizado con el programa *tree*

- `db.sqlite`: es la base de datos del proyecto.
- `lib`: contiene las bibliotecas utilizadas en el proyecto. Entre ellas se encuentran *Bootstrap* y bibliotecas de *JavaScript*.
- `static`: contiene ficheros que se sirven de forma estática y que se han creado para la aplicación desarrollada. Los documentos servidos son ficheros *JavaScript* para añadir funcionalidad en el lado del cliente, hojas de estilo *CSS* para definir el diseño de las páginas web, documentos *CSV* de ejemplo, imágenes y un documento para la creación de la aplicación de *FirefoxOS*.
- `templates`: contiene las plantillas para generar los documentos *HTML* que se envían al cliente.
- `URJCheckIn`: contiene ficheros necesarios para el funcionamiento del proyecto *Django*. Entre ellos se encuentran `settings.py` que define la configuración del proyecto *Django* y `urls.py` que define las distintas *URLs* y las vincula con las *vistas* o funciones

encargadas de procesar las peticiones de los clientes.

- userfiles: en este directorio se almacenan los ficheros que envían los usuarios. Cada usuario puede tener una imagen guardada para su perfil, en caso de enviar una nueva se reemplaza. Además, almacena temporalmente los ficheros CSV mientras se procesan (una vez procesados son eliminados).

4.2. Diseño de la base de datos

La Figura 4.4 representa las distintas tablas de la base de datos de la aplicación desarrollada y las relaciones entre ellas, habiendo sido obtenido dicho esquema mediante una extensión de *Django*, *Graph models*.

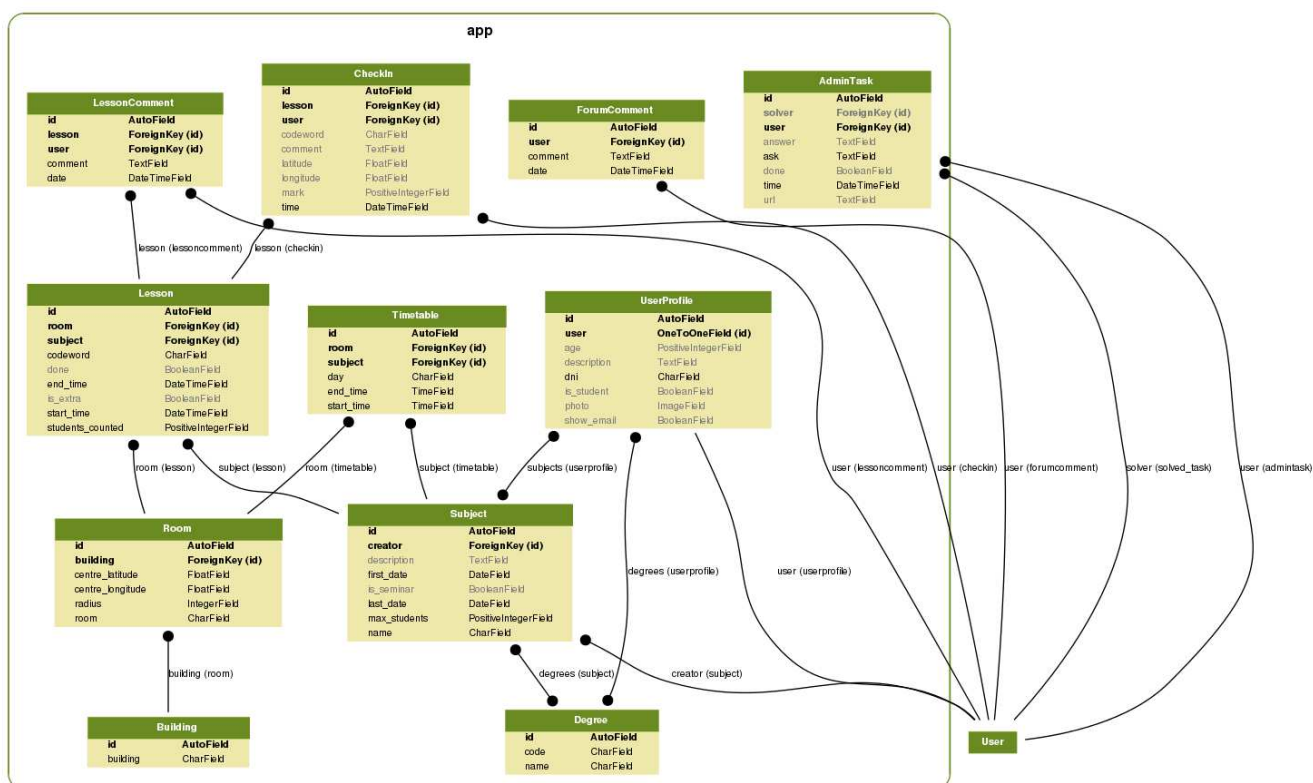


Figura 4.4: Esquema de la base de datos de la aplicación

4.2.1. *Models o modelos de Django*

En una aplicación de un proyecto *Django* existe un documento denominado `models.py` que contiene la descripción de los distintos elementos u objetos que almacenará la base de datos.

La aplicación desarrollada tiene los siguientes modelos:

- **Grado:** representa un grado universitario. Contiene el nombre del grado y un código identificativo para resumir el nombre del grado. Ambos campos contienen un *string* (cadena de caracteres) y deben ser únicos.
- **Asignatura:** representa una asignatura o un seminario; no se diferencia entre ambos debido a que su estructura y la forma de interactuar con ambos es prácticamente la misma. Contiene el nombre de la asignatura, la fecha de inicio de la asignatura y la fecha de finalización de la asignatura, un *booleano* para indicar si es un seminario, el máximo número de estudiantes que puede haber matriculados en dicha asignatura, una breve descripción opcional, una referencia al grado o grados a los que pertenece la asignatura y una referencia al usuario que creó la asignatura.

Contiene además métodos internos para obtener el número de estudiantes matriculados, el porcentaje de asistencia de los profesores, el porcentaje de asistencia de los alumnos, la valoración media de la asignatura y el estado de la asignatura (actual, antigua o futura, según las fechas de inicio y de finalización).

En cuanto al porcentaje de asistencia de los profesores, se tiene en cuenta la relación del número de clases impartidas entre el número de clases obligatorias anteriores al momento actual, por lo que el porcentaje de asistencia puede ser mayor que el 100 % si el profesor imparte más clases que las estrictamente obligatorias —se realiza de esta manera para que un profesor pueda recuperar una clase obligatoria que no pudo impartir, generando el mismo una clase extra posterior, además de permitir observar a los profesores que se esfuerzan y trabajan más de lo obligatorio, superando el 100 % de la asistencia obligatoria—.

- **Edificio:** representa un edificio. Contiene el nombre del edificio, el cual no puede repetirse.

- **Aula:** representa un aula. Contiene el nombre del aula, una referencia al edificio en el que se encuentra, la longitud y latitud del centro del aula y el radio de la misma (considerándola redonda). El par compuesto por el nombre del aula y la referencia a un edificio debe ser único, es decir, no pueden existir dos aulas con el mismo nombre dentro de un mismo edificio.
- **Perfil de usuario:** complementa al modelo usuario creado por *Django*.

Un usuario de *Django* contiene, entre los campos que interesan para esta explicación, un nombre de usuario, el nombre real del usuario, un campo para su apellido o apellidos y su dirección de correo electrónico. El modelo de perfil de usuario creado debe referenciar a un usuario, pero no puede existir más de un perfil de usuario que referencie al mismo usuario —lo cual no implica que todos los usuarios deban tener un perfil de usuario—.

El perfil de usuario contiene, además de la referencia al usuario que extiende, una imagen de perfil, una descripción opcional, referencias a las asignaturas en las que está matriculado o que imparte, referencias a los grados en los que está matriculado o en los que enseña, un *booleano* para indicar si el usuario es un estudiante (si está como *verdadero*, en caso contrario será tratado como un profesor; por defecto se crea como alumno), la edad del usuario, su *DNI* o documento identificativo que no puede repetirse, y un *booleano* para indicar si se debe mostrar su dirección de correo electrónico públicamente —se deberá tener en cuenta la ley de protección de datos—.

- **Clase:** representa una lección. Contiene la fecha y hora de inicio y la de finalización, una referencia a la asignatura a la que corresponde la clase, una referencia al aula en la que se realiza la clase, un campo *booleano* para indicar si la clase es extra (si no se considera obligatoria), un *booleano* para registrar que la clase se ha impartido (es posible obtener esta información de forma implícita, pero es más eficiente almacenarla de forma explícita), un código de caracteres aleatorio (que se utiliza para demostrar que un alumno o profesor ha asistido a la clase), y el número de estudiantes que el profesor contó durante la clase.

Al crear una clase se comprueba, además de que la fecha de inicio sea menor que la de finalización y que ambas estén dentro del mismo día, que la clase no se solape con otra de la misma asignatura y que no se solape con otra clase en el mismo aula. En caso de

no cumplirse alguna de estas condiciones se impide la creación de la clase y se indica el error.

Contiene además métodos internos definidos para calcular el número de estudiantes que asistieron a la clase, el porcentaje de estudiantes matriculados que asistieron, y la valoración media de la clase.

- **Tarea de administración:** representa una petición de un usuario a los administradores, generalmente para solicitar la realización de una operación que no tiene permitido realizar el peticionario, debido a un error como puede ser el caso de que un profesor tenga un perfil de alumno, o para reportar un fallo en la aplicación.

Contiene una referencia al usuario que realizó la petición, el mensaje con la petición, el momento en el que se realizó la petición, un *booleano* para indicar si la tarea se ha gestionado —este campo está inicialmente como falso y cuando un administrador atienda el problema debe marcarlo como verdadero—, una referencia al usuario del administrador que gestione la tarea y un campo para que el administrador pueda responder al peticionario en caso de que sea necesario debido al tipo de petición.

- **Check in:** representa el registro de asistencia de un usuario a una clase. Contiene una referencia al usuario que realizó el registro, una referencia a la clase en la que realiza el *check in*, la puntuación otorgada a la clase, una opinión sobre la clase, la longitud y latitud en la que se realizó el registro, la palabra código relativa a la clase y la fecha y la hora en la que se generó.

Al guardarse comprueba que se guarda con la localización y/o el código, y en caso de guardarse con el código que se corresponda con el de la clase. En caso de no cumplirse alguna de estas condiciones no se guarda el *check in*, informando de la causa. En caso de guardarse, si el usuario que ha realizado el registro es un profesor, se activará el campo de la clase que indica si se ha realizado.

- **Comentario en clase:** representa un comentario realizado sobre una clase, el cual no debe confundirse con el comentario valorativo que se realiza al efectuar un *check in*. Está pensado para que los profesores puedan informar sobre algún aspecto de la clase, como puede ser un recordatorio sobre la necesidad de llevar determinado material a la

clase o para poner algún enlace de interés, o cualquier otro dato relevante sobre la clase.

Contiene una referencia al usuario que escribió el comentario, una referencia a la clase a la que está vinculado, la fecha y hora de su publicación y el propio contenido del comentario.

- **Comentario del foro:** representa un comentario publicado en el foro de la página web. Contiene una referencia al usuario que escribió el comentario, la fecha y hora de su publicación y el propio contenido del comentario.
- **Horario:** representa el horario de una asignatura. Contiene una referencia a la asignatura correspondiente, el día de la semana en el que se imparten las clases —una asignatura puede tener varios horarios, cada horario declara las clases en un solo día de la semana—, la hora de inicio y de finalización de las clases, y el aula en el que se realizan (aunque no asegura que dichas clases se vayan a realizar en ese aula).

Al guardarse comprueba, además de que la fecha de inicio sea menor que la de finalización, que el horario no se solape con otro horario de la misma asignatura y que el horario no se solape con otro horario de otra asignatura en el mismo aula (teniendo solo en cuenta las asignaturas cuya fecha de inicio y finalización se solapen con la asignatura del horario a crear). En caso de que no se cumpla alguna de estas condiciones, el elemento no se guarda, indicando la causa.

Además de ser un elemento informativo en la información general que se muestra sobre una asignatura, es un modelo activo que al guardarse genera de forma automática las clases, en el día de la semana dado y la hora indicada, desde la fecha de inicio de la asignatura hasta la fecha de finalización (o desde la fecha actual si la asignatura ya ha empezado) en el aula asignada. Para la creación de las clases se deben cumplir las condiciones indicadas previamente al describir dicho modelo, tomando las decisiones que se han considerado más racionales, por ejemplo, en caso de que una clase se solape con otra en el mismo aula, se buscará otro aula libre dentro del mismo edificio de forma automática y en caso de no existir un aula disponible en esa franja horaria no se creará dicha clase (considerándolo un caso muy poco probable debido a las medidas adoptadas en la creación de los horarios); igualmente, en caso de existir una clase de la propia asignatura que se solape, dicha clase no se creará, prevaleciendo la clase que se creó anteriormente

—la cual debe haber sido creada de forma manual, al no poder haber sido creada por otro horario por las restricciones existentes—.

4.3. Tipos de usuarios

Dentro de la aplicación se distinguen distintos tipos de usuarios, con distintos roles y diferentes permisos:

- **Alumno:** no tiene asignado ningún permiso especial, se le identifica por tener un perfil de usuario con el campo que indica que es estudiante activado. Se indica de esta forma para que al crear el usuario sea obligatorio indicarlo y no sea necesario buscar el permiso manualmente, lo que podría provocar que se olvidase en muchas ocasiones. Por defecto, un perfil de usuario está definido como estudiante. Este tipo de usuario puede consultar su horario, apuntarse a seminarios, revisar sus asignaturas y seminarios y hacer *check in* en la clases con la posibilidad de valorarlas.
- **Profesor:** no tiene asignado ningún permiso especial, se le identifica por tener un perfil de usuario con el campo que indica que es estudiante desactivado. Se indica de esta forma por las razones expuestas en el usuario de tipo alumno. Este tipo de usuario puede consultar su horario, hacerse organizador de un seminario además de poder crearlos, revisar sus asignaturas y comentar en las clases, hacer *check in* en la clases, consultar la valoración de sus clases y la asistencia de los alumnos a éstas, y crear, editar y eliminar clases, seminarios y asignaturas con ciertas limitaciones (por ejemplo, no puede eliminar clases obligatorias aunque sí puede cambiar su horario).
- **Controlador:** no necesita un perfil de usuario. Tienen un permiso especial, `can_see_statistics`, que se le debe asignar manualmente. Su función consiste en comprobar el correcto desarrollo de las asignaturas, teniendo acceso a la información y estadísticas sobre la asistencia de los profesores y de los alumnos y la valoración de las asignaturas.
- **Bedel:** no necesita un perfil de usuario. Tienen un permiso especial, `can_see_codes`. Tienen acceso a los códigos de las clases, los cuales tienen que estar disponibles en las aulas correspondientes durante la clase correspondiente.

- **Administrador:** no necesita un perfil de usuario. Su función es la gestión de la aplicación, teniendo que atender a las *tareas de administración* realizadas por otros usuarios y encargados de la creación de usuarios, asignaturas y otros modelos; su principal función consiste en el mantenimiento de la base de datos.

Un mismo usuario puede desempeñar varios roles sin problema (excepto ser profesor y alumno al mismo tiempo), pero es preferible crear varios usuarios para una misma persona en caso de que vaya a desempeñar las funciones de distintos tipos de usuarios.

4.4. Diseño de las vistas y las *URLs*

En una aplicación de un proyecto *Django* existe, normalmente, un documento denominado `views.py`, con la definición de las *vistas* que procesan peticiones web y devuelven una respuesta web, como puede ser una página *HTML* o un *JSON*. Las *vistas* se relacionan con una petición del cliente a una determinada *URL* mediante un fichero llamado `urls.py`.

En la presente aplicación, generalmente se realiza en las *vistas* una distinción entre peticiones realizadas mediante *AJAX* y peticiones realizadas sin *AJAX*.

En este apartado se explican cada una de las *vistas* y sus *URLs*.

Todas las páginas *HTML* descritas en las siguientes subsecciones, a no ser que se indique explícitamente lo contrario, siguen un esquema común (ver Figura 4.5), que consta de un pie de página con información sobre la página, un enlace a una página de ayuda y otro enlace a una página para reportar problemas, un contenedor central cuya información dependerá del usuario registrado y de la *URL* solicitada, y una barra de menú flotante (siempre presente en la parte superior de la ventana del navegador) que contiene enlaces que varían según el tipo de usuario que ha iniciado sesión o incluso si la sesión está iniciada o no, apareciendo las siguientes configuraciones:

- Estudiante o profesor: enlaces a la página de inicio, la página del perfil del usuario registrado, la página para hacer el *check in*, la página con las asignaturas del usuario, el foro general y para cerrar sesión.
- Administrador: enlaces a la página de inicio, la página del perfil del usuario registrado, la página de administración, el foro general y para cerrar sesión.

- Controlador: enlaces a la página de inicio, la página del perfil del usuario registrado, la página de las estadísticas de asistencia, el foro general y para cerrar sesión.
- Bedel: enlaces a la página de inicio, la página del perfil del usuario registrado, la página con los códigos de las clases, el foro general y para cerrar sesión.

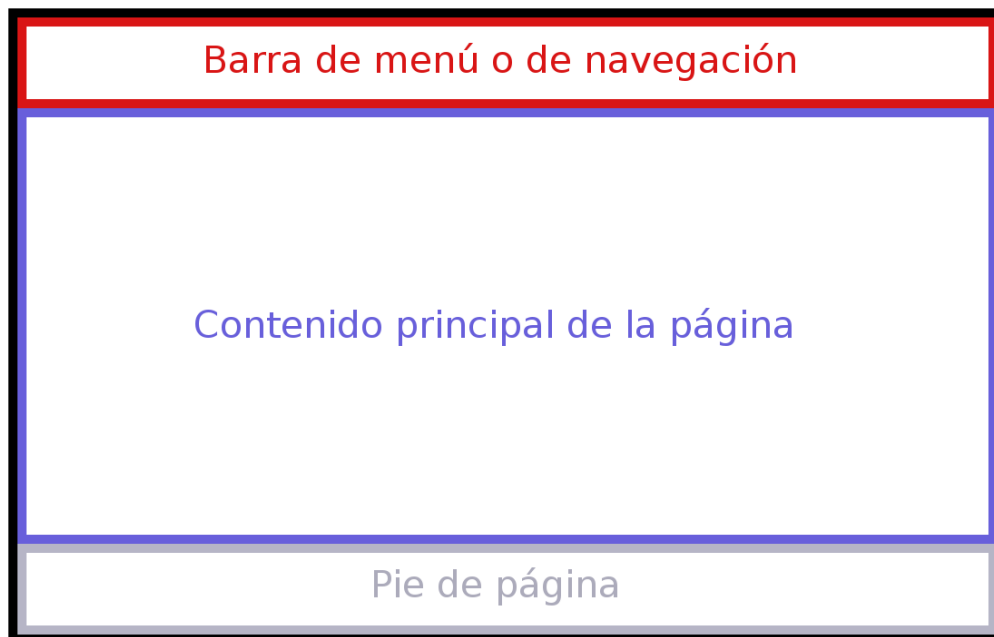


Figura 4.5: Esquema básico de una página

Además, todas las páginas redirigen al usuario a la página de inicio de sesión si intenta acceder a ellas sin haber iniciado sesión, a no ser que se especifique lo contrario.

Página de inicio

La página de inicio es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ¹, a través de la *vista* *home*, utilizando la plantilla *home.html* para generar la página *HTML* de la respuesta (ver Figura 4.6).

El contenido de la página *HTML* que recibe el cliente depende del tipo de usuario que solicite la página:

¹URL de la página de inicio: /



Figura 4.6: Página de inicio de un usuario con perfil

- Estudiante o profesor: la página muestra un calendario con las clases del usuario en el transcurso de una semana, de lunes a sábado. Si no se indica nada, la semana mostrada será la actual. Si la *URL* de la solicitud presenta una *querystring* del tipo *page=X*, siendo *X* un número, se mostrará la siguiente *X-ésima* semana a partir de la actual (o la *X-ésima* anterior en caso de ser un número negativo, o la actual en caso de ser 0).
- Administrador: la página muestra un enlace al panel de administración. Además, aparece una lista con las últimas tareas de administración (peticiones de los usuarios) pendientes de gestionar. Estas tareas enlazan a su recurso correspondiente dentro del panel de administración, para poder contestarlas o marcarlas como gestionadas, por ejemplo. Además, hay un enlace después de la última tarea mostrada que dirige a la lista de tareas de administración desde el panel de administración.
- Controlador: la página muestra un enlace a la página para controlar la asistencia.
- Bedel: la página muestra un enlace a la página con los códigos de las clases.

Página para realizar el *check in*

La página para realizar el *check in* es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ², a través de la *vista* `checkin_page`, utilizando la plantilla `checkin.html` para generar la página *HTML* de la respuesta (ver Figura 4.7).

The screenshot shows the URJCheckIn web application. At the top is a red navigation bar with links: Inicio, Perfil, Checkin, Asignaturas, and Foro. On the right of the bar, it says 'Bienvenido, estudiante1.' and a 'Salir' button. Below the navigation bar, the main heading is 'URJCheckIn' with the subtitle 'Realiza el Check In'. The central content area is titled 'Check in' and contains a QR code with the text 'Leer código' next to it. Below the QR code are four form fields: 'Asignatura:' with a dropdown menu showing 'Señales y Sistemas', 'Código:' with an empty text input, 'Puntuación:' with a text input containing the number '3', and 'Comentario:' with a larger text area containing the placeholder '(Comentario y puntuación anónimos para el profesor)'. At the bottom of the form is a green button labeled 'Check In'. The footer of the page contains the text: 'Para reportar problemas, informa a los administradores. | Ayuda. | URJCheckIn Jorge Bazaco Caloto © 2014'.

Figura 4.7: Página de checkin para un alumno

Esta página solo es accesible para usuarios que tienen un perfil de usuario.

Contiene un formulario para realizar un *check in*, es decir, para demostrar la asistencia a una clase del usuario registrado. Este formulario contiene siempre un campo con la asignatura sobre la que se quiere hacer el *check in* (permitiendo al usuario seleccionar una de sus asignaturas) y un campo para indicar el código de la clase. En caso de que el usuario sea un alumno, tendrá dos campos más, uno numérico para valorar la clase entre 0 y 5, y otro campo para escribir una opinión sobre la clase; y en caso de ser un profesor tendrá un campo numérico para indicar cuantos alumnos asistieron a la clase.

La página contiene además un botón para leer el código *QR* con la clave de la clase, evitando al usuario la necesidad de escribirlo a mano.

Al enviar el formulario, utilizando el botón correspondiente, si el usuario lo permite, se enviará la localización a través de unos campos ocultos en el formulario. El envío se realiza mediante una petición *POST* sobre la misma *URL*, procesada, por tanto, por la misma *vista*. La

²*URL* de la página para realizar el checkin: `/checkin`

vista, al identificarlo como un *POST*, almacena el *check in* si todo es correcto, realizando los procedimientos oportunos, e informa al usuario si el *check in* se ha realizado con éxito o no; en caso de que la petición sea mediante *AJAX*, se envía un *JSON* al cliente indicándole si se ha realizado con éxito o no, y el código *JavaScript* se encarga de informar al usuario con un mensaje en pantalla, mientras que si por el contrario se realiza sin *AJAX*, el servidor envía la página para realizar el *check in* completa que, a diferencia de la del *GET*, contiene un mensaje informando de si la operación se ha realizado con éxito.

Página de ayuda

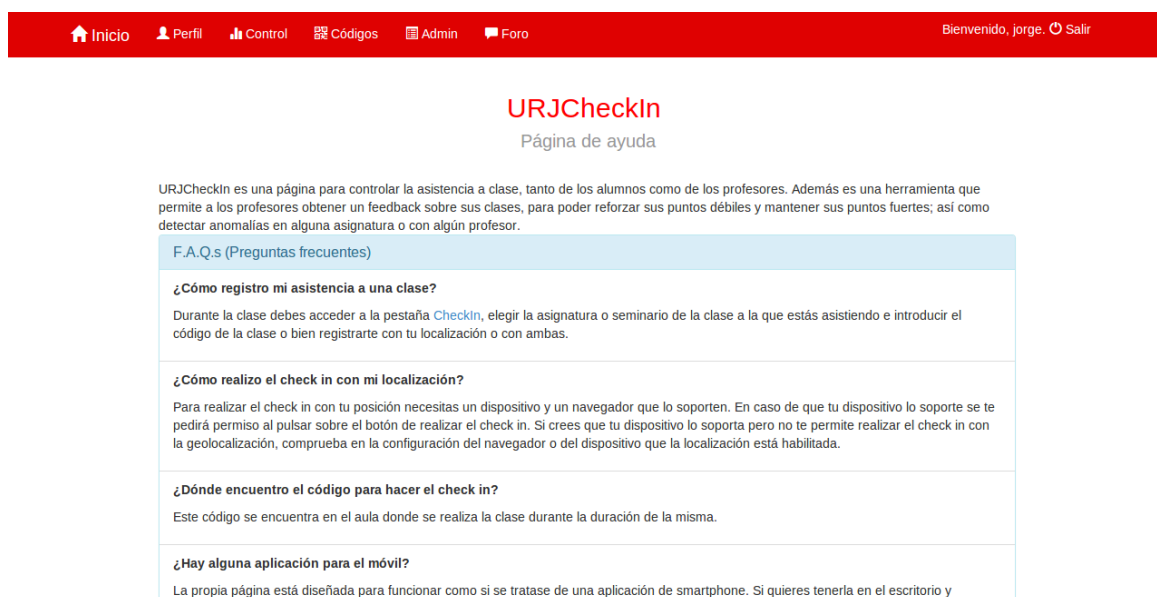


Figura 4.8: Página de ayuda

La página de ayuda es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ³, a través de la *vista* `help_page`, utilizando la plantilla `help.html` para generar la página *HTML* de la respuesta (ver Figura 4.8).

Esta página es la misma para todos los usuarios. Contiene una descripción sobre la función de la aplicación y una sección resolviendo preguntas que pueden ser frecuentes durante el empleo de la aplicación en un entorno real.

³URL de la página de ayuda: `/help`

Página de perfil

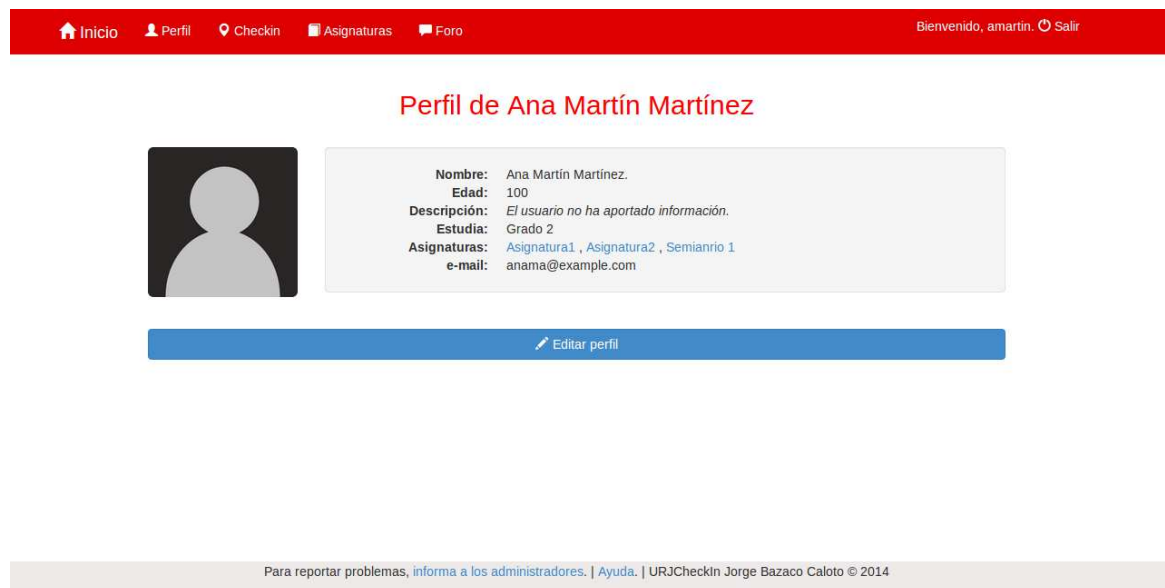


Figura 4.9: Página de perfil del propio usuario

La página de perfil es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ⁴, a través de la *vista* `profile_page`, utilizando la plantilla `profile.html` para generar la página *HTML* de la respuesta. En la *URL* mediante un identificador numérico se identifica al usuario del que se solicita el perfil (ver Figura 4.9).

El contenido varía dependiendo de si el cliente está solicitando el perfil de un usuario que tiene un perfil de usuario asignado o no lo tiene, y también depende de si el usuario solicita su página de perfil o la de otro usuario.

En caso de que un usuario solicite el perfil de otro usuario, si este último no tiene perfil se muestra un mensaje indicando que el usuario con dicho identificador no tiene perfil, y si se solicita un identificador que no se corresponde con ningún usuario, se muestra un mensaje indicando que dicho usuario no existe. Si se solicita el perfil de un usuario existente que tenga un perfil, se muestra su foto de perfil, su nombre y apellidos, su edad, su descripción, las asignaturas y el grado que enseña o estudia, y el correo electrónico en caso de que el usuario haya decidido compartirlo. La foto de perfil se añade en el documento *HTML* con un enlace a la foto del usuario ⁵, o con una foto por defecto ⁶ si no ha subido ninguna.

⁴*URL* de la página de perfil del usuario con id `iduser`: `/profile/view/iduser`

⁵*URL* de la foto de perfil personalizada del usuario con id `iduser`: `/media/profile_photos/iduser`

⁶*URL* de la foto de perfil por defecto: `/img/default_profile_img.png`

Para el caso de un usuario solicitando su propio perfil, en caso de que el usuario no tenga un perfil de usuario, se muestran dos formularios, uno para cambiar la dirección de correo electrónico y otro para cambiar la contraseña. Si el usuario sí tiene un perfil de usuario (es un profesor o un alumno), aparece un botón para editar su perfil debajo del mismo. Al pulsarlo desaparece su información de perfil y se muestran cuatro formularios y un botón para cancelar la edición y volver al perfil. Dos de los formularios son los descritos anteriormente, entre los otros se encuentra un formulario para cambiar la foto de perfil y otro para cambiar la edad, la descripción del perfil y la opción de mostrar públicamente el correo electrónico.

Para el envío de cada uno de los formularios:

- Cambio de imagen: se envía un *POST* al recurso correspondiente ⁷, que es procesado por la *vista* `change_profile_img`. La *URL* contiene al final la palabra `edit` o `delete`, dependiendo de si se pulsa sobre el botón “Cambiar foto” o “Eliminar mi foto de perfil”, respectivamente. En el caso de `edit` se reemplaza la foto actual por la foto enviada en la petición, y en el caso de `delete` se elimina y se le asigna al peticionario una imagen por defecto. El usuario es redirigido a su página de perfil si no emplea *AJAX* o recibirá una respuesta *JSON* indicando si se ha realizado con éxito y en tal caso se envía también la *URL* de la imagen, para que la reemplace el código JavaScript.
- Edición del perfil de usuario: se envía un *POST* sobre el mismo recurso en el que se muestra ⁸. La *vista* `profile_page` detecta que se trata de un *POST* y del contenido de la petición extrae la información y modifica el perfil de usuario del peticionario en función a ésta. Si no se realiza con *AJAX* se envía la página del perfil del usuario, y si se emplea *AJAX* se envía un *JSON* indicando si se ha realizado con éxito y enviando los datos del perfil de usuario para que el *script* de JavaScript actualice los cambios en el documento *HTML*.
- Cambio de correo electrónico: se envía un *POST* al recurso correspondiente ⁹, procesado por la *vista* `email_change`. Si la información de la petición es correcta se actualiza la dirección de correo del peticionario. El usuario es redirigido a su página de perfil si no emplea *AJAX*, y si lo emplea recibe una respuesta *JSON* indicando si se ha realizado con

⁷URL para cambiar la foto de perfil: `profile/img/(edit/delete)`

⁸URL de la página de perfil del usuario con id `iduser`: `/profile/view/iduser`

⁹URL para cambiar el e-mail: `/email_change`

éxito y en tal caso se envía también la nueva dirección de correo, para que la reemplace el código JavaScript.

- Cambio de contraseña: sin *AJAX* se envía un *POST* al recurso correspondiente ¹⁰, procesado por una *vista* definida por *Django*, que redirige al usuario a una página que muestra un mensaje de éxito si se realiza el cambio de contraseña correctamente, o a el recurso solicitado en el *POST* en caso contrario, que contiene el formulario para cambiar la contraseña, indicando los errores de la petición anterior. Si se realiza con *AJAX*, se realiza un *POST* sobre un recurso diferente ¹¹, que modifica la contraseña si la información recibida es correcta y envía al usuario un *JSON* indicándole si se ha cambiado su contraseña o los errores producidos en su petición.

Páginas de una clase



Figura 4.10: Página de una clase

La página de una clase es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ¹², que contiene un identificador numérico para indicar la clase que se

¹⁰URL para cambiar la contraseña: `/password_change/`

¹¹URL para cambiar la contraseña con *AJAX*: `/password_change/ajax`

¹²URL de la página de la clase con id `idlesson`: `/lesson/idlesson`

solicita, a través de la *vista* `process_lesson`, utilizando la plantilla `lesson.html` para generar la página *HTML* de la respuesta (ver Figura 4.10).

Esta página es accesible para profesores y alumnos que enseñan o estudian la asignatura a la que pertenece la clase y para los usuarios con el permiso `can_see_statistics`.

Esta página presenta una breve descripción de la clase indicada en la *URL*, seguida de los comentarios que los profesores hayan escrito sobre dicha clase. En caso de que haya muchos mensajes, no se muestran inicialmente todos, y se solicitarán sin *AJAX* mediante una *querystring* de la forma `page=X`, siendo `X` un número, indicando la página de comentarios que se desea, enviando el servidor de nuevo la página de la clase pero con los comentarios solicitados; o mediante *AJAX* solicitando mensajes más recientes o anteriores, en cuyo caso el servidor solo envía los mensajes nuevos que el código *JavaScript* incorpora al *árbol DOM*. La *URL* empleada para pedir más comentarios con *AJAX* ¹³ consta de una serie de variables, siendo `idcurrent` el identificador del mensaje más reciente o más antiguo mostrado en la página que está viendo el usuario, `idlesson` el identificador de la clase, y mediante la palabra `true` o `false` se indica si se quieren mensajes más nuevos o más antiguos, respectivamente.

Si una clase se ha impartido y la está visualizando un profesor o un usuario con el permiso `can_see_statistics`, se muestra un enlace a una página con los *check ins* de la clase. Estos usuarios son los únicos con acceso a dicha página, la cual muestra mediante la *vista* `lesson_attendance`, que utiliza la plantilla `lesson_attendance.html`, la información de los *check ins* realizados en la clase con el identificador indicado en la *URL* ¹⁴, pudiendo ver en un mapa dónde se realizó el *check in* de cada usuario.

También, si una clase se ha impartido y la está visualizando un profesor o un usuario con el permiso `can_see_statistics`, se muestra un listado con los comentarios realizados por los alumnos en el *check in* de la clase junto con su puntuación, de forma completamente anónima.

Si el usuario es un profesor, encima de los mensajes tendrá un espacio para escribir y enviar un nuevo comentario, el cual se envía con un *POST* a la *URL* de la clase, respondiendo el servidor la página con el nuevo comentario, o en caso de emplear *AJAX* con un *JSON* con la información del nuevo comentario para añadirlo al resto de mensajes.

En caso de que el usuario sea un profesor y que la fecha y hora de la clase sea posterior

¹³*URL* para pedir comentarios con *AJAX*: `/more/comments/idcurrent/idlesson/(true|false)`

¹⁴*URL* de la página con los *check ins* de la clase con id `idlesson`: `/lesson/idlesson/attendance`

a la actual, aparece un enlace a la página para editar la clase. Si se solicita la *URL* ¹⁵ mediante una petición *GET*, se envía al solicitante, gracias a la *vista* `edit_lesson` y la plantilla `lesson_edit.html`, una página con un formulario para editar la clase con el identificador indicado en la *URL*. Esta página solo se muestra a los profesores de la asignatura a la que corresponde dicha clase, y contiene, además del formulario para editar la clase, un formulario para buscar aulas libres y un enlace para volver a la página principal de la clase. Los formularios indicados funcionan de la siguiente forma:

- Formulario de edición de la clase: contiene campos de tipo fecha y de tipo hora para indicar el inicio y el fin de las clases, y un campo para seleccionar el aula, previamente iniciados con la información de la clase; presenta también un campo oculto para indicar si se solicita la edición o la eliminación de la clase. Mediante el botón “Editar clase”, se solicita con un *POST* a la *URL* actual que se realicen los cambios, enviándose de nuevo la página al usuario o indicando simplemente el resultado de la operación si la petición se ha realizado con *AJAX*.

Además, teniendo en cuenta que existen clases obligatorias y clases extra (añadidas por el profesor), si la clase es extra aparece otro botón para eliminar la clase, realizando un *POST* a la misma *URL* pero utilizando el campo oculto para solicitar la eliminación de la clase. Después, el usuario es redirigido a la página de la asignatura de dicha clase. Aunque el usuario tratase de eliminar una clase obligatoria de forma manual, se le denegaría la operación.

- Formulario para buscar aulas libres: este formulario tiene campos para indicar la hora y fecha de inicio y finalización de una clase, y para seleccionar el edificio. Al pulsar el botón “Buscar aula”, se realiza una petición *GET* a la *URL* correspondiente ¹⁶ con una *querystring* con la información, que es gestionada por la *vista* `free_room`. En caso de realizarse mediante *AJAX*, el servidor envía un *JSON* con el aula libre en el edificio indicado en la franja horaria indicada; si no se utiliza *AJAX*, se devuelve una página utilizando la plantilla `freeroom.html`, que contiene el mismo formulario para buscar aulas libres y el nombre del aula libre según la información de la solicitud (en este caso, la página se muestra en una nueva pestaña del navegador).

¹⁵*URL* para editar la clase con id `idlesson`: `/lesson/idlesson/edit`

¹⁶*URL* para buscar aulas libres: `/freeroom`

Página del foro

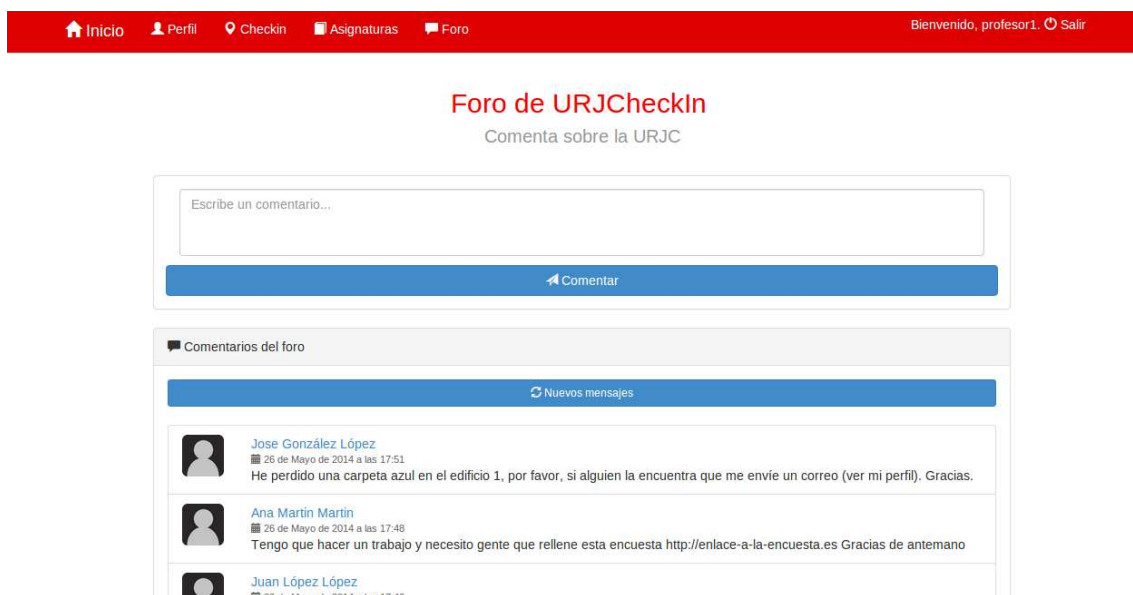


Figura 4.11: Página del foro

La página del foro es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ¹⁷, a través de la *vista* `forum`, utilizando la plantilla `forum.html` para generar la página *HTML* de la respuesta ((ver Figura 4.11).

La página del foro contiene los últimos mensajes publicados en el foro, si se quieren ver más mensajes, hay botones para pedirlos con un *GET*, sin *AJAX* mediante una *querystring* de la forma `page=X`, siendo `X` un número, indicando la página de comentarios que se desea, enviando el servidor de nuevo la página del foro pero con los comentarios solicitados; o mediante *AJAX* solicitando mensajes más recientes o anteriores, en cuyo caso el servidor solo enviará los mensajes nuevos que el código *JavaScript* incorporará al *árbol DOM*. La *URL* empleada para pedir más comentarios con *AJAX* ¹⁸ contiene algunas variables, siendo `idcurrent` el identificador del mensaje más reciente o más antiguo mostrado en la página que está viendo el usuario, y mediante la palabra `true` o `false` se indica si se quieren mensajes más nuevos o más antiguos, respectivamente.

Encima de los mensajes aparece un formulario para enviar un comentario, mediante un *POST* sobre la misma *URL*. La *vista* reconoce que es un *POST* y almacena el mensaje recibido

¹⁷*URL* de la página del foro: `/forum`

¹⁸*URL* para pedir más comentarios en el foro: `/more/comments/idcurrent/0/(true/false)`

a nombre del usuario que ha realizado la petición. En la respuesta, el servidor devuelve la página del foro, o si la petición se ha realizado con *AJAX*, devuelve solo la información del nuevo mensaje para que el código *JavaScript* lo incluya en la página.

El objetivo inicial del foro era dotar a la aplicación de un aspecto social, según lo comentado en los objetivos iniciales.

La función del foro es la de conectar a toda la universidad, desde alumnos y profesores hasta otros trabajadores de la misma, permitiendo un entorno universitario conectado y permitiendo mejorar el mismo. Algunos ejemplos prácticos de su uso pueden consistir en informar sobre problemas de tráfico o transporte público para llegar a la universidad, problemas en las instalaciones del campus universitario, permitir a alumnos solicitar colaboración de otros —por ejemplo en trabajos o trabajos fin de grado en los que necesiten la participación de otros alumnos para realizar pruebas del mismo o para realizar alguna encuesta—, o informar sobre la pérdida de alguna pertenencia por si alguien la encontrase, entre otros casos.

Página de los seminarios

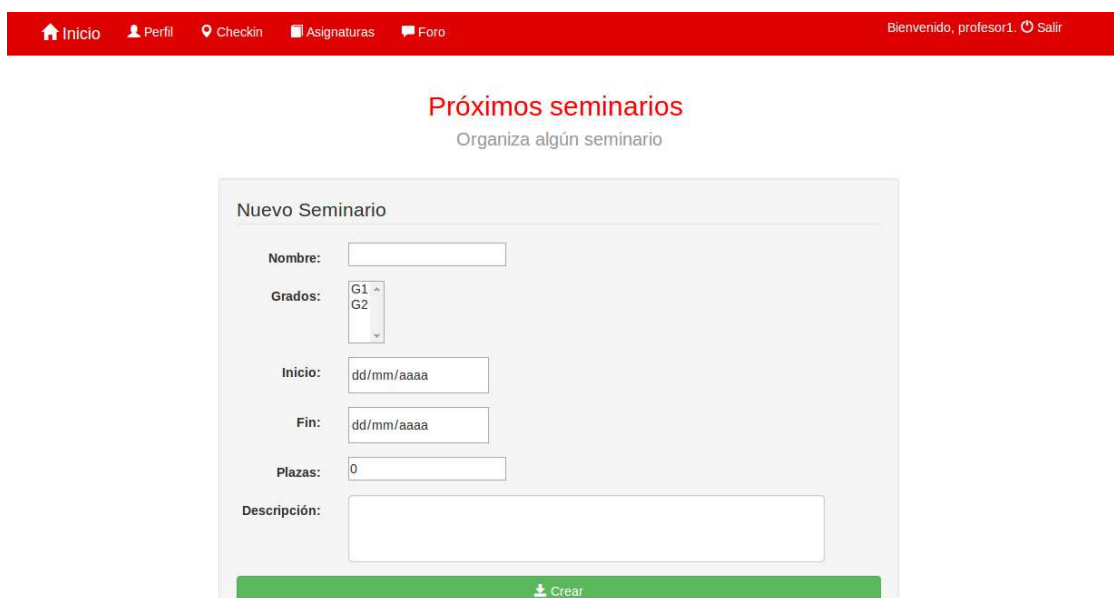


Figura 4.12: Página de los seminarios para un profesor

La página de los seminarios es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ¹⁹, , a través de la *vista* `seminars`, utilizando la plantilla

¹⁹URL de la página de los seminarios: `/seminars`

`seminars.html` para generar la página *HTML* de la respuesta (ver Figura 4.12).

Para acceder a esta página se debe tener un perfil de usuario, es decir, se debe ser un alumno o un profesor. Para ambos, se mostrará una lista de los seminarios que aún no han empezado que estén relacionados con sus grados, enlazando a la página específica de cada seminario. Los profesores tendrán además un formulario para la creación de un nuevo seminario.

El formulario para la creación de seminarios consta de un campo de texto para indicar el nombre que tendrá el seminario, un campo de selección múltiple para indicar el grado o los grados con los que está relacionado, un campo de fecha para indicar el inicio del seminario y otro para indicar la fecha de finalización, un campo numérico para indicar el número de plazas y un campo de texto para escribir una breve descripción del seminario. Al enviar el seminario se realiza un *POST* sobre la misma *URL*, cuya *vista* al reconocer que es un *POST* crea el seminario y, si la información de la petición es correcta, redirige al usuario a la página principal del seminario que acaba de crear.

Página de las asignaturas

Inicio Perfil Checkin Asignaturas Foro Bienvenido, profesor1. Salir

Asignaturas y seminarios

Accede a la información de tus asignaturas

Mis asignaturas	Mis seminarios
Señales y Sistemas	Próximos seminarios
Introducción a la programación	Seminario de Accesibilidad

Últimos mensajes en tus asignaturas

- Clase de Señales y Sistemas (05/06/2014 13:00)
2 de Junio de 2014 a las 13:25
Por favor, traed la calculadora a esta clase.
- Clase de Introducción a la programación (03/06/2014 14:00)
2 de Junio de 2014 a las 13:24
Realizaremos un pequeño test para que pueda saber que conocimientos previos tenéis.
- Clase de Señales y Sistemas (04/06/2014 13:00)
2 de Junio de 2014 a las 13:22
Ésta es la primera clase, en ella realizaré una introducción de la asignatura.

Figura 4.13: Página de las asignaturas

La página de las asignaturas es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ²⁰, a través de la *vista* `subjects_page`, utilizando la plantilla

²⁰*URL* de la página de las asignaturas: `/subjects`

`subjects.html` para generar la página *HTML* de la respuesta (ver Figura 4.13).

Es necesario tener un perfil de usuario para poder acceder a esta página, es decir, ser un alumno o un profesor. Se muestra una lista con las asignaturas del usuario, con enlaces a la página de cada una de ellas, otra lista con un enlace a la página de los seminarios, seguido de los seminarios del usuario, los cuales también enlazan a sus páginas correspondientes, y una última lista con los últimos mensajes escritos en las clases de las asignaturas y seminarios del usuario, con enlaces a dichas clases para verlos en su contexto.

Páginas de una asignatura

Inicio **Perfil** **Checkin** **Asignaturas** **Foro** Bienvenido, profesor1. [Salir](#)

Nombre: Señales y Sistemas
Número de alumnos: 1
Grado: • Grado1
Profesor: • [Profesor1 Apellido1P1 Apellido2P1](#)
Fecha inicio/fin: 01-06-2014/02-12-2014
Estado: 0 clases pasadas y 10 restantes
Valoración media: 3/5
Asistencia profesor: 100%
Asistencia alumnos: 100%
Horario:

- Miércoles de 13:00 a 13:59
- Jueves de 13:00 a 13:59

[Comprobar asistencia](#)
[Crear clases](#)

[Ver estadísticas](#)

Próximas clases	Últimas clases
04-06-2014 13:00/13:59 dentro de 1 día, 23 horas	Todavía no ha habido clases
05-06-2014 13:00/13:59 dentro de 2 días, 23 horas	No hay más clases

Figura 4.14: Página de una asignatura para un profesor

La página de una asignatura es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ²¹, , que contiene un identificador numérico para indicar la asignatura solicitada, a través de la *vista* `subject_page`, utilizando la plantilla `subject.html` para generar la página *HTML* de la respuesta (ver Figura 4.14). En esta sección cuando se habla de asignaturas se está incluyendo también a los seminarios.

Para acceder a la página de una asignatura el usuario debe ser un alumno o profesor relacionado con la asignatura, o tener el permiso `can_see_statistics` (para el caso de un

²¹URL de la página de la asignatura con id `idsubject`: `/subjects/idsubject`

seminario los profesores y alumnos pueden acceder a su información aunque no estén relacionados con dicho seminario).

La parte más básica de la página está formada por una descripción de la asignatura y dos listas con las clases antiguas y las clases futuras. Además, en caso de estar impartándose una clase aparece la misma en otra lista. Las listas de las clases tienen un botón para pedir más clases en caso de que no se muestren todas. Estos botones, si no está disponible *JavaScript* realizan un *GET* con una *querystring* del tipo `page=X`, siendo `X` un número, indicando la página de clases que se quiere, estando las clases de una asignatura paginadas; de esta forma el servidor envía la página de la asignatura con las clases de la página indicada. Por el contrario, si hay *JavaScript*, el *GET* se realiza mediante *AJAX* a una *URL* ²² que contiene algunas variables, siendo `idcurrent` el identificador de la última clase que ve el usuario, e indicando con `true` si las clases que se están solicitando son posteriores a la clase correspondiente a `idcurrent`, o `false` si son las anteriores; el servidor devuelve la información de las siguientes diez clases solicitadas y el código *JavaScript* la añade en el lugar correspondiente.

Los profesores de la asignatura y los usuarios con el permiso `can_see_statistics`, tienen además un enlace a una página con información sobre la asistencia de los alumnos a la asignatura ²³, gestionado por la vista `subject_attendance` que emplea la plantilla `subject_attendance.html`, que contiene un listado de los alumnos de la asignatura indicada con su nombre y apellidos, junto con su DNI y su porcentaje de asistencia a clase; y otro enlace ²⁴, gestionado por la vista `subject_statistics` que emplea la plantilla `subject_statistics.html`, que muestra una página que contiene gráficas con el porcentaje de asistencia de los alumnos a cada clase, la puntuación media de cada clase, el porcentaje de asistencia medio de los alumnos y el de los profesores (ver Figura 4.15).

Además, los profesores de la asignatura tienen un enlace a una página para crear nuevas clases ²⁵, gestionado por la vista `create_lesson` que emplea la plantilla `new_lesson.html`, que contiene un formulario para crear una nueva clase, otro formulario para buscar aulas libres

²²*URL* para pedir más clases de una asignatura: `/more/lessons/idcurrent/(true/false)`

²³*URL* de la página con el porcentaje de asistencia de los alumnos a la asignatura con id `idsubject`:
`/subjects/idsubject/attendance`

²⁴*URL* de la página con las estadísticas de la asignatura con id `idsubject`:
`/subjects/idsubject/statistics`

²⁵*URL* de la página para crear clases de la asignatura con id `idsubject`:
`/subjects/idsubject/new_lesson`

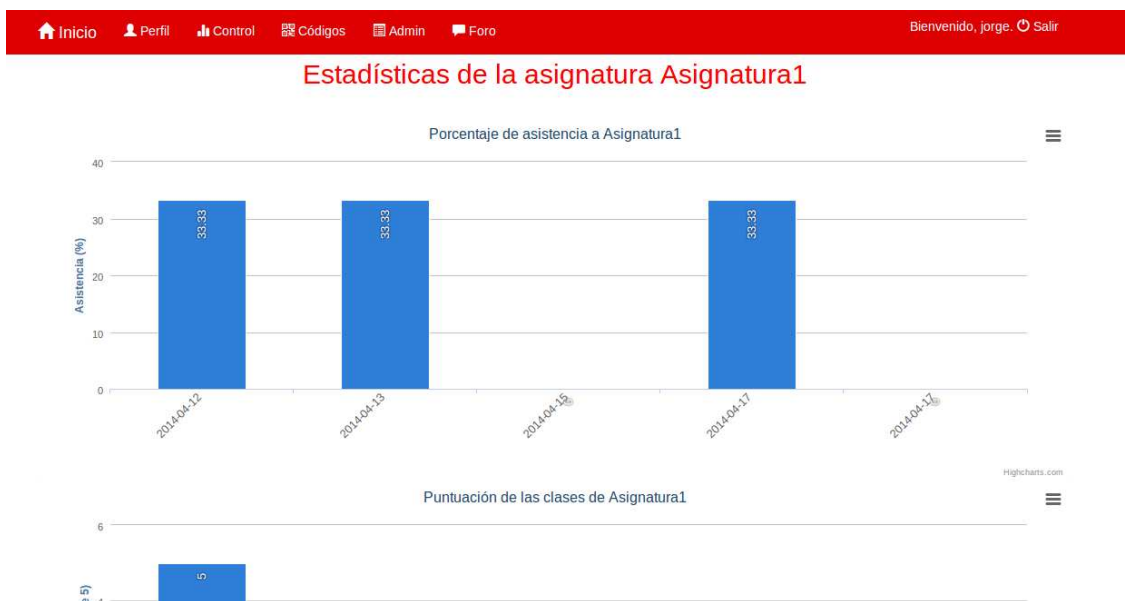


Figura 4.15: Página de las estadísticas de una asignatura

y un enlace para volver a la asignatura. El formulario para buscar aulas libres es exactamente igual al descrito en la sección de las páginas de una clase (ver 4.4).

En cuanto al formulario para crear una clase contiene campos de tipo fecha y de tipo hora para indicar el inicio y el fin de las clases, y un campo para seleccionar el aula. Al enviar el formulario se solicita con un *POST* a la *URL* actual, y la *vista* al detectar un *POST* comprueba la información de la petición y en caso de ser correcta crea la clase y redirige al usuario a la página de la clase creada.

Si el usuario es un profesor y además es el creador de la asignatura, tiene otro enlace ²⁶, gestionado por la *vista* `subject_edit`, que con la plantilla `subject_edit.html`, genera una página con un formulario para editar o eliminar la clase, relleno con la información actual de la asignatura, y con un enlace para volver a la página de la asignatura. Este formulario contiene los mismos campos que el formulario para la creación de un seminario, más un campo oculto para indicar si se quiere editar o eliminar la asignatura, pero en este caso el *POST* se envía a la *URL* de la página para editar la asignatura; el servidor guarda los cambios en caso de solicitarse una edición, y en caso de realizarse la petición sin *AJAX* envía la página con la nueva información, y en caso de realizarse con *AJAX* envía un *JSON* con el resultado de la operación (si se ha realizado con éxito o los errores que se han producido en caso contrario). En caso

²⁶*URL* de la página para editar la asignatura con id `idsubject`: `/subjects/idsubject/edit`

de que se pulse sobre el botón para eliminar la asignatura, el servidor la elimina y redirige al usuario a la página con sus asignaturas.

Por último, la página de un seminario que aún no ha empezado, incluye para los alumnos y profesores un botón para apuntarse/desapuntarse del seminario, o hacerse organizador del mismo o dejar de serlo, respectivamente. Al pulsar el botón se envía un *POST* sobre la página del seminario, y en caso de ser un alumno, si está apuntado se desapuntará y si no está apuntado al seminario se apuntará si quedan plazas, y de la misma forma para un profesor pero sin preocuparse por las plazas. El servidor, después de guardar los cambios, envía la página completa si la petición es sin *AJAX*, y si se ha realizado con *AJAX*, el servidor solo envía un *JSON* con la información necesaria para modificar la página en función a la acción realizada.

Página de reportes

Reportar problema

Url:

Problema:

Enviar

Mis reportes

ID: #19	Estado: pendiente [solicitado hace 1 día, 21 horas]
Problema:	Creo que tengo asignado un perfil de alumno, podéis cambiármelo en el caso de que esté equivocado
ID: #18	Estado: pendiente [solicitado hace 1 día, 21 horas]
Problema:	Levante

Figura 4.16: Página de reportes

La página de reportes es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ²⁷, a través de la *vista* `reports_page`, utilizando la plantilla `reports.html` para generar la página *HTML* de la respuesta (ver Figura 4.16).

Esta página está formada por un formulario para reportar un problema y por un listado de los reportes realizados por el usuario registrado. En el listado, aparecen inicialmente los reportes

²⁷URL de la página de reportes: `/reports`

más recientes, con su información detallada, tanto sobre la petición como su gestión (si ha sido gestionada y la respuesta del administrador, si la hay).

En caso de querer visualizar reportes más antiguos, el usuario dispone de un botón para pedirlos; se solicitarán sin *AJAX* mediante una *querystring* de la forma `page=X`, siendo `X` un número, indicando la página de reportes que se desea, enviando el servidor de nuevo la página de reportes pero con los reportes solicitados; o mediante *AJAX* solicitando reportes más recientes (para el caso en el que esté en una página distinta de la primera) o anteriores, en cuyo caso el servidor solo envía los reportes nuevos que el código *JavaScript* incorpora al *árbol DOM*. La *URL* empleada para pedir más reportes con *AJAX* ²⁸ contiene algunas variables, siendo `idcurrent` el identificador del reporte más reciente o más antiguo mostrado en la página que está viendo el usuario, y mediante la palabra `true` o `false` se indica si se quieren mensajes más nuevos o más antiguos, respectivamente.

Sobre el formulario, está formado por dos campos de texto, uno para indicar la *URL* en la que el usuario tiene un problema, en caso de que la haya (es un campo opcional), y otro para describir el problema. Al enviar el formulario, se realiza un *POST* sobre la página de reportes, y el servidor guarda el reporte como una tarea de administración y responde con la página completa, o simplemente con un *JSON* con la información de la tarea de administración creada si la petición se ha realizado con *AJAX*.

Página para controlar el desarrollo de las asignaturas

La página para controlar las asignaturas es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ²⁹, a través de la *vista* `control_attendance`, utilizando la plantilla `control_attendance.html` para generar la página *HTML* de la respuesta (ver Figura 4.17).

A esta página solo tienen acceso los usuarios con el permiso `can_see_statistics`.

Esta página muestra un listado con las asignaturas, que contiene el nombre de la asignatura, el grado o grados a los que pertenece, el profesor o profesores que la imparten, el porcentaje de asistencia de los profesores y el de los alumnos y la valoración media. Inicialmente se muestran diez asignaturas y para cargar más existen unos botones, que realizan una petición *GET* con

²⁸*URL* para pedir más reportes: `/more/reports/idcurrent/(true/false)`

²⁹*URL* de la página de control de asignaturas: `/control/attendance`

Inicio Perfil Control Códigos Admin Foro
Bienvenido, jorge. Salir

Revisa las estadísticas de las asignaturas

Filtrar

Seminarios y asignaturas ▼

Ordenar por: Nombre ☐ orden inverso

Filtrar

Porcentaje de asistencia					
Asignatura	Grado	Profesores	Asist. (prof.)	Asist. (alum.)	Valor. media
Asignatura1	◦ Grado1	◦ Jose González López	6,94%	20,0%	3,4/5
Asignatura2	◦ Grado1	◦ Jose González López	0,0%	100%	3/5
Asignatura3	◦ Grado1 ◦ Grado 2	Ninguno	100%	100%	3/5

Figura 4.17: Página para el control de asignaturas

una *querystring* de la forma `page=X`, siendo `X` un número, indicando la página de asignaturas que se desea, enviando el servidor de nuevo la página de control de asignaturas pero con las asignaturas solicitados.

Además, encima del listado de asignaturas, se muestra un formulario para filtrar la búsqueda de asignaturas. Este formulario puede ser rellenado con el nombre de una asignatura, nombre del grado, nombre del profesor, apellido o apellidos del profesor, y seleccionar si se desean seminarios y asignaturas o solo uno de los dos, y si se desea ordenar por nombre de la asignatura, fecha de inicio o fecha de finalización de la misma y en orden lógico o inverso. Todos estos campos son opcionales, pudiendo realizarse cualquier combinación, y además no es necesario que tengan el nombre completo, por ejemplo, si una asignatura se llama “Asignatura de introducción”, se muestra escribiendo en el campo asignatura, por ejemplo, “de intro”, y lo mismo para el resto de campos.

Una vez relleno el formulario, si se pulsa sobre el botón “Filtrar”, se envía un *GET* sobre la misma página, pero con una *querystring* con la información del filtro, recibiendo el usuario la página con las asignaturas que cumplan los criterios de búsqueda, estando también paginadas si existiesen más de diez asignaturas que lo cumplan. La paginación y el filtrado son perfectamente compatibles, sin producirse ningún tipo de conflicto.

Página con los códigos de las clases

La página con los códigos de las clases es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ³⁰, a través de la *vista* `show_codes`, utilizando la plantilla `control_codes.html` para generar la página *HTML* de la respuesta (ver Figura 4.18).

Inicio Perfil Control Códigos Admin Foro Bienvenido, jorge. Salir

Accede a los códigos

Filtrar

Códigos del día: dd/mm/aaaa Edificio: todos Aula: cualquiera Tipo: Seminarios y asignaturas Desde las: --:--

Hasta las: --:--

Ordenar por: Hora de inicio ☐ orden inverso

*Por defecto se mostrarán las clases desde este momento hasta el final del día

Obtener códigos

Aula 1, Edificio I 02-06-2014

Horario: 14:12-14:12

Asignatura: Asignatura1

Código: mqjysjnigfaxihjhjui

Figura 4.18: Página de los códigos de las clases

A esta página solo tienen acceso los usuarios con el permiso `can_see_codes`.

Inicialmente, la página solo muestra un formulario para filtrar los códigos de las clases que se quieren ver. Este formulario permite indicar un día, seleccionar un edificio, seleccionar un aula, seleccionar si se quieren códigos de asignaturas y seminarios o solo de uno de los dos, indicar una hora de inicio y una hora de finalización para generar una franja en la que tiene que estar la hora de comienzo de la clase, y un campo para indicar si se quieren ordenar los resultados por hora de inicio, aula o edificio y otro campo para indicar un orden lógico o inverso. Todos estos campos son opcionales, pudiendo realizarse cualquier combinación, y por defecto se muestran los códigos del día actual, con una franja de tiempo desde el momento de la petición hasta el final del día.

Una vez relleno el formulario, si se pulsa sobre el botón “Obtener códigos”, se envía un *GET* sobre la misma página, pero con una *querystring* con la información del filtro, recibiendo

³⁰URL de la página con los códigos de las clases: `/control/codes`

el usuario la página con los códigos de las clases que cumplan los criterios de búsqueda.

Cada código se muestra dentro de una caja que contiene el nombre del aula y del edificio, la fecha y el horario de la clase, el nombre de la asignatura y el código, tanto como cadena de caracteres como con la imagen *QR*.



Figura 4.19: Código de una clase para imprimir

Estos códigos están pensados para ser impresos y puestos en las clases, por lo que al imprimirlos se utilizan propiedades *CSS* específicas. Así en la impresión, cada página contendrá un solo código, con un tamaño de letra mayor, y un tamaño de la imagen del código *QR* mucho más grande, y mostrando además una breve instrucción sobre como escanear el código desde la página de *check in* (ver Figura 4.19).

Página de inicio de sesión

La página de inicio de sesión es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente³¹ o al intentar acceder a algún recurso que requiera inicio de sesión si el usuario no la ha iniciado; a través de una *vista* por defecto existente en *Django*, aunque utilizando una plantilla desarrollada específicamente para esta aplicación (ver Figura 4.20).

³¹URL de la página para iniciar sesión: `/login`

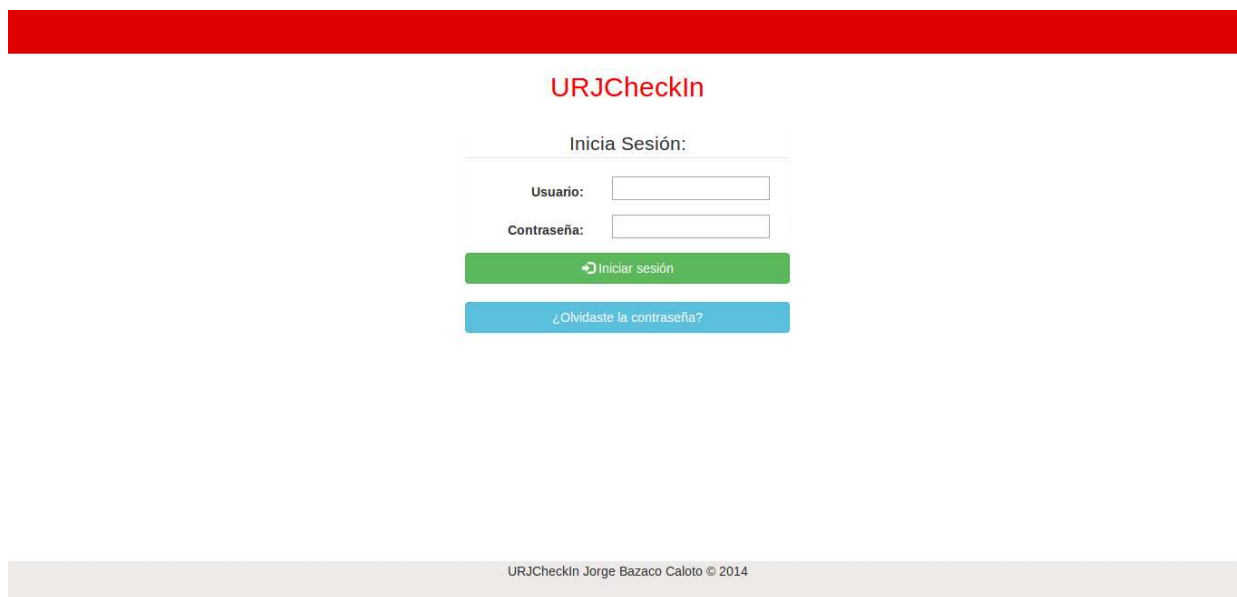


Figura 4.20: Página de inicio de sesión

La página de inicio de sesión tiene una barra de menú sin ningún elemento, simplemente decorativa para mantener el estilo con el resto de la página.

Contiene un formulario para iniciar sesión y un enlace por si el usuario ha olvidado su contraseña. Al iniciar sesión se envía un *POST* gestionado por la *vista* de *Django*, y si el usuario y contraseña son correctos se redirige al usuario a la página de inicio a no ser que se indique una *URL* concreta mediante una *querystring*.

En caso de acceder a la página de inicio de sesión con la sesión iniciada, la barra de menú tendrá un enlace a la página de inicio y la página contendrá un botón para cerrar sesión. Este caso solo se produce si el usuario escribe manualmente la *URL* del inicio de sesión, ya que no hay ninguna página que, estando con la sesión iniciada, enlace a la misma, al ser solo útil cuando la sesión está cerrada, en cuyo caso se redirigirá a la misma automáticamente.

Página de cierre de sesión

La página de cierre de sesión es enviada al cliente cuando se recibe una petición *GET* sobre el recurso correspondiente ³², a través de la *vista* `my_logout`, utilizando la plantilla `registration/logged_out.html` para generar la página *HTML* de la respuesta.

Esta página presenta una barra de menú sin ningún enlace, simplemente para mantener la

³²*URL* para cerrar sesión: `/logout`

uniformidad con el resto de la página. La página contiene un enlace a la página para iniciar sesión.

Solo se muestra en el caso de escribir la *URL* manualmente, o en el caso de cerrar sesión desde el panel de administración (aunque en este caso la página se muestra bajo una *URL* diferente). Esto se debe a que la forma en la que está implementada `my_logout` utiliza la función existente en *Django* para cerrar sesión. El comportamiento al cerrar sesión en cualquier sitio menos en el panel de administración —y excepto cuando se escribe la *URL* manualmente— provoca una redirección a la página para iniciar sesión. La razón por la que no se utiliza directamente la funcionalidad por defecto de *Django* es para añadir una diferencia en su comportamiento cuando se utiliza *AJAX*, para evitar perder el efecto de estar trabajando con una aplicación nativa.

Páginas para resetear la contraseña

The screenshot shows a web page for password reset. At the top is a red navigation bar with a white house icon and the text 'Inicio'. Below this is a red heading '¿Has olvidado tu clave?'. Under the heading is a line of text: 'Introduce tu dirección de correo a continuación y te enviaremos por correo electrónico las instrucciones para establecer una nueva.' Below this text is a form titled 'Restablecer contraseña'. The form contains a label 'Correo electrónico:' followed by a text input field. At the bottom of the form is a blue button with a white arrow icon and the text 'Solicitar reseteo'. At the very bottom of the page is a light gray footer bar containing the text 'URJCheckin Jorge Bazaco Caloto © 2014'.

Figura 4.21: Página para resetear la contraseña

En el caso de olvidar la contraseña, la página para iniciar sesión contiene un enlace a una *URL* para resetear la contraseña. Para ello se han utilizado las *vistas* que define *Django* para dicho proceso. Lo único que difiere con el comportamiento por defecto es que se han creado nuevas plantillas para mantener el diseño del resto de la aplicación (ver Figura 4.21).

Tras seguir los pasos que se van indicando el usuario puede generar una contraseña nueva.

La barra de menú que aparece durante todo el proceso tiene un único enlace a la página de inicio, y no es necesario iniciar sesión para acceder a estas páginas.

URLs a documentos estáticos

Hay documentos que no necesitan ser procesados, ya que no varía su contenido según el contexto de la petición. Para ello, se ha utilizado la *vista* que ofrece *Django* para servir ficheros estáticos.

El tipo de ficheros que sirve la presente aplicación de forma estática son, ficheros *JavaScript*, hojas de estilo *CSS*, imágenes y ficheros *CSV*.

Panel de administración

El panel de administración se ha gestionado con las *vistas* y las *URLs* definidas por *Django* por defecto, realizando pequeñas modificaciones en algunas plantillas, y utilizando otras presentes en *Django* sin necesidad de generar nuevas, al adaptarse a las necesidades de la aplicación (ver Figura 4.22).



Figura 4.22: Página de administración

Todas las páginas del panel de administración se encuentran partiendo del *path* `/admin/`.

A pesar de que *Django* ofrece un comportamiento por defecto para la creación y modificación de los modelos de la aplicación, también permite flexibilidad, permitiendo definir la forma de administrarlos a través de un fichero denominado `admin.py`. Así, tras modificarlo,

la gestión de cada modelo, cuya mención en el presente trabajo es relevante, se presenta de la siguiente forma:

- Usuarios: cuando se crea o se edita un usuario, además se permite crear o editar un perfil de usuario para dicho usuario directamente, de hecho, para crear o gestionar los perfiles de usuario se debe realizar a través del usuario. De esta forma se consigue reforzar el hecho de que un perfil de usuario extiende al usuario.

Además, en el listado de los usuarios se ha decidido mostrar en columnas el nombre y los apellidos (como identificación principal), la dirección de correo electrónico, el nombre y los apellidos por separado, si es *staff* (administrador) y si es estudiante, profesor o ninguno de los dos. Dentro de este listado, se pueden filtrar los usuarios según sean *staff*, sean *superusuarios*, sean activos, sean alumnos o profesores y por el grado al que pertenecen. A parte de esa selección se puede realizar una búsqueda por palabras entre los campos de las asignaturas, los grados y el DNI, además de los campos de búsqueda por defecto del modelo usuarios.

Por último, encima del listado aparece la opción de subir un documento CSV para generar varios usuarios y sus respectivos perfiles de usuario de forma más rápida, subiendo el fichero CSV mediante un *POST* a `/admin/auth/user/csv/`, ejecutándose un *script* en el servidor que generará los usuarios y perfiles de usuarios según la información del documento. Además, se facilita un enlace a un fichero CSV de ejemplo —servido de forma estática con *Django*—, que puede ser utilizado para crear los usuarios.

- Asignaturas: al crear o editar una asignatura, aparece plegada la información relevante para los seminarios, de esta forma al gestionar las asignaturas que no son seminarios, se hará de una forma más eficiente y clara. Permiten además crear horarios directamente y ver los horarios que tiene dicha asignatura, ya que no tienen sentido si no están relacionados con una asignatura y así es más fácil su gestión, de hecho, la única forma que hay de crearlos y gestionarlos es desde el recurso de una asignatura en el panel de administración (cada horario se muestra en una línea ya que su información es lo suficientemente ligera como para compactarla en una línea y de esta forma no se hace la página de administración de una asignatura demasiado larga, ver Figura 4.23). Para facilitar aun más la creación de una asignatura, el campo del creador de una asignatura estará inicialmente

asignado al administrador que la está creando.

Administración de URJCheckin

Inicio > App > Asignaturas > Seminario de Accesibilidad

Modificar asignatura

Añadir usuarios a la asignatura a partir de un fichero CSV

Seleccionar archivo | Ningún archivo seleccionado | Subir y enlazar usuarios | Descargar plantilla

Histórico | Ver en el sitio

Nombre: Seminario de Accesibilidad

Grados: G1, G2

Mantenga presionado "Control", o "Command" en un Mac, para seleccionar más de una opción.

Fecha de inicio: 03/06/2014 Hoy |

Fecha de finalización: 16/06/2014 Hoy |

Creador: profesor1

Seminario (Mostrar)

Day	Hora de inicio	Hora de finalización	Aula	¿Eliminar?
-----	Ahora	Ahora	-----	
-----	Ahora	Ahora	-----	

Agregar otro Horario.

Eliminar | Grabar y añadir otro | Grabar y continuar editando | Grabar

Figura 4.23: Página de administración de una asignatura

La página de administración de una asignatura presenta, además, un aparatado para subir un documento *CSV* mediante un *POST* a `/admin/app/subject/idsubject/csv/`, siendo *idsubject* el identificador de la asignatura que se está gestionando. Este documento es procesado en el servidor, que vincula la asignatura con los usuarios cuyos DNIs están listados en el fichero *CSV*. Junto al formulario de envío existe un enlace para descargar un documento *CSV* de ejemplo —servido de forma estática con *Django*—, que puede ser utilizado para realizar la tarea descrita.

En cuanto al listado de asignaturas se ha decidido mostrar en columnas el nombre de la asignatura (como identificación principal), la fecha de inicio, la fecha de finalización y su estado (si es antigua, actual o futura a la fecha actual). Además, se pueden filtrar según sean o no seminarios, según su estado y según su grado; y realizar búsquedas por palabras presentes en los campos del nombre, nombre del grado y nombre y apellidos de los perfiles de usuario relacionados con una asignatura.

- Aulas: no tienen ningún detalle particular que resaltar.
- Check ins: en el listado de *check ins* se ha decidido mostrar en columnas la representación de la clase relacionada (como identificación principal) y el nombre del usuario que

realizó el *check in*. Además, se puede realizar una búsqueda por palabras presentes en los campos del nombre de la asignatura relacionada, el nombre o el código del grado relacionado, y por el nombre o apellidos del usuario que realizó el *check in*.

- Clases: en el listado de clases se ha decidido mostrar en columnas la representación de la clase (como identificador principal), la hora y fecha de inicio y la de finalización, si se ha realizado o no, si es clase extra o no y el aula y edificio donde se imparte. Además, se pueden filtrar según se haya realizado o no, según sea clase extra o no y según su estado (anterior, actual o futura, según el momento actual); y se puede realizar una búsqueda por palabras presentes en los campos del nombre de la asignatura relacionada y el nombre o el código del grado relacionado.
- Comentarios del foro: desde el panel de administración no se permite crear comentarios nuevos. En el listado de comentarios del foro se ha decidido mostrar en columnas el número identificativo del comentario (como identificador principal), la hora de publicación y el nombre del usuario que lo publicó. Se pueden filtrar los comentarios según la fecha de publicación (hoy, últimos 7 días, este mes, este año) y se puede realizar una búsqueda por palabras presentes en los campos del comentario en sí y del nombre, apellido o nombre de usuario del usuario que lo publicó.
- Comentarios en clase: presentan las mismas características que los comentarios del foro salvo porque el identificador principal contiene la representación de la clase en que fue escrito, y que añade a la búsqueda por palabras el nombre de la asignatura relacionada.
- Edificios: en la página de administración de un edificio aparecen en línea las aulas de dicho edificio, pudiendo editarlas o crear nuevas aulas dentro de ese edificio.
- Grados: en el listado de grados se muestran el nombre del grado (como identificador principal) y el código del grado. Se pueden realizar búsquedas por palabras contenidas en los campos del nombre y el código del grado.
- Tareas de administración: en el listado de tareas de administración se muestran en columnas el identificador de la tarea (como identificador principal), la fecha y hora de la petición, si está gestionada o no, el nombre de usuario del peticionario y el nombre de

usuario del administrador que la gestionó en caso de que haya sido gestionada. Se pueden filtrar según esté gestionada o no, según la fecha de publicación (hoy, últimos 7 días, este mes, este año) y para seleccionar exclusivamente las que hayan sido resueltas por el administrador que está viendo la página. Además, se puede realizar una búsqueda por palabras contenidas en los campos del nombre de usuario del peticionario o del administrador que lo gestionó o del identificador de la tarea.

Página no encontrada

En caso de que no se solicite ninguno de los recursos indicados anteriormente, se devuelve una página informando del error de *HTTP 404*, de página no encontrada, la cual mantiene el diseño del resto de la página.

4.5. Ficheros *JavaScript*

Entre los documentos *JavaScript* que se utilizan en la aplicación se diferencian dos grandes grupos: las bibliotecas de terceros desarrolladas bajo licencias de software libre y los *scripts* creados específicamente para esta aplicación.

Referente a las bibliotecas externas, se han utilizado:

- *Bootstrap*: esta biblioteca, explicada en la sección 3.5, no solo consta de *scripts* de *JavaScript*, sino que principalmente contiene hojas de estilo *CSS*. La funcionalidad *JavaScript* que aporta a la aplicación se limita al pliegue y despliegue de la barra de menú en dispositivos con pantalla pequeña.
- *jQuery*: esta biblioteca, explicada en la sección 3.3.2, facilita la modificación del *árbol DOM* y la realización y gestión de peticiones *AJAX*, así como aportar algunas funciones para simplificar el código *JavaScript* y la posibilidad de utilizar el *plugin* o añadido *jQuery-form* que gestiona el cambio de la foto de perfil de un usuario mediante *AJAX*.
- *Highcharts*: esta biblioteca aporta la funcionalidad para crear gráficas, las cuales se utilizan para mostrar estadísticas de las asignaturas.
- *Jsqr*: esta biblioteca permite la lectura de códigos *QR*.

- *QRcode-js*: esta biblioteca permite la generación de códigos *QR*.

La funcionalidad de las bibliotecas externas es aprovechada gracias a su *API* o interfaz de aplicación, desde los *scripts* desarrollados específicamente para la aplicación, los cuales se han dividido en varios ficheros para una mejor organización del código. Las funcionalidades que aportan dichos *scripts* consisten fundamentalmente en gestionar las peticiones *AJAX* —incluyendo el cambio entre páginas, el envío de formularios, la petición de más elementos (como es el caso de la petición de comentarios y clases) y la gestión de los botones del navegador para retroceder o avanzar en el historial de navegación—, leer y generar códigos *QR* y modificar ciertos elementos del *árbol DOM* —como por ejemplo cambiar entre la vista del perfil del propio usuario y la configuración del mismo—.

4.6. Diseño de la aplicación y diferencias entre dispositivos

Las principales diferencias en el diseño de la aplicación, considerando tanto aspectos funcionales como la presentación visual de la aplicación, dependen del tamaño de la pantalla del dispositivo y del navegador utilizado, más concretamente del soporte de *JavaScript* y *jQuery*.

4.6.1. Distribución y tamaño del contenido

La aplicación se desarrolló al mismo tiempo para dispositivos móviles, con pantalla pequeña, y para ordenadores, con un tamaño de pantalla mayor; también se tuvo en cuenta un tercer pilar, de pantallas medianas de *tablets* u ordenadores portátiles con pantallas pequeñas en comparación a los ordenadores normales, pero los cambios son menores. El objetivo era que la aplicación fuese cómoda de utilizar en pantallas pequeñas, y táctiles, y que no se mostrase todo demasiado grande en pantallas grandes y sin aprovechar el mayor tamaño de pantalla. *Bootstrap* facilitó bastante la tarea, por ejemplo, creándose en muchos casos listas que en el móvil ocupan todo el ancho de la pantalla y se colocan una debajo de otra, y en el ordenador se disponen unas al lado de otras, o el caso de tablas como la del control de las asignaturas que son muy anchas debido a la presencia de muchos campos y en el móvil se muestran unos campos debajo de otros y en el ordenador todos en una fila, todo ello utilizando exclusivamente clases *CSS* de *Bootstrap*.

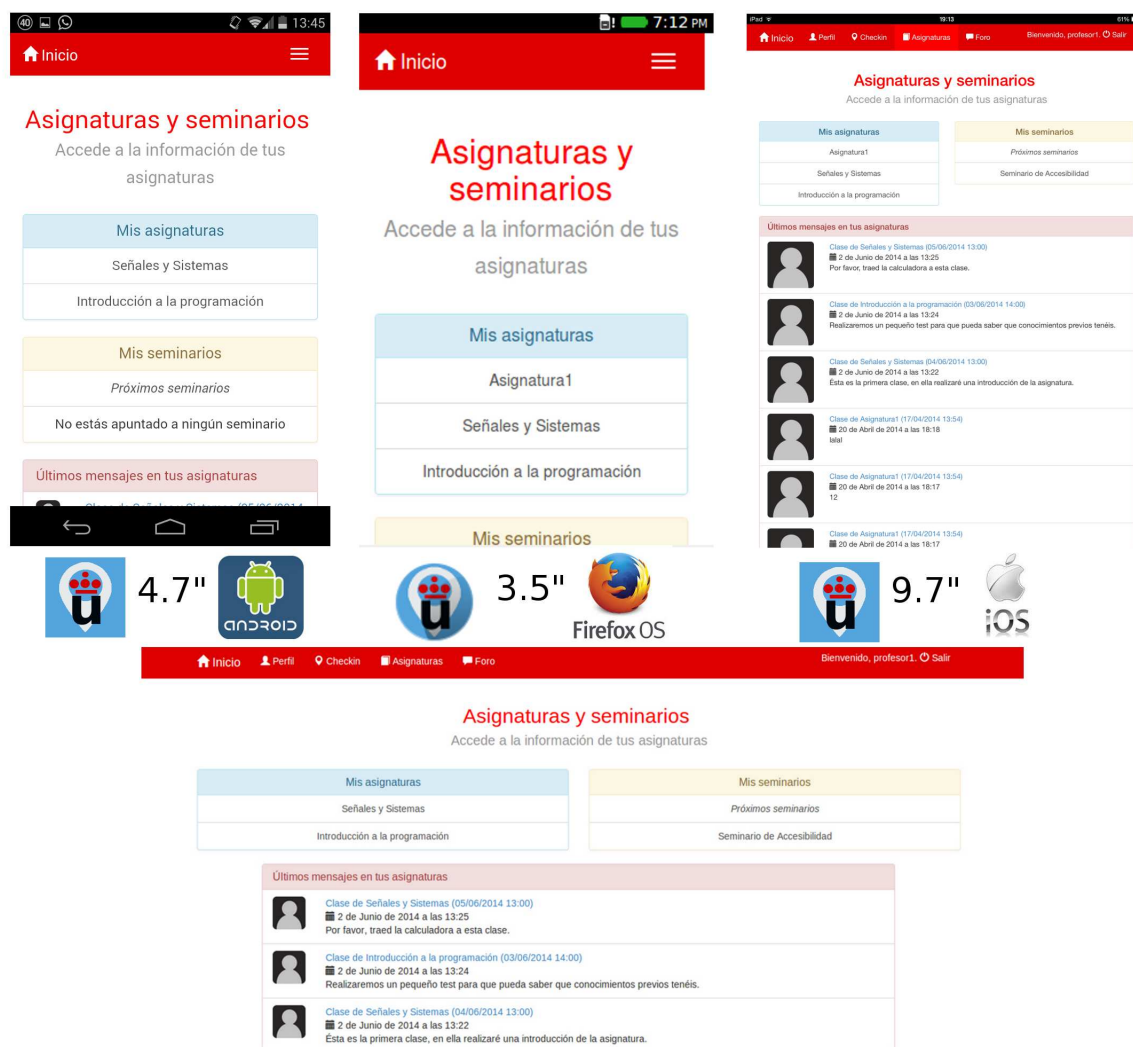


Figura 4.24: Vista de la página en diferentes dispositivos y sistemas operativos (por orden, móvil Android con pantalla de 4.7", móvil FirefoxOS de 3.5", tablet iOS de 9.7" y abajo ordenador de 15")

4.6.2. Simulación de una aplicación nativa

Otro objetivo era que la página fuese bastante dinámica y simulase una aplicación nativa, siguiendo la filosofía de una *single-web application*, sin embargo, esta aplicación requiere más de una página con contenido bastante diferente.

Para que la aplicación pareciese un programa, era necesario establecer algún elemento común en todas las páginas que diese la sensación de estar siempre en el mismo sitio, para lo que se utilizó una barra de navegación presente en todas las páginas y presente siempre en la parte superior de la ventana del navegador, aunque se realice *scroll*.

Otro problema era que en muchos navegadores de móviles se puede hacer desaparecer la barra del navegador para que parezca que no se está en el navegador y que se está utilizando una aplicación nativa, pero al cargar una página de forma *tradicional* la barra vuelve a aparecer rompiendo el efecto. Además, con unas pocas etiquetas *meta* de *HTML* —las cuales se han introducido en la presente aplicación— se pueden instalar páginas web en móviles con sistema operativo *Android* o *iOS*, utilizándose como si realmente fuese una aplicación nativa —teniendo acceso a ellas desde el menú de aplicaciones o la página de inicio y desapareciendo la barra del navegador—, pero en el caso de *iOS* se abre el navegador cada vez que se realiza una petición *tradicional* a alguna *URL*. Ambas situaciones desembocaron en la necesidad de utilizar *AJAX*, no solo para tareas pequeñas como el envío de un formulario y la petición de mensajes de foro, sino también para cambiar de una página a otra.

Además de las etiquetas en el *HTML* para instalar la aplicación en *iOS* y en *Android*, se ha creado un fichero `urjcheckin.webapp` para instalarla en *FirefoxOS*.

Se requería además cargar las páginas sin *AJAX* por si se introduce la *URL* o se accede desde algún enlace externo, aunque principalmente para mantener una alternativa con un sistema más tradicional y, en mayor medida, para los navegadores que no soportan *JavaScript* o *jQuery*.

Como se ha comentado anteriormente, todas las páginas comparten la barra de navegación, la cual está siempre en la parte superior de la ventana del navegador. Tras unas pruebas, al utilizar *AJAX* para cambiar de un recurso a otro, cuando el código *JavaScript* modificaba el contenido del *árbol DOM*, algunos navegadores detectaban que la barra de navegación no cambiaba y no la sustituían, sin embargo, otros navegadores sustituían la barra ya presente por otra exactamente igual produciéndose un parpadeo innecesario de la misma, rompiendo el efecto de aplicación nativa. Para solucionar esto, se decidió que la barra permanecería fija y lo que enviaría el *JSON* de la respuesta del servidor sería el contenido que se coloca en un contenedor específico presente en todas las páginas en el documento *HTML*.

De esta forma, en vez de utilizar el sistema tradicional de extensión de plantillas de *Django* —que consiste en crear plantillas con elementos comunes y que las más específicas incluyan las capas externas extendiendo las plantillas con los elementos comunes— se diseñó en sentido inverso; existe una plantilla con los elementos comunes (la barra de navegación y el pie, la plantilla `main.html`) y dentro de ella se incluye el contenido específico presente en otras plantillas. De esta forma si la petición es *AJAX* el servidor devuelve con un *JSON* el resultado de

renderizar la plantilla específica (la que contiene el contenido principal que varía según la página y el contexto de la petición), y si la petición no se realiza mediante *AJAX* se envía al cliente la página *HTML renderizada* a partir de la plantilla general conteniendo la plantilla específica—esto último se realiza gracias a la etiqueta `{% include htmlname %}` en la plantilla general, y cuando se *renderiza* la misma se le pasa en el contexto la variable `htmlname` con el nombre de la plantilla específica deseada—.

La única parte de la página que no mantiene el efecto aplicación es el panel de administración, ya que se ha considerado que los administradores van a utilizar un ordenador para gestionar la aplicación, y se ha considerado que no merecía la pena sacrificar tiempo para esta función en detrimento de otras más importantes. Además, las páginas para resetear la contraseña también carecen de un funcionamiento con *AJAX*, considerándola una situación poco frecuente y que no afecta de forma significativa al funcionamiento de la aplicación.

4.6.3. Otras diferencias

En el caso de pantallas grandes y medianas, la barra de navegación es lo suficientemente ancha como para mostrar todos los enlaces, sin embargo, en una pantalla pequeña no, sobre todo teniendo en cuenta que los enlaces deben tener un tamaño suficiente para ser fácilmente seleccionables con un dedo, en vez de con un ratón. Por tanto, gracias a *Bootstrap*, la barra de navegación en pantallas pequeñas se pliega, mostrando el enlace a la página de inicio y otro botón para desplegar la barra y acceder así al resto de enlaces. Esto supone un problema en dispositivos con pantalla pequeña cuyo navegador no soporte *JavaScript*, al no poder desplegar el menú, por lo que se deberían escribir las *URLs* a mano. Para solucionarlo, exclusivamente en los dispositivos que cumplan ambas condiciones (pantalla pequeña y sin *JavaScript*), en la página de inicio se muestran los enlaces que hay en la barra de navegación encima del contenido principal, pudiendo de esta forma acceder al resto de *URLs*, y pudiendo volver a la página de inicio, ya que el enlace a la página de inicio sí está accesible desde la barra de navegación (ya que no requiere ser desplegada).

Otros cambios en dispositivos con ausencia de *JavaScript* o incompatibilidad con *jQuery* son la eliminación de los botones para mostrar el mapa de los *check ins* en la página que muestra los *check ins* de una clase; en la página de las gráficas sobre las estadísticas de una asignatura, en vez de aparecer las gráficas aparecen unas tablas con los datos; en la página para realizar

el *check in* no aparece el botón para escanear el código *QR*; en la página de perfil del propio usuario, aparece tanto su información como los formularios de edición, en vez de disponer de un botón para alternar entre ambos; y en la página que muestra los códigos de las clases no muestra las imágenes de los códigos *QR*, ya que son generados con *JavaScript*. Además, en estos dispositivos se pierde el efecto de aplicación nativa, al realizarse todas las peticiones sin *AJAX*, recargándose la página completamente cada vez que se solicita una página o se envía un formulario.

Capítulo 5

Conclusiones

5.1. Evaluación del resultado

El trabajo buscaba ofrecer una solución informatizada para el control de asistencia de alumnos y profesores, objetivo que se ha conseguido realizar, añadiendo además otras funcionalidades y un mejor aprovechamiento de dicho registro, más allá de servir exclusivamente para comprobar el cumplimiento de las normas de asistencia.

Analizando cada uno de los subobjetivos planteados en la sección 2, también se obtiene un resultado satisfactorio:

- Aplicación universal: se ha conseguido crear una aplicación compatible con multitud de dispositivos, sistemas operativos y navegadores de internet, no limitándose a ser compatible, sino también adaptándose para ofrecer un control y una apariencia adecuados.
- Registrar la asistencia a clase: la aplicación registra la asistencia de profesores y alumnos mediante la realización de *check ins*, demostrando su asistencia mediante una clave y/o enviando su localización.
- Introducir mejoras en el ámbito académico y no limitarse exclusivamente a ofrecer una alternativa a métodos más tradicionales. La aplicación es capaz de mostrar estadísticas (tanto numéricas como mediante el empleo de gráficas) sobre la calidad de las clases y asignaturas y la asistencia. También permite la valoración de las clases por parte de los alumnos, para que los profesores puedan mantener sus puntos fuertes y mejorar los débiles.

- Sencillez en su uso: se ha creado una interfaz bastante intuitiva para todos los usuarios. Se han facilitado herramientas para simplificar las tareas de administración mediante cambios en el panel de administración, y la aplicación presenta en su mayoría tablas, botones y formularios con bastante información para entenderlos la primera vez que se utilizan y sin ser excesiva y molesta cuando se ha utilizado durante mucho tiempo; además de presentar una página de ayuda y una página para comunicarse con los administradores.
- Aportar un elemento social: se ha creado un foro para que todos los miembros de la universidad puedan comunicarse y colaborar para lograr un mejor entorno universitario. Además, se ha proporcionado a los profesores la opción de comunicar a los alumnos información sobre las clases y que los alumnos puedan dar su opinión a los profesores.
- Demostrar la capacidad de *HTML5*: se considera demostrada la capacidad de *HTML5*, *CSS3* y *JavaScript* para desarrollar aplicaciones, las cuales pueden ser además muy versátiles en cuanto a compatibilidad con dispositivos, sin haberse echado en falta en ningún momento elementos que podrían haber ofrecido aplicaciones nativas, ya que todo se ha podido realizar con las tres tecnologías anteriormente citadas; todo esto de un modo más sencillo y rápido que si se hubiesen tenido que desarrollar aplicaciones nativas para diferentes sistemas operativos.

5.2. Conocimientos aplicados

Para el desarrollo del presente trabajo he utilizado conocimientos adquiridos durante la realización del Grado en Ingeniería en Tecnologías de la Telecomunicación.

Entre dichos conocimientos se incluye el desarrollo de software mediante el empleo de diferentes lenguajes de programación, destacando el aprendizaje de *Python*, al ser uno de los lenguajes utilizados para la creación de la aplicación, pero también siendo útil la habilidad adquirida con todos ellos. Además, paralelamente a la realización de este trabajo, estuve cursando asignaturas en las cuales se trabajaba con *JavaScript*, por lo que también fueron útiles dichos conocimientos, aunque al ser al mismo tiempo también fue necesario el aprendizaje autodidacta.

También he aplicado conocimientos sobre el funcionamiento de la web, como es el caso del protocolo *HTTP*, y conocimientos del *framework Django*.

Durante el grado, también he adquirido capacidad para la búsqueda de soluciones y el análisis de las mismas, para obtener soluciones óptimas y viables.

Por último, aunque no es algo específico del grado estudiado, he adquirido conocimientos sobre el funcionamiento de la universidad durante los años de estudio en la misma, los cuales, debido a la temática del proyecto, han sido esenciales para obtener un buen resultado.

5.3. Conocimientos adquiridos

Además del desarrollo de una aplicación funcional y la obtención de una posible solución a una necesidad real, durante la realización del presente trabajo he obtenido conocimientos y también confianza en mis capacidades.

Al iniciar el trabajo, no me había enfrentado anteriormente a la creación de un programa de tal envergadura, y sin un guión estricto. Sin embargo, me he demostrado a mí mismo que tengo capacidades de realizarlo, y que no solo soy capaz de programar una idea, sino también de tomar grandes decisiones como su diseño, las funciones específicas y el camino a seguir para lograr algo como el resultado obtenido.

En cuanto a conocimientos, he solidificado y madurado mis conocimientos de *Django* y *Python*, con los cuales ya había trabajado pero no en tanta profundidad. He adquirido conocimientos de *JavaScript*, *HTML5* y *CSS3*, tecnologías que apenas había utilizado antes del comienzo del proyecto, y de la biblioteca *jQuery* y el *framework Bootstrap*, tecnologías muy utilizadas actualmente. También he utilizado por primera vez herramientas de depuración de código, como es la herramienta *Pylint*.

Por último, pero no menos importante, he aprendido bastante sobre el ámbito web, la evolución de las páginas web, la situación actual y el camino que parecen estar siguiendo; conocimientos que me servirán en el futuro en caso de trabajar con alguna tecnología relacionada.

5.4. Trabajos futuros

Finalizado el trabajo, se plantea como posible evolución del mismo aumentar la eficiencia del manejo de la base de datos, gracias a funciones de *Django* no empleadas, que permiten controlar con una mayor precisión qué datos se deben leer de la base de datos en cada lectura

de la misma.

Además, se podría intentar combinar con otras aplicaciones educativas ya existentes, más centradas en las asignaturas, como aquellas que ofrecen herramientas a los profesores para compartir documentos de una asignatura con los alumnos o para la entrega de trabajos por parte de los alumnos, y para la gestión de las notas de las asignaturas, ya que es mucho mejor utilizar una sola aplicación, tanto para que los alumnos y profesores no tengan que acceder a distintas páginas, como para simplificar la gestión y reducir la cantidad de información a almacenar, gracias a una base de datos unificada.

Por último, sería interesante llevarlo a un entorno simulado, realizando el despliegue, pasando de una versión de desarrollo a una versión funcional, y en caso de ser posible realizar pruebas simulando un entorno real, utilizando la aplicación entre varios usuarios.

Apéndice A

Instalación y uso

Para el empleo de la presente herramienta, no es necesaria su instalación, más allá de tener instalado *Python* y *Django*, e instalar el módulo de *Python* *pytz* (en *Linux* se puede instalar mediante el comando `sudo pip install pytz`).

Para probar la aplicación se debe iniciar el servidor ejecutando en un terminal, desde la carpeta *URJCheckIn*, el comando `python manage.py runserver IP:Puerto`, indicando la dirección *IP* y el puerto, o mediante `python manage.py runserver Puerto`. Al ejecutar el comando se iniciará el servidor y la terminal indica la *URL* a través de la cual es accesible, como se muestra a continuación:

```
jorge@ubuntu:~/Proyecto/URJCheckIn$ python manage.py runserver 1025
Validating models...

0 errors found
May 28, 2014 - 12:17:24
Django version 1.5.4, using settings 'URJCheckIn.settings'
Development server is running at http://127.0.0.1:1025/
Quit the server with CONTROL-C.
```

Por tanto, será suficiente con introducir la *URL* en cualquier navegador —habiéndose probado con éxito en los principales navegadores y en diferentes versiones de los mismos—.

Sin embargo, es necesario otro paso para evitar errores al resetear la contraseña de un usuario o al crear usuarios mediante un *CSV*, ya que necesita un servidor capaz de enviar correos electrónicos. Para la simulación, es suficiente con ejecutar en un terminal el comando `python`

`-m smtpd -n -c DebuggingServer localhost:1026`. En caso de no estar disponible el puerto 1026, sería necesario abrir el fichero `URJCheckIn/URJCheckIn/settings.py` y modificar el campo `EMAIL_PORT` escribiendo el puerto que se va a utilizar.

Se debe tener en cuenta que el presente programa es una versión de desarrollo, para su despliegue se deberían realizar ciertos cambios y el resultado sería ligeramente diferente (sin afectar de cara al usuario final).

El programa entregado presenta una base de datos de prueba, para poder probarlo en menos tiempo. Si se desea crear una base de datos desde cero, se debe borrar el fichero `URJCheckIn/db.sqlite` y ejecutar desde el directorio `URJCheckIn` el comando `python manage.py syncdb` (para el caso de *Django 1.5.4* que es la versión que se ha utilizado durante el desarrollo, ya que este comando puede variar según la versión). Para iniciar sesión con la base de datos de prueba, se facilitan los siguientes credenciales, teniendo un administrador con todos los permisos o *superusuario* con nombre de usuario `admin`, un profesor con nombre de usuario `profesor1` y un alumno con nombre de usuario `alumno1`, todos ellos con la contraseña `password`.

Apéndice B

Recurso para descargar la aplicación

La aplicación se encuentra disponible en *GitHub*, en la siguiente *URL*:

`https://github.com/jbazaco/URJCheckIn`

Apéndice C

Licencia

La aplicación desarrollada se publica bajo la licencia de software libre Apache v2.0.

Copyright 2014 Jorge Bazaco Caloto

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

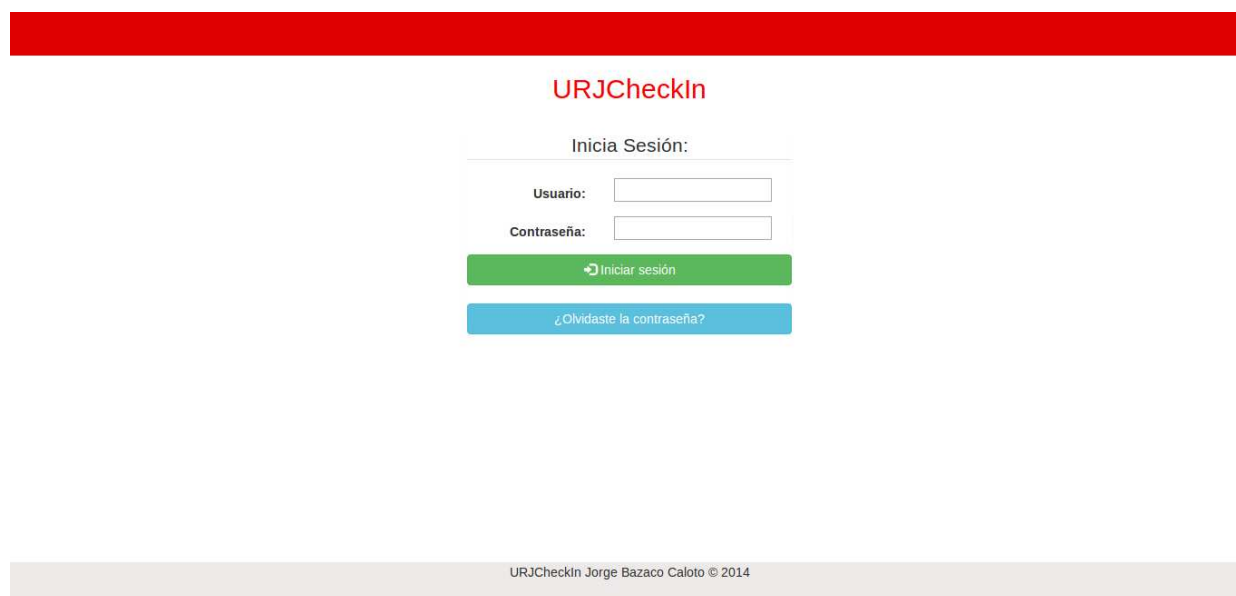
<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

Apéndice D

Manual de usuario

Inicio de sesión



URJCheckIn

Inicia Sesión:

Usuario:

Contraseña:

[Iniciar sesión](#)

[¿Olvidaste la contraseña?](#)

URJCheckIn Jorge Bazaco Caloto © 2014

Figura D.1: Página de inicio de sesión

Para acceder a la página URJCheckIn es necesario iniciar sesión. Al acceder a la página aparece un formulario para iniciar sesión. Se debe indicar el nombre de usuario y la contraseña y pulsar sobre el botón “Iniciar sesión”.

En caso de olvidar la contraseña o el nombre de usuario, se debe pulsar sobre el botón con el texto “¿Olvidaste la contraseña?” y seguir los pasos indicados.



Figura D.2: Página para resetear la contraseña

Barra de navegación

En la parte superior de la pantalla aparece una barra roja que permite acceder a distintas páginas de la aplicación. En caso de tener los enlaces directamente visibles (Figura D.3) es suficiente con pulsar sobre el enlace deseado; si por el contrario se está utilizando desde un móvil y aparece plegada, para acceder a la página de inicio basta con pulsar sobre el enlace “Inicio”, pero para acceder al resto de enlaces se debe pulsar sobre las tres rayas horizontales (Figura D.4).

En la barra de navegación aparece además un enlace para cerrar sesión.



Figura D.3: Barra de navegación en pantallas grandes



Figura D.4: Barra de navegación plegada en pantallas pequeñas

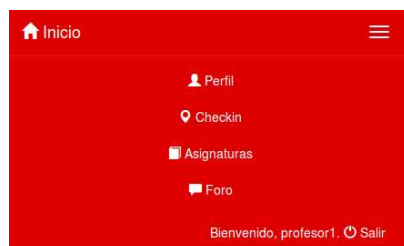


Figura D.5: Barra de navegación desplegada en pantallas pequeñas

Realizar el *check in* o registro en una clase

Figura D.6: Página de checkin para un alumno

The screenshot shows the URJCheckIn web application interface. At the top is a red navigation bar with links: Inicio, Perfil, Checkin, Asignaturas, and Foro. On the right of the bar, it says 'Bienvenido, profesor1.' and a 'Salir' button. Below the navigation bar, the page title is 'URJCheckIn' with the subtitle 'Realiza el Check In'. The main content area is a 'Check in' form. It includes a QR code icon and the text 'Leer código'. There are three input fields: 'Asignatura:' with a dropdown menu showing 'Señales y Sistemas', 'Código:' with a text input field, and 'Número de estudiantes:' with a text input field containing '0'. A green 'Check In' button is at the bottom of the form. At the very bottom of the page, there is a footer with the text: 'Para reportar problemas, informa a los administradores. | Ayuda. | URJCheckIn Jorge Bazaco Caloto © 2014'.

Figura D.7: Página de checkin para un profesor

Para acceder al recurso se puede pulsar sobre el enlace “Checkin” en la barra de navegación.

En la página aparece un formulario para registrar la asistencia a una clase. Se debe seleccionar la asignatura o seminario en el que se quiere realizar el *check in* e indicar el código manualmente o pulsando sobre “Leer código” y escaneando la imagen con el código *QR*.

En caso de tratarse de un profesor debe contar el número de alumnos en la clase e indicarlo en el campo “Número de estudiantes”. En caso de ser un alumno debe puntuar de 0 a 5 la clase en la que se está registrando y valorarla brevemente de forma anónima (se advierte que a pesar de ser anónimo para el profesor, en caso de realizarse insultos o amenazas, los administradores tienen acceso al usuario que realizó el comentario).

Finalmente, una vez rellenado el formulario, se debe pulsar sobre el botón “Check In”. En caso de preguntar si se quiere compartir la ubicación, se recomienda aceptar para realizar un *check in* más completo y fiable.

Ayuda y reporte de problemas

Figura D.8: Pie de página

En caso de tener alguna duda, al final de cada página aparece un enlace a una página de ayuda con respuestas a preguntas frecuentes.

URJCheckIn es una página para controlar la asistencia a clase, tanto de los alumnos como de los profesores. Además es una herramienta que permite a los profesores obtener un feedback sobre sus clases, para poder reforzar sus puntos débiles y mantener sus puntos fuertes; así como detectar anomalías en alguna asignatura o con algún profesor.

F.A.Q.s (Preguntas frecuentes)

¿Cómo registro mi asistencia a una clase?
Durante la clase debes acceder a la pestaña [CheckIn](#), elegir la asignatura o seminario de la clase a la que estás asistiendo e introducir el código de la clase o bien registrarte con tu localización o con ambas.

¿Cómo realizo el check in con mi localización?
Para realizar el check in con tu posición necesitas un dispositivo y un navegador que lo soporten. En caso de que tu dispositivo lo soporte se te pedirá permiso al pulsar sobre el botón de realizar el check in. Si crees que tu dispositivo lo soporta pero no te permite realizar el check in con la geolocalización, comprueba en la configuración del navegador o del dispositivo que la localización está habilitada.

¿Dónde encuentro el código para hacer el check in?
Este código se encuentra en el aula donde se realiza la clase durante la duración de la misma.

¿Hay alguna aplicación para el móvil?
La propia página está diseñada para funcionar como si se tratase de una aplicación de smartphone. Si quieres tenerla en el escritorio y

Figura D.9: Página de ayuda

Si se tiene algún problema, al final de cada página aparece además un enlace a una página para reportar el problema a los administradores, para que lo solucionen con la mayor brevedad posible.

Reporte de problemas

Reportar problema

Url:

Problema:

[Enviar](#)

Mis reportes

ID: #19	Estado: pendiente [solicitado hace 1 día, 21 horas]
Problema:	Creo que tengo asignado un perfil de alumno, podéis cambiármelo en el caso de que esté equivocado
ID: #18	Estado: pendiente [solicitado hace 1 día, 21 horas]
Problema:	Levante

Figura D.10: Página de reportes

Configuración

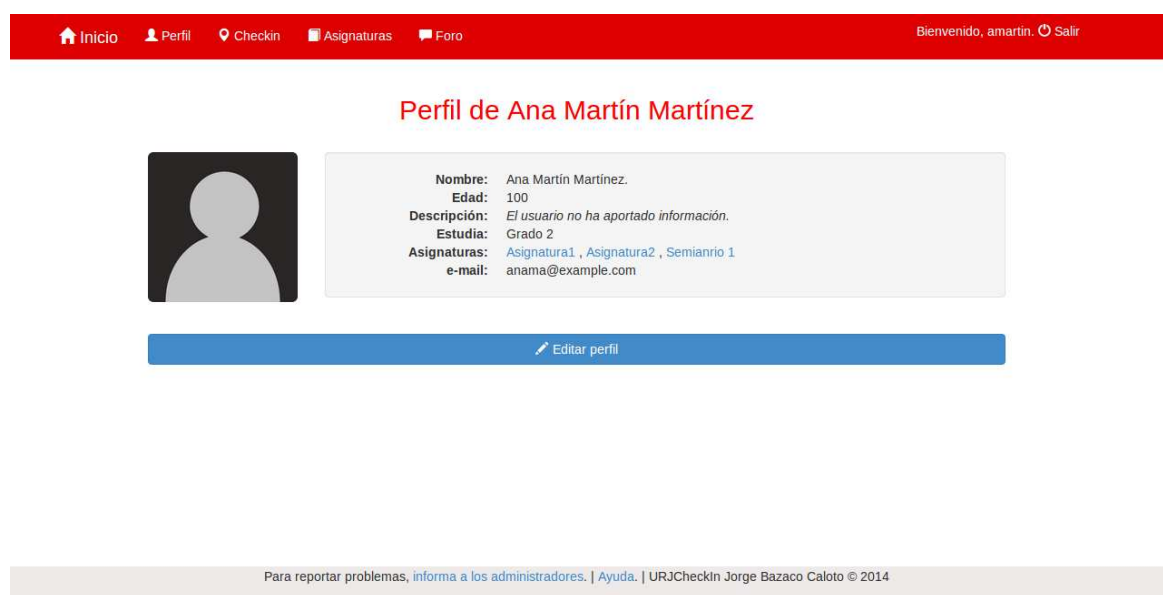


Figura D.11: Página de perfil

Para realizar algún cambio sobre su cuenta, se debe pulsar sobre el enlace “Perfil” de la barra de navegación. En dicha página, si no aparecen directamente los formularios de edición, se debe pulsar sobre “Editar perfil”.

Es posible modificar la foto de perfil, la edad y la descripción del usuario, la decisión sobre mostrar públicamente la dirección de correo electrónico, la dirección de correo electrónico y la contraseña.

Algunas de las anteriores configuraciones pueden no estar disponibles dependiendo del usuario registrado.

The image shows four stacked web forms for user configuration:

- Imagen de perfil:** Includes a file selection button 'Examinar...', a message 'No se ha seleccionado ningún archivo.', a green 'Cambiar foto' button, and a red 'Eliminar foto de perfil' button.
- Información del perfil:** Includes fields for 'Nombre:' (Profesor1 Apellido1P1 Ap), 'Edad:' (43), and 'Descripción:' (Soy profesor). It also has a checkbox for 'e-mail público:' which is checked. A green 'Guardar Cambios' button is at the bottom.
- Cambiar e-mail:** Includes an 'e-mail:' field with 'prof1@example.com' and a green 'Cambiar e-mail' button.
- Cambiar contraseña:** Includes three password fields: 'Contraseña antigua:', 'Contraseña nueva:', and 'Repetir contraseña:'. A green 'Cambiar contraseña' button is at the bottom.

Figura D.12: Formularios de configuración del usuario

Página de las asignaturas

The image shows the 'Asignaturas y seminarios' page with a red navigation bar at the top containing links: Inicio, Perfil, Checkin, Asignaturas, and Foro. The user is logged in as 'profesor1' and can click 'Salir'.

The main heading is 'Asignaturas y seminarios' with the subtitle 'Accede a la información de tus asignaturas'.

There are two main sections:

- Mis asignaturas:** A list with 'Señales y Sistemas' and 'Introducción a la programación'.
- Mis seminarios:** A list with 'Próximos seminarios' and 'Seminario de Accesibilidad'.

Below these is a section titled 'Últimos mensajes en tus asignaturas' showing three messages:

- Clase de Señales y Sistemas (05/06/2014 13:00):** 2 de Junio de 2014 a las 13:25. Por favor, traed la calculadora a esta clase.
- Clase de Introducción a la programación (03/06/2014 14:00):** 2 de Junio de 2014 a las 13:24. Realizaremos un pequeño test para que pueda saber que conocimientos previos tenéis.
- Clase de Señales y Sistemas (04/06/2014 13:00):** 2 de Junio de 2014 a las 13:22. Ésta es la primera clase, en ella realizaré una introducción de la asignatura.

Figura D.13: Página de las asignaturas

Pulsando sobre el enlace de asignaturas, se accede a una página que muestra las asignaturas y seminarios del usuario. Además, debajo de las asignaturas y seminarios aparecen los últimos mensajes realizados en las clases de dichas asignaturas y seminarios.

Es posible acceder a la información de las asignaturas y seminarios pulsando sobre ellos, y acceder a las clases de los comentarios a partir del enlace facilitado en cada comentario.

Apuntarse a un seminario

Desde la página de las asignaturas, al pulsar sobre “Próximos seminarios”, se accede a una página con el listado de los próximos seminarios relacionados con el grado o los grados del usuario. Al pulsar sobre algún seminario, se accede a su página.

Seminario de Accesibilidad
Gestiona el seminario y accede a las clases

Nombre:	Seminario de Accesibilidad
Grados:	<ul style="list-style-type: none">• Grado1• Grado 2
Profesor:	<ul style="list-style-type: none">• Profesor1 Apellido1P1 Apellido2P1
Fecha inicio/fin:	03-08-2014/16-08-2014
Estado:	0 clases pasadas y 0 restantes
Valoración media:	3/5
Asistencia profesor:	100%
Asistencia alumnos:	100%
Plazas ocupadas:	0/10
Descripción:	Seminario sobre la accesibilidad en las tecnologías para personas discapacitadas.

[Dejar de ser organizador](#)

[Comprobar asistencia](#) [Crear clases](#)

Figura D.14: Página de un seminario futuro para uno de sus organizadores

Inicio
Perfil
Checkin
Asignaturas
Foro

Bienvenido, estudiante1. Salir

Seminario de Accesibilidad

Gestiona el seminario y accede a las clases

Nombre: Seminario de Accesibilidad

Grados:

- Grado 1
- Grado 2

Profesor:

- Profesor1 Apellido1P1 Apellido2P1

Fecha inicio/fin: 03-08-2014/16-08-2014

Estado: 0 clases pasadas y 0 restantes

Valoración media: 3/5

Asistencia profesor: 100%

Asistencia alumnos: 100%

Plazas ocupadas: 0/10

Descripción: Seminario sobre la accesibilidad en las tecnologías para personas discapacitadas.

Apuntarme

Próximas clases
Últimas clases

Figura D.15: Página de un seminario futuro para un alumno no apuntado

En la página de un seminario que aún no ha empezado, en caso de ser un profesor aparece un botón para hacerse organizador o para dejar de serlo, mientras que en caso de ser un alumno, si está apuntado dispone de un botón para desapuntarse y si no está apuntado y quedan plazas libres se muestra un botón para apuntarse al seminario.

Crear un seminario (solo profesores)

Desde la página de las asignaturas, al pulsar sobre “Próximos seminarios”, se muestra para los profesores un formulario para organizar un nuevo seminario.



Figura D.16: Página de los seminarios para un profesor

En el formulario se debe indicar, en el siguiente orden, el nombre del seminario, el grado o los grados relacionados con el mismo (en algunos ordenadores y navegadores, se debe mantener pulsado el botón “ctrl” del teclado para seleccionar varios seminarios), la fecha de inicio, la fecha de finalización, el número de plazas y una breve descripción del seminario.

Una vez completado el formulario se debe pulsar sobre el botón “Crear” para crear el seminario. El creador del seminario es por defecto organizador del mismo (para dejar de serlo ver sección D). Además, el creador del mismo es el único que puede editarlo y eliminarlo desde la página del seminario pulsando sobre el botón “Gestión del seminario”, mostrando un formulario igual que el de la creación del seminario pero relleno con los datos del seminario (los administradores también pueden editarlo y eliminarlo, pero desde el panel de administración).

Página de una clase

Para acceder a las clases de una asignatura o seminario, se debe ir a la página de las asignaturas y seleccionar la asignatura deseada. En la página de la asignatura se muestran listas con las clases (diferenciadas según sean antiguas, futuras o se estén impartiendo en el mismo momento en que se solicita la página). Al pulsar sobre una de esas clases se accede a su página que contiene información de la clase y los comentarios de los profesores.

[Inicio](#) [Perfil](#) [Checkin](#) [Asignaturas](#) [Foro](#) Bienvenido, profesor1. [Salir](#)

Clase de Señales y Sistemas (04/06/2014 13:00)

Asignatura: Señales y Sistemas
Estado: sin realizar
Número de alumnos: 1
Fecha: 04-06-2014 13:00/13:59
Profesor:

- Profesor1 Apellido1P1 Apellido2P1

Aula: Aula 1, Edificio I

Comentarios del profesor

Comentar

Nuevos mensajes

 Profesor1 Apellido1P1 Apellido2P1
2 de Junio de 2014 a las 13:22
Esta es la primera clase, en ella realizaré una introducción de la asignatura.

Figura D.17: Página de una clase

En caso de ser profesor dispone de un formulario para publicar un mensaje sobre la clase. Para ello, se debe escribir el mensaje en el cuadro de texto y pulsar sobre “Comentar” para publicar el mensaje.

En caso de ser profesor de la clase o controlador, y haberse impartido la clase, se muestran los comentarios realizados por los alumnos que asistieron junto con su valoración.

Crear y editar clases (solo profesores)



Figura D.18: Página para crear una clase

Un profesor de una asignatura o seminario puede crear clases nuevas. Para ello, desde la página de la asignatura o seminario debe pulsar sobre el botón con el texto “Crear clases”. De esta forma accede a una página con un formulario para crear una clase. Además de ese formulario, la página presenta otro formulario para buscar aulas libres en la franja horaria en la que se quiera crear la clase y en un edificio concreto. Las clases creadas de esta forma serán clases extra (no obligatorias), que permiten mejorar el porcentaje de asistencia y que pueden ser eliminadas por el profesor.

Además, desde la página de una clase futura, abajo del todo, aparece un botón con el texto “Editar clase” que enlaza a una página para editar una clase. La página de edición es igual que la de creación de una clase pero tiene el formulario relleno con los datos de la clase. Además, si la clase es una clase extra no solo se puede editar, sino que también se puede eliminar.

Comprobar asistencia (solo profesores y controladores)

Desde la página de una asignatura los profesores de la misma y los controladores tienen dos botones. Uno con el texto “Comprobar asistencia” para mostrar un listado de los alumnos de la asignatura y su asistencia, y otro con el texto “Ver estadísticas” que muestra gráficas con la asistencia de los alumnos y los profesores y con la valoración de las clases de la asignatura.

Además, pulsando sobre una clase se accede a la página de la misma, la cual, en caso de haberse realizado, presenta en su descripción un enlace para ver los *check ins* realizados por los alumnos y profesores en dicha clase.

Publicar un mensaje en el foro

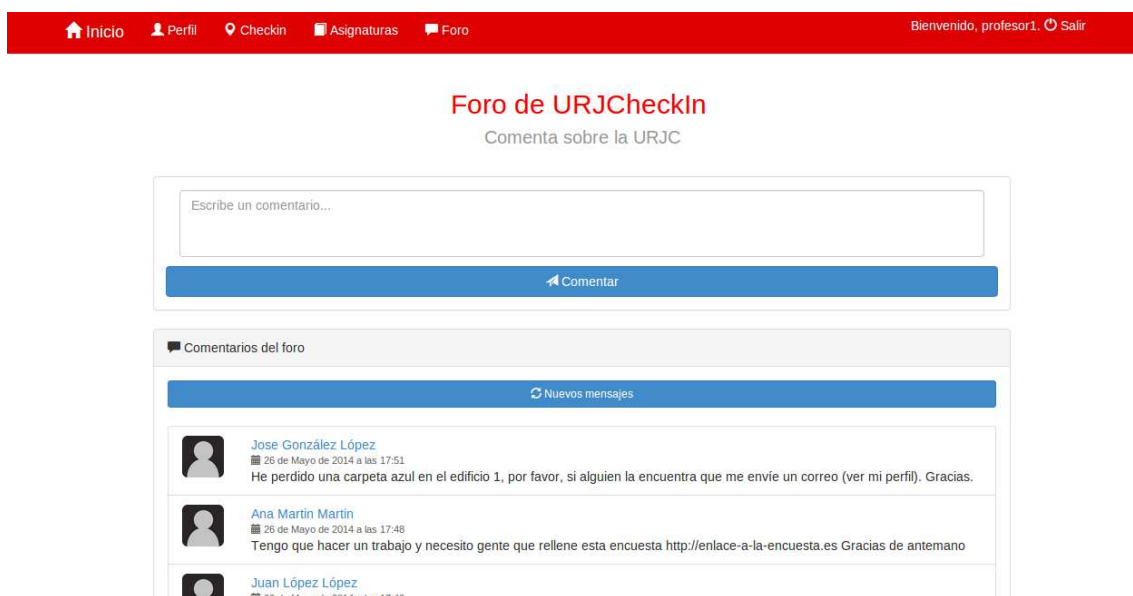


Figura D.19: Página del foro

Para acceder al foro se debe pulsar sobre el enlace “Foro”. En la página aparecen los comentarios realizados por los usuarios y sobre ellos un cuadro de texto para escribir un mensaje, el cual se publica pulsando sobre el botón “Comentar”.

Control de las asignaturas (solo controladores)

Inicio Perfil Control Códigos Admin Foro
Bienvenido, jorge. Salir

Revisa las estadísticas de las asignaturas

Filtrar
asignatura grado nombre profesor apellido profesor Seminarios y asignaturas
Ordenar por: Nombre orden inverso
Filtrar

Porcentaje de asistencia					
Asignatura	Grado	Profesores	Asist. (prof.)	Asist. (alum.)	Valor. media
Asignatura1	◦ Grado1	◦ Jose González López	6,94%	20,0%	3,4/5
Asignatura2	◦ Grado1	◦ Jose González López	0,0%	100%	3/5
Asignatura3	◦ Grado1 ◦ Grado 2	Ninguno	100%	100%	3/5

Figura D.20: Página para el control de asignaturas

Pulsando sobre el enlace “Control” de la barra de navegación, se accede a una página con un listado de asignaturas resumiendo su información y estadísticas. Para mayor información, se puede pulsar sobre el nombre de la asignatura para acceder a su página. También es posible seleccionar un profesor para ver su perfil.

Sobre el listado de asignaturas, se muestra un formulario que permite filtrar las asignaturas mostradas. Este formulario puede ser rellenado con el nombre de una asignatura, nombre del grado, nombre del profesor, apellido o apellidos del profesor, y seleccionar si se desean seminarios y asignaturas o solo uno de los dos, y si se desea ordenar por nombre de la asignatura, fecha de inicio o fecha de finalización de la misma y en orden lógico o inverso. Todos estos campos son opcionales, pudiendo realizarse cualquier combinación, y además no es necesario que tengan el nombre completo, por ejemplo, si una asignatura se llama “Asignatura de introducción”, se muestra escribiendo en el campo asignatura, por ejemplo, “de intro”, y lo mismo para el resto de campos.

Una vez relleno el formulario se debe pulsar sobre el botón “Filtrar”.

Obtener códigos de las clases (solo bedeles)

Inicio Perfil Control Códigos Admin Foro Bienvenido, jorge Salir

Accede a los códigos

Filtrar

Códigos del día: Edificio: Aula: Tipo: Desde las:

Hasta las:

Ordenar por: ☐ orden inverso

*Por defecto se mostrarán las clases desde este momento hasta el final del día

[Obtener códigos](#)

Aula 1, Edificio I	02-06-2014
Horario: 14:12-14:12 Asignatura: Asignatura1 Código: mqjysjnigfaxihjhjui	

Figura D.21: Página de los códigos de las clases

Pulsando sobre el enlace “Códigos” de la barra de navegación, se accede a una página con un formulario para indicar los códigos que se quieren visualizar.

Este formulario permite indicar un día, seleccionar un edificio, seleccionar un aula, seleccionar si se quieren códigos de asignaturas y seminarios o solo de uno de los dos, indicar una hora de inicio y una hora de finalización para generar una franja en la que tiene que estar la hora de comienzo de la clase, y un campo para indicar si se quieren ordenar los resultados por hora de inicio, aula o edificio y otro campo para indicar un orden lógico o inverso. Todos estos campos son opcionales, pudiendo realizarse cualquier combinación, y por defecto se muestran los códigos del día actual, con una franja de tiempo desde el momento de la petición hasta el final del día.

Una vez relleno el formulario se debe pulsar sobre el botón “Obtener códigos”.

Al imprimir los códigos no se visualiza lo mismo que se observa en la página del navegador (ver Figura D.22).



Figura D.22: Código de una clase para imprimir

Bibliografía

[1] Página para desarrolladores web de Mozilla.

<https://developer.mozilla.org/es/docs/Web>

[2] Página oficial de jQuery.

<http://jquery.com/>

[3] Página sobre JSON.

<http://www.json.org/>

[4] Libro de estándares web. *Varios autores.*

<http://mosaic.uoc.edu/ac/le/es/>

[5] Página oficial de Bootstrap.

<http://getbootstrap.com/>

[6] Página oficial de Python.

<https://www.python.org/>

[7] Página oficial de Django.

<https://docs.djangoproject.com/>

[8] Página oficial de Pylint.

<http://www.pylint.org/>

[9] Página con las normas de estilo de Python.

<http://legacy.python.org/dev/peps/pep-0008/>

[10] Página de Git.

<http://git-scm.com/>