ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

INGENIERÍA DE TELECOMUNICACIÓN-LADE

PROYECTO FIN DE CARRERA

SISTEMA DE CONTROL DE PRESENCIA IMPLEMENTADO EN SMARTPHONES VÍA GEOLOCALIZACIÓN

Autor : Félix Redondo Sierra
Tutor : Gregorio Robles Martínez

Curso Académico 2011/2012

Proyecto Fin de Carrera

Sistema de control de presencia implementado en smartphones vía geolocalización

Autor

Félix Redondo Sierra

Tutor

Gregorio Robles Martínez

La defensa del presente Proyecto Fin de Carrera se realizó el día de de , siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Fuenlabrada, a de de

Sólo es capaz de realizar los sueños el que,

cuando llega la hora, sabe estar despierto.

León Daudí

Agradecimientos

Para comenzar, me gustaría agradecer al lector de esta memoria el interés por la lectura de este proyecto, deseando que al cerrar la última página del libro sea capaz de hacerse una idea muy aproximada a la que he tenido yo desarrollando este proyecto de principio a fin.

Agradecer a Gregorio Robles Martínez, tutor de este proyecto, la idea, la paciencia y todo el tiempo dedicado junto a mí para sacar adelante todo el trabajo. También querría extender estos agradecimientos a mi compañero Jorge Fernández González, al cual también he molestado mucho y que, sin su ayuda, me habría costado más trabajo haber finalizado el proyecto.

Quisiera dar las gracias también a los alumnos y profesores que han participado en las pruebas de este Proyecto Fin de Carrera. Sin su ayuda y su paciencia no habría sido posible mejorar y perfeccionar el proyecto. Gracias por su interés y la amabilidad mostrada durante todo el proceso.

También agradecer a mi novia Jenifer, ingeniera informática, su constante apoyo y su ayuda en todo el proceso de trabajo del proyecto. Su experiencia y su espíritu crítico me han ayudado mucho para mejorar todo el trabajo desarrollado.

Y por último, y no por ello menos importante, agradecimientos infinitos para mi familia, mi *piña*, por el eterno sacrificio, cariño y apoyo que me han otorgado desde mi primer día de vida. Sin ellos jamás habría podido llegar a este punto. Y entre todos ellos en especial a ti, papá, porque aunque nos dejaste hace más de dos años y no estás aquí conmigo, te desviviste siempre para que llegara este día y sé que a ti te hacía mucha ilusión verme aquí. Sé que desde ahí arriba estarás muy orgulloso, tanto como yo de ti porque has sido el mejor padre del mundo.

Resumen

El desarrollo tecnológico que se ha experimentado en el ámbito de la telefonía móvil en la última década con la aparición y evolución de los *smartphones*, el auge de las redes de datos en dispositivos móviles y el interés de la sociedad en todas estas nuevas tecnologías, ha provocado un desarrollo importante tanto en los sistemas operativos que trabajan en estos dispositivos como en las aplicaciones que se desarrollan. Además, el crecimiento de las redes sociales y su uso tan habitual en la actualidad hace que la sociedad se encuentre más conectada que nunca. Todo este conjunto de causas ha provocado que los desarrolladores hayan abierto una nueva vía de investigación hasta hace poco abandonada por la intromisión en la privacidad que provocaba en el usuario. Esta vía de desarrollo está centrada en dos características principales: conexión con la red social y geolocalización de los usuarios.

Basándonos en esta disyuntiva, este Proyecto Fin de Carrera tiene como objetivo principal el desarrollo e implementación de un sistema automático de control de asistencia y geolocalización de eventos académicos. Esta herramienta permite tanto a profesores como alumnos realizar su control de asistencia a las clases o conferencias sin necesidad de firmar en ningún listado como hasta ahora. El proceso, automatizado, permite desde un *smartphone* realizar el proceso de control de asistencia y otorga una herramienta que permite la interacción social por parte de los miembros de un mismo evento.

El desarrollo e implementación está basado en una arquitectura cliente-servidor: por una parte, una aplicación para *smartphones* con sistema operativo *Android* que será utilizada para realizar el proceso de control de asistencia. Por otro lado, una aplicación *web* desarrollada en *Django* que actuará como servidor y almacenará y mostrará toda la información.

Índice general

1 Introducción **1**

1.1 Motivaciones 1

1.2 Objetivos 6

1.3 Otros proyectos de Sistemas de Geolocalización 8

1.3.1 *Foursquare* 8

1.3.2 *Do it Social* 9

1.4 Estructura de la Memoria 10

2 Estado del Arte **13**

2.1 Aplicaciones cliente-servidor 13

2.1.1 Definiciones 13

2.1.2 Funcionamiento 15

2.2 *Python* y *Django* 16

2.2.1 *Python* 17

2.2.2 *Django* 18

2.3 *Android* 20

2.3.1 Conceptos necesarios para entender una aplicación desarrollada en *Android* 24

2.3.1.1 ¿Qué es *Eclipse*? 25

2.3.1.2 ¿Qué elementos componen una aplicación *Android*? 25

2.4 *Códigos QR* 28

3 Diseño del sistema **31**

3.1 Introducción 31

3.2 Diseño del servidor 33
3.2.1 Diseño del modelo de datos 33
3.2.2 Diseño de la aplicación *web* 40
3.2.2.1 Modelo 41
3.2.2.2 Vista 44
3.2.2.3 Controlador 47
3.3 Diseño del cliente 48

4 Implementación del sistema 51
4.1 Introducción 51
4.2 Implementación del servidor 52
4.2.1 Estructura de directorios y ficheros 52
4.2.2 Explicación del Modelo Vista Controlador implementado 54
4.2.2.1 Modelos 55
4.2.2.2 Controladores 61
4.2.2.3 Vistas de la interfaz gráfica 73
4.3 Implementación del cliente 85
4.3.1 Estructura de directorios y ficheros 85
4.3.2 Explicación de la aplicación 87

5 Despliegue y resultados 95
5.1 Introducción 95
5.2 Primer Experimento 96
5.2.1 Resultados 97
5.2.2 Conclusiones del Primer Experimento 98
5.3 Segundo Experimento 98
5.3.1 Resultados 100
5.3.2 Conclusiones del Segundo Experimento 101

6 Conclusiones y Futuras Líneas de Trabajo 103
6.1 Conclusiones 103
6.2 Futuras Líneas de Trabajo 105

A Resultado de los experimentos 109

B Presupuesto del Proyecto **113**
B.1 Costes Materiales 113
B.2 Costes de los Recursos Humanos 114
B.3 Coste Total 116

C Planificación del Proyecto **117**
C.1 Diagrama de Gantt 117

Bibliografía **120**

D Glosario **123**

Índice de figuras

2.1 Diagrama de flujo de la interacción de un servidor interactivo y un cliente 15
2.2 Capturas de pantalla de la *Android Market* desde un dispositivo móvil 21
2.3 Ciclo de vida de una *Activity* en *Android* 26
2.4 Ejemplo de estructura de un código QR 29

3.1 Esquema general de diseño del sistema 32
3.2 Diagrama relacional de la base de datos del servidor 39
3.3 Diagrama de relaciones entre el Modelo, la Vista y el Controlador 41

4.1 Estructura de directorios y ficheros del proyecto *Django* implementado para este sistema 52
4.2 Diagrama relacional de los modelos desarrollados en *Django* . 57
4.3 Plantilla original de la que parte el diseño de la interfaz del servidor de *GeoURJC* 74
4.4 Captura de imagen de la vista *login.html* 75
4.5 Captura de imagen de la vista *manual.html* 76
4.6 Captura de imagen de la vista *newuser.html* 77
4.7 Cuatro capturas de imagen de las secciones de la vista *menu.html* 78
4.8 Dos capturas de imagen de la vista *checkin.html* 79
4.9 Tres capturas de las secciones de la vista *signature.html* . . . 80
4.10 Cuatro capturas de imagen de las secciones con permisos de administrador de la vista *signature.html* 81

4.11 Cuatro capturas de imagen de las secciones de la vista *class.html* 82
4.12 Dos capturas de imagen de las secciones de la vista *profile.html* 83
4.13 Cuatro capturas de imagen del panel de Administración de *GeoURJC* 84
4.14 Estructura de directorios y ficheros del cliente programado para *Android* 86
4.15 Captura de pantalla de la *Activity Login* 88
4.16 Captura de pantalla de la *Activity CheckIn* para un usuario profesor 90
4.17 Captura de pantalla de la aplicación *Barcode Scanner* 91
4.18 Captura de pantalla del menú de ajustes 92
4.19 Dos capturas de pantalla del menú de Configuración y Acerca de... 93

C.1 Diagrama de Gantt 118

Índice de cuadros

B.1 Coste de los recursos materiales necesarios para el proyecto . . 114
B.2 Coste de los recursos humanos involucrados para el proyecto . 115
B.3 Coste total de los recursos necesarios para realizar el proyecto 116

Capítulo 1

Introducción

1.1 Motivaciones

La principal motivación del autor de este Proyecto Fin de Carrera al escogerlo fue poder aplicar los conocimientos adquiridos durante todos los años de estudio en un proyecto específico de un modo más riguroso y con cierta similitud a lo que sería el trabajo en una empresa. Elegir un área de desarrollo afín a sus gustos en el que el poder implementar una idea desde cero y plasmarla en un producto listo para ser utilizado. Asimismo, este proyecto otorgaba la posibilidad de aprender a desarrollar aplicaciones en *smartphones* y permitía poner en práctica los conocimientos en desarrollo de aplicaciones *web* de un tamaño más o menos considerable que se adquirieron durante el aprendizaje académico.

Los sistemas de geolocalización implementados en *smartphones* son una herramienta para un futuro a corto plazo. Desde un año a esta parte, se ha producido una importante proliferación de aplicaciones que cada vez más interactúan con el usuario y comparten más información privada al resto de su red social, destacando por encima del resto compartir la geolocalización del usuario.

Es inevitable comprobar cómo, año tras año, la telefonía se está abriendo paso en líneas mucho más profundas que las que trabajaban hasta hace poco tiempo: de ser un medio de comunicación que permitía la comunicación per-

sona a persona se ha convertido en un medio que permite la interacción social y conectividad entre grupos, además de un medio de comunicación persona a persona. Las posibilidades que otorgan los nuevos dispositivos (por ejemplo, cámaras de fotos de gran calidad, incorporación de antena *GPS*, conexión de datos, etc.) hace que el desarrollo de aplicaciones sea mucho más exigente que en el pasado. Dentro del desarrollo de aplicaciones para dispositivos móviles, la gran mayoría de informes y estudios indican que la tendencia del mercado en el próximo año 2012 sea que *Android* disponga de la mitad de la cuota del mercado dentro del mundo de los *smartphones*.

Otro aspecto a resaltar es el surgimiento de las llamadas *tablets*. Estos dispositivos, destinados a sustituir en un medio plazo a los ordenadores personales, incorporan en su gran mayoría sistemas operativos de *smartphones*. En el caso de este Proyecto Fin de Carrera, es una ventaja puesto que el sistema operativo *Android* también es el entorno que domina dentro del mundo de las *tablets* con más del 60 % de la cuota de mercado existente.

Por tanto, y como conclusión a todo lo indicado anteriormente, si se analiza el mercado socialmente hablando (esto es, en cuanto a la potencialidad de uso de la aplicación), desarrollar un Proyecto Fin de Carrera para un *smartphone* es sinónimo de llevarlo a cabo en el entorno *Android*. Si se analiza desde la óptica del desarrollador, *Android* otorga una serie de facilidades mucho más interesantes que otros sistemas operativos móviles disponibles. A continuación, se realiza un listado de aquellas más importantes:

- **Programación en Java.**

  El entorno de programación que ofrece *Android* está basado en uno de los lenguajes más utilizados en el presente y en un futuro cercano. Permite la incorporación de multitud de proyectos distribuidos bajo licencias de libre uso y dispone de una *API* (*Application Programming Interface*) muy potente llamada *SDK* (*Software Development Kit*) que permite, sin tener muchos conocimientos sobre programación para dispositivos móviles, desarrollar aplicaciones muy interesantes para ser compartidas con la garantía de que funcionarán en prácticamente todas las versiones del sistema operativo que vayan apareciendo próximamen-

te.

- **Código abierto.**

  El hecho de que *Android* esté liberado con licencia *Apache* y código abierto lo convierte en un sistema operativo totalmente libre para que un desarrollador pueda modificar su código y mejorarlo. A través de esas mejoras puede publicar el nuevo código, y con él, ayudar a mejorar el sistema operativo para futuras versiones sin depender de fabricantes u operadoras. Del mismo modo, al ser código abierto garantiza que en caso de haber un error, éste pueda ser detectado y reparado con mayor presteza al no existir ninguna traba legal para indagar en su interior ni depender de nadie para pedir autorización a su cambio.

- **Comunidad de desarrolladores.**

  Existen cientos de comunidades de desarrolladores en Internet que se ayudan mutuamente para el desarrollo de nuevas aplicaciones y entornos. *Android* no solo cuenta con la comunidad más grande mundial de desarrolladores sino también el mayor movimiento de éstos con multitud de eventos, concursos, competiciones y reuniones así como múltiples vías de comunicación como foros y chats oficiales para fomentar la participación y la colaboración para encontrar mejoras e ideas para futuras versiones.

El desarrollo de aplicaciones *web* como medio de interacción de un proyecto con los usuarios es un aspecto que se encuentra completamente en auge en la actualidad. Al igual que ocurre con el desarrollo para dispositivos *smartphone*, el desarrollo de sitios *web* ha experimentado un crecimiento muy importante en los últimos años. Los usuarios del producto son cada vez usuarios más exigentes que piden, por una parte, un nivel de información mucho mayor y, por otra, un dinamismo y rapidez suficiente para mostrar todo el contenido que desean. La implantación del entorno *web* como una de las principales fuentes de trabajo y de uso tanto para las personas como para las organizaciones, provoca que cada vez sea mucho más complejo el desarrollo de las mismas. Esto hace que, debido a las necesidades, surjan nuevas herramientas

que faciliten la labor de implementación de estas aplicaciones *web*, del mismo modo que anteriormente se comentaba con el sistema operativo *Android*.

Aunque en los años de la carrera se estudia alguna de las herramientas utilizadas a tal efecto (*Ruby on Rails*), desde la óptica del alumno no se llega a aprender con la profundidad necesaria como para hacer proyectos de cierta envergadura. Dentro de todas las alternativas que se disponen a día de hoy, la herramienta elegida fue *Django*. Tanto *Ruby on Rails* como *Django* son las alternativas más utilizadas en el desarrollo de aplicaciones *web* en la actualidad. El motivo fundamental por el que se ha elegido *Django* es el auge que está teniendo actualmente. Además, complementado a este incremento de uso de aplicaciones *web* en la red, el desconocimiento por parte del autor en el desarrollo de aplicaciones *web* con herramientas de este tipo, permitía recopilar una serie de conocimientos importantes para el futuro.

Con todo lo anterior, una aplicación interesante de todas las tecnologías descritas anteriormente es la realización de este Proyecto Fin de Carrera, consistente en el desarrollo e implementación de un sistema automático de control de asistencia y geolocalización orientado a los miembros de cualquier organización académica. El sistema que se utiliza en la actualidad es un sistema anticuado para las tecnologías que existen hoy en día. Las principales ventajas que ofrece son las siguientes:

- **Automatización del proceso de control de asistencia.**

  El sistema automáticamente genera todas las clases, claves y códigos QR necesarios para poder llevar a cabo el proceso de control de asistencia y geolocalización. Asimismo, genera las conexiones necesarias para que cualquier usuario, desde su cuenta personal, acceda a toda la información de cada una de las asignaturas a las que esté inscrito. Además, vigila toda la información para que el administrador del sistema pueda analizar todo el contenido de una manera más rápida. Esto otorga un mayor dinamismo y una reducción considerable del tiempo empleado en preparar y administrar todo el sistema.

- **Reducción de los recursos humanos.**

Debido a la característica anterior, se necesitará un número menor de personas que lleven a cabo todo el proceso de control de eventos académicos, reduciéndose únicamente a las personas que lleven a cabo la administración del sistema.

- **Disponibilidad absoluta en cualquier parte del planeta.**

  Debido a las tecnologías empleadas (geolocalización a través de redes y localización en interiores mediante el uso de códigos QR), es posible realizar el control de asistencia en cualquier parte teniendo conexión de red de datos. El lugar donde se realice el evento ya no será un problema. Además, estos datos serán accesibles desde cualquier punto del globo gracias a la interfaz de la aplicación *web* implementada.

- **Notificaciones a distancia.**

  En la actualidad, cada profesor de cada clase de la universidad tiene la obligación de firmar todos los días para atestiguar que efectivamente ha realizado la clase. Es algo recurrente que al profesor, por cualquier motivo, se le olvide firmar su clase. Esta herramienta permite llevar todo este control de una manera mucho más dinámica que la que se realiza en la actualidad. Para evitar este problema, la herramienta incorpora un notificador que avisa de manera personalizada a cada miembro de la organización de la obligación de llevar a cabo el *check-in* durante el transcurso de la clase.

Así pues, disponiendo de una serie de recursos como son los *smartphones*, que cuentan con la innovadora plataforma *Android* y con unas capacidades aún por explotar, unido a las ya conocidas facilidades de la *web 2.0*, se podrá automatizar de una forma considerable este sistema y reducir todo tipo de costes que lleva asociados, haciendo de una obligación para los usuarios algo más dinámico y provechoso desde el punto de vista del desarrollo tecnológico.

1.2 Objetivos

El objetivo global de este Proyecto Fin de Carrera es el de desarrollar e implementar un sistema automático de control de asistencia y geolocalización de eventos académicos de una organización. Puesto que el objetivo principal es muy ambicioso, es necesario para lograrlo dividirlo en subobjetivos o hitos que se deben ir cumpliendo para, finalmente, cumplir dicho objetivo global. A continuación se realiza una enumeración de los objetivos alcanzados con este Proyecto Fin de Carrera:

1. **Diseño e implementación de una base de datos que permita almacenar toda la información que se manejará por el sistema.**

   Este objetivo es uno de los principales para que el sistema actúe de la forma más óptima posible. Es indispensable almacenar toda la información necesaria para poder llevar a cabo el control de asistencia y geolocalización y el posterior tratamiento de toda la información. Por ello, es indispensable diseñar el contenido de la base de datos de tal forma que las relaciones entre sus diferentes campos permitan que el sistema sea escalable a cuántos usuarios vayan a utilizarlo.

2. **Diseño e implementación de la aplicación *web* que generará las clases de las asignaturas para almacenar la información del control de asistencia.**

   Dotar al sistema de las herramientas suficientes para generar toda la información necesaria para llevar a cabo el proceso de almacenamiento de información y los mecanismos de análisis necesarios para posteriormente poder mostrar, a través de su interfaz *web*, toda esta información a los usuarios y administradores del sistema.

3. **Diseño e implementación de la Aplicación para el sistema operativo *Android* que permita al usuario guardar su asistencia.**

   Desarrollar una aplicación para terminales de telefonía móvil con sistema operativo *Android*, a través de la cual puedan conectarse al sistema

y poder guardar su posición y asistencia a cualquier evento académico de la organización y configurar el notificador para que le avise de manera personalizada de sus eventos.

4. **Búsqueda de mecanismos de geolocalización para *smartphones* en interiores de edificios.**

   Debido al problema que supone no contar con un sistema de geolocalización fiable para la localización en interiores de edificios, es indispensable encontrar un mecanismo de seguridad que garantice al sistema que la información referente a la localización de usuarios sea feaciente y verdadera. Para ello, hay que buscar diferentes alternativas existentes a día de hoy y ver cuál de ellas es más interesante de implementar.

5. **Comprobar el correcto funcionamiento del sistema en un escenario de pruebas real y valorar los comentarios de los usuarios que realicen pruebas con el sistema.**

   Una vez finalizado todo el proceso de desarrollo e implementación es de vital importancia probar el sistema con gente ajena al Proyecto Fin de Carrera. Con ello, se obtendrá información técnica que permitirá pulir todos los errores que puedan surgir en el uso del sistema y, además, se podrán recabar opiniones sobre la percepción que los usuarios tienen del sistema, con tal de mejorar aquellos aspectos que no sean de su agrado o que sean mejorables.

Por tanto, además de estudiar y trabajar a fondo sobre las tecnologías que se han ido refiriendo hasta ahora, el objetivo principal de este Proyecto Fin de Carrera consistirá en el desarrollo, por un lado, de una aplicación *web* implementada a través de un *framework* de desarrollo, *Django*, que hará las labores de servidor de almacenamiento de toda la información y de interfaz *web* para acceder a todos los contenidos disponibles por parte de los usuarios. Por otro lado, una aplicación para dispositivos móviles con sistema operativo *Android* que permitirá notificar a cada usuario en tiempo real cuándo tiene una clase y debe realizar el control de asistencia y, asimismo, realizar automáticamente todo el proceso de almacenamiento de su asistencia al evento

académico que esté disponible en esos momentos.

1.3 Otros proyectos de Sistemas de Geolocalización

En este apartado se procede a analizar algunos sistemas implementados similares en cuanto a la funcionalidad que se quiere dar por parte de este Proyecto Fin de Carrera aplicados en áreas parecidas. Los proyectos que se van a explicar a continuación están más bien orientados a áreas relacionadas con redes sociales mientras que este Proyecto Fin de Carrera está centrado más en áreas académicas y de ámbito organizativo.

1.3.1 *Foursquare*

Foursquare[1] es un servicio basado en localización *web* aplicada a las redes sociales. Basada en la geolocalización de los usuarios, *Foursquare* hace que las ciudades sean más fáciles e interesantes de explorar. El servicio fue creado en 2009 por Dennis Crowley y Selvadurai Naveen. La aplicación, dirigida principalmente a *smartphones*, cuenta con versiones para *iOS*, *Android*, *Windows Phone*, *Symbian* y *Blackberry*.

La idea principal de esta plataforma móvil es la de realizar *check-ins* a través de la aplicación móvil conectada a una red de datos o a través de *SMS* de un *smartphone* en un lugar específico dónde uno se encuentra y así permitir informar sobre lugares que visitar y encontrar sugerencias sobre lugares cercanos. Con cada *check-in*, el usuario va ganando puntos. Por ejemplo, por descubrir nuevos lugares los usuarios son recompensados con una especie de medallas llamadas *badges*, o los *mayorships*, que son medallas ganadas por las personas que hacen más *check-ins* en un cierto lugar.

Aunque *Foursquare* basa su funcionamiento a través de un *smartphone*, este sistema incorpora distintos servidores que trabajan en paralelo y se encargan de almacenar toda la información y responder a las peticiones

[1]Página *web* oficial de Foursquare: `http://es.foursquare.com`

realizadas por los clientes.

Al igual que otras redes sociales como *Facebook* o *Twitter*, *Foursquare* tiene una *API* que permite a los desarrolladores llevar a cabo sus ideas sobre esta plataforma. Los desarrolladores han llevado a cabo ya nuevas funcionalidades para hacer *check-in*, juegos y visualización de datos interesantes. Esto permite, como ocurre con el sistema operativo *Android*, una continua actualización de contenidos y mejoras que consigue que cada vez más usuarios sigan utilizando el sistema después de dos años desde su creación.

1.3.2 *Do it Social*

Do it Social[2] es un sistema de control, información y gestión en tiempo real orientado principalmente para universidades y escuelas de negocios que tiene como objetivo mejorar la calidad del servicio ofrecido así como promocionar las diferentes actividades que se realizan. Cada usuario dispone de una tarjeta, pulsera o acreditación con tecnología *RFID* (*Radio Frequency IDentification*) que debe hacer pasar cada vez que accede a un evento de la organización por un puesto especial que se encuentra a la entrada. Este puesto detecta al usuario y manda la información a los servidores que almacenan sus *check-in*s. Además, opcionalmente, se puede configurar la cuenta del usuario para que el servidor publique automáticamente un mensaje en sus redes sociales indicando su asistencia al evento.

Este sistema permite conocer en tiempo real el número de asistentes a un acto determinado. Además, otorga la posibilidad de enviar encuestas a todos los asistentes si fuera necesario para completar el evento. Otra de las herramientas destacables de *Do it Social*, es la posibilidad de asociar documentos a un evento determinado, por ejemplo una clase, haciendo que el sistema automáticamente mande ese documento a todos los usuarios a través de su correo electrónico (por ejemplo, los enunciados de unos ejercicios o el documento que incluye las transparencias que se estudiarán durante la clase).

Como se observa, este sistema es muy parecido al que se desarrollará en este Proyecto Fin de Carrera, con algunas diferencias considerables. Aunque

[2]Página *web* oficial de *Do it Social*: `http://www.doitsocial.net`

tiene muchos puntos positivos, uno de los puntos débiles de *Do it Social* es el alto coste de cada puesto de identificación y mantenimiento (600€/puesto y 300€/mes de mantenimiento de los dispositivos). El sistema que se propone en el presente Proyecto Fin de Carrera es una alternativa mucho menos costosa de mantener y con la que se pueden obtener resultados muy similares a los descritos para *Do it Social*. De todos modos, es un sistema muy interesante y que otorga publicidad y éxito de difusión del evento en las redes sociales pioneras en la actualidad (*Twitter*, *Facebook*, *Foursquare* o *LinkedIn*).

1.4 Estructura de la Memoria

A modo de pequeño resumen sobre los contenidos disponibles en esta memoria que faciliten su lectura, a continuación se lleva a cabo un listado de todos y cada uno de los capítulos que la componen y unas pinceladas sobre su contenido:

- ***Capítulo 1: Introducción.***

  En el primer capítulo de la memoria se presentan las motivaciones que han hecho que se vaya a llevar a cabo el desarrollo del presente Proyecto Fin de Carrera, así como los objetivos marcados para su consecución final. Además, se comentan algunos proyectos similares que se utilizan en la actualidad y, finalmente, se dan unas pinceladas sobre la estructura y contenido de cada uno de los capítulos de la memoria.

- ***Capítulo 2: Estado del Arte.***

  En el capítulo 2 de la memoria se analizan las características más importantes de las tecnologías que se utilizan dentro de este Proyecto Fin de Carrera, siendo recomendable complementar la lectura de las cuestiones técnicas de dichas tecnologías con la lectura del glosario de términos que se encuentra al final de esta memoria.

- ***Capítulo 3: Diseño del Sistema.***

  En el tercer capítulo de la memoria se presenta con todo detalle el diseño del sistema que se ha ido explicando conceptualmente en los

anteriores capítulos. Comenzando con un análisis general del sistema, se estudiarán todos y cada una de las características fundamentales de cada uno de los componentes que lo conforman.

- ***Capítulo 4: Implementación.***

  En el capítulo 4 de la memoria se desarrolla la implementación del sistema de tal forma que se consigan los subobjetivos o hitos explicados en el Apartado 1.2 y, por tanto, el objetivo global descrito en el mismo capítulo. En definitiva, la implementación de la aplicación *web* (servidor e interfaz *web*), así como su base de datos y la aplicación desarrollada para el sistema operativo *Android*.

- ***Capítulo 5: Despliegue y Resultados.***

  En el quinto capítulo de la memoria se explican las acciones a realizar para llevar a cabo el despliegue del sistema y su puesta en funcionamiento para llevar a cabo dos fases de pruebas por parte de usuarios ajenos al Proyecto Fin de Carrera. Asimismo, se exponen los resultados obtenidos gracias a la colaboración de dichos usuarios.

- ***Capítulo 6: Conclusiones y Futuras Líneas de Trabajo.***

  En el sexto y último capítulo, a modo de cierre de la memoria, se muestran una serie de conclusiones sobre el presente Proyecto Fin de Carrera y se desarrollan posibles líneas futuras de trabajo que permitan mejorar y perfeccionar la implementación del sistema que aquí se presenta.

- ***Bibliografía.***

  En este apartado se presentan las referencias bibliográficas, artículos, informes, definiciones, etc., consultados durante todo el proceso de realización del Proyecto Fin de Carrera.

- ***Apéndices.***

  En el siguiente apartado se presentan los apéndices que se han creído necesario incluir para clarificar algunos aspectos de la memoria: por un lado, un primer apéndice que recoge los resultados obtenidos en las

encuestas realizadas a los participantes en la fase de pruebas del Proyecto Fin de Carrera. Un segundo apéndice que recoge el presupuesto necesario para llevar a cabo el proyecto. Por último, un tercer apéndice que incluye toda la planificación de este Proyecto Fin de Carrera.

- ***Glosario.***

  En el último apartado de la memoria que se presenta, se incorpora un breve glosario de términos que intervienen en mayor o menor medida en la realización del presente Proyecto Fin de Carrera. Con ello se pretende que el lector, aunque no tenga conocimientos en las materias que se tratan en esta memoria, pueda hacerse una idea rápida de aquellos conceptos que no entienda con claridad.

Capítulo 2

Estado del Arte

En este capítulo se procede a explicar con cierto nivel de detalle las características de las principales tecnologías utilizadas en el desarrollo e implementación del presente Proyecto Fin de Carrera. Antes de continuar con la lectura, se recomienda al lector acompañar este capítulo junto con el Glosario de términos disponible en el Apéndice D que se anexa al final de la presente memoria.

2.1 Aplicaciones cliente-servidor

Uno de los principales dilemas para todo desarrollador de aplicaciones que deben trabajar en entornos de red distribuidos es qué arquitectura utilizar para que los dispositivos se comuniquen entre sí. Esta es la primera pregunta que debe responder todo diseñador antes de implementar cualquier tipo de mecanismo. La historia del desarrollo de redes distribuidas atribuye mayor importancia por encima de cualquier otro diseño al modelo cliente-servidor [Márquez-García(1996)].

2.1.1 Definiciones

El modelo cliente-servidor, como su propio nombre indica, está compuesto por dos entidades lógicas: el servidor y el cliente. La separación de ambas

entidades es una separación de tipo lógica. Antes de avanzar con algo más de detalle, es indispensable explicar qué es un servidor y qué es un cliente:

- ***Servidor.***

  El servidor es la parte con mayor lógica y peso en un sistema diseñado para un entorno cliente-servidor. El servidor es aquel elemento de la aplicación que proporciona los recursos y servicios solicitados por los clientes de manera remota.

  Existen diferentes diseños de servidores en la actualidad. Los dos diseños principales son los servidores interactivos y los servidores concurrentes:

  - ***Interactivos.***

    El servidor en el momento que le llega la petición de servicio, la atiende. El problema de este tipo de servidores es que, si existen muchas peticiones o el servidor es lento, puede originar tiempos de espera largos.

  - ***Concurrentes.***

    El servidor recoge la petición de servicio, pero en lugar de atenderla, crea otros procesos que se encargan de ejecutar el código necesario para atender la petición (son lo que se llaman procesos hijo). Este modelo de servidor sólo se puede aplicar en máquinas con sistemas multiproceso. Con este tipo de servidor, el tiempo de espera se reduce drásticamente ya que aumenta la velocidad del sistema.

- ***Cliente.***

  El cliente es aquella parte lógica que se encarga de solicitar servicios y recursos al servidor para ofrecer una serie de contenidos al usuario final de la aplicación.

Como se puede comprobar, cada dispositivo tiene una serie de obligaciones organizadas siguiendo el patrón consumidor/productor. En el paradigma que

nos atañe, el cliente consume los servicios que le proporciona el productor, llamado servidor.

2.1.2 Funcionamiento

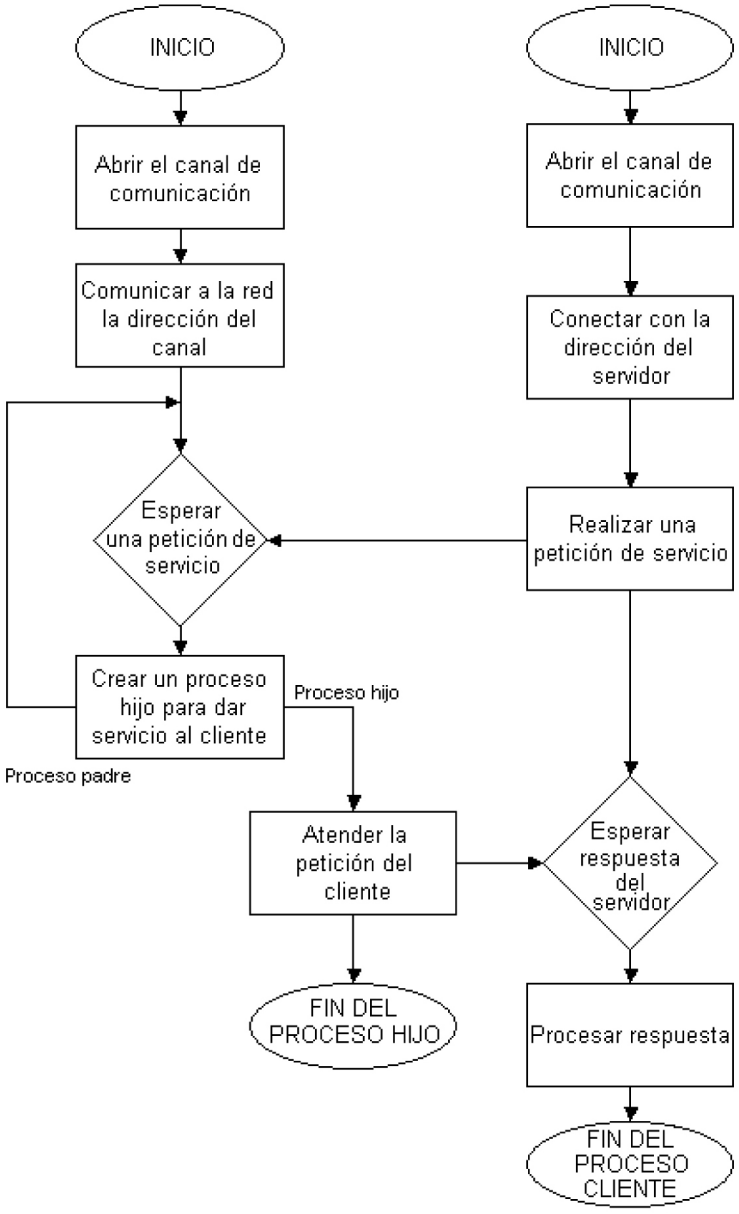


Figura 2.1: Diagrama de flujo de la interacción de un servidor interactivo y un cliente

El funcionamiento [González(2010)] del modelo cliente-servidor es el siguiente: partiendo del hecho de que un servidor será un proceso ejecutándose en una determinada máquina de la red (se explicará con un proceso por simplificar el problema, pero realmente puede estar compuesto por distintos procesos ejecutándose en diferentes máquinas), su principal objetivo será el de gestionar diversos servicios y recursos que son solicitados por los dispositivos clientes y devolverles los contenidos solicitados a modo de respuesta.

En la Figura 2.1 se observa el diagrama de flujo de funcionamiento de un servidor concurrente y un cliente. No le será difícil al lector descubrir cuál es el diagrama de flujo simplificado en el caso de utilizar un servidor interactivo.

Para hacer posible esta gestión, el servidor se encontrará en un estado de espera continua hasta que reciba una solicitud de servicio. En el momento en el que esa solicitud se produzca por parte de uno de los clientes de la aplicación, el servidor comenzará a ejecutar las acciones necesarias para atenderla, volviendo al estado de espera una vez se haya terminado de ofrecer el servicio solicitado.

El cliente, por el contrario, se encontrará en funcionamiento siempre que el usuario final de la aplicación lo esté ejecutando. Cuando el usuario final le solicite a su dispositivo cliente cierta petición de recursos o servicio, la entidad cliente automáticamente mandará una petición del servicio correspondiente que será atendida por el servidor.

2.2 *Python* y *Django*

Una vez elegida la arquitectura del sistema que se desea desarrollar, el segundo punto importante es la elección de las tecnologías que se van a emplear para el desarrollo del servidor que almacenará toda la información y, además, ofrecerá la interfaz gráfica para mostrarla de manera útil a los administradores y usuarios. Para ello se ha elegido el *framework* de desarrollo *web Django*, escrito en el lenguaje de programación *Python* y utilizado para el desarrollo de *webs 2.0* [Vlist(2007)]. Primero se analizará dicho lenguaje de programa-

ción y, posteriormente, se desarrollará con mayor profundidad el concepto de *Django*.

2.2.1 *Python*

A la hora de diseñar la aplicación, es de vital importancia elegir un lenguaje de programación que se adecúe de la mejor forma posible al sistema que se tiene pensado implementar. Dentro de todos los lenguajes de programación utilizados para el desarrollo de aplicaciones *web* (*Ruby*, *Python, PHP*, *ASP*, *JSP*, etc.), una de las motivaciones que se han indicado en el Capítulo 1 era la de tener un mayor nivel de conocimientos al final del desarrollo del presente Proyecto Fin de Carrera. Puesto que el autor ya disponía de conocimientos de *PHP*, *JSP* o *Ruby*, una alternativa interesante para potenciar el aprendizaje en esta travesía era desarrollar el proyecto en el lenguaje *Python*. A continuación se explica con algo más de detalle el concepto y características de este lenguaje.

Python[1] es un lenguaje de programación interpretado con alto nivel de abstracción para el desarrollador que otorga gracias a su dinamismo una alternativa muy interesante para el desarrollo de aplicaciones. Creado en Holanda por Guido van Rossum a finales de los años 80 y distribuido bajo una licencia de código abierto (*Python Software Foundation License*) compatible con la licencia *GPL* (*General Public License*), ofrece ventajas muy parecidas a las de los lenguajes de *scripting*, como puede ser un rápido desarrollo de las aplicaciones, además de buscar una mayor facilidad de lectura del propio código y facilidad en el diseño.

Una de las características más importantes de *Python* es que es un lenguaje de programación multiparadigma, es decir, permite al desarrollador llevar a cabo aplicaciones de diferente índole gracias a la posibilidad de programación con varios estilos: programación orientada a objetos, programación imperativa o programación funcional.

La principal filosofía de este lenguaje de programación es hacer hincapié en una sintaxis muy limpia y que favorezca un código legible. La distribución

[1]Página *web* oficial de Python: `http://python.org/`

bajo licencia de código abierto otorga a la comunidad de programadores la posibilidad de desarrollo de multitud de librerías o *API*s libres que facilitan el trabajo en aplicaciones de gran envergadura.

Se puede decir que *Python* es un lenguaje dinámico y fuertemente tipado, sensible a letras mayúsculas y minúsculas (*case sensitive*) en el que la declaración de variables se realiza sin necesidad de declararlas de manera explícita dentro del código, con la consecuente comodidad para el programador. Además, permite un manejo muy útil de las excepciones, otorgando autonomía a la propia aplicación. Computacionalmente hablando, cabe destacar que el propio lenguaje administra de manera automática el uso de memoria de la aplicación (reserva memoria cuando es necesario y la libera cuando deja de utilizarse) de manera que sea transparente para el desarrollador. Pero si algo lo hace principalmente interesante y potencia sus posibilidades es el uso de estructuras de datos sencillas y flexibles como los diccionarios, listas o *strings*, haciendo de estos tipos parte integral del lenguaje y que su uso se realice con un alto nivel de abstracción. Todo ello hace que *Python* se suela considerar como un lenguaje interpretado moderno, especialmente al compararlo con otros anteriores como puedan ser *Perl* o *PHP*.

2.2.2 *Django*

Dentro del desarrollo de aplicaciones *web*, existen multitud de alternativas muy interesantes para su desarrollo y posterior mantenimiento. Dependiendo del objetivo que se imponga el desarrollador, deberá analizar estas herramientas y decidir cuál es la que se ajusta mejor a las características de su aplicación. En el caso de este Proyecto Fin de Carrera, era indispensable encontrar una herramienta que permita el desarrollo de una aplicación para un nivel de usuarios final realmente importante (el total de profesores y alumnos de la *Universidad Rey Juan Carlos*). Se descartó el uso de tecnologías como *PHP* o *Python* por el simple hecho de que el desarrollo de una aplicación de grandes dimensiones era mucho más costosa aplicando lenguajes de programación únicamente. Era necesario dar un paso más allá y utilizar un *framework* de código abierto que a día de hoy cada vez es más utiliza-

do por la comunidad de desarrolladores de aplicaciones *web*. Dentro de las alternativas de desarrollo, *Django* es aquella herramienta que más se acerca a cumplir todas las motivaciones y objetivos planteados en el Capítulo 1 de esta memoria.

Django[2] es un *framework* de desarrollo *web* de código abierto, escrito en *Python*, basado en el paradigma MVC (Modelo Vista Controlador). Inicialmente, surgió con un objetivo principal: gestionar páginas orientadas a noticias de la *World Company* de Lawrence, Kansas. Sin embargo, y debido a su continuo desarrollo, fue liberado al público bajo una licencia *BSD* (*Berkeley Software Distribution*) en julio de 2005. El MVC (Modelo Vista Controlador) es un patrón de arquitectura de *software* que separa la lógica, los datos de una aplicación y la interfaz de usuario en tres componentes distintos, otorgando estabilidad y orden al desarrollo de la aplicación.

La filosofía principal de *Django [Adrian Holovaty(2009)]* es facilitar la creación de sitios *web* complejos otorgando herramientas potentes de alto nivel al programador. *Django* centra todo su esfuerzo en la reutilización de funciones y código, evitando la replicación siempre que sea posible (*DRY, Don't Repeat Yourself*). Además, hace hincapié en la conectividad y extensibilidad de componentes, añadiendo a su valor principal toda la potencialidad del lenguaje de programación *Python* (usado en todas las partes del *framework*, incluso en configuraciones, archivos y en modelos de datos). Todas las características explicadas anteriormente para *Python* son necesariamente aplicables en *Django*. A continuación, se enumeran a modo de resumen las características más importantes de este *framework*:

- **Herramientas para la gestión de la aplicación.**
- **Manipulación de bases de datos mediante el mapeo de objetos de la aplicación a entradas relacionales.**
- **Seguridad (*XSS*, *SQL Injection*, etc.).**
- **Un sistema de serialización para generar y procesar instancias del modelo de la aplicación *Django* representadas en formatos**

[2]Página *web* oficial de Django: `https://www.djangoproject.com/`

como *JSON* o *XML*, caché, internacionalización, plantillas, etc.

- **Alto nivel de abstracción gracias al uso de funciones muy potentes para el desarrollador que permiten mover gran volumen de datos sin dificultad.**
- **Servidor *web* de desarrollo básico que permite lanzar una aplicación en pruebas de inmediato.**

A modo de conclusión, cabe indicar que *Django* es una herramienta muy interesante para el presente pero también para un futuro a medio plazo. Las facilidades otorgadas potencian aún más su utilidad para la comunidad de desarrolladores de aplicaciones *web*. Gracias a que es de código abierto, estos desarrolladores proporcionan nuevas herramientas que completan algunas lagunas que las versiones oficiales no tienen exploradas y complementan brillantemente una herramienta sin duda interesante.

2.3 *Android*

Android[3] es una plataforma *software* cuyo núcleo está basado en sistemas *Linux* que incorpora un sistema operativo destinado a dispositivos móviles. En sus inicios, fue desarrollado por *Google* tras adquirir en julio de 2005 una pequeña empresa de California llamada *Android, Inc.* dedicada al desarrollo de *software* para teléfonos móviles. Desde ese momento, no fueron pocos los rumores que comenzaron a surgir sobre el deseo de *Google* de querer entrar en el mercado de la telefonía móvil, y más concretamente con el lanzamiento de su primer teléfono móvil llamado *Gphone*.

El 5 de noviembre de 2007 se fundó la *OHA* (*Open Handset Alliance*), un consorcio de importantes compañías internacionales entre las que se encuentran fabricantes de dispositivos *hardware*, empresas de *software* y operadores de telecomunicaciones como *Google*, *HTC*, *Sony, Dell, Intel*, *Samsung, LG* o *T-Mobile*. Saliendo al mismo tiempo su primer producto, *Android*, y teniendo

[3]Página *web* oficial de *Android*: `http://www.android.com/`

como objetivo el desarrollo de estándares abiertos para dispositivos móviles, a este consorcio se le han han ido sumando multitud de empresas de distinta índole hasta configurar el total de 84 empresas integrantes de la *OHA* en Diciembre de 2011.

Desarrollado en la actualidad por la *OHA*, *Android* se distribuye bajo una licencia *Apache 2.0* y *GPLv2*, lo que hace que reciba el tratamiento de *software* libre y código abierto. Esto permite que cualquier empresa o desarrollador, pueda realizar aplicaciones desarrolladas para *Android* sin la necesidad de que ninguna empresa u organismo tenga que validarlas previamente.

Figura 2.2: Capturas de pantalla de la *Android Market* desde un dispositivo móvil

Android otorga una herramienta muy interesante a todos los usuarios con *smartphones* que incorporan este sistema operativo para la descarga y difusión de las aplicaciones desarrolladas: la *Android Market*. En la *Android Market*, cualquier desarrollador puede subir su propia aplicación para

Android (ya sea de pago o gratuita). Los usuarios pueden descargarlas, difundirlas, poner comentarios al desarrollador y votar por todas y cada una de ellas. En la Figura 2.2 se pueden observar varias capturas de la *Android Market* desde un dispositivo *smartphone*.

Dentro de las características que hacen de *Android* un sistema operativo líder dentro del mundo de los *smartphones*, caben destacar las siguientes [DiMarzio(2008)]:

- **Reutilización y reemplazo de componentes gracias al *framework* de aplicaciones o la máquina virtual *Dalvik*.**

  Todos los desarrolladores de aplicaciones *Android*, tienen acceso total al código fuente usado en las aplicaciones base. Esto ha sido diseñado de esta forma para que no se generen cientos de componentes de aplicaciones distintas, que respondan a la misma acción. Mediante la reutilización de los componentes *Intent*, que son aquellos que se utilizan para pasar de una pantalla a otra dentro de una aplicación, *Android* permite utilizar pequeñas partes de otros programas en otras aplicaciones reutilizando código y ahorrando en recursos de forma que las aplicaciones sean lo más limpias y eficientes posibles. Para ello, debe ser el propio desarrollador el que configure su aplicación para que se lleve a cabo esta acción.

  Dentro de la arquitectura de *Android*, una de las partes que le hace muy interesante desde el punto de vista del programador es el uso por parte del sistema operativo de un *middleware*. Un *middleware* es un *software* que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, *software*, redes, *hardware*, etc. Éste simplifica el trabajo de los programadores en la compleja tarea de generar las conexiones que son necesarias en los sistemas distribuidos. El *middleware* de *Android* no es más que una máquina virtual similar a las *JVM* de *Java* (*Java Virtual Machine*) basada en registro, permitiendo por ello una ejecución más rápida. Incluye las librerías o *API*s fundacionales de *Java* y por tanto, las funcionalidades básicas del lenguaje (pero no cumple con sus estándares). Cada aplicación de *Android* se ejecuta co-

mo un proceso con su propia instancia de la máquina *Dalvik*, estando dicha máquina virtual especialmente diseñada para un funcionamiento eficiente durante la ejecución de varias máquinas virtuales.

- **Incluye de manera integrada un navegador *web* (basado en el motor *WebKit*).**

  Aunque la versión de fábrica de *Android* trae incorporado un navegador propio, existen multitud de alternativas muy interesantes dentro de la *Android Market* promocionadas incluso por el mismo equipo de *Android* para su utilización en los dispositivos móviles.

- **Optimización de gráficos, incluyendo tanto una librería de gráficos 2D como una en 3D.**

- **Base de datos *SQLite* para el almacenamiento de datos estructurados.**

  Android ofrece al desarrollador dos alternativas imprescindibles a la hora de almacenar contenido por parte de las aplicaciones del sistema operativo. La primera alternativa es el almacenamiento en ficheros de preferencias (para almacenar nombres de usuario, contraseñas, opciones de la aplicación, etc.) y archivos locales (almacenamiento del estado de una aplicación) que cuentan con permisos privados y de accesibilidad única por aplicación. Con ello se consigue que ningún usuario malicioso pueda acceder al contenido privado de la aplicación. Queda en manos del desarrollador permitir, mediante código implementado dentro de la aplicación, que esa información se comparta con otras aplicaciones. La segunda opción que permite *Android* es el uso de pequeñas bases de datos *SQLite*, orientadas a utilizarse cuando el volumen de información a almacenar es considerable. Con ello, el proceso de acceso a la información se hace más dinámico, rápido y menos pesado para *Android* ya que no es necesario generar muchos ficheros locales, con el tiempo que ello conlleva tanto en acceso, escritura y control.

- **Telefonía móvil, *Bluetooth*, *EDGE*, *3G*, *Wi-Fi*, cámara, *GPS***

(*Global Positioning System*), brújula y acelerómetro, dependiente del *hardware*.

Android otorga cierta independencia con los elementos que conforman nuestro dispositivo móvil. Para ello, otorga una herramienta vital para los desarrolladores como es el *SDK* (*Software Development Kit*), el entorno de programación de aplicaciones para *Android*. El *SDK* consta de un conjunto de librerías para el desarrollo, tutoriales de aprendizaje y un emulador de un dispositivo móvil con *Android*. Por supuesto, también se cuenta con un *plug-in* para un *IDE* (*Integrated Development Environment*) como *Eclipse*, con el cual se facilita la tarea de programación, compilación, ejecución y depuración. Gracias a esta herramienta, el desarrollador toma el control de todos los componentes de cualquier dispositivo, por lo que se desentiende por completo de éstos a la hora del desarrollo de aplicaciones ya que serán las propias empresas que desarrollan dichos componentes las que se preocupen de que funcionen con *Android* y con cualquier aplicación desarrollada con esta herramienta.

- **Soporte para pantalla táctil y para formatos de vídeo, audio e imágenes (*MPEG4*, *H.264*, *MP3*, *AAC*, *AMR*, *JPG*, *PNG*, *GIF*, etc.).**

2.3.1 Conceptos necesarios para entender una aplicación desarrollada en *Android*

A continuación se explican conceptualmente algunos aspectos importantes referentes al desarrollo de una aplicación en *Android*. Si el lector desea ampliar aún más los conceptos de una aplicación *Android* y profundizar en ellos a más bajo nivel se recomienda la siguiente lectura [González(2010)] (concretamente, el Apartado 2.3).

2.3.1.1 ¿Qué es *Eclipse*?

Eclipse[4] es un *IDE* (*Integrated Development Environment*), es decir, un entorno de desarrollo de aplicaciones muy potente diseñado originalmente para *Java*. Es distribuido con licencia libre y fue creado originalmente por la empresa *IBM*. *Eclipse* se está convirtiendo en una herramienta de uso obligado para todos los proyectos que se desarrollan en este lenguaje de programación, pero cada vez con más frecuencia *Eclipse* se emplea para el desarrollo de programas de toda índole y diferentes lenguajes de programación.

Una de las características fundamentales de este entorno de desarrollo es que se trata de un marco de trabajo modular ampliable mediante complementos (*plug-ins*). Así, cualquier desarrollador puede trabajar en un *plug-in* que haga posible programar en otros lenguajes de programación con este entorno de desarrollo. *Android* facilita un *plug-in* específico para que cualquier persona o equipo de trabajo pueda iniciar el desarrollo de aplicaciones para su propio sistema. Este *plug-in* es el anteriormente mencionado *SDK*, que pone a disposición multitud de librerías que abstraen al desarrollador y facilitan la tarea de programación para este sistema operativo. Ambas herramientas en conjunto facilitan al programador una serie de utilidades muy interesantes, otorgando un entorno de trabajo agradable para el desarrollador.

2.3.1.2 ¿Qué elementos componen una aplicación *Android*?

Toda aplicación en *Android* tiene varios elementos fundamentales característicos [Meier(2008)] que son necesarios comprender para poder desarrollar y entender cualquier programa que se lleve a cabo para este sistema operativo. La explicación de estos elementos tiene como objetivo que el lector se familiarice con cada uno de ellos puesto que en los próximos capítulos se hará referencia a casi todos con relativa frecuencia.

- ***Activity.***

  Las actividades (*Activities*) representan el componente principal de la interfaz gráfica de una aplicación *Android*. Se puede pensar en una *Acti-*

[4]Página *web* oficial de Eclipse: `http://www.eclipse.org/`

vity como el elemento análogo a una ventana en cualquier otro lenguaje visual. Estas *Activities* a su vez, cuentan con un ciclo de vida desde que son creadas hasta que se destruyen, tal y como se muestra en la Figura 2.3, ejecutándose en cada transición entre estados un determinado método.

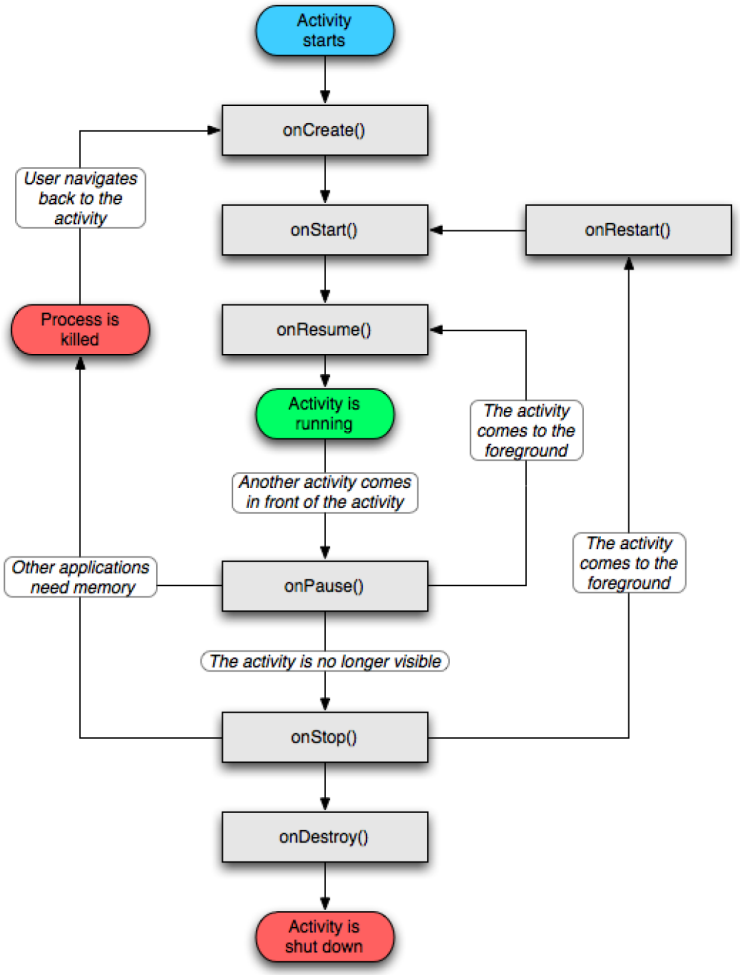


Figura 2.3: Ciclo de vida de una *Activity* en *Android*

A continuación se describen los principales métodos que permiten dentro del desarrollo de una aplicación *Android* dar lógica a una *Activity* en función

de su estado dentro del ciclo de vida:

onCreate() Ejecutado cuando la actividad ha sido lanzada (y por tanto creada) por primera vez.

onPause() Estado en el que la actividad se encuentra cuando se lanza otra nueva actividad (acción que se realizará, normalmente, desde la actividad que queda ahora parada), permaneciendo en segundo plano hasta que se regrese a su ejecución.

onResume() Tras un estado de pausa, se ejecuta este método de manera que la *Activity* vuelva a encontrarse en primer plano de ejecución.

onDestroy() Este método se ejecutará cuando se haya indicado explícitamente la finalización de la *Activity*. En caso de que esta finalización no sea señalada de manera explícita, lo habitual será que la *Activity* permanezca en segundo plano (tras ser pausada, pasará a estado de parada), a no ser que el propio sistema operativo necesite analizarla para reasignar los recursos (fundamentalmente memoria) que ésta tuviera asignados.

- ***View.***

  Los objetos *View* son los componentes básicos con los que se construye la interfaz gráfica de la aplicación. De inicio, *Android* pone a nuestra disposición una gran cantidad de controles básicos como cuadros de texto, botones, listas desplegables o imágenes, aunque también existe la posibilidad de extender la funcionalidad de estos controles básicos o crear controles personalizados.

- ***Service.***

  Los objetos *Service* son componentes sin interfaz gráfica que se ejecutan en segundo plano. En concepto, son exactamente iguales a los servicios presentes en cualquier otro sistema operativo. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo actualizar datos, lanzar

notificaciones, o incluso mostrar elementos visuales (*Activities*) si se necesita en algún momento la interacción con el usuario. Al igual que las *Activities*, los *Services* también disponen de un ciclo de vida desde que son creados hasta que se destruyen análogo al explicado anteriormente y que se puede recordar en la Figura 2.3.

- ***Content Provider.***

  Un *Content Provider* es el mecanismo que se ha definido en *Android* para compartir datos entre aplicaciones. Mediante estos componentes es posible compartir determinados datos de una aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. De la misma forma, la aplicación podrá acceder a los datos de otra a través de los *Content Provider* que se hayan definido.

- ***Intent.***

  Un *Intent* es el elemento básico de comunicación entre los distintos componentes *Android* que se han descrito anteriormente. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un *Intent* se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje *broadcast*, iniciar otra aplicación, etc.

2.4 *Códigos QR*

Un código QR[5] (*Quick Response Code*) es un sistema destinado al almacenamiento de información en un código de barras [Hiroko Kato(2010)] bidimensional o en una matriz de puntos creado por la compañía japonesa *Denso-Wave* en el año 1994. Una de las características más importantes que ha permitido su expansión en multitud de aplicaciones es que se trata de código abierto ya que sus derechos de patente (propiedad de *Denso Wave*)

[5]Página *web* oficial de los códigos QR: `http://www.denso-wave.com/qrcode/index-e.html`

no son ejercidos. El estándar japonés para códigos QR (*JIS X 0510*) fue publicado en enero de 1999 y su correspondiente estándar internacional *ISO* (*ISO/IEC18004*) fue aprobado en junio de 2000. Los códigos QR son muy comunes en Japón y de hecho son el código bidimensional más popular en ese país. Aunque inicialmente se usaron para registrar repuestos en el área de la fabricación de vehículos, en la actualidad los códigos QR se usan para administración de inventarios en una gran variedad de industrias y su uso es cada vez más habitual en dispositivos móviles para compartir información pública de usuarios, productos, anuncios, etc.

Estos códigos se caracterizan por los tres cuadrados que se encuentran en las esquinas y que permiten detectar la posición del código al lector de códigos del dispositivo móvil. Dentro de la propia estructura del código QR, se encuentra incluido además de la información útil para el usuario, información de control como información de la versión, información del formato, información de alineamiento y sincronización, etc. En la Figura 2.4 se puede observar un pequeño gráfico de ejemplo que explica qué partes del código QR corresponden a cada tipo de información explicado anteriormente.

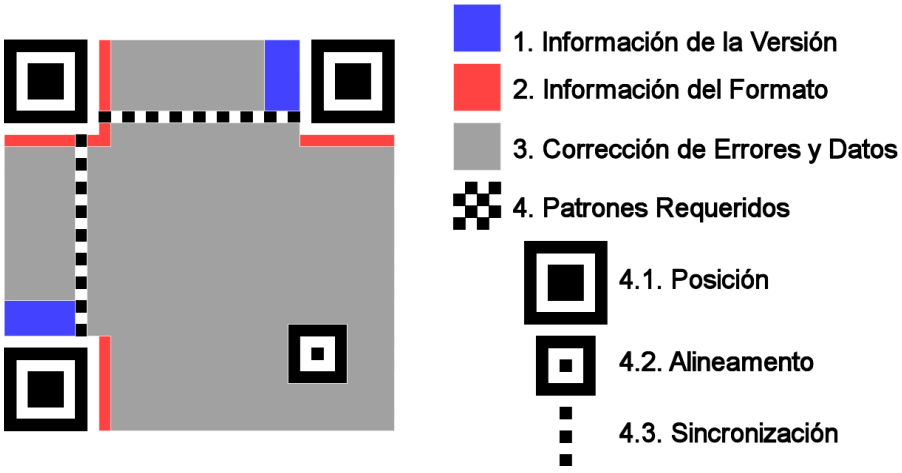


Figura 2.4: Ejemplo de estructura de un código QR

La posibilidad que otorgan estos códigos es infinita para el uso en diferentes aplicaciones tanto para dispositivos móviles como para aplicaciones

web. Los códigos QR permiten compartir todo tipo de información, desde una *URL*, una cadena de texto, un número de teléfono o un *SMS* con simplemente compartir una imagen y disponer de un dispositivo con cámara que tenga instalado un programa que decodifique códigos QR.

Capítulo 3

Diseño del sistema

En este capítulo se explica con detalle cómo se ha llevado a cabo el diseño del sistema del presente Proyecto Fin de Carrera. Para ello, primero se realizará una breve introducción donde se explicarán los rasgos globales del mismo. A continuación, se procederá a explicar con detalle los componentes del sistema, primero la aplicación *web* que se ejecutará en el servidor y finalmente la aplicación para *smartphones* que actuará como cliente.

3.1 Introducción

Para poder lograr el objetivo principal es necesario, en un primer momento, analizar el problema y estudiar qué elementos deben componer el sistema para poder lograrlo. El objetivo principal con el que partió este Proyecto Fin de Carrera es el desarrollo e implementación de un sistema automático de control de asistencia y geolocalización de eventos académicos de una organización. Esto quiere decir que habrá dos partes bien diferenciadas.

Por un lado, habrá que configurar un servidor que, conectado a Internet, pueda almacenar toda la información de la asistencia y geolocalización de eventos académicos. Asimismo, esta máquina deberá poder ser administrada por parte de personas ajenas a los desarrolladores (gente cualificada de la propia organización) que serán los que lleven el control y mantenimiento de la misma. Además, los usuarios (tanto profesores como alumnos de la

organización académica) deberán poder acceder a su información personal, sus últimos *check-ins* realizados y poder apuntarse a los eventos previamente antes de acudir para poder llevar a cabo el almacenamiento automático de su asistencia. Por tanto, será necesario que esta máquina pueda ser accedida por cualquier usuario a través de una interfaz *web* adaptada a las necesidades de la organización. Por otro lado, será necesario un dispositivo cliente que, conectado a Internet, permita confirmar qué personas se encuentran efectivamente en el evento (mediante geolocalización y algún mecanismo de localización en interiores) y se comunique con el servidor para poder almacenar dicha información.

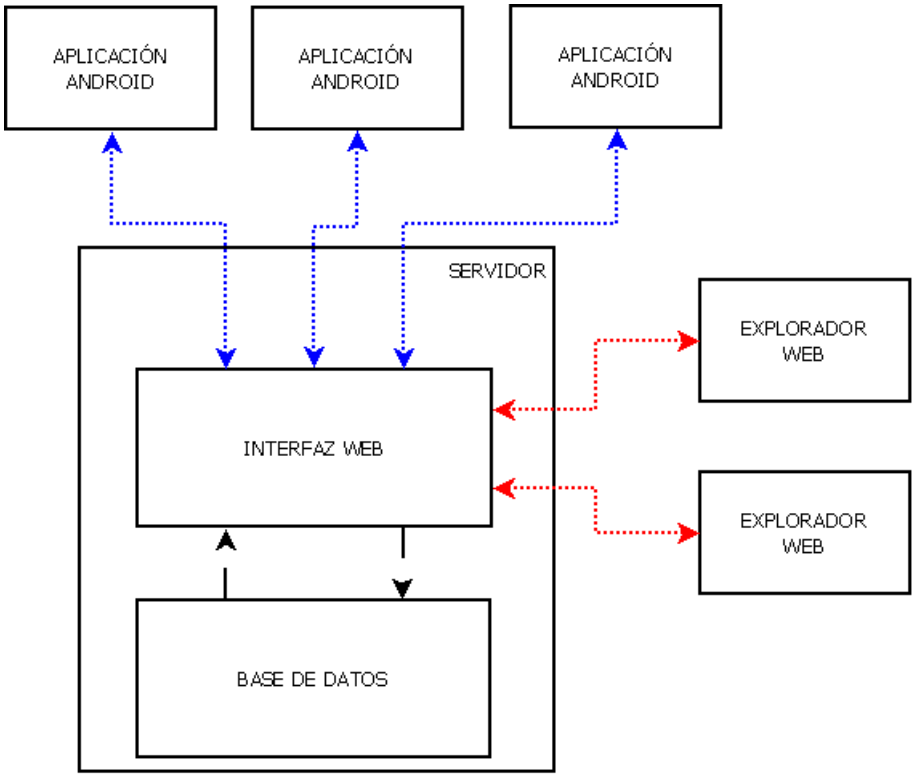


Figura 3.1: Esquema general de diseño del sistema

En la Figura 3.1 se puede observar un pequeño gráfico a nivel muy primario de cómo debe funcionar y comunicarse el sistema a desarrollar, marcando

qué partes corresponden al servidor y qué partes corresponden a los clientes de la misma. A modo aclaratorio, indicar que en el gráfico expuesto los elementos denominados "Aplicación *Android*" corresponden a la parte del cliente que se ejecutará en los *smartphones* y están pensados para que sean utilizados únicamente por los usuarios de la aplicación (se excluyen a los administradores), mientras que los elementos denominados "Explorador *Web*" corresponden a la parte del cliente que se ejecutará desde un navegador *web* y están pensados que sean utilizados por los usuarios y los administradores de la aplicación.

3.2 Diseño del servidor

A la hora de diseñar el servidor, es de vital importancia dividir su diseño en problemas más pequeños que permitan, una vez diseñado todo el conjunto, atender a las necesidades que precisa el sistema del presente Proyecto Fin de Carrera. Como se ha analizado en el apartado anterior, el servidor deberá constar de una interfaz que hará, por un lado, de pasarela con los clientes que utilicen *Android* para sincronizar su geolocalización y almacenar su asistencia a un evento, y por otro, un entorno *web* que permita a los usuarios, a través de un explorador, poder acceder a su cuenta de usuario y ver toda la información referente a sus *check-in*s, eventos a los que se han apuntado, escribir comentarios, etc. Por ello, primero se analizará el modelo de datos que debe disponer el servidor y, posteriormente, se analizará el servicio *web* que permitirá diseñar una interfaz para mostrar toda la información.

3.2.1 Diseño del modelo de datos

Para diseñar el modelo de datos es indispensable resumir todos y cada uno de los requisitos necesarios para que la aplicación cumpla con los objetivos marcados. Este es el proceso más lento y complicado ya que un error a la hora de contemplar los aspectos principales de la aplicación, hará que se tenga que modificar el modelo de datos a la par de que se esté llevando a cabo la fase de implementación y provocará un retraso importante en la ejecución del

proyecto. A continuación se listan las especificaciones que debe tener la base de datos:

- **Grupos de usuarios que utilizan la aplicación: alumnos y profesores.**

  Existen dos 'roles' de usuario principales que pueden utilizar esta aplicación: los alumnos y los profesores. Tanto unos como otros deberán tener acceso únicamente a su información propia como usuarios del sistema (sus asignaturas o eventos y su información de perfil) y podrán ver únicamente los perfiles públicos y comentarios de otros usuarios, estos últimos siempre que sean sobre eventos en los que el usuario tenga permisos de lectura. La diferencia fundamental de ambos grupos es que los alumnos se podrán apuntar y borrar libremente a cualquiera de las asignaturas de su carrera mientras que los profesores deberán solicitar permisos al grupo de administradores para poder aparecer con ese rango en cualquiera de las asignaturas generadas en la aplicación.

  En este punto es necesario referir a los administradores, que tendrán acceso a todos los contenidos de la aplicación *web* y al Panel de Administración desde donde podrán controlar y mantener toda la información del sistema. Cualquier usuario de la aplicación, sea alumno o profesor, podrá ser administrador del sistema siempre que se le den los permisos necesarios.

- **Eventos académicos: titulaciones, asignaturas y clases.**

  La aplicación a desarrollar en el presente Proyecto Fin de Carrera se centrará en el control de asistencia a clases de asignaturas de cualquier titulación. Es por ello que los únicos eventos que se van a tener en cuenta en el desarrollo del sistema serán asignaturas con sus respectivas clases por asignatura. Cada titulación tendrá una relación de uno a muchos con cada asignatura (una titulación puede estar relacionada con muchas asignaturas). Asimismo, la relación entre clases y asignaturas será de uno a uno (una clase sólo puede estar relacionada con una asignatura) y la relación de asignaturas y clases será de uno a muchos

(una asignatura puede estar relacionada con muchas clases).

La información que se deberá almacenar para cada objeto será la siguiente: en el caso de las titulaciones, únicamente se almacenará el nombre de la titulación. En el caso de las asignaturas, se almacenará el nombre de la asignatura, la titulación a la que pertenece dicha asignatura y el curso al que corresponde dentro del plan de estudios. Por último, para cada clase se registrará la fecha de inicio y finalización de la misma para poder llevar el control de usuarios malintencionados, así como algún tipo de mecanismo de seguridad para la localización en interiores ya que en estos casos la geolocalización mediante *GPS* es muy complicada. El mecanismo de seguridad elegido de entre varias alternativas es la generación de una cadena de texto aleatoria de 16 caracteres formados por letras y números, que será enviado por el cliente a la hora de hacer el *check-in* gracias al uso de códigos QR para el control en interiores. Además, el sistema comprobará si la posición del usuario corresponde con la del edificio donde se debe impartir la clase. En el caso de que no sea así, generará una alarma a los administradores para que realicen las operaciones oportunas con el usuario malicioso.

- **Permisos de los diferentes grupos de usuarios: alumnos y profesores.**

  Con el fin de facilitar la labor de administración del modelo de datos y que la aplicación sea más eficiente y consistente desde el punto de vista de la escalabilidad para su uso por una población potencial de miles de usuarios, será necesario distinguir en dos grupos diferentes los permisos de acceso a las asignaturas de los estudiantes y de los usuarios. Por ello deberán generarse dos grupos diferentes de permisos, uno que almacenará el alumno y la asignatura a la que se ha apuntado él mismo de manera voluntaria, y otro en el que se almacenará el profesor y la asignatura a la que tiene permisos otorgados por la administración desde el Panel de Administración de *Django*.

- **Almacenamiento de la posición de un usuario: formato de los**

check-in.

A la hora de almacenar el *check-in* es necesario guardar una serie de campos de información relevante. Por un lado, es vital almacenar la fecha y la hora (controlada por el servidor para evitar usuarios malintencionados) a la que se realice el almacenamiento a modo de control, la asignatura a la que corresponde el *check-in*, la geolocalización mediante *GPS* del usuario en ese momento (para evitar a aquellos usuarios malintencionados que busquen almacenar la información y no se encuentren en la clase), el usuario que está realizando el almacenamiento y, por último, a modo de estadísticas el número de alumnos que hay en el aula, dato que se almacenará sólo en el caso de que sea un profesor el que realiza el *check-in*.

- **Mecanismos de seguridad y control.**

  Como en cualquier sistema de comunicaciones, será de vital importancia la implementación de diversos mecanismos de control con el fin de poder asegurar la veracidad de la información y la dificultad de encontrar agujeros de seguridad en el servidor que permitan a usuarios maliciosos el acceso a información confidencial como es la que se tiene por objeto guardar en el sistema. Por tanto, el servidor deberá restringir cualquier acceso a la base de datos de cualquier usuario que no disponga de permisos de administrador del sistema. Asimismo, sólo los administradores tendrán la potestad de dar permisos de superusuario a aquellos usuarios de su elección. Además, se deberá restringir el acceso a cualquier información de otro usuario siempre que se detecte que el usuario que intenta acceder a dichos contenidos no es el propietario de los mismos.

Una vez analizados los puntos más importantes del diseño de los datos que almacenará la base de datos, hay que elegir qué gestor de base de datos se utilizará para las conexiones con la base de datos relacional de la aplicación. *Django* ofrece mecanismos de acceso y soporte a diferentes gestores de bases de datos: *PostgreSQL*, *MySQL*, *SQLite* y *Oracle*. Siguiendo el pensamiento

expresado en más de una ocasión en esta memoria, el objetivo fundamental es utilizar tecnologías que sean *software* libre y que permitan mover un gran volumen de información de la manera más eficiente, rápida y consistente posible. Por este motivo, tanto *SQLite* (uso en proyectos de pequeña embergadura) y *Oracle* (no ser de código abierto) hacen que sean descartados desde el inicio, quedando la elección de la base de datos entre dos tecnologías: *PostgreSQL* y *MySQL*.

Las características positivas y negativas que posee *PostgreSQL* son:

- **(+) Posee una gran escalabilidad.** Es capaz de ajustarse al número de *CPU*s y a la cantidad de memoria que posee el sistema de forma óptima, haciéndole capaz de soportar una mayor cantidad de peticiones simultáneas de manera correcta (en algunos benchmarks se dice que ha llegado a soportar el triple de carga de lo que soporta *MySQL*).

- **(+) Implementa el uso de rollback's, subconsultas y transacciones.** Gracias a la implementación de dichos mecanimos, hace que su funcionamiento sea mucho más eficaz ofreciendo soluciones en campos en las que *MySQL* no podría.

- **(+) Tiene la capacidad de comprobar la integridad referencial, así como también la de almacenar procedimientos en la propia base de datos, equiparándolo con los gestores de bases de datos de alto nivel.**

- **(-) Consume gran cantidad de recursos.**

- **(-) Tiene un límite de 8K por fila, aunque se puede aumentar a 32K, con una disminución considerable del rendimiento.**

- **(-) Es de 2 a 3 veces más lento que *MySQL*.**

- **(-) La sintaxis no es nada intuitiva, dificultando sobremanera su utilización a usuarios que se enfrenten a él y carezcan de los conocimientos específicos del gestor.**

Igualmente, se muestran a continuación las características más relevantes de *MySQL*:

- **(+) Lo mejor de *MySQL* es su velocidad a la hora de realizar las operaciones, lo que le hace uno de los gestores que ofrecen mayor rendimiento.**
- **(+) Su bajo consumo lo hacen apto para ser ejecutado en una máquina con escasos recursos sin ningún problema.**
- **(+) Las utilidades de administración de este gestor son envidiables para muchos de los gestores comerciales existentes, debido a su gran facilidad de configuración e instalación.**
- **(+) Tiene una probabilidad muy reducida de corromper los datos, incluso en los casos en los que los errores no se produzcan en el propio gestor, sino en el sistema en el que está.**
- **(+) El conjunto de aplicaciones *Apache-MySQL* es uno de los más utilizados en Internet para el desarrollo de aplicaciones *web*.**
- **(-) Carece de soporte para transacciones, *rollback*'s y subconsultas.**
- **(-) El hecho de que no maneje la integridad referencial, hace de este gestor una solución pobre para muchos campos de aplicación, sobre todo para aquellos programadores que provienen de otros gestores que sí que poseen esta característica.**
- **(-) No es viable para su uso con grandes bases de datos a las que accedan del orden de centenares de miles de usuarios, a las que se acceda continuamente, ya que no implementa una buena escalabilidad.**

Analizando los pros y los contras de ambos gestores de bases de datos, se ha decidido utilizar *MySQL* por su rapidez y porque la aplicación está pensada para su uso en organizaciones académicas que muy difícilmente podrán

llegar a varios centenares de miles de usuarios. De todos modos, los accesos a la base de datos que serán necesarios no son continuos y habrá que implementar el uso de la aplicación de tal forma que se restrinja al máximo el número de accesos a la base de datos y filtrar de la mejor manera posible la información a la que se accede dentro de ella para no extraer información que posteriormente no se va a utilizar.

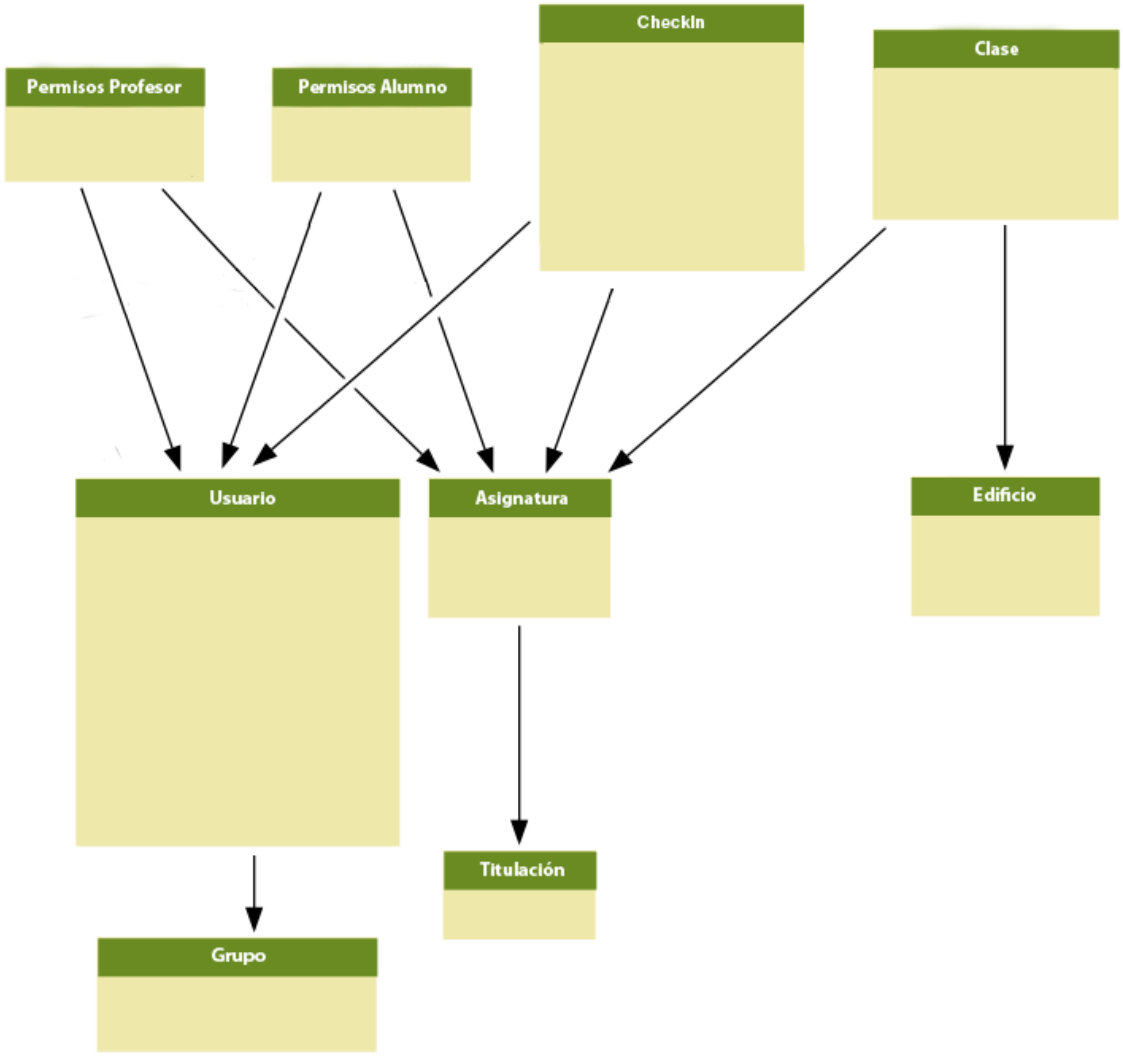


Figura 3.2: Diagrama relacional de la base de datos del servidor

Una vez explicadas las características del modelo de datos de la aplicación y argumentado por qué se utiliza *MySQL* como gestor de la base de datos, a continuación se puede observar en la Figura 3.2 el diagrama relacional del diseño que finalmente se ha llevado a cabo en la base de datos.

3.2.2 Diseño de la aplicación *web*

Una vez diseñado lo que va a ser el modelo de datos que tendrá la parte del servidor, es necesario a su vez diseñar la estructura que va a tener la aplicación *web* para posteriormente poder ser desarrollada en *Django*. Antes de comenzar a explicar el diseño que deberá tener la aplicación *web*, es necesario explicar más en detalle qué es la arquitectura MVC (Modelo Vista Controlador) y, una vez entendido el concepto, proceder al diseño de la aplicación.

La arquitectura MVC [Sonia Jaramillo Valbuena(2008), Reenskaug(1979)] fue creada en 1979 por Trygve Reenskaug. Es un patrón que permite separar la *GUI* (*Graphical User Interface*) de los datos y la lógica apoyándose en tres componentes:

- **Modelo.**

  Esta es la representación específica de la información con la cual el sistema opera, facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.

- **Vista.**

  Permite mostrar la información del modelo en un formato adecuado que permita que se dé la interacción. Presenta el modelo en un formato adecuado para interactuar. También recibe usualmente el nombre de interfaz de usuario.

- **Controlador.**

Es la parte que engloba la lógica de la aplicación. Responde a los eventos solicitados por el usuario que implican cambios en el modelo y la vista, dando una correcta gestión a las entradas de usuario.

En la Figura 3.3 se representa un diagrama sencillo que muestra la relación entre el modelo, la vista y el controlador. A modo aclaratorio, indicar que las líneas sólidas indican una asociación directa entre elementos y las punteadas una asociación indirecta.

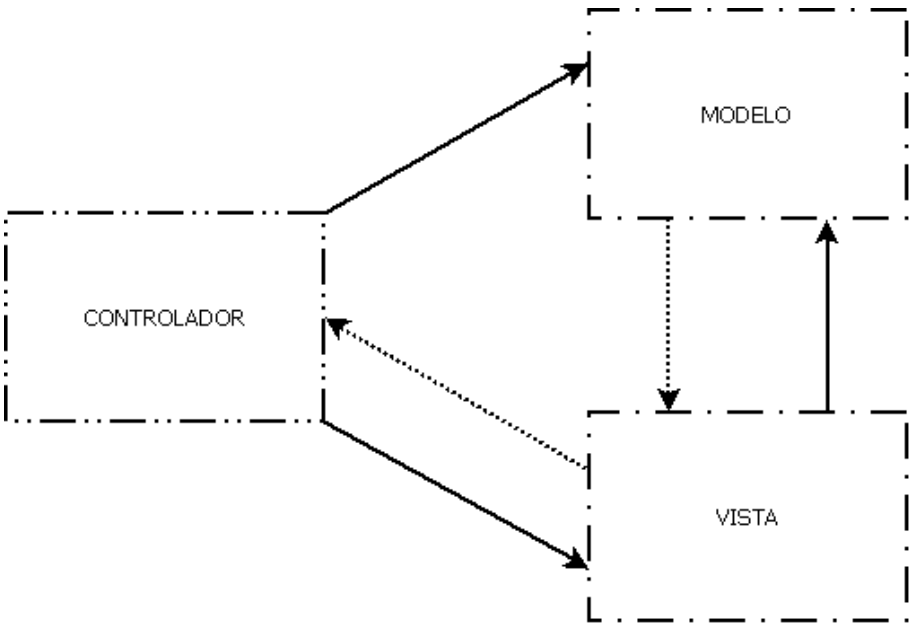


Figura 3.3: Diagrama de relaciones entre el Modelo, la Vista y el Controlador

Una vez entendida la arquitectura MVC, se procede a analizar cada elemento y las necesidades del sistema que se va a desarrollar en el presente Proyecto Fin de Carrera.

3.2.2.1 Modelo

El primer elemento de la arquitectura MVC que se va a analizar es la parte Modelo de la aplicación. Esta parte, ciñiéndose a la defínción explicada en

el punto anterior, estará compuesta por todas las clases u objetos necesarios para facilitar al desarrollador la programación del sistema y, además, que la parte Controlador pueda desarrollar su tarea lógica.

Si el lector se remonta a los requisitos de diseño explicados en el apartado 3.2.1 de la presente memoria, podrá observar que los objetos que se diseñan en este apartado tienen grandes similitudes con ellos con el fin de que las partes Controlador y Vista puedan desarrollar su tarea lo más eficientemente posible. A continuación, se enumeran los modelos a diseñar en la parte servidor:

- **Titulación.**

  Objeto que almacenará una titulación de la organización académica. Los campos que deberá contener son: una clave primaria única que sirva como identificador de la titulación y el nombre de la titulación.

- **Asignatura.**

  Elemento que almacenará la información de una asignatura de una titulación de la organización académica. Los campos que deberá contener son: una clave primaria única que sirva como identificador de la asignatura, el nombre de la asignatura, la titulación a la que pertenece y el curso al que corresponde dentro de la titulación.

- **Edificio.**

  Objeto que almacenará la información de un edificio de la *Universidad Rey Juan Carlos* donde se imparten una o varias asignaturas. Los campos que deberá contener son: una clave única que sirva como identificador del edificio, latitud y longitud de su situación y el nombre correspondiente (Aulario, Laboratorio, etc.).

- **Clase.**

  Objeto que almacenará la información de una clase de una asignatura de una titulación de la organización académica. Los campos que deberá contener son: una clave primaria única que sirva como identificador de la clase, el día y la hora de inicio y finalización de la clase, la asignatura

a la que pertenece, un campo que permita indicar si la clase se ha efectuado o si no se ha dado y el edificio en la que se impartirá.

- **Permiso de estudiante.**

  Elemento que almacenará un permiso para que un alumno pueda acceder con permisos de alumno a los eventos y contenidos de una asignatura. Los campos que deberá contener son: una clave primaria única que sirva como identificador del permiso de alumno, el alumno que dispone del permiso de alumno en la asignatura y la asignatura a la que se le da permiso.

- **Permiso de profesor.**

  Objeto que almacenará un permiso para que un profesor pueda acceder con permisos de profesor a los eventos y contenidos de una asignatura. Los campos que deberá contener son: una clave primaria única que sirva como identificador del permiso de profesor, el usuario que dispone del permiso de profesor en la asignatura y la asignatura a la que se le da permiso.

- ***Check-in.***

  Elemento que almacenará la información de geolocalización y control de asistencia de un usuario. Los campos que deberá contener son: una clave primaria única que sirva como identificador de *check-in*, el día y la hora a la que se almacene la posición, la geolocalización (latitud y longitud) del usuario en ese momento, el usuario que ha guardado su posición, la asignatura a la que corresponde el *check-in*, el número de estudiantes que hay en el aula (sólo en el caso de que el usuario tenga permiso de profesor), un campo que permita incluir un comentario por parte del usuario (sea alumno o profesor) que complemente los contenidos dados en la clase y por último un apartado que identifique si el *check-in* se ha realizado de manera maliciosa por el usuario (por ejemplo, haciendo *check-in* cuando realmente no se encuentra en el edificio).

- **Formato de los códigos QR.**

Como se ha indicado anteriormente, el mecanismo de seguridad que será utilizado por *GeoURJC* para atestiguar que un usuario se encuentra efectivamente en la clase será el uso de Códigos QR. Cada código QR se compondrá por una serie de campos que identifican unívocamente a una clase de una asignatura. Estos códigos estarán disponibles en formato impreso cada día que haya una clase y sólo tendrán acceso a ellos los administradores de la aplicación. El código QR de *GeoURJC* codifica una cadena de texto que debe seguir un patrón. El elegido por el diseñador es el siguiente: `{sid};{año};{mes};{día};{key}`, donde `{sid}` corresponderá con el identificador único de la asignatura creada en la aplicación *web* de *GeoURJC* por los administradores, `{año};{mes};{día}` corresponderán con la fecha en la que se debe impartir la clase de la asignatura y `{key}`, que será una cadena aleatoria compuesta por 16 letras y números que deberá generar el servidor cuando cree una clase de una asignatura.

3.2.2.2 Vista

El siguiente elemento de la arquitectura MVC que se va a analizar es la Vista de la aplicación. Esta parte estará compuesta por todas las interfaces de usuario necesarias para mostrar la información a cada uno de los usuarios. A continuación, al igual que a la hora de diseñar el Modelo de la aplicación *web*, se procede a enumerar los requisitos (a nivel de vistas) que debe cumplir la interfaz de usuario:

- **Página de acceso y registro de cuentas de usuario.**

  La primera interfaz que se debe encontrar cualquier usuario que no ha iniciado sesión en el sistema o que no tiene una cuenta de usuario debe contener un formulario que le permita acceder a la aplicación con su usuario y contraseña. Además, deberá contener otro formulario para que el usuario anónimo pueda crearse su cuenta personal y poder acceder a la aplicación.

- **Menú personal.**

Una vez que el usuario haya realizado el proceso de autenticación correctamente, deberá acceder a su menú personal. En él, deberá encontrar una serie de accesos directos que le faciliten el acceso a los diferentes contenidos del menú. Tendrá acceso a un calendario personal que mostrará todas y cada una de las asignaturas y clases en las que está apuntado, tendrá permiso para acceder a sus últimos *check-ins* realizados, deberá poder editar su información privada y pública del perfil y, por último, deberá disponer de alguna herramienta que le permita administrar las asignaturas a las que se encuentra apuntado. En el caso de que sea profesor, este último acceso directo no deberá estar disponible.

- **Información de una asignatura.**

  Desde el menú personal de cada usuario, se accederá al menú de cada asignatura. Este menú de asignatura tendrá disponible un nuevo calendario que indicará qué clases se han efectuado y cuáles no, y permitirá acceder a la información de aquellas clases que efectivamente se han impartido. Además, permitirá ver un listado de los profesores y alumnos que están apuntados a la asignatura (con el objetivo de que haya una interacción social por parte de los usuarios) que permitirá mostrar todos los perfiles públicos de todos ellos. Desde el punto de vista de los administradores de la aplicación, encontrarán accesos directos adicionales como pueden ser un listado de los códigos QR necesarios para realizar el proceso de control de asistencia y un apartado donde se pueda editar la información de la asignatura, generar nuevas clases e incluso eliminar la asignatura del sistema.

- **Información de una clase.**

  Desde el menú de información de la asignatura, se accederá al menú de cada clase que se haya impartido de dicha asignatura. En este apartado, se deberá mostrar todos los comentarios que tanto profesores como alumnos (que hayan asistido a la asignatura) escriban para complementar los contenidos de la clase. Asimismo, se tendrá acceso a un listado

de alumnos que han asistido (esto es, que hayan realizado el proceso de almacenamiento de su asistencia). Por otro lado, los administradores tendrán acceso al código QR de esa clase en concreto y a un apartado desde donde podrán editar la información referente a la clase (editar la fecha y hora de inicio o final de la misma, por ejemplo).

- **Perfil público de los usuarios.**

  Cada usuario dispondrá de un perfil público que podrán ver el resto de usuarios (pertenezcan al grupo que pertenezcan). En él se mostrará el nombre y apellidos del usuario, el email de contacto para que otros compañeros o profesores puedan ponerse en contacto con él, la fecha de registro en el sistema y la última hora a la que se ha conectado al sistema. Asimismo, se mostrará la última posición (a través de geolocalización) que ha llevado a cabo con su dispositivo cliente.

- **Panel de administración.**

  Como toda aplicación *web*, será necesario realizar un panel de administración que automatice algunos procesos a los administradores como puede ser el crear una nueva asignatura, eliminarla, editarla, añadir profesores a una asignatura o eliminarlos, etc. El panel de administración será personalizado para esta aplicación pero será necesario complementarlo con el panel de administración que ofrece *Django* (por ejemplo, para dar permisos a los profesores o comprobar los permisos de los usuarios).

- **Manual de usuario.**

  Se ofrecerá a todos los usuarios un manual que permita, paso a paso, configurar todo el sistema necesario (tanto la parte del servidor como la parte del cliente). Este manual sólo ofrecerá información sobre el uso de la interfaz. Aquella información confidencial para los administradores no se publicará en el manual de uso.

3.2.2.3 Controlador

El último elemento a analizar dentro de la arquitectura MVC será el diseño de los controladores que darán toda la lógica al servidor. Cabe destacar que en este punto únicamente se va a definir conceptualmente qué controladores se tienen que desarrollar, siendo en el capítulo siguiente donde se entre en detalle sobre cada uno de ellos.

- **Controlador de acceso a la aplicación.**

  Este controlador dispondrá de las funciones necesarias para el acceso y la desconexión de usuarios registrados y la lógica para crear nuevas cuentas de usuario.

- **Controlador del menú personal.**

  Es uno de los controladores más importantes. Incluye todas las funciones que recopilan toda la información imprescindible para mostrar a cada usuario sus asignaturas, clases, información privada, etc.

- **Controlador de sincronización con los clientes.**

  Es otro de los controladores vital. Incluirá todas las funciones necesarias para la sincronización y guardado de los *check-ins* con la aplicación cliente que los usuarios tendrán disponible en sus dispositivos móviles con sistema operativo *Android*.

- **Controlador de asignaturas y clases.**

  Este controlador incluirá todas las funciones necesarias que generarán toda la información que permite a los usuarios poder acceder tanto a los menús de asignaturas como a la información de cada una de las clases.

- **Controlador de gestión de *check-in*s.**

  Este controlador estará compuesto por toda la lógica que recopila la información necesaria para poder ver y editar los propios *check-ins* de un usuario.

- **Controlador de perfiles públicos.**

  Este controlador incluye la lógica que permite recopilar la información pública de un usuario para poder ser mostrada al resto de usuarios.

- **Controlador de administración.**

  Este controlador incluirá todas las funciones necesarias que generarán toda la información que permita a los administradores interactuar en el panel de administración .

3.3 Diseño del cliente

Una vez diseñado el servidor, su modelo de datos y cada una de las partes de la arquitectura MVC, se debe diseñar un cliente que se desarrollará en el lenguaje de programación *Java* para sistemas operativos *Android*. Antes de proceder al diseño de la aplicación, es necesario realizar algunas explicaciones sobre cómo se desarrolla una aplicación en este sistema operativo. Como se indicó en el Apartado 2.3 de la presente memoria, para el desarrollo de aplicaciones para *Android* es necesario utilizar la herramienta *SDK* junto al entorno de desarrollo integrado *Eclipse*.

El objetivo fundamental es desarrollar una aplicación lo más simple posible. Esto es debido a que la lógica de la aplicación nunca debe estar en la parte cliente y debe ser de único control por parte del servidor. La comunicación con el servidor es necesario que sea lo más sencilla posible. Puesto que el servidor únicamente va a mandar mensajes de estado al cliente (que será el que mande la información importante al servidor), se ha decidido que para la comunicación entre ambos entes se utilice únicamente el protocolo *HTTP*[1] [R. Fielding(1999)] y los métodos *GET* y *POST* para el intercambio de información y estados.

Por ello, las características de la aplicación desarrollada en *Android* deben ser las siguientes:

[1]RFC 2616 del protocolo *HTTP*: `http://www.w3.org/Protocols/rfc2616/rfc2616.html`

- **Conexión con la parte servidor.**

  La aplicación deberá ser capaz de, con el usuario y contraseña del usuario, realizar una conexión a la aplicación que se encuentra en el servidor para realizar las operaciones pertinentes.

- **Detectar la geolocalización del usuario de la forma más aproximada posible.**

  Además, deberá ser capaz de utilizar la detección de la posición del dispositivo móvil (latitud y longitud) con fines informativos para los administradores del sistema.

- **Decodificar códigos QR**.

  Otro requisito de la aplicación cliente debe ser la detección y tratamiento de códigos QR. Para ello, se deberá utilizar la herramienta que otorga *Android* de reutilización de código y aprovechar la utilidad de otra aplicación de libre distribuición y gratuita que permita realizar este proceso y devuelva el resultado del código decodificado. En el caso de este Proyecto Fin de Carrera, se ha decidido utilizar la aplicación *Barcode Scanner*[2].

- **Desarrollar un servicio de notificaciones.**

  Otra herramienta de la que debe disponer el sistema es el desarrollo de un servicio que permita conectarse al servidor y comprobar, periódicamente, si en ese mismo instante o en un futuro a corto plazo el usuario tendrá una clase de una asignatura en la que está apuntado. En el caso de que sea así, deberá lanzar una notificación en la barra de notificaciones del sistema operativo para que el usuario no olvide acudir a ninguna clase.

- **Proceso automatizado de la detección de la posición.**

  Con todos los requisitos anteriores, la aplicación debe ser capaz de filtrar la información y mandársela al servidor para efectuar el proceso

[2]Página *web* oficial del proyecto: `http://code.google.com/p/zxing/`

de control de asistencia de manera que sea completamente transparente para el usuario y el administrador.

- **Vista personalizada para profesor y alumno.**

  Como se indicó en el Apartado 3.2.2 de la presente memoria, el profesor tiene la obligación de enviar cuando realice su control de asistencia el número de alumnos que hay en el aula. Este campo no es un requisito para los alumnos, por lo que dependiendo del grupo al que pertenezca el usuario, será indispensable crear dos vistas personalizadas y comprobar en el momento de la autenticación a qué grupo pertenece el usuario que está utilizando la aplicación cliente.

Como se observa, los requisitos de la aplicación que actuará como cliente no son muy exigentes. El funcionamiento del cliente debe ser consistente y lo más eficiente posible, permitiendo enviar el mínimo de información posible a través de la red de datos para llevar a cabo su proceso de control y asistencia.

Capítulo 4

Implementación del sistema

En el siguiente capítulo se explicará con detalle la implementación de la parte servidor y la parte cliente del presente Proyecto Fin de Carrera para cumplir los objetivos marcados en el Apartado 1.2. Se va a seguir los apartados del capítulo anterior, profundizando en los aspectos más importantes que sea necesario resaltar para comprender la implementación.

4.1 Introducción

Una vez indicadas las motivaciones y objetivos, explicado de manera conceptual las tecnologías que se van a implementar y llevado a cabo el proceso de diseño del sistema, el siguiente paso es la implementación tanto del servidor como del cliente. Antes de comenzar el trabajo, es necesario elegir un buen nombre para el sistema. En el caso de este Proyecto Fin de Carrera, se ha elegido el nombre *GeoURJC*.

Es necesario aclarar al lector de esta memoria que el desarrollo tanto de la parte servidor como de la parte cliente se ha llevado a cabo en paralelo durante todo el tiempo de trabajo. Sin embargo, es necesario seguir un orden para que el lector no se pierda en las explicaciones de ambos elementos del sistema, por lo que primero se explicará la implementación de la parte servidor y, posteriormente, se explicará la parte cliente.

4.2 Implementación del servidor

4.2.1 Estructura de directorios y ficheros

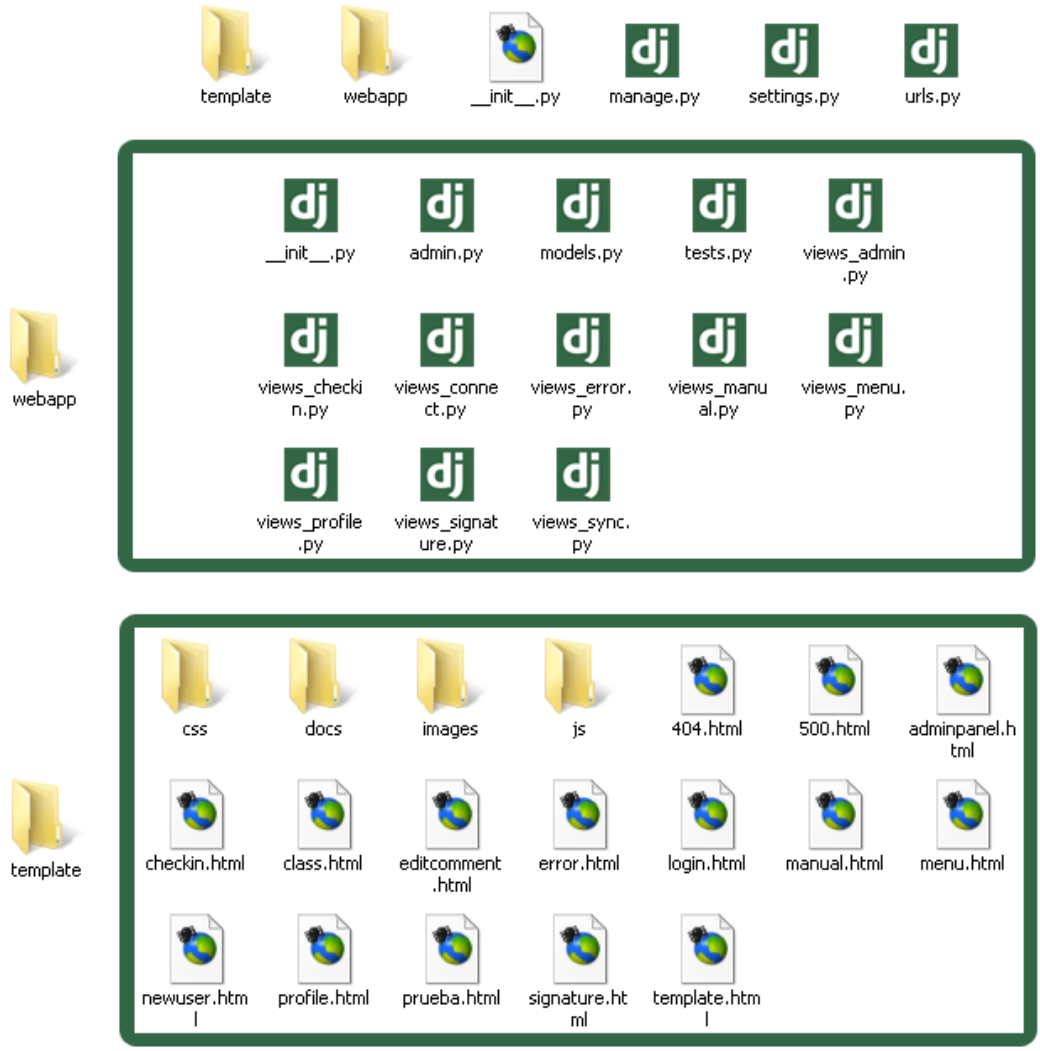


Figura 4.1: Estructura de directorios y ficheros del proyecto *Django* implementado para este sistema

Como se ha indicado en varias ocasiones a lo largo de la memoria, el servidor se desarrollará utilizando el *framework* de desarrollo *web Django*.

Entender la estructura de directorios y ficheros es de vital importancia para poder entender igualmente el desarrollo de la aplicación. En la Figura 4.1 se muestra en un gráfico el árbol de directorios implementado para el servidor de este sistema. A continuación se realiza una breve introducción a los ficheros y directorios más importantes:

- ***template***

  En esta carpeta se almacena toda la información necesaria para el diseño de la plantilla de la aplicación: imágenes, archivos de estilo *CSS (Cascading Style Sheets)*, *scripts* desarrollados en *Javascript* y *AJAX*. Asimismo, se incluyen en esta carpeta todas las vistas necesarias para la interfaz gráfica del servidor (arquitectura MVC) que se explicarán más adelante (los ficheros llamados **.html*).

- ***webapp***

  En este directorio se encuentra toda la lógica de la aplicación. Su nombre es definido por el usuario cuando se inicia el proyecto. En él se encuentran tanto los modelos necesarios para la implementación del modelo de datos (en el fichero *Models.py*) como los controladores que otorgan la funcionalidad al servidor (todos los ficheros llamados *views_ *.py*). Todos ellos se explicarán con más profundidad más adelante. Asimismo, se encuentra el fichero *Admin.py* que permite personalizar al máximo el panel de administración que otorga *Django* a sus proyectos (y que permite el acceso a todos los contenidos almacenados en la base de datos).

- ***settings.py***

  En este fichero se configuran todas las características de la aplicación *web*: configuración de la base de datos que utilizará la aplicación, el idioma, las rutas de directorios a la plantilla de diseño, a los archivos disponibles públicamente para acceso de los usuarios, etc.

- ***url.py***

En este archivo se almacenan las url de acceso a la aplicación *web*. Para un desarrollo lo más eficiente posible de una aplicación *web*, es de vital importancia que la estructura de *URL* (*Uniform Resource Locator*) sea lo más clara posible. Para ello, *Django* ofrece un sistema que puede ser completamente personalizado a los requisitos de la aplicación. Algunos ejemplos de *URL* en el caso del sistema de este Proyecto Fin de Carrera son los siguientes:

- `http://localhost/menu`: *URL* que llevará al usuario a su menú personal
- `http://localhost/signature/1`: *URL* que permite al usuario ver la información de la asignatura cuyo identificador tenga valor 1.
- `http://localhost/signature/1/2011-11-11`: *URL* que permite al usuario ver la información de la clase del día 11 de noviembre de 2011 de la asignatura cuyo identificador tenga valor 1.
- `http://localhost/checkin/37`: *URL* que permite al usuario ver la información del *check-in* cuyo identificador tenga valor 37 (si no es el propietario del *check-in* se mostrará un error).
- `http://localhost/profile/3`: *URL* que permite al usuario ver la información pública del usuario cuyo identificador de usuario tenga valor 3.

4.2.2 Explicación del Modelo Vista Controlador implementado

Una vez entendida la estructura de ficheros del servidor, se procede a la explicación de cada una de las partes que componen el proyecto *Django [Adrian Holovaty(2009)]*. Primero, se procede a la explicación de los Modelos implementados para poder trabajar con abstracción en el desarrollo de la aplicación. Seguidamente, se explicarán cada uno de los controladores que ha sido necesario implementar, así como las funciones que lo componen. Por

último, se hará una explicación de cada una de las vistas que componen la interfaz gráfica para el acceso de los usuarios.

4.2.2.1 Modelos

Como se indicó en el Apartado 4.2.1 en el que se habló de la estructura de directorios y ficheros de la aplicación, los objetos necesarios para el desarrollo tanto del modelo de datos como de la lógica de la aplicación se implementan dentro del fichero *Models.py* que se encuentra dentro del directorio *webapp*. La generación de modelos en *Django* es fundamental para cualquier aplicación. Una de las características más potentes de este *framework* es que, una vez generados los modelos, otorga multitud de funciones de acceso, borrado, modificación y creación de objetos a todos los campos que los componen sin ser necesario que el desarrollador haga ese trabajo. Además, se encarga de generar todas las tablas necesarias en la base de datos para que el tratamiento con ella sea ajena por parte del desarrollador y hacer mucho más sencillo su mantenimiento. Dentro de todas estas tablas que genera *Django*, es necesario explicar de manera conceptual algunas de ellas.

Todo modelo de datos de una aplicación *Django* cuenta con los siguientes elementos dentro de su modelo:

django.contrib.sessions *Django* ofrece un soporte completo para sesiones de usuarios anónimos. El *framework* de sesiones permite almacenar y recuperar información arbitraria sobre cada uno de los usuarios. En esta tabla se almacenan los datos necesarios por parte del servidor permitiendo un nivel de abstracción en la tarea del envío y recepción de *cookies*, facilitando el proceso. Las *cookies* guardan un identificador de sesión (en ningún caso se almacena información personal del usuario anónimo).

django.contrib.sites *Django* incorpora de manera opcional un *framework* de sitios *web* (aunque más que una serie de herramientas otorga un mecanismo de convención para sitios *web*). Otorga la posibilidad de guardar de manera asociada nombres de dominio y direcciones *web* de los sitios que se desarrollen en *Django*. Tiene

especial relevancia cuando un mismo proyecto dispone de varios sitios *web*.

django.contrib.admin Una de las partes más potentes de *Django* es el panel de administración que genera de manera automática nada más crear un proyecto. *Django* lee los metadatos de los modelos añadidos al proyecto y proporciona una potente interfaz que permite incorporar contenido al sitio *web* en el momento que es generado. En este apartado se almacena toda la información necesaria para el uso de dicho panel de administración.

django.contrib.contenttypes *Django* incorpora un mecanismo que le permite diferenciar todo el tipo de contenido que se puede añadir en cada uno de los modelos del proyecto que se está desarrollando. La finalidad principal es la de ofrecer una interfaz genérica al desarrollador independientemente de los modelos que se incluyan en el proyecto, otorgando un grado de abstracción. En este apartado se almacena la información necesaria para llevarlo a cabo.

django.contrib.auth Otra de las características más potentes de *Django* es la incorporación de un completo sistema de autenticación de usuarios. Este sistema maneja cuentas de usuarios, grupos, permisos de usuarios y utiliza el sistema de *cookies* mediante sesiones explicado anteriormente. Pone a disposición del programador funciones de acceso a este sistema, ofreciendo, de nuevo, un alto nivel de abstracción al desarrollador. Todas las tablas de este grupo almacenan toda la información necesaria para que cualquier proyecto pueda disponer de usuarios de manera prácticamente inmediata.

En cualquier proyecto *Django*, cada aplicación nueva que se incorpora al proyecto cuenta con su propio grupo de tablas dentro del modelo de datos del proyecto. Todos los modelos que se incorporen a la aplicación serán nombrados como *nombre-de-aplicacion_nombre-de-modelo*. En el proyecto *GeoURJC* sólo se ha incorporado una aplicación, llamada *webapp* (el nombre

lo puede elegir el desarrollador al iniciar el proyecto). Por tanto, todos los modelos de datos incorporados a la aplicación tienen como nombre en la base de datos *webapp_ nombre-de-modelo*. En la Figura 4.2 se muestra un diagrama realizado por una utilidad de *Django* que ofrece cada uno de los componentes del modelo de datos de la aplicación *webapp*. En él, se pueden observar las relaciones entre las diferentes clases y los campos que componen cada uno de los modelos incluidos en la aplicación.

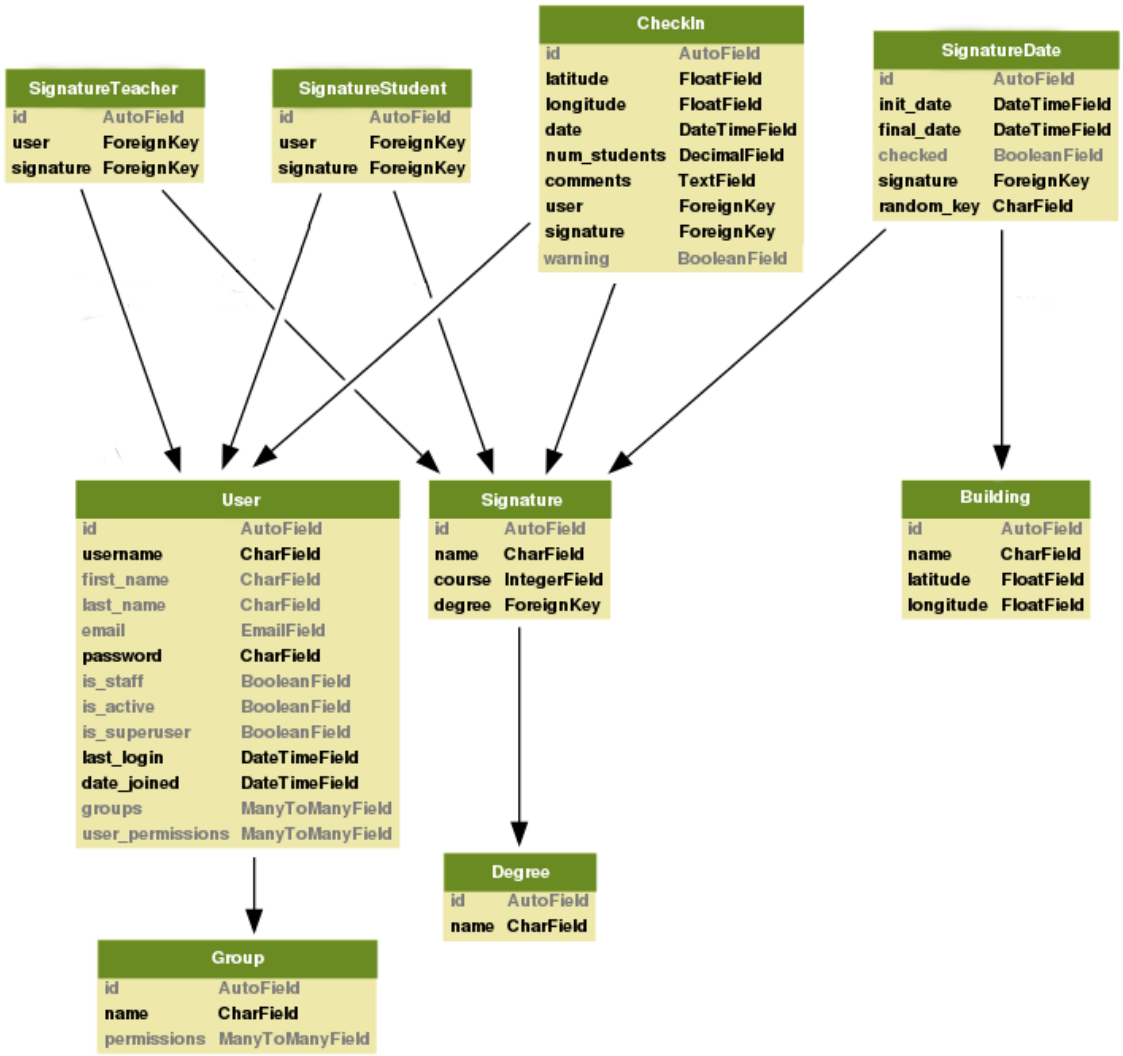


Figura 4.2: Diagrama relacional de los modelos desarrollados en *Django*

Si el lector se remonta a los requisitos de diseño explicados en el Apartado 3.2.2.1 de la presente memoria, podrá observar que los modelos que se han implementado cumplen las características de diseño que se solicitaban en ese punto. A modo de resumen final y conclusión de este primer elemento de la arquitectura MVC, se explica cada uno de ellos:

- ***Degree***

  Modelo que almacena una titulación de la organización académica. El nombre de la tabla en el que se almacenarán todos los objetos *Degree* es *webapp_degree*. Los campos que contiene son:

  - *id: AutoField*. Identificador único de cada titulación.
  - *name: CharField*. Nombre de la titulación. Contiene una restricción máxima de 100 caracteres.

- ***Signature***

  Modelo que almacena la información de una asignatura de una titulación de la organización académica. El nombre de la tabla en el que se almacenarán todos los objetos *Signature* es *webapp_signature*. Los campos que contiene son:

  - *id: AutoField*. Identificador único de cada asignatura.
  - *name: CharField*. Nombre de la asignatura. Contiene una restricción máxima de 100 caracteres.
  - *course: IntegerField*. Curso de la titulación a la que corresponde la asignatura.
  - *degree: ForeignKey*. Relación foránea con la titulación a la que corresponde esta asignatura.

- ***Building***

  Modelo que almacena la información de un edificio donde se imparte una clase de una asignatura dentro de la organización académica. El

nombre de la tabla en el que se almacenarán todos los objetos *Building* es *webapp_building*. Los campos que contiene son:

- *id: AutoField*. Identificador único de cada edificio.
- *latitude: FloatField*. Latitud de la geolocalización del edificio.
- *longitude: FloatField*. Longitud de la geolocalización del edificio.

- ***SignatureDate***

  Modelo que almacena la información de una clase de una asignatura de una titulación de la organización académica. El nombre de la tabla en el que se almacenarán todos los objetos *SignatureDate* es *webapp_signaturedate*. Los campos que contiene son:

  - *id: AutoField*. Identificador único de cada clase.
  - *init_date: DateTimeField*. Fecha y hora de inicio de la clase.
  - *final_date: DateTimeField*. Fecha y hora de finalización de la clase.
  - *checked: BooleanField*. Campo que almacenará el estado de la clase. Si el profesor ha hecho *check-in*, la clase se entiende por efectuada y su valor será *True*. Si el profesor no ha hecho *check-in*, se entenderá la clase como no realizada y su valor será *False*.
  - *random_key: CharField*. Campo que almacenará una cadena de caracteres y números aleatoria y que servirá como mecanismo de seguridad a la hora de hacer *check-in*.
  - *signature: ForeignKey*. Relación foránea con la asignatura a la que corresponde la clase.
  - *building: ForeignKey*. Relación foránea con el edificio en el que se imparte la clase.

- ***SignatureStudent***

  Modelo que almacena un permiso para que un alumno pueda acceder con permisos de alumno a los eventos y contenidos de una asignatura.

El nombre de la tabla en el que se almacenarán todos los objetos *SignatureStudent* es *webapp_ signaturestudent*. Los campos que contiene son:

- *id: AutoField*. Identificador único de cada permiso de acceso de alumnos.
- *user: ForeignKey*. Relación foránea con el usuario al que corresponde el permiso de acceso.
- *signature: ForeignKey*. Relación foránea con la asignatura a la que corresponde el permiso de acceso.

- ***SignatureTeacher***

  Modelo que almacena un permiso para que un profesor pueda acceder con permisos de profesor a los eventos y contenidos de una asignatura. El nombre de la tabla en el que se almacenarán todos los objetos *SignatureTeacher* es *webapp_ signatureteacher*. Los campos que contiene son:

  - *id: AutoField*. Identificador único de cada permiso de acceso de profesores.
  - *user: ForeignKey*. Relación foránea con el usuario al que corresponde el permiso de acceso.
  - *signature: ForeignKey*. Relación foránea con la asignatura a la que corresponde el permiso de acceso.

- ***CheckIn***

  Modelo que almacena la información de geolocalización y control de asistencia de un usuario. El nombre de la tabla en el que se almacenarán todos los objetos *CheckIn* es *webapp_ checkin*. Los campos que contiene son:

  - *id: AutoField*. Identificador único de cada *check-in*.
  - *latitude: FloatField*. Latitud de la geolocalización del usuario en el momento del *check-in*.

- *longitude: FloatField.* Longitud de la geolocalización del usuario en el momento del *check-in.*
- *date: DateTimeField.* Fecha y hora en el momento en el que se ha realizado el *check-in.*
- *num_students: DecimalField.* Número de estudiantes en el aula en el momento del *check-in.* Este campo sólo puede ser rellenado por los profesores, teniendo el valor -1 siempre que el *check-in* es realizado por los alumnos.
- *comments: TextField.* Comentario opcional que puede realizar el usuario que ha asistido a clase.
- *warning: BooleanField.* Campo que almacenará el estado del *check-in.* Si el usuario ha hecho *check-in* y su localización no corresponde con la del edificio donde se está impartiendo la clase, su valor será *True.* Si el usuario se encuentra en el edificio, su valor será *False.*
- *user: ForeignKey.* Relación foránea con el usuario al que corresponde el *check-in.*
- *signature: ForeignKey.* Relación foránea con la asignatura a la que corresponde el *check-in.*

4.2.2.2 Controladores

Una vez se dispone de los modelos implementados, el siguiente paso a la hora del desarrollo de la aplicación es incluir toda la lógica necesaria para lograr los objetivos marcados. La lógica de toda aplicación con arquitectura MVC se debe implementar siempre en los controladores [Dana Moore(2008)]. Como se indicó en el Apartado 4.2.1 en el que se habló de la estructura de directorios y ficheros de la aplicación, todos los controladores deben ser implementados dentro del directorio *webapp* y su nombre de fichero debe seguir el siguiente formato: *views_ *.py.* A la hora de la implementación, es necesario distribuir todas las funciones lógicas en diferentes controladores con el objetivo de facilitar la tarea de implementación y de posterior mantenimiento a otros desarrolladores diferentes del desarrollador inicial.

Los controladores tienen en *Django* dos funciones fundamentales: primero, recoger aquellas funciones lógicas que permitan automatizar diferentes procesos necesarios para el desarrollo de la aplicación (generar listados de contenidos, algoritmos que generan información y la almacenan o borran de la base dedatos, ejecutar bucles para devolver un resultado a otra función, etc.). La segunda función fundamental es la de servir de pasarela entre la información de la base de datos y las vistas de la arquitectura MVC de la aplicación que muestran el contenido a través de la interfaz *web* a todos los usuarios. Generalmente, los controladores disponen de funciones que cumplen ambas funciones. Sin embargo, pueden existir controladores que incluyan sólo funciones que cumplan una de ellas y no la otra, como se mostrará más adelante.

Para implementar la parte servidor de *GeoURJC* ha sido necesario crear nueve controladores. A continuación se exponen todos ellos con una breve explicación conceptual de lo que realizan cada una de las funciones que los componen:

- ***views_admin.py***

  Este controlador incluye todas las funciones necesarias para el uso del panel de administración por parte de los administradores encargados de esta labor en *GeoURJC*. Las funciones que se incluyen en este controlador se enumeran a continuación:

  - `adminPanel(request)`

    Es la función principal del panel de control. Recopila toda la información necesaria para pasársela a la vista *adminpanel* y así poder mostrar el panel de administración personalizado para *GeoURJC* a través de la siguiente URL: `http://localhost/adminpanel`.

  - `saveAddSignature(request)`

    Esta función recoge toda la información necesaria del panel de administración personalizado para *GeoURJC* y genera mediante un algoritmo una nueva asignatura y las clases que la componen.

  - `checkActionAdminPanel(request)`

Esta función genera los permisos para las secciones del panel de administración (en función de si tienes permisos de administrador o no) para dar el efecto *AJAX* que permita mostrar en la vista *adminpanel* las secciones a las que tiene acceso el usuario.

- `handlerNotification(user, request)`

  Esta función es llamada por la función `adminPanel(request)` y maneja las notificaciones de estado de la aplicación en el panel de administración personalizado para *GeoURJC*, de modo que cualquier error o notificación que se pase como parámetro *GET* (por ejemplo, */adminpanel?error=no_signatures*) en la vista *adminpanel* es mostrado al usuario (es decir, que se muestre un error, un mensaje de éxito, etc.). Existen tres tipos de notificaciones en la interfaz de *GeoURJC*: *error*, cuando se solicita contenido que no existe en la aplicación o al que no tiene permisos de acceso; *success*, cuando se realiza con éxito una acción (por ejemplo, crear una asignatura nueva y que se haya creado bien); *info*, que da un mensaje informativo al usuario.

- `getMsg(request)`

  Esta función es llamada por la función `handlerNotification(user, request)` y devuelve el mensaje que informa del estado de la aplicación (pasado como parámetro en la variable *request*) para que el usuario sepa el por qué de un error o confirme que la acción solicitada se ha procesado con éxito.

- ***views_checkin.py***

  Este controlador engloba las funciones necesarias para mostrar la información de cualquier *check-in* y poder añadir comentarios al mismo. Cada usuario tiene permisos únicamente para poder acceder y añadir comentarios a los *check-in* realizados por sí mismo. Las funciones que se incluyen en este controlador se enumeran a continuación:

  - `checkin(request, cid)`

Es la función principal del controlador. Recopila toda la información necesaria para pasársela a la vista *checkin* y así poder mostrar la información del *check-in* al usuario propietario del mismo, a través de la siguiente *URL*: `http://localhost/checkin/{cid}`, donde *{cid}* es el identificador único del *check-in*.

- `checkin_save_comment(request,cid)`

  Esta función guarda o modifica el comentario del usuario en el *check-in* identificado por *{cid}* en la base de datos.

- ***views_ connect.py***

  Este controlador incluye todas las funciones de conexión, desconexión y creación de cuentas de usuario en el servidor. Las funciones que se incluyen en este controlador se enumeran a continuación:

  - `mylogin(request)`

    Esta función recibe el nombre de usuario y contraseña (pasados como parámetro en la variable *request*) y comprueba si son correctos para autenticar al usuario y que pueda acceder inmediatamente a su menú personal.

  - `mylogout(request)`

    Esta función desconecta al usuario que recibe como parámetro en la variable *request* del sistema.

  - `newuser(request)`

    Función que recibe los datos de formulario de la vista *newuser* y que comprueba que todos sean correctos para crear una nueva cuenta en el sistema que no existiera con anterioridad. En el caso de que algún campo no sea correcto o el usuario ya exista en la base de datos, devolverá un mensaje de error para que la vista *newuser* se la muestre al usuario en cuestión a través de su interfaz gráfica.

  - `handlerNotification(user, request)`

    Esta función es llamada por la función `mylogin(request)` y maneja las notificaciones de estado de la aplicación en los formularios de

login y creación de cuentas de usuario, de modo que cualquier posible error o notificación que se pase como parámetro *GET* en la vista *login* es mostrado al usuario (es decir, que se muestre un error, por ejemplo).

- `getMsg(request)`

  Esta función es llamada por la función `handlerNotification(user, request)` y devuelve el mensaje que informa del estado de la aplicación (pasado como parámetro en la variable *request*) para que el usuario sepa el por qué de un error o confirme que la acción solicitada se ha procesado con éxito.

- ***views_error.py***

  Este controlador incluye las funciones que manejan los errores del servidor haciendo que se muestren vistas personalizadas en la interfaz de la aplicación.

  - `custom_404_view(request)`

    Esta función maneja los errores 404 del servidor *HTTP*, producidos por intentar acceder a contenido que no existe en él. Esta función muestra la vista *404* personalizada al usuario.

  - `custom_500_view(request)`

    Esta función maneja los errores 500 del servidor *HTTP*, producidos por una condición inesperada que le impidió completar la solicitud del cliente. Esta función muestra la vista *500* personalizada al usuario.

- ***views_manual.py***

  Este controlador incluye las funciones que muestran el manual de usuario en la interfaz gráfica del servidor.

  - `manual_pid(request,pid)`

Esta función recoge toda la información necesaria para pasársela a la vista *manual* y mostrar la página *{pid}* del manual solicitada por el visitante.

- `manual(request)`

  Esta función recoge toda la información necesaria para pasársela a la vista *manual* y mostrar la primera página del manual solicitada por el visitante.

- `step_list()`

  Esta función es llamada por las funciones `manual(request)` y `manual_pid(request,pid)` y devuelve una lista de números enteros que va desde 1 hasta el número de páginas que componen el manual. Esta lista será utilizada por la vista *manual* para dividir todo el contenido del manual en las diferentes páginas que lo componen.

- ***views_ menu.py***

  Es uno de los controladores más importantes de la aplicación. Engloba todas las funciones necesarias para el uso del menú personal por parte de los usuarios de *GeoURJC*. Las funciones que se incluyen en este controlador se enumeran a continuación:

  - `menu(request)`

    Es la función principal del menú personal. Recopila toda la información necesaria para pasársela a la vista *menu* y así poder mostrar el menú personalizado y cada una de las secciones que la componen para cada usuario de *GeoURJC* a través de la siguiente *URL*: `http://localhost/menu`.

  - `editStudentSignatures(user, allsignatures)`

    Esta función recibe como parámetro un objeto *user* correspondiente al usuario que solicita la información y una lista de objetos *allsignatures* que engloba todas las asignaturas (objetos *Signature*) almacenadas en el servidor. Es llamada por la función

`menu(request)`. Su utilidad es la de devolver un listado de objetos *Signature* compuesto por las asignaturas a las que tiene acceso el usuario (independientemente de que sea profesor o estudiante).

- `saveSignatures(request)`

  Esta función recibe toda la información necesaria como parámetro *POST* del formulario de actualización de asignaturas del menú personal de los alumnos y modifica los permisos a las asignaturas seleccionadas para que pueda acceder a ellas y borrarle de aquellas en las que estuviera apuntado y no quiera seguir.

- `saveProfile(request)`

  Esta función recibe toda la información necesaria como parámetro *POST* del formulario que permite editar la información privada del usuario y actualiza dicha información en la base de datos.

- `coloursArray()`

  Esta función es llamada por la función `menu(request)` y genera una lista de colores para llevar a cabo la leyenda de colores dentro del calendario personal de clases y asignaturas disponible para cada usuario de manera personalizada en su menú.

- `handlerNotification(user, request)`

  Esta función es llamada por la función `menu(request)` y maneja las notificaciones de estado de la aplicación en el menú personal del usuario, de modo que cualquier error o notificación que se pase como parámetro *GET* en la vista *menu* es mostrado al usuario (es decir, que se muestre un error, un mensaje de éxito, etc.).

- `getMsg(request)`

  Esta función es llamada por la función `handlerNotification(user, request)` y devuelve el mensaje que informa del estado de la aplicación (pasado como parámetro en la variable *request*) para que el usuario sepa el por qué de un error o confirme que la acción solicitada se ha procesado con éxito.

- ***views_ profile.py***

  Este controlador está compuesto por todas las funciones encargadas de recopilar la información necesaria para mostrar los perfiles públicos de los usuarios. Las funciones que se incluyen en este controlador se enumeran a continuación:

  - `user_profile(request)`

    Es la función principal que genera toda la información necesaria para pasársela a la vista *profile* y poder mostrar tanto la última posición almacenada por el usuario y mostrar un resumen de su información más relevante a través de la siguiente *URL*: `http://localhost/profile/{uid}`, donde *{uid}* es el identificador único de usuario.

  - `handlerNotification(user, request)`

    Esta función es llamada por la función `user_profile(request)` y maneja las notificaciones de estado de la aplicación en el perfil público de los usuarios, de modo que cualquier error o notificación que se pase como parámetro *GET* en la vista *profile* es mostrado al usuario (es decir, que se muestre un error, un mensaje de éxito, etc.).

  - `getMsg(request)`

    Esta función es llamada por la función `handlerNotification(user, request)` y devuelve el mensaje que informa del estado de la aplicación (pasado como parámetro en la variable *request*) para que el usuario sepa el por qué de un error o confirme que la acción solicitada se ha procesado con éxito.

- ***views_ signature.py***

  Este es otro de los controladores más importantes para la interfaz de usuario. Está formado por todas las funciones encargadas de recopilar la información necesaria para mostrar todas las secciones de las asignaturas generadas en *GeoURJC*, así como cada una de las vistas de

las clases que las componen. Primero se explicarán las funciones encargadas de dar la lógica a las asignaturas y, en segundo lugar, aquellas que se encargan de las clases de éstas. Las funciones encargadas de las asignaturas son las siguientes:

- `signature(request, sid)`

  Es la función principal que genera toda la información necesaria para pasársela a la vista *signature* y poder mostrar todas las secciones que componen la asignatura (el calendario personal, los listados de profesores y alumnos que componen la asignatura y las secciones de administración que permiten añadir clases, editar la información de la asignatura o borrarla del sistema), todo ello a través de la siguiente *URL*: `http://localhost/signature/{sid}`, donde *{sid}* es el identificador único de asignatura.

- `editSignature(request, sid)`

  Esta función recibe toda la información necesaria como parámetro *POST* del formulario que permite modificar la información de la asignatura por parte de cualquier administrador y almacena dicha información en la base de datos.

- `deleteSignature(request, sid)`

  Esta función se encarga de realizar todas las acciones necesarias para borrar toda la información almacenada en el sistema relacionada con esta asignatura: *check-in*, clases, permisos, etc.

- `checkActionSignature(request)`

  Esta función genera los permisos para las secciones de la vista de la asignatura (en función de si tiene permisos de administrador o no) para dar el efecto *AJAX* que permita mostrar en la vista *signature* las secciones a las que tiene acceso el usuario.

- `funtopcomments(listcheckins)`

  Esta función es llamada por la funciones `signature(request, sid)` y `classsignature(request, sid, year, month, day)`, recibiendo como parámetro una lista con los *check-ins* almacenados

de la asignatura, genera una lista de usuarios con el número de comentarios de cada uno ordenados de mayor a menor.

- `funtopcheckins(listcheckins)`

  Esta función es llamada por las funciones `signature(request, sid)` y `classsignature(request, sid, year, month, day)`, recibiendo como parámetro una lista con los *check-ins* almacenados de la asignatura, genera una lista de usuarios con el número correspondiente de *check-ins* de cada uno ordenados de mayor a menor.

- `handlerNotification(user, request)`

  Esta función es llamada por las funciones `signature(request)` y `classsignature(request, sid, year, month, day)` y maneja las notificaciones de estado de la aplicación en el perfil público de los usuarios, de modo que cualquier error o notificación que se pase como parámetro *GET* en la vista *profile* es mostrado al usuario (es decir, que se muestre un error, un mensaje de éxito, etc.).

- `getMsg(request)`

  Esta función es llamada por la función `handlerNotification(user, request)` y devuelve el mensaje que informa del estado de la aplicación (pasado como parámetro en la variable *request*) para que el usuario sepa el por qué de un error o confirme que la acción solicitada se ha procesado con éxito.

A continuación, se enumeran las funciones que se encargan de dar la lógica a las clases de las asignaturas:

- `classsignature(request, sid, year, month, day)`

  Es la función principal que genera toda la información necesaria para pasársela a la vista *class* y poder mostrar todas las secciones que componen la clase (el apartado de comentarios de los usuarios que han acudido a la clase, la lista de asistencia de estudiantes y las secciones de administración que permiten editar la información de la clase o ver el código QR asociado), a través de la *URL*:

`http://localhost/signature/{sid}/{año}-{mes}-{día}`, donde *{sid}* es el identificador único de asignatura y *{año}*, *{mes}* y *{día}* corresponden a la fecha de la clase.

- `editClass(request, sid)`

  Esta función recibe toda la información necesaria como parámetro *POST* del formulario disponible en la sección que permite editar la información de la clase por parte de cualquier administrador y modifica dicha información en la base de datos.

- `addClass(request, sid)`

  Esta función se encarga de recoger las variables necesarias como parámetro *POST* del formulario disponible en el menú de administración de la asignatura y se encarga de crear la clase para que esté disponible para todos los usuarios con acceso a la asignatura.

- `checkActionClass(request)`

  Esta función genera los permisos para las secciones de la vista de la clase seleccionada (en función de si tiene permisos de administrador o no) para dar el efecto *AJAX* que permita mostrar en la vista *class* las secciones a las que tiene acceso el usuario.

- ***views_sync.py***

  Este controlador es sin duda el más importante. En él se incluye toda la lógica encargada de ser pasarela entre los clientes y el servidor, tanto para realizar *check-in* como para la conexión de los clientes a la aplicación. Como se explicó en apartados anteriores, la necesidad de que las conexiones sean lo más sencillas posibles hace que la comunicación entre clientes y servidores sea a través de respuestas *HTTP* con *GET* (las respuestas del servidor a los clientes) y *POST* (las solicitudes de los clientes a los servidores). El servidor devolverá un mensaje de estado a los clientes indicando si el comando se ha realizado con éxito o no. Las funciones que se incluyen se enumeran en el siguiente listado:

  - `sync(request)`

Es la función principal de este controlador. Recibe mediante parámetro *POST* el comando que se quiere ejecutar desde el cliente y los campos necesarios para ejecutarlo y llama a la función que se encarga de tratar dicha solicitud de servicio. En el caso de que el comando no exista en el servidor, la respuesta del servidor será vacía.

- `sync_login(user)`

  Esta función es llamada desde la función `sync(request)` y permite al cliente *Android* autenticarse en el sistema para, posteriormente, poder hacer *check-in*. Recibe como parámetros *POST* el usuario y la contraseña, y en función del grupo al que corresponda el usuario le indica al cliente *Android* que muestre la vista que le corresponda. En el caso de que el *login* sea correcto y el usuario sea un alumno, devuelve el estado *login-ok-student*. Si el grupo al que pertenece el usuario es un profesor, devuelve como respuesta al cliente el estado *login-ok-teacher*. Si el login no es correcto, devuelve el estado *login-failed*.

- `sync_checkclass(user)`

  Esta función es llamada por la función `sync(request)` y comprueba si el usuario solicitante con su dispositivo *Android* tiene una clase ahora mismo o en los próximos minutos, devolviendo un mensaje de estado dependiendo de cada situación. Si el usuario tiene una clase en el instante de comprobación, devuelve el estado *class-in-yes* y los minutos restantes. Si el usuario ha hecho *check-in* en una clase que tiene en el momento de comprobación, devuelve el estado *class-in-checkin*. Si por el contrario el alumno o profesor tiene clase en los próximos minutos, responderá con el estado *class-next-yes*. Si la clase ha finalizado hace menos de 10 minutos, el estado que devuelve el servidor será *class-finished-yes*. Si no tiene ninguna clase ni en ese momento ni en un futuro a corto plazo, devolverá el estado *class-no*.

- `sync_setGPS(user)`

Esta función es llamada por la función `sync(request)` y es la encargada de tratar las solicitudes de *check-in* de los clientes en *Android*. Se encarga de comprobar si el usuario tiene en ese mismo instante clase y no ha hecho *check-in* con anterioridad. En el caso de que se cumplan ambas condiciones, comprueba que el usuario y contraseña sean correctos y que el código QR se haya decodificado de manera correcta. Además, se encarga de analizar si el usuario se encuentra en el mismo edificio donde se está impartiendo la clase (para indicar si el *check-in* es malicioso o no). Si todo ello se cumple, almacena el *check-in* y devuelve como estado *class-checkin-ok*. En el caso de que el código QR no coincida con el código QR generado para esa clase, se devolverá el mensaje de error *class-checkin-qr-error*.

4.2.2.3 Vistas de la interfaz gráfica

Tras la implementación de los modelos y los controladores, el servidor ya puede funcionar. Sin embargo, es necesario generar la interfaz gráfica necesaria para que los usuarios y los administradores puedan acceder a sus contenidos de un modo cómodo y sencillo. Para ello, es necesario desarrollar las vistas necesarias para lograr que cualquier persona que desconoce por completo el funcionamiento interno de la aplicación pueda acceder sin ningún contratiempo.

Como se explicó en el Apartado 4.2.1 en el que se habló de la estructura de directorios y ficheros de la aplicación, todas las vistas se guardarán dentro del directorio *template* y sus nombres de fichero debe seguir el siguiente formato: **.html*. Es necesario recordar al lector que ninguna vista debe incluir líneas de código que formulen cierta lógica de la aplicación, ya que toda ella debe ser desarrollada íntegramente en los controladores. Por ello, las vistas se encargan de recibir la información por parte de éstos y adaptarla para mostrársela al usuario de la forma más cómoda que sea posible.

Toda página *web* que se precie debe estar compuesta por una plantilla de estilo con el fin de hacer más agradable la navegación a los usuarios. En el

caso del presente Proyecto Fin de Carrera, se ha optado por incorporar a la aplicación *Django* una plantilla[1] que tiene licencia *Creative Commons 2.5* y código abierto para incorporarlo a cualquier proyecto de manera totalmente gratuita. En la Figura 4.3 se muestra la plantilla original de la que partió el diseño.

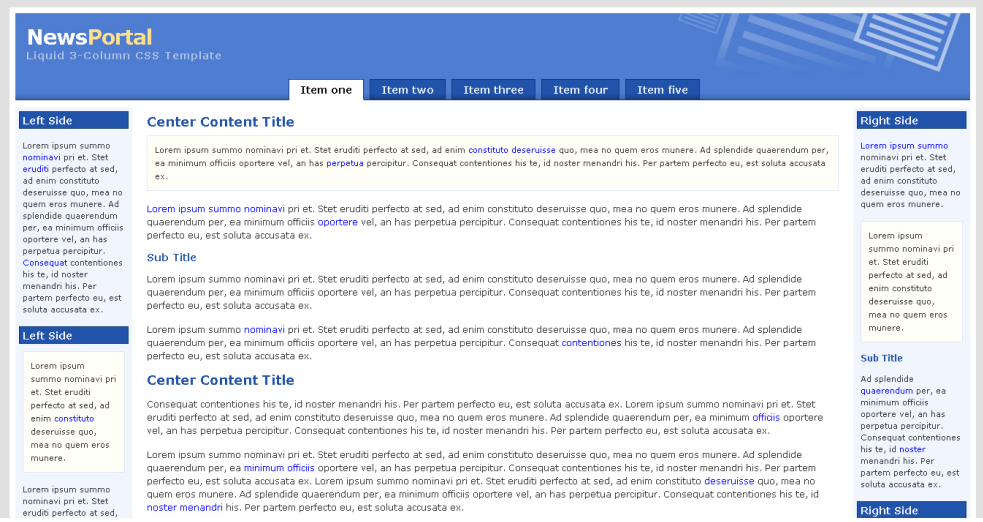


Figura 4.3: Plantilla original de la que parte el diseño de la interfaz del servidor de *GeoURJC*

La plantilla permite tener un archivo en el que se diseña toda la plantilla de la aplicación. En el proyecto que se está desarrollando, la plantilla se almacena en el fichero *template.html* y es de éste del que parten todas las vistas. La plantilla está dividida en bloques diferentes que permiten a las vistas adaptar su contenido siempre a la plantilla original (títulos de texto, divisiones dentro de la pantalla con etiquetas * *, etc.), de modo que otorga total estabilidad y un diseño común a toda la aplicación.

Además de una plantilla, en este Proyecto Fin de Carrera se han incorporado diversos *plug-ins* desarrollados a partir de la librería *jQuery*[2]. *JQuery*

[1] Página *web* oficial de la plantilla en la que se basa la aplicación *Django*: `http://www.designsbydarren.com/preview/news-portal/`

[2] Página *web* oficial de la librería *jQuery*: `http://jquery.com/`

es una biblioteca de *JavaScript* que permite simplificar la manera de interactuar con los documentos *HTML*, manejar eventos, desarrollar animaciones y agregar interacción con la técnica *AJAX* a páginas *web*. De esta forma, se le da dinamismo a la aplicación *web* y hace que la navegación por las distintas vistas sea más intuitiva. Estos *plug-ins* implementados se distribuyen igualmente con licencia *Creative Commons* que permite su uso libre en cualquier proyecto.

A continuación se explican todas y cada una de las vistas desarrolladas para la interfaz *web* del servidor.

- ***login.html***

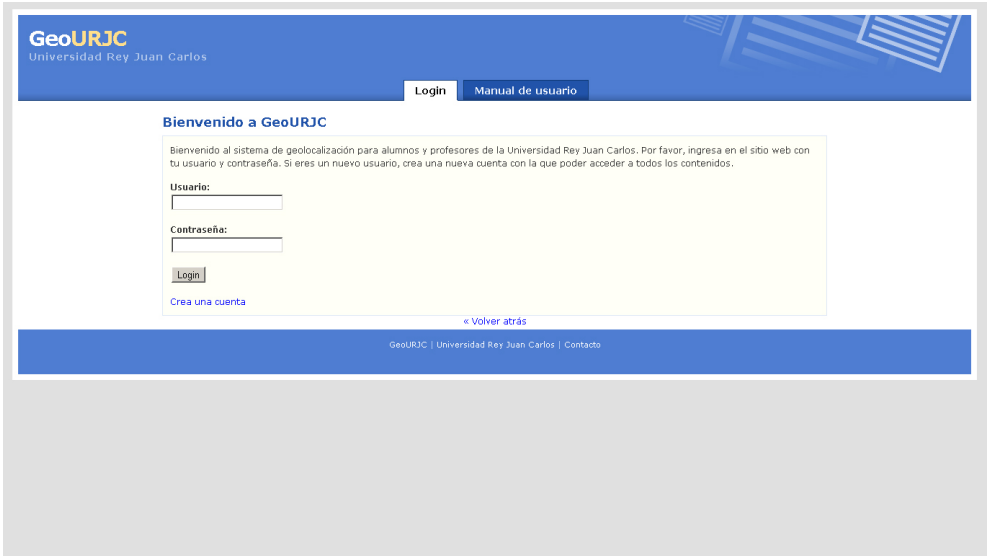


Figura 4.4: Captura de imagen de la vista *login.html*

Nada más acceder a la interfaz *web*, si el usuario no se ha autenticado antes, la primera vista que aparece es la que contiene el formulario de acceso a la aplicación. En la Figura 4.4 se muestra una imagen en la que se puede observar que, además del formulario de acceso, existe la posibilidad de acceder al manual de usuario de la aplicación y al apartado donde un visitante puede crearse una cuenta de usuario.

- ***manual.html***

  Antes de crearse una cuenta de usuario, el visitante puede tener interés en leerse el manual de uso de *GeoURJC* y acceder a él a través de la siguiente *URL*: `http://localhost/manual`, donde poder leer con detalle los pasos a dar para la creación de la cuenta y el uso de la aplicación. Como se puede observar en la Figura 4.5, el manual está dividido en diferentes páginas donde poder leer con comodidad todos y cada uno de los pasos que debe realizar un usuario que quiera utilizar tanto el servidor como el cliente.

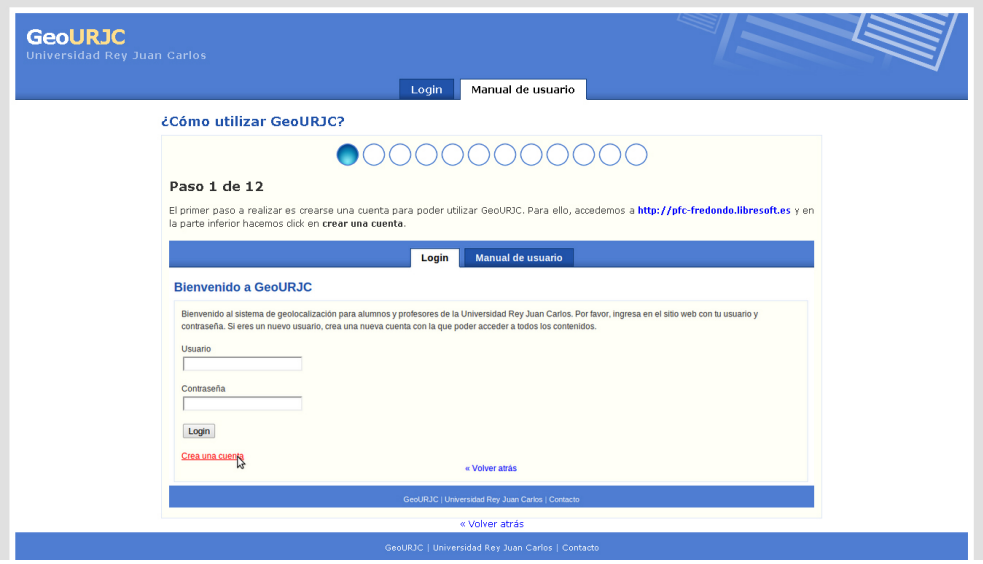


Figura 4.5: Captura de imagen de la vista *manual.html*

- ***newuser.html***

  La otra opción disponible desde la página de acceso es la opción de que el visitante quiera crearse una cuenta de usuario. Para ello, accede a través de la *URL* `http://localhost/newuser` y verá el formulario que debe rellenar para crearse una cuenta de usuario. Como se puede observar en la Figura 4.6, el formulario está compuesto por el nombre

de acceso a la aplicación, una contraseña, el nombre y apellidos del visitante y por último el correo electrónico de la *Universidad Rey Juan Carlos*. Cabe aclarar al lector en este punto que la elección del correo electrónico determinará la asignación del grupo al que corresponderá su cuenta de usuario. Si su cuenta de correo tiene como dominio *@urjc.es* corresponde a un profesor y si es *@alumnos.urjc.es* la cuenta del visitante es de un alumno de la universidad. Una vez rellenados todos los datos, se creará la cuenta si todo el proceso ha ido correctamente y ya podrá acceder a todos los contenidos de la aplicación.

Figura 4.6: Captura de imagen de la vista *newuser.html*

- ***menu.html***

  Una vez el usuario accede con su nombre y su contraseña, la primera vista a la que se le redirige es a su menú personal. La vista predefinida al acceder es la vista de su calendario personal de clases y asignaturas, viendo en la columna izquierda las asignaturas a las que tiene acceso el usuario, ya sea porque se ha apuntado como alumno o porque el

administrador le ha dado permisos al profesor. Además de este calendario, como se puede observar en la Figura 4.7 existen diferentes iconos en la parte central que permiten al usuario moverse por las diferentes secciones que componen este menú: ver el calendario personal, ver los últimos *check-ins* del usuario, editar la información privada de su perfil y el apartado donde los alumnos pueden añadir/borrar asignaturas de su menú personal.

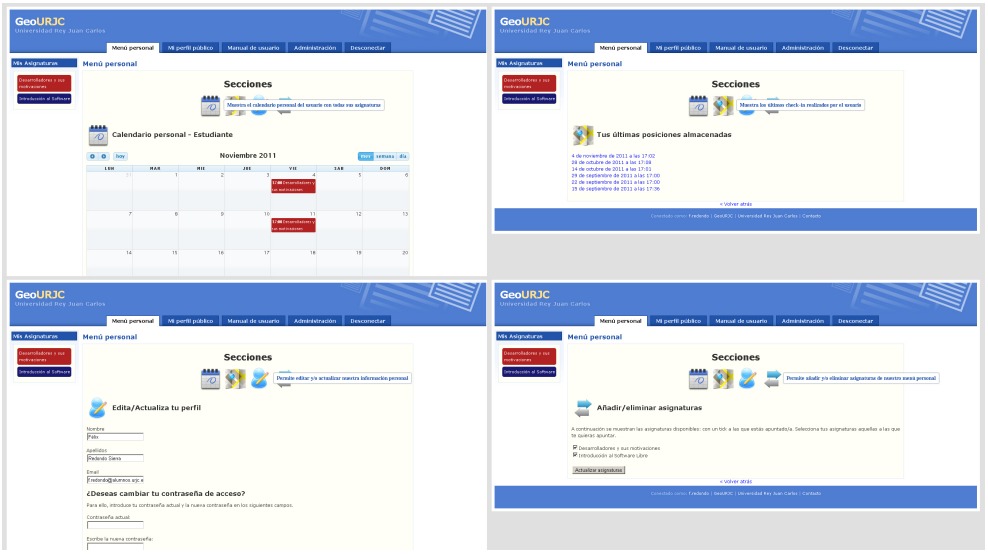


Figura 4.7: Cuatro capturas de imagen de las secciones de la vista *menu.html*

El mosaico de las cuatro imágenes de la Figura 4.7 está realizado desde una cuenta de alumno con permisos de administrador. En esta vista no hay diferencias si el usuario es administrador o no, pero sí hay una pequeña diferencia si el usuario es alumno o profesor. El último icono de las secciones (cuarta captura de la Figura 4.7) es el que utiliza el alumno para poder apuntarse a las asignaturas. Los profesores no tienen acceso a esta sección, ya que para poder tener permisos de profesor deben ponerse en contacto con los administradores de la aplicación.

Esto es así para evitar a usuarios malintencionados que puedan usurpar la identidad y darse permisos que realmente no deben tener.

- ***checkin.html***

  En la segunda sección disponible dentro del menú personal del usuario, se puede acceder a los últimos *check-ins* realizados por el usuario. A la hora de acceder a uno de ellos, se ofrece la posibilidad de añadir un comentario que facilite o aclare los contenidos de la clase a la que pertenece dicho *check-in* a otros alumnos que no hayan podido asistir.

  Dentro de estas vistas, se ofrece al usuario un resumen de su contenido: la asignatura y la clase a la que corresponde el *check-in* y se da la posibilidad al usuario de añadir el mensaje sobre la clase indicado anteriormente. Además, se incorporan dos iconos que sirven como acceso directo a la clase y a la asignatura a la que corresponde el *check-in*. Se muestra en la Figura 4.8 dos capturas de imagen que muestran la vista *checkin.html* y sus secciones disponibles.

Figura 4.8: Dos capturas de imagen de la vista *checkin.html*

- ***signature.html***

  Desde el menú situado a la izquierda del menú personal o desde el mismo calendario personal, se puede acceder a la vista personalizada de cada una de las asignaturas a las que el usuario tiene permiso. Como

se puede observar, existen dos clasificaciones o *rankings* para cada asignatura en la parte izquierda de la vista. Por un lado, se muestra una clasificación de los alumnos que han realizado más *check-ins* dentro de la asignatura y que, por tanto, son los alumnos que más acuden a clase. Asimismo, existe una segunda clasificación que muestra los alumnos que más comentarios han escrito dentro de las clases de la asignatura y son, por tanto, los alumnos más activos en la asignatura dentro de *GeoURJC*. Como en vistas anteriores, también hay disponibles diferentes secciones que están disponibles dependiendo de si el usuario es tiene permisos de administrador o no.

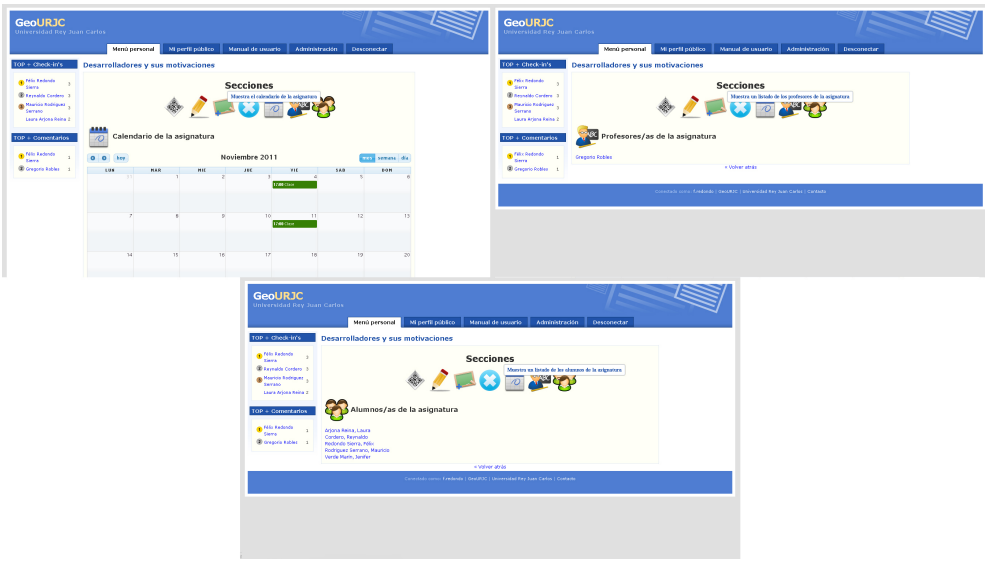


Figura 4.9: Tres capturas de las secciones de la vista *signature.html*

Las secciones disponibles por cualquier usuario (independientemente de sus permisos) son el calendario de clases de la asignatura (en el cuál aparecen de color azul las clases que no se han efectuado y en color verde aquellas que sí se han realizado porque al menos un profesor ha hecho *check-in*), un listado con los profesores de la asignatura y otro

listado con los alumnos que están apuntados a la misma. En la Figura 4.9 se muestran tres capturas con cada una de las secciones indicadas.

Si el usuario es administrador, tendrá acceso a unas secciones privilegiadas: un apartado donde encontrará un listado con todos y cada uno de los códigos QR generados por la aplicación para cada clase, una sección donde podrá editar la información de la asignatura, otro apartado donde podrá añadir clases extraordinarias o adicionales que no estuvieran contempladas en un inicio y, por último, la sección desde donde podrá borrar la asignatura y toda la información que está directamente relacionada con ella: *check-ins*, clases, permisos de usuarios y profesores, etc. En la Figura 4.10 se muestran cuatro capturas de estos apartados disponibles sólo con permisos de administrador.

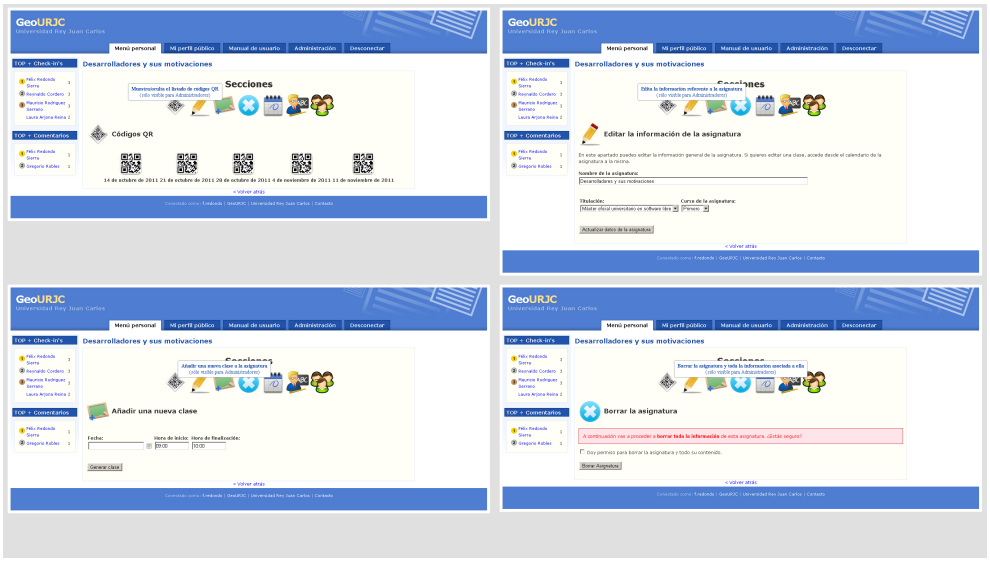


Figura 4.10: Cuatro capturas de imagen de las secciones con permisos de administrador de la vista *signature.html*

- ***class.html***

  Una vez que el profesor haya hecho *check-in* en su clase, se habilitará dentro del menú de la asignatura en la vista anterior la posibilidad de

acceder al contenido de la clase. Como en algunas de las anteriores vistas, el usuario podrá acceder a diferentes secciones independientemente de que tenga permisos o no de administrador.

Las secciones disponibles para cualquier usuario son las siguientes: un apartado donde se podrán ver todos los comentarios de los profesores y alumnos que han asistido a la clase (y hayan editado su *check-in* como se explicó en la vista *checkin.html*) y una sección donde ver el listado completo de alumnos que han estado en ella. Los administradores tendrán acceso a dos secciones adicionales: una sección donde podrán ver el código QR generado para esa clase y que sirve para hacer *check-in* por parte de los usuarios y un formulario donde podrán editar la información de la clase (fecha, hora de inicio y hora de finalización).

En la Figura 4.11 se muestran cuatro capturas de las secciones que componen esta vista a modo informativo para el lector.

Figura 4.11: Cuatro capturas de imagen de las secciones de la vista *class.html*

- ***profile.html***

Como habrá podido observar el lector a lo largo de todas las vistas, es muy habitual generar listados de usuarios (clasificaciones, alumnos que están apuntados a una asignatura, profesores, etc.). Todos ellos a la hora de la navegación están enlazados en todo momento a sus respectivos perfiles públicos para que el usuario pueda contactar con ellos o conocer mayor información de ellos.

Estos perfiles públicos tienen como objetivo que otros usuarios (profesores y alumnos) tengan conocimiento sobre quién es el usuario y cuál ha sido su último *check-in* para poder contactar con ellos. De este modo, en el perfil público se encuentran dos apartados o secciones: por un lado, una ficha en la que poder ver la información pública del usuario, tal como su nombre y apellidos, su email o la hora a la que se conectó por última vez al sistema. Por otro lado, se muestra un mapa de *Google Maps* que muestra la última posición almacenada por el propietario del perfil público y la fecha y hora a la que se produjo, simplemente a nivel informativo y con la única intención de que la aplicación sirva como mecanismo de socialización y comunicación entre los participantes. Como con las anteriores vistas, se adjunta en la Figura 4.12 dos capturas de las dos secciones que se pueden encontrar en esta vista.

Figura 4.12: Dos capturas de imagen de las secciones de la vista *profile.html*

- ***adminpanel.html***

Aunque *Django* ofrece un panel de administración propio para cada proyecto que se desarrolla con este *framework*, *GeoURJC* cuenta con su propio panel personalizado compuesto por tres secciones adicionales que son de utilidad para los administradores. La primera sección útil para los administradores se compone por un formulario que permite crear una asignatura de manera completamente automática (asignatura, clases y códigos QR). La segunda sección muestra un listado de éstas y ofrece accesos directos a los apartados que permiten añadir una nueva clase y borrar por completo una asignatura (dentro de la vista *signature.html*). La tercera y última sección, muestra un listado de los *check-ins* maliciosos. Un *check-in* malicioso es aquel realizado por un usuario que no se encuentra en el mismo edificio donde se está impartiendo la clase correspondiente. Con este listado, los administradores podrán sancionar a aquellos usuarios que infrinjan las normas.

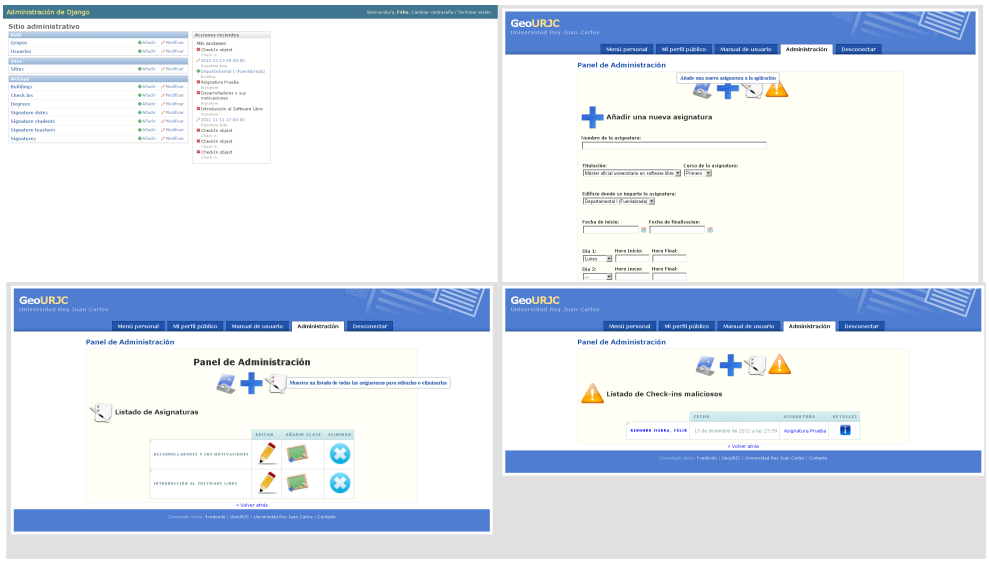


Figura 4.13: Cuatro capturas de imagen del panel de Administración de *GeoURJC*

Asimismo, se incorpora un cuarto icono que sirve como acceso directo al panel de administración de *Django*. La administración de los permisos de los profesores y de los alumnos, así como su mantenimiento, se ha decidido que se haga a través de él.

En la Figura 4.13 se muestran cuatro capturas del panel de Administración que muestran el panel predefinido de *Django* y las tres secciones que componen el panel personalizado de *GeoURJC*.

- ***400.html y 500.html***

  Django permite personalizar aquellos errores por acceso a contenido que no existe en el servidor o por fallos en las consultas necesarias para mostrar un contenido solicitado. Para ello, se han generado dos vistas que contemplan estos errores más comunes dentro de un sitio *web*.

Y una vez explicadas las vistas, se da por finalizada la implementación de la parte servidor de *GeoURJC*. En el siguiente punto se estudiará la implementación de la aplicación *Android* como cliente del sistema.

4.3 Implementación del cliente

4.3.1 Estructura de directorios y ficheros

Al igual que se ha realizado en el Apartado 4.2 para la implementación del servidor, se va a explicar la estructura de directorios y ficheros que componen la aplicación desarrollada en *Java* para *Android* representado en la Figura 4.14. A continuación se realiza una breve introducción a los ficheros y directorios más importantes:

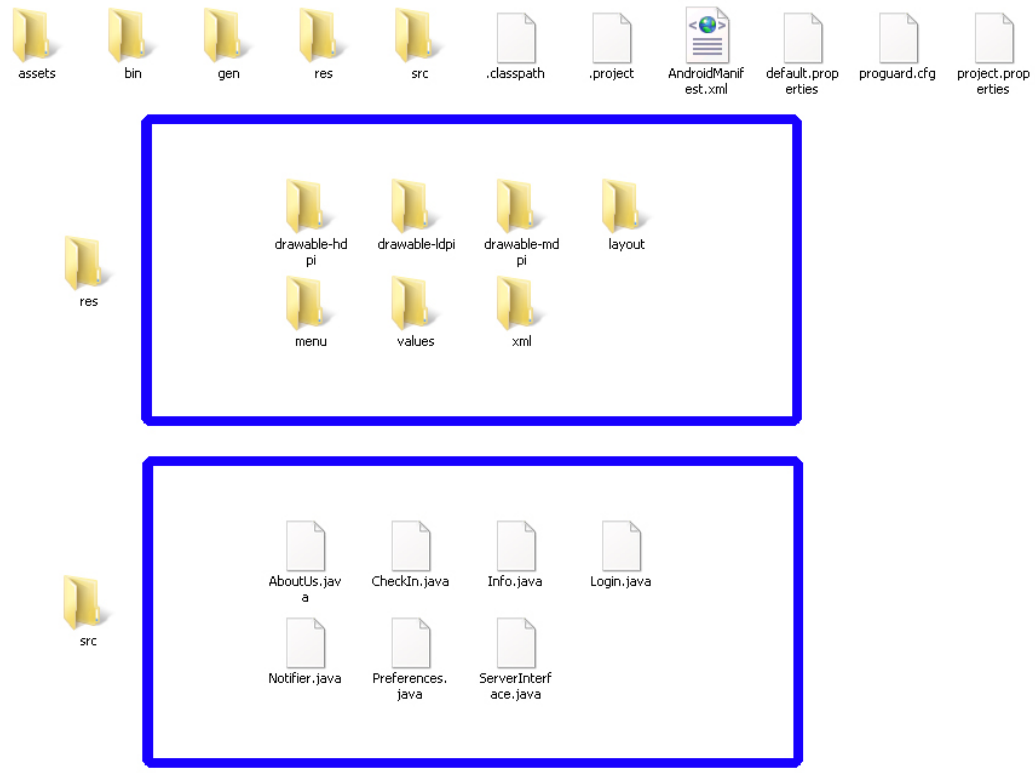


Figura 4.14: Estructura de directorios y ficheros del cliente programado para *Android*

- ***res***

  En esta carpeta del proyecto se almacenan todos los ficheros relacionados con las vistas de las *Activities*.

  Entre otras cosas, se almacenan todas las imágenes utilizadas por la aplicación distribuidas en diferentes carpetas en función de su tamaño en píxeles: *drawable-hdpi* (para imágenes grandes), *drawable-mdpi* (para imágenes grandes) y *drawable-ldpi* (para imágenes grandes). Debido a que las aplicaciones *Android* se utilizan en diferentes dispositivos con diferentes resoluciones, es de vital importancia generar cada imagen con el tamaño correspondiente y que esté disponible en las tres resoluciones para que el sistema operativo ofrezca la aplicación con la mejor calidad posible.

En la carpeta *layout* se almacenan todos los diseños de las vistas (que son generados en ficheros **.xml*) y que se explicarán más adelante. También se almacena dentro del directorio *menu* la estructura del menú de la aplicación (desde el que se podrá acceder a los ajustes de la aplicación o activar/desactivar el *GPS*). En la carpeta *values* se almacenan todas las constantes de la aplicación *Android*. En el caso de este proyecto se almacena un fichero *strings.xml* que almacena los *strings* utilizados en el diseño y funcionamiento de las *Activities* y un fichero *arrays.xml* en el que se almacena la configuración del notificador de clases. Por último, en la carpeta *xml* se almacena el fichero preferences.xml que almacena la estructura de los ajustes de la aplicación que se acceden desde el menú como se dijo anteriormente.

- ***src/es/gsyc/geourjc***

  En este directorio se almacenan todas las *Activities* y *Services* que componen la aplicación cliente de *GeoURJC*. Estas *Activities* son clases programadas en *Java* y son las encargadas de efectuar toda la lógica de la aplicación.

- ***AndroidManifest.xml***

  Es uno de los ficheros principales de cualquier proyecto *Android*. En él, se define la versión de la aplicación (para realizar actualizaciones futuras), se enumeran todas y cada una de las *Activities* y *Services* que trabajan en ella (indicando cuál es la *Activity* inicial al arrancar la aplicación) y, por último, se almacenan también los permisos de acceso de las aplicaciones a herramientas y datos del dispositivo *smartphone* (por ejemplo, acceso a Internet, al uso de la geolocalización de la posición del dispositivo, etc.).

4.3.2 Explicación de la aplicación

Como se ha indicado en multitud de ocasiones en el desarrollo de la memoria, la aplicación cliente debe ser lo más ligera y sencilla posible porque toda la lógica del sistema debe recaer sobre el servidor. Por ello, no es necesario

dividir la explicación de la implementación de la misma sino que se irán explicando las vistas de las diferentes *Activities* y la lógica de la aplicación conforme se vaya navegando por ella a través de estas palabras.

Antes de comenzar con la explicación de cada una de las *Activities* que componen la aplicación cliente de *GeoURJC*, se hace un inciso para explicar conceptualmente cómo realiza la conexión el cliente. Como ya se indicó durante la implementación del servidor, el cliente se comunica mediante mensajes *POST* enviados a través del protocolo *HTTP* a la *URI /sync* del servidor. En ellos, se incluye el comando de solicitud de servicio y, además, todas las variables necesarias para que el servidor haga las comprobaciones pertinentes antes de devolver una respuesta al cliente.

Figura 4.15: Captura de pantalla de la *Activity Login*

Todas las funciones de comunicación están almacenadas en la clase *ServerInterface.java* del paquete *es.gsyc.geourjc*. La función o procedimiento más importante de esta clase es `executeHttpRequest(String data)`, que envía a la *URI* indicada anteriormente los datos que recibe como parámetro en la variable *data* (el comando y las variables necesarias) y devuelve un *String*

con la respuesta del servidor a la *Activity* que ha hecho la solicitud. Una vez hecha esta aclaración, se puede comenzar con el desarrollo de cada una de las *Activities*.

Al iniciar la aplicación *GeoURJC* para *Android*, la *Activity* que se muestra al usuario es la que se almacena en el fichero *Login.java* del paquete *es.gsyc.geourjc*. En la Figura 4.15 se muestra una captura del diseño de la *Activity*. En ella, el usuario deberá autenticarse para poder acceder a la *Activity* que se encarga de efectuar el *check-in*.

La *Activity Login*, una vez el usuario escriba su usuario y contraseña y pulse el botón Conectar, lanza un nuevo hilo de ejecución (o *Thread*) asíncrono que envía una solicitud de autenticación al servidor a través del comando *login*. Si el proceso de autenticación es correcto, la *Activity Login* lanza una nueva *Activity* llamada *CheckIn* (*CheckIn.java*) desde la que se deberá realizar el *check-in* por parte del usuario y finaliza su ciclo de vida de *Activity* matándose con la función *finish()*. Además, la *Activity Login* le pasará a la *Activity CheckIn* el nombre de usuario y contraseña para que pueda ser enviado sin volver a ser escrito por el usuario cuando el cliente quiera realizar cualquier otra solicitud al servidor.

En función de la respuesta que el cliente reciba del servidor (*login-ok-student* o *login-ok-teacher*), la aplicación deberá cargar un diseño diferente (*checkin_student.xml* o *check-in_teacher.xml*). La única diferencia de diseño es que si el usuario es un profesor, aparecerá un campo de texto en el que deberá escribir el número de estudiantes que hay en el aula, mientras que en la vista del alumno este campo no estará disponible, siendo el resto del diseño exactamente igual. En la Figura 4.16 se muestra la vista de la *Activity CheckIn* si el usuario es un profesor por ser más completa.

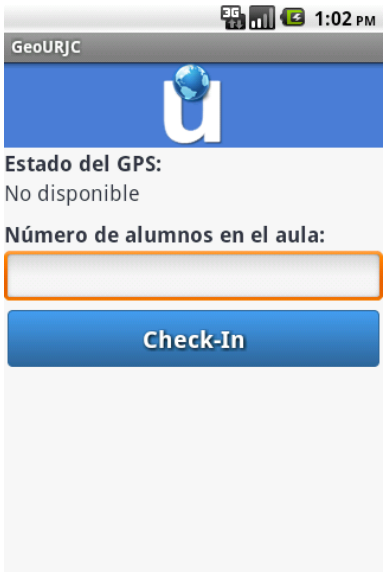


Figura 4.16: Captura de pantalla de la *Activity CheckIn* para un usuario profesor

Esta *Activity* nada más iniciarse tiene una función fundamental: lanzar el *Service* que funcionará como notificador de clases del usuario. Este *Service* periódicamente preguntará al servidor si el usuario tiene una clase en ese instante o en un futuro cercano para lanzar una notificación a través de la barra de notificaciones al usuario y recordarle cuándo debe realizar los *check-ins*. El tiempo que tardará en preguntar este servicio es configurado por el usuario a través de los ajustes de la aplicación que se explicarán más adelante. Además, esta *Activity* configura todos los aspectos necesarios para poder tomar la geolocalización del dispositivo.

Cuando el usuario quiere realizar un *check-in* y pulsa el botón de la aplicación, la *Activity* comprueba si la detección de la geolocalización está activada. En el caso de que no lo esté, la propia aplicación muestra al usuario los ajustes de geolocalización para que active la detección a través de redes. El motivo principal por el que se utiliza la detección a través de redes es debido a que la geolocalización a través de los satélites *GPS* en edificios no es viable. La detección a través de redes no es exacta, pero otorga una posición

aproximada que permite al sistema detectar, al menos, que el dispositivo se encuentra cerca del punto donde debe realizar el *login*.

Una vez activada la detección, la *Activity* debe comprobar si el dispositivo tiene instalada la aplicación *Barcode Scanner* que se encargará de realizar la decodificación de los códigos QR. En el caso de que el usuario no la tenga disponible en su dispositivo, lanzará un aviso al usuario para que instale la aplicación, abriendo el enlace a la *Android Market* de la aplicación para que sea instalada.

Con todo preparado y configurado correctamente, al pulsar en el botón *Check-In* la *Activity* envía una solicitud de comprobación de clases al servidor a través del comando *checkinclass*. Si el servidor confirma que hay una clase en esos momentos a la que se debe hacer *check-in*, la *Activity CheckIn* lanza una nueva *Activity*, pero en esta ocasión será una *Activity* de la aplicación *Barcode Scanner* que permitirá leer y decodificar el código QR para confirmar la asistencia del usuario a la clase, finalizándose de inmediato una vez decodifique el código QR y devolviendo el resultado a la *Activity CheckIn*. En la Figura 4.17 se observa una captura del funcionamiento de esta *Activity* de la aplicación *Barcode Scanner*.

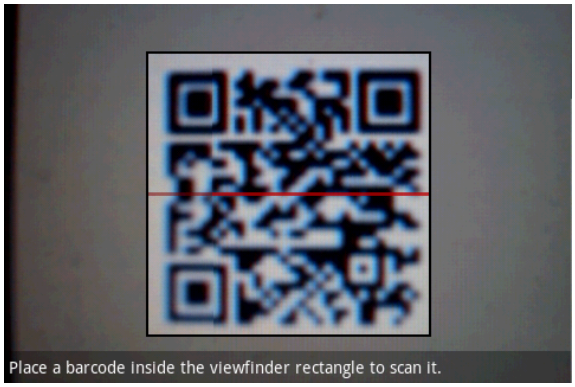


Figura 4.17: Captura de pantalla de la aplicación *Barcode Scanner*

La *Activity CheckIn* comprobará que el código QR mantenga el formato que se ha diseñado para los códigos que se utilizan en *GeoURJC* y que se

explicó en el Apartado 3.2.2.1. Si el formato cumple el estándar dispuesto, la *Activity CheckIn* almacenará todo el contenido necesario para el *check-in* en la clase *Info* recogida en el fichero *Info.java* del directorio *src* del proyecto: latitud, longitud, año, mes, día y clave aleatoria de la clase. A continuación, lanza un nuevo hilo de ejecución (o *Thread*) que será el encargado de mandar toda la información al servidor a través del comando *setGPS*. Si todo ha ido bien, el servidor responderá con un mensaje de estado *class-checkin-ok* o devolverá un mensaje de error (por ejemplo, si alguno de los parámetros del código QR leído es incorrecto con respecto a la clase del usuario).

Para finalizar la explicación de la aplicación para *Android*, queda por explicar el menú de ajustes de la aplicación. Al pulsar en el botón correspondiente de nuestro dispositivo, se observará una nueva pestaña que incluye tres enlaces: acceso a los ajustes propios de la aplicación, acceso a los ajustes de geolocalización del *smartphone* y un acceso a una página de información de la aplicación. En la Figura 4.18 se muestra una captura de imagen de estos accesos.

Figura 4.18: Captura de pantalla del menú de ajustes

Si se accede a los ajustes propios de la aplicación, se observará que en él se puede configurar la posibilidad de guardar el usuario y contraseña para no tener la necesidad de escribirlos en cada acceso del usuario a la aplicación. Además, se da la opción de configurar la periodicidad de comprobación de clases del *Service Notifier*. El código de esta *Activity* está disponible en el fichero *Preferences.java* del directorio *src* del proyecto. Además, la estructura de las duraciones seleccionables para configurar el notificador se encuentran disponibles en el fichero *arrays.xml* del directorio *res/xml*. Si se pulsa en la opción *Acerca de...* del menú de la aplicación, se observará toda la información referente al proyecto *GeoURJC*, nombre del autor de la aplicación, versión, fecha de publicación, correo electrónico, etc. A modo de conclusión del capítulo, en la Figura 4.19 se muestran dos capturas de pantalla de ambos apartados del menú.

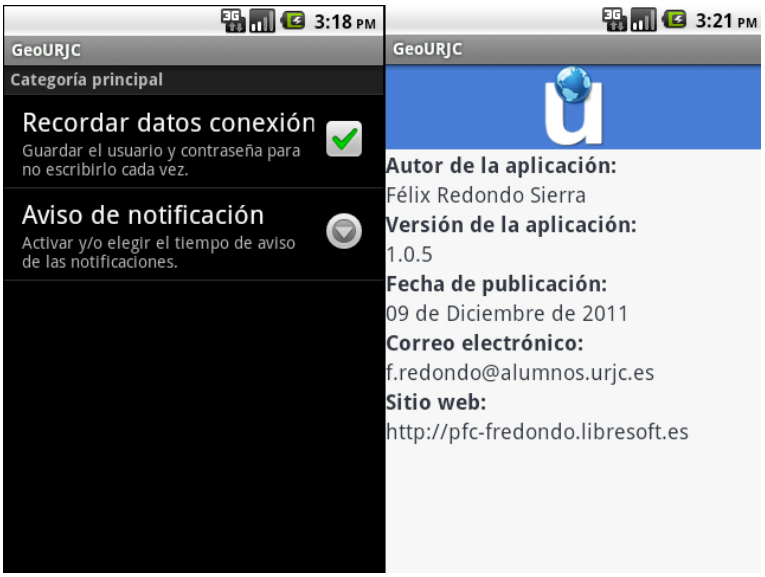


Figura 4.19: Dos capturas de pantalla del menú de Configuración y Acerca de...

Capítulo 5

Despliegue y resultados

En este capítulo se presenta el proceso de pruebas que se han llevado a cabo para comprobar el funcionamiento del sistema implementado. En él, se explicarán las condiciones de todos los experimentos, los objetivos iniciales que se buscaban, los resultados obtenidos y unas conclusiones finales tras el proceso.

5.1 Introducción

A la hora del desarrollo de un sistema con cierta envergadura y complejidad, el paso fundamental por el que toda creación debe pasar es la fase de pruebas y experimentación con personal ajeno al equipo que ha trabajado en el proyecto. La finalidad de esta fase no es más que comprobar que el sistema funciona con diferentes dispositivos, solucionar todos los problemas que puedan surgir y comprobar el grado de aceptación por parte de los usuarios, en el caso de este Proyecto Fin de Carrera, profesores y alumnos de la universidad.

Para ello, se han realizado dos pruebas o experimentos, ambos en un entorno controlado donde el número de alumnos potenciales no superaran la quincena. Ambas pruebas se han llevado a cabo en el *Máster oficial en Software Libre* que se imparte en el *Campus de Fuenlabrada* de la *Universidad Rey Juan Carlos*. A continuación se explican ambos experimentos, en orden cronológico.

5.2 Primer Experimento

En este primer experimento el sistema todavía estaba en fase de implementación. La funcionalidad era básica, cumpliendo únicamente el objetivo más prioritario de todos: detectar la posición del usuario que utilizara la aplicación y realizar el *check-in* de la clase por parte de los usuarios.

En la aplicación para *Android* no estaban contemplados los códigos QR como mecanismo de seguridad, no existía el notificador de clases ni los menús de ajustes y el diseño era muy básico. En la aplicación para *Django* no existían los calendarios personales, las clasificaciones de *check-ins* realizados y comentarios, el diseño era mucho más simple (sin el uso de plantillas) y no se disponía de *plug-ins* desarrollados con *jQuery* y *AJAX*.

Los datos del experimento son los siguientes:

- **Objetivos iniciales del experimento:**
  - Comprobar el funcionamiento básico de la aplicación en otros dispositivos que el utilizado para las pruebas.
  - Comprobar la usabilidad de la aplicación por personas ajenas al desarrollo.
  - Recopilar información para solucionar el problema de la geolocalización en interiores.
- **Fecha de inicio:** 15 de septiembre de 2011.
- **Fecha de finalización:** 06 de octubre de 2011.
- **Asignatura en la que se han realizado las pruebas:** Introducción al *Software* Libre.
- **Titulación de la asignatura:** Máster oficial en *Software* Libre.
- **Lugar donde se han realizado las pruebas:** Seminario 225 del Departamental I del *Campus de Fuenlabrada* de la *Universidad Rey Juan Carlos*.

- **Pruebas diseñadas:** Se han planteado cuatro sesiones de prueba para que aquellos usuarios que lo deseen puedan utilizar la aplicación.

5.2.1 Resultados

En esta primera fase de pruebas, el éxito o fracaso de la aplicación se analizará desde el punto de vista de los usuarios que han utilizado el sistema y las encuestas respondidas por éstos que se podrán leer en el Apéndice A. En este primer experimento, existían cuatro alumnos potenciales y dos profesores que podían utilizar la aplicación al disponer de un *smartphone* con sistema operativo *Android*. En esta fase de pruebas, tres de los siete usuarios potenciales se decidieron a probar *GeoURJC*, es decir, un 43 % del público potencial.

Los resultados técnicos fueron muy positivos. El mecanismo de realización de *check-in* funcionaba correctamente y la interfaz *web* básica desarrollada almacenaba y mostraba toda la información mandada por los clientes sin ningún problema. Existieron algunos problemas técnicos en la aplicación cliente de *GeoURJC*, algunas excepciones no manejadas que provocaban una finalización repentina de la aplicación. Estos errores fueron informados al desarrollador para subsanarlos en futuras versiones a través de correo electrónico y de la *Android Market*. Además, se comprobó el problema de la detección de la localización de los usuarios, ya que no es exacta y no se detecta correctamente la posición del usuario.

Los resultados de la parte social del sistema se puede decir sin miedo que fueron un fracaso. Ninguno de los usuarios ajenos al desarrollo de la aplicación escribió ningún mensaje para ninguna de las clases de la asignatura. Habría que trabajar en algún mecanismo que permitiera llamar más la atención de este punto a los usuarios para el siguiente experimento. Además, en esta parte hubo algunos comentarios en los que los usuarios que llevaron a cabo las pruebas hacían hincapié en que se les olvidaba realizar *check-in* y que se podría desarrollar un notificador que, cuando hubiera una clase, te avisara periódicamente. De esta manera ningún usuario tendría la excusa de que se le ha olvidado porque el propio dispositivo se encargaría de avisarle (siempre

que estuviera conectado a Internet, evidentemente).

5.2.2 Conclusiones del Primer Experimento

Como conclusiones por parte del desarrollador tras la primera fase de pruebas, a continuación se enumeran aquellas más importantes:

- **El sistema de manera básica funciona.** Se comprueba que el sistema desarrollado ha funcionado en diferentes *smartphones* obteniéndose datos interesantes para continuar con el desarrollo del sistema.
- **La geolocalización del dispositivo no es exacta.** Como ya se comprobó en la parte de desarrollo, no se puede detectar la posición del usuario que realiza el *check-in* únicamente con la geolocalización de su posición. Es necesario el desarrollo de algún mecanismo (no muy costoso) que permita una mejor localización y sirva, a su vez, como mecanismo de seguridad para usuarios malintencionados.
- **La parte social de la aplicación ha sido un fracaso.** Explicado en los resultados del primer experimento, ningún usuario ha utilizado el mecanismo de comentarios de las clases implementado en el servidor. Es necesario desarrollar alguna utilidad que permita levantar la curiosidad de los usuarios para utilizarlo. Por ejemplo, basándonos en *FourSquare*, generar unos *rankings* de alumnos que más asisten a clase y más comentarios que otorguen reconocimiento a los usuarios.
- **Algunos usuarios se han olvidado de hacer *check-in*.** Se ha comprobado gracias a la fase de pruebas, que es necesario implementar un notificador de clases del usuario para que no se le olvide realizar los *check-in*. Es un problema importante para la viabilidad del sistema el hecho de que los usuarios no guarden su asistencia por un olvido.

5.3 Segundo Experimento

En este segundo experimento el desarrollo del sistema había sido finalizado. La funcionalidad era más compleja, puesto que además de realizar la geo-

localización, se ponía en funcionamiento varias herramientas desarrolladas a partir de los resultados del primer experimento.

En la aplicación para *Android* ya estaban contemplados los códigos QR como mecanismo de seguridad. Se había implementado el notificador de clases del usuario para evitar los olvidos de guardar su asistencia a las clases totalmente personalizable por el usuario en los ajustes de la aplicación. Se desarrolló un diseño personalizado para que la navegación por la interfaz fuera lo más cómoda posible. Se optimizó el proceso automatizado de almacenamiento de la posición del usuario para evitar duplicidades en la aplicación.

En la aplicación para *Django* se incorporan muchas mejoras técnicas: desarrollo de calendarios personales personalizados para cada usuario y calendarios para las asignaturas, se incorporan multitud de *plug-ins* desarrollados con la librería *jQuery* para hacer la navegación por la interfaz mucho más dinámica, se desarrollan clasificaciones personalizadas para cada asignatura en la que los alumnos verán qué estudiantes son los que más comentan y van a clase, se implementa el proceso automatizado de generación de códigos QR para la seguridad del sistema, se implementa la plantilla definitiva para el diseño de la interfaz, etc.

Los datos del experimento son los siguientes:

- **Objetivos iniciales del experimento:**
  - Comprobar el funcionamiento básico de la aplicación en otros dispositivos como las *tablets* (sabíamos que al menos, un usuario potencial tenía una).
  - Comprobar, de nuevo, la usabilidad de la aplicación por personas ajenas al desarrollo.
  - Poner en funcionamiento el notificador de clases.
- **Fecha de inicio:** 14 de octubre de 2011.
- **Fecha de finalización:** 04 de noviembre de 2011.
- **Asignatura en la que se ha realizado la segunda fase de pruebas:** Desarrolladores y sus motivaciones.

- **Titulación de la asignatura:** Máster oficial en *Software* Libre.
- **Lugar donde se han realizado las pruebas:** Seminario 225 del Departamental I del *Campus de Fuenlabrada* de la *Universidad Rey Juan Carlos*.
- **Pruebas diseñadas:** Se han planteado cinco sesiones de prueba para que aquellos usuarios que lo deseen puedan utilizar la aplicación.

5.3.1 Resultados

En esta segunda fase de pruebas, el éxito o fracaso del sistema se analizará de nuevo desde el punto de vista de los usuarios que han utilizado el sistema y las encuestas respondidas por éstos que se podrán leer en el Apéndice A. En este segundo experimento, existían unos diez alumnos potenciales ya que los profesores de la asignatura no disponían de un *smartphone* con sistema operativo *Android*. En esta fase de pruebas, tres de los diez usuarios potenciales se decidió a probar *GeoURJC*, es decir, un 30 % del público potencial. Sin embargo, el tutor de este Proyecto Fin de Carrera, Gregorio Robles Martínez, se prestó voluntario a actuar como profesor a la hora de realizar los *check-ins*. Por tanto, fueron cuatro los usuarios que realizaron pruebas en este segundo experimento.

Los resultados técnicos volvieron a ser muy positivos. Uno de los principales objetivos en esta segunda fase de pruebas era que se consiguiera utilizar el cliente desde una *tablet* con sistema operativo *Android*. Se consiguió que la usuaria probara la aplicación y se comprobó que no funcionaba. Después de unos arreglos y lanzar una nueva versión a través de la *Android Market*, la aplicación funcionó correctamente para una versión de *Android* superior a la 3.0. Sin esta prueba, no se habrían corregido algunos aspectos necesarios para el funcionamiento en versiones futuras de *Android*. Además, se volvió a certificar que el mecanismo de realización de *check-ins* funcionaba correctamente y las nuevas interfaces de los clientes y del servidor mostraban toda la información sin problemas. Además, el notificador funcionó correctamente

durante esta fase de pruebas ya que, aunque la primera sesión fue la introducción a los alumnos y nadie hizo *check-in*, en las tres restantes los alumnos que utilizaron la aplicación hicieron *check-in*, por lo que gracias al notificador no hubo ningún olvido.

Los resultados de la parte social del sistema se puede decir que, de nuevo, fueron un fracaso. El hecho de incluir las clasificaciones para los alumnos no cambió la forma de ver el mecanismo social de comentarios de la aplicación. Ninguno de los usuarios ajenos al desarrollo de la aplicación escribió ningún mensaje para ninguna de las clases de la asignatura.

5.3.2 Conclusiones del Segundo Experimento

Como conclusiones por parte del desarrollador tras la primera fase de pruebas, a continuación se enumeran aquellas más importantes:

- **El sistema final funciona.** Se comprueba que el sistema desarrollado y finalizado, técnicamente, funciona correctamente.
- **La aplicación para *Android* funciona en otros dispositivos como las *tablets*.** Gracias a los experimentos de esta segunda fase de pruebas, se corrigieron ciertos problemas que daba la aplicación en versiones posteriores a la 3.0. Con ello, se garantiza el uso en un mayor número de dispositivos, y no sólo en *smartphones*, sino también en *tablets* que incorporen este sistema operativo.
- **La parte social de nuevo vuelve a ser un fracaso.** Explicado en los resultados del segundo experimento, ningún usuario ha utilizado de nuevo el mecanismo de comentarios de las clases implementado en el servidor. Quizás sería uno de los apartados que se pueden considerar prescindibles al no alcanzar el éxito esperado.
- **Los usuarios ya no se olvidan de hacer *check-in*s.** Sin duda, la herramienta más positiva en esta segunda fase de pruebas ha sido la incorporación del notificador de clases a los usuarios. Con él, se ha comprobado que ningún usuario se ha olvidado de realizar su *check-in*, problema que sí se dio en la primera fase de pruebas.

Capítulo 6

Conclusiones y Futuras Líneas de Trabajo

6.1 Conclusiones

Una vez terminado el proyecto, es importante mirar hacia atrás y analizar, desde el primer día que se decidió realizar este Proyecto Fin de Carrera, si se han cumplido los objetivos marcados y cuáles son las impresiones del autor tras casi un año de trabajo. El objetivo principal que se marcó al inicio era el desarrollo e implementación de un sistema automático de control de asistencia y geolocalización de eventos académicos. Este objetivo se dividió en pequeños subobjetivos o hitos que debían irse superando para lograr el objetivo principal.

Después de un año de trabajo, de horas y horas delante del ordenador, se ha conseguido desarrollar un sistema bastante automatizado que permite a cualquier persona que no disponga de conocimientos de desarrollo de aplicaciones, utilizarla para mantener el control de asistencia de los alumnos de cualquier organización académica. Por lo tanto, en base al trabajo desempeñado en este proyecto, se pueden señalar como principales conclusiones las que se indican a continuación:

- Es cierto que en la actualidad, hay muchas aplicaciones que realizan funciones parecidas a la que se realiza en este Proyecto Fin de Carrera,

pero ninguna ha sido orientada al ámbito académico. En general, el objetivo de estas aplicaciones siempre ha ido más por un lado consumista y social (por ejemplo, comentarios y puntuaciones sobre hoteles, restaurantes, discotecas, etc.), con ánimo de dar publicidad a ciertos establecimientos, por ejemplo. No se tiene constancia de ningún proyecto desarrollado para *Android* y *Django* en el que la principal función sea la de facilitar la tarea administrativa de asistencia a eventos académicos. Y es que, en pleno siglo XXI, la era de la comunicación y conectividad global, es un pequeño paso atrás que no existan sistemas similares al desarrollado en las organizaciones académicas.

- Como se indicó al principio de esta memoria, una de las motivaciones principales era adquirir el máximo de conocimientos posibles, y por eso se decidió utilizar herramientas con las que el autor nunca había trabajado antes. Después de un año de trabajo, son muchos los conocimientos que se han adquirido, no sólo de tecnologías de desarrollo como pueden ser *Java* o *Python* (lenguajes de programación de aplicaciones para *Android* o *Django*), sino de arquitecturas de sistemas de comunicaciones, acudir al análisis y lectura de *RFC*s y tener cierta soltura a la hora de diseñar modelos de datos para bases de datos relacionales.

- El proyecto aquí presentado cuenta con ciertas similitudes con los proyectos que se desarrollan actualmente en el mundo empresarial. La principal razón de estas similitudes es que se han cubierto todas las fases por las que debe atravesar un proyecto empresarial, desde marcar los objetivos iniciales, realizar el diseño e implementación, y hasta su posterior despliegue y pruebas en escenarios reales con usuarios que no tienen que ver con el desarrollo, permitiendo un mayor grado de perfeccionamiento y resolución de problemas del sistema. Otra de las razones está versada en la combinación y uso dentro de un mismo proyecto de múltiples tecnologías que a día de hoy son de las más utilizadas en sus ámbitos.

- Como en cualquier desarrollo de una aplicación que parte desde cero, la

finalización de una primera versión consistente no es más que un punto de partida para que en un futuro se pueda continuar trabajando en esta línea implementando multitud de herramientas mucho más complejas basadas en los desarrollos ya implementados en esta primera versión. Por ello, en el Apartado 6.2 se darán algunas pautas que el autor de este Proyecto Fin de Carrera cree necesarios para trabajar e incidir con el objetivo de mejorar este sistema.

- Como última conclusión, reflejar el grado de satisfacción del autor de este Proyecto Fin de Carrera con el trabajo realizado. Son muchísimas horas las que se han dedicado al desarrollo del sistema y resulta tremendamente gratificante comprobar que se ha desarrollado un sistema, primero, que hasta ahora no existía y, segundo, que puede servir como punto de partida para que, en un futuro no muy lejano, las organizaciones académicas se planteen incorporar un sistema similar al desarrollado en este proyecto.

6.2 Futuras Líneas de Trabajo

Como se indicaba en las conclusiones de esta memoria, este Proyecto Fin de Carrera no es más que una herramienta que debe servir como punto de partida para que se convierta en un sistema empleado por cualquier tipo de organización en un futuro no muy lejano. Existen varias líneas de trabajo en las que seguir profundizando para perfeccionar el sistema. A continuación, se indican algunas de ellas a modo informativo para dar algunas pautas a aquellos lectores que estén interesados.

- **Mejora del protocolo de comunicaciones cliente-servidor.**

  Aunque el protocolo de comunicaciones entre clientes y servidor se ha elegido de tal modo que la comunicación entre ellos fuera la mínima posible, una de las líneas de trabajo en la que se puede incidir es la de un protocolo más complejo que permita a los clientes recoger más información que unos estados concretos. Existen multitud de alternativas

como *XML*, *JSON*, etc., que ofrecerían al protocolo de comunicaciones nuevas posibilidades y permitirían a los dispositivos móviles tener un mayor peso dentro del sistema, por ejemplo, pudiendo acceder a los contenidos de su cuenta de usuario directamente desde el cliente *Android*, apuntarse a las asignaturas de inmediato desde una pestaña de la interfaz o poder editar su perfil añadiendo su imagen personal o validar sus direcciones de email sin necesitar del servidor.

- **Mejora de los mecanismos de seguridad de la aplicación.**

  Aunque el servidor cuenta con herramientas para limitar todo lo posible la conectividad de usuarios malintencionados que puedan falsear la información, se deberían incorporar nuevos mecanismos de seguridad, por ejemplo, incorporar la transferencia segura de datos del protocolo *HTTP*, *HTTPS*.

  Otra línea de trabajo que puede ayudar a aumentar la seguridad es implementar mecanismos de encriptación de las peticiones de modo que sólo tengan la clave privada que los desencripte los clientes y el servidor. De este modo, aunque se capture el tráfico por un usuario malicioso, sería muy costoso conseguir la información.

  Por último, habría que trabajar en otro problema que puede suceder. Si un usuario sabe que un compañero no va a asistir a clase y éste le da su usuario y contraseña, el usuario que sí acude a clase hace *check-in* por los dos. Otra línea de trabajo puede ir encaminada en solucionar este problema, accediendo a la *dirección MAC* de la tarjeta de red del dispositivo y permitiendo sólo 1 *check-in* por clase y dirección *MAC* (aunque esto haría que si a un usuario se le olvida su dispositivo, no pueda hacer *check-in* con el dispositivo de ningún compañero) u otra alternativa que solucione este contratiempo.

- **Desarrollo de técnicas de geolocalización en interiores compatibles con sistemas operativos móviles.**

  El mayor problema sin dudas a la hora de poner en funcionamiento un proyecto como este es la dificultad de conseguir una geolocalización

del usuario lo más certera posible que permita atestiguar que efectivamente, el usuario se encuentra donde se está impartiendo el evento académico. Este problema se ha solventado en esta primera versión con el uso de unos códigos QR necesarios para poder hacer el *check-in* y que son únicos para cada clase creada en *GeoURJC*. Además, se comprueba en cada *check-in* que la localización del usuario esté muy próxima a la localización del edificio donde se imparte la clase y en el caso de que no sea así, avisa a los administradores de la irregularidad. Sin embargo, este mecanismo de seguridad no es 100 % efectivo si un usuario malintencionado consigue dicho código y se lo pasa a otro usuario (por ejemplo, mediante una foto) y éste se encuentra en el mismo edificio. Por ello, una línea de trabajo muy importante debe ser la del desarrollo de sistemas de geolocalización en edificios (por ejemplo, poder localizar al usuario a través de motas), y combinar tanto el código QR como la posibilidad de obtener la posición exacta en el edificio.

- **Desarrollo de mecanismos que ayuden a mejorar los aspectos sociales del sistema.**

  Sin duda otra línea en la que se puede trabajar para mejorar la aplicación es la interactuación de los usuarios, que se involucren con la aplicación. Para ello, una buena herramienta sería la creación de un sistema de mensajería privado entre ellos, por ejemplo, que permitiera un contacto rápido desde sus dispositivos móviles. Además, se podría seguir indagando en la posibilidad de las clasificaciones de los usuarios, otorgando medallas a cada uno de ellos como lo realiza con mucho éxito aplicaciones para *Android* como *Foursquare*.

- **Creación de un sistema de estadísticas para los profesores.**

  Esta línea de trabajo se intentó desarrollar en el presente Proyecto Fin de Carrera pero debido a los problemas de seguridad y localización explicados anteriormente se decidió dejar como línea de trabajo para futuras versiones de *GeoURJC*. Y es que, sería muy importante otorgar al profesorado herramientas de análisis de la asistencia a sus clases:

gráficos de asistencia, porcentaje de alumnos que ayudan a sus compañeros, etc.

Apéndice A

Resultado de los experimentos

En este apéndice se presenta las respuestas obtenidas del cuestionario que se pasó a los voluntarios después del segundo experimento realizado para probar el sistema que se explica en el presente Proyecto Fin de Carrera.

1. ¿Qué puntuación le das al diseño del sitio web de GeoURJC?

- 1 (Muy mala): 0
- 2: 0
- 3: 1 (25.00 %)
- 4: 2 (50.00 %)
- 5 (Muy buena): 1 (25.00 %)

2. ¿Qué puntuación le das al diseño de la aplicación para Android de GeoURJC?

- 1 (Muy mala): 0
- 2: 0
- 3: 0
- 4: 3 (75.00 %)
- 5 (Muy buena): 1 (25.00 %)

3. ¿Has necesitado utilizar el manual de usuario de la web o el subido al Moodle de la asignatura?

- Sí: 2 (50.00 %)
- No: 2 (50.00 %)

4. ¿Has utilizado el sitio web de GeoURJC para comprobar tu información y la de la asignatura?

- Sí: 3 (75.00 %)
- No: 1 (25.00 %)

5. En el caso de que hayas respondido No en la anterior pregunta, ¿cuál es el motivo por el que no accediste?

-
-
- Me bastaba con la confirmación de que se había hecho el apunte correctamente.
-

6. ¿Te parece útil la aplicación GeoURJC?

- Sí: 4 (100.00 %)
- No: 0

7. ¿Podrías explicar por qué te parece (o no) útil la aplicación GeoURJC?

- Porque permite a los profesores tener un control de los alumnos que van a sus clases y a los propios alumnos les permite llevar un control de a qué clases han faltado.
-
- Es útil, ya que es una alternativa a la firma en papel, que tiene sus problemas y molestias. Tiene posibilidades de integración con otra información útil para el alumno.
- Me parece muy util, ya que proporciona una forma muy comoda para el profesor de controlar la asistencia de los alumnos y para los alumnos puede ser algo beneficioso, ya que a través de los comentarios los profesores pueden facilitarles un resumen de lo dado en esa clase, exterder la información que por falta de tiempo no se haya podido ver, puede colgar ejercicios resueltos y/o simulacros de ejercicios de examen, aclaraciones, ect... cualquier cosa que pueda facilitar el buen entendimiento de la asignatura.

8. ¿Qué has echado en falta en el sitio web de GeoURJC?

- Quizás algo más de interactividad por parte de los usuarios. Un sistema de mensajería privada, recordatorios de las clases que te salten si estás navegando y tienes una nueva clase.
- Que esté en inglés (a veces hay alumnos extranjeros). Un apartado "Privacidad" donde se explique dónde se almacenan los datos, política LOPD, seguridad... Si es software libre, enlace al código fuente o a la forja donde esté alojado. Un apartado "Acerca de..." donde te presentes e indiques que es un proyecto fin de carrera, cuáles son los objetivos...
- Para mi gusto debería estar pensada para su acceso desde el móvi desde la propia aplicación Android.
- Lo que he echado en falta es la posibilidad de que si se te olvida la contraseña te la recuerden o te den la posibilidad de cambiarla via email facilitando el usuario y la cuenta de correo.

9. ¿Qué te parece prescindible del sitio web de GeoURJC?

- La verdad es que nada, es muy útil sobretodo el calendario personal que permite ver a cada usuario sus propias clases y asignaturas y mantener su propio control.
- No sé.
- Me centraría en la usabilidad desde un smartphone.
- Nada.

10. ¿Qué has echado en falta en la aplicación para Android de GeoURJC?

- Una interfaz gráfica que permita acceder a los mismos contenidos que el sitio web. Tener que acceder a mi información privada a través de la web y no a través de la aplicación es un punto negativo.
- Que se pueda escoger idioma.
- Un botón para acceder a la información web.
- Nada.

11. ¿Qué te parece prescindible de la aplicación para Android de GeoURJC?

- La verdad es que nada. Es una aplicación sencilla pero muy útil. Destaco positivamente el notificador de clases que ayuda a que no se nos olvide guardar nuestra asistencia.
- No sé.
- Nada.
- Nada.

12. ¿Qué añadirías a la aplicación para que fuera más útil para los alumnos?

- Un sistema de mensajería entre usuarios.
- Poder enviar un mensaje junto con la ubicación actual, si no puedo hacer checkin a la hora, para explicar que llego tarde o algo así.
- Se puede añadir un sistema de sugerencias/notificaciones de uso.
- Quizas fomentar la ayuda entre alumnos utilizando el rankin de comentarios, que el alumno que más dudas resuelva de manera correcta se le premie con puntos extras en la asignatura.

13. ¿Algún comentario más sobre GeoURJC?

- Buena idea y buen proyecto. Felicidades.
- Nada, enhorabuena por el trabajo.
- En la versión 1.0.4 el programa, una vez se accede con usuario y contraseña, hay que matarlo para salir. Con la anterior versión se salía sin problemas.
- En mi opinión creo que se pueden mejorar ciertas cosas como la exactitud con la que se muestra en el mapa la posición desde la que se hace checkin,creo que se puede mejorar el margen de error. Y otra de las cosas que veo que se podría mejorar son los aspectos referentes a la seguridad,de alguna forma cerciorarse de que quien esta haciendo checkin es quien dice ser, para las futuros usos que se le pueda dar a esta aplicación. Por otra parte me parece que tiene un diseño sencillo pero a la vez elegante e intuitivo y me parece de gran utilidad.

Apéndice B

Presupuesto del Proyecto

En este apéndice se presenta el desglose por partidas del presupuesto para la realización del presente proyecto, contemplándose tanto los costes materiales como el de los recursos humanos implicados en el mismo y necesarios para la finalización del proyecto contando con el cumplimiento de los objetivos marcados en un principio.

B.1 Costes Materiales

En el Cuadro B.1 se presenta el desglose del coste de los recursos materiales necesarios para la realización del proyecto. A continuación se enumeran todos ellos, explicados brevemente:

- Ordenador para que el ingeniero lleve a cabo sus labores de desarrollo.
- Máquina virtual en la que implementar el servidor que servirá como interfaz *web* a los usuarios y como servidor de almacenamiento de la información de los usuarios.
- *Smartphone* modelo *Samsung Galaxy Spica* empleado durante la fase de desarrollo para la depuración de errores.
- *Smartphones/tablets* con sistema operativo *Android* empleados durante los experimentos realizados con voluntarios para las pruebas del sistema. Estos dispositivos son propiedad de los voluntarios. Además,

Concepto	**Uds.**	**Coste/Ud.**	**Coste Total**
Ordenador portátil de gama media	1	800 euros	800 euros
Máquina virtual	1	200 euros	200 euros
Teléfono *Samsung Galaxy Spica*	1	225 euros	225 euros
Licencias de *software*			0 euros
Impresora Láser	1	200 euros	200 euros
Material de Oficina (tóner, papel, etc.)	1	200 euros	200 euros
Local de Trabajo	11	200 euros	2.200 euros
Coste Total Recursos Materiales			**3.825 euros**

Cuadro B.1: Coste de los recursos materiales necesarios para el proyecto

todos ellos pueden funcionar con la *Wi-Fi* de la *Universidad Rey Juan Carlos*, por lo que no hay coste de tarifa de datos.

- Coste nulo en concepto de adquisición de licencias por uso de *software*. Todo el *software* empleado en el proyecto cuenta con licencias de *software* libre: *Django*, *MySQL*, *Android*, *LATEX*, etc.

- Impresora para la impresión de la memoria del proyecto. El manual de usuario está disponible electrónicamente para los usuarios.

- Alquiler de un local de trabajo debidamente equipado y acondicionado durante un total de once meses (tiempo dedicado a la realización del Proyecto Fin de Carrera). Para el cálculo del coste por este concepto, téngase en cuenta que se consideran once unidades (once meses) a un coste de doscientos euros por unidad (200 euros/mes).

B.2 Costes de los Recursos Humanos

Según el último estudio realizado por el COIT (*Colegio Oficial de Ingenieros de Telecomunicación*)[1], el intervalo salarial en el que la mayoría de los ingenieros de telecomunicación tienen su salario comprende entre los 18.000 y los

[1]Resultados accesibles en `http://coit.es/descargar.php?idfichero=463` y publicados en abril de 2005 en la página web del COIT.

Concepto	**Coste Total**
Ingeniero	24.750 euros
Dirección y consultas a expertos	2.857 euros
Administrador de Sistemas	100 euros
Gratificación a voluntarios	0 euros
Coste Total Recursos Humanos	**27.607 euros**

Cuadro B.2: Coste de los recursos humanos involucrados para el proyecto

36.000 euros brutos anuales (remuneración propia de cuatro de cada diez encuestados). Sin embargo, este intervalo se acota aún más ya que el salario más frecuente es el comprendido entre los 24.000 y 30.000 euros brutos anuales. Así, se estimará el coste salarial del autor del proyecto, considerando que se trata de un recién titulado, en unos 20.000 euros brutos anuales. A estos 20.000 euros habrá que sumarle los costes sociales del trabajador (alrededor del 35% del salario bruto), obteniendo como importe final 27.000 euros. Si se tiene en cuenta que la realización del proyecto se ha llevado a cabo en un total de once meses, y considerándose también doce pagas anuales, el coste del ingeniero resulta un total redondeando cifras de 24.750 euros brutos.

Por otro lado, se estima que el coste relacionado a la dirección del proyecto, consultas a expertos, etc., sea de un 10% sobre la suma del coste material y el coste de personal. La suma total de ambas partidas es de 28.575 euros, por lo que el coste relacionado a la dirección del proyecto será de 2.857 euros. Adicionalmente, se deben incluir los costes asociados al administrador de sistemas necesario para la creación, configuración y administración de la máquina virtual en la que se ha desplegado el servidor del sistema desarrollado en este proyecto, considerando que dicho coste sea de alrededor de unos 100 euros.

Por último, el coste de las personas ajenas al proyecto para la participación en las fases de pruebas del proyecto aquí presentado y obtener un posterior *feedback*, es nulo dado su carácter altruista.

En el Cuadro B.2 se presenta el desglose del coste de los recursos humanos involucrados en la realización del proyecto.

Concepto	**Coste Total**
Coste Recursos Materiales	3.825 euros
Coste Recursos Humanos	27.607 euros
Coste Total	**31.432 euros**

Cuadro B.3: Coste total de los recursos necesarios para realizar el proyecto

B.3 Coste Total

Finalmente, una vez calculadas las partidas que componen el coste del proyecto, se calcula el coste total del proyecto como la suma de los costes de los recursos materiales y los costes de los recursos humanos, obteniéndose un coste total de el resultado obtenido es de un total de 31.432 euros, como se recoge en el Cuadro B.3.

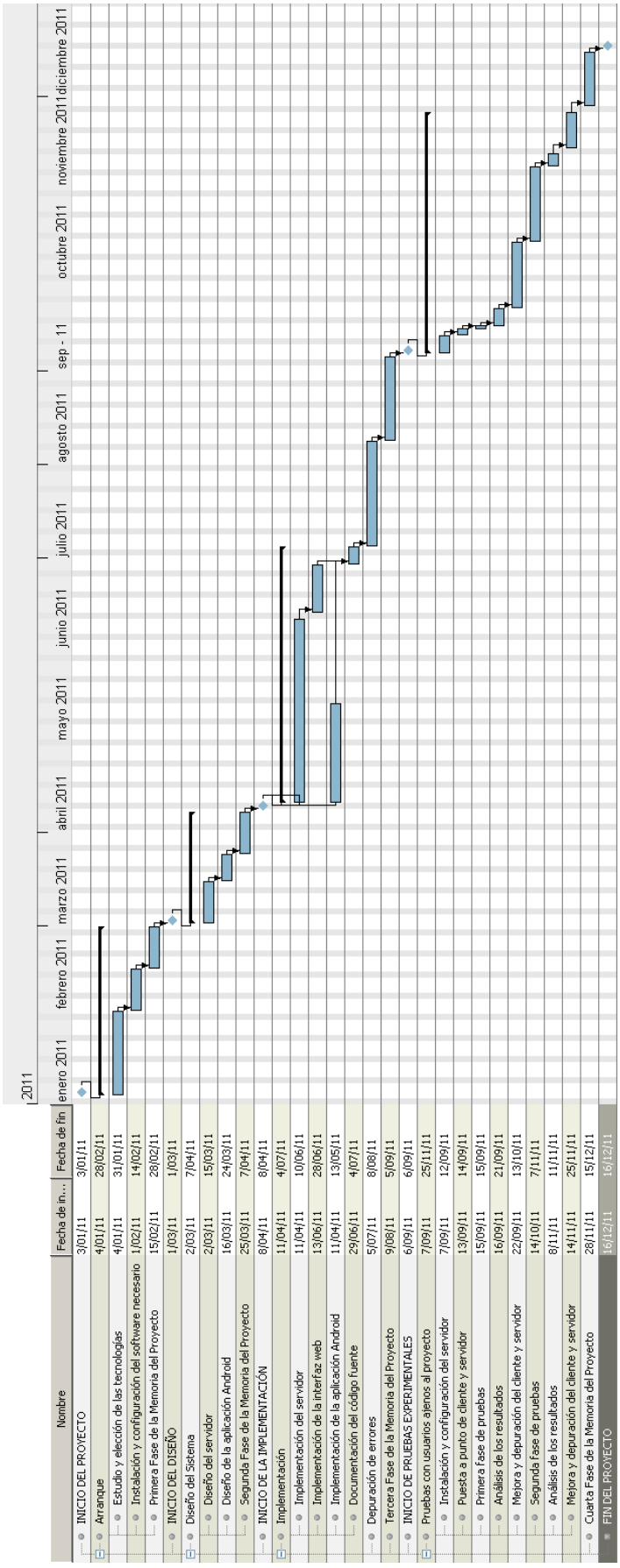
Apéndice C

Planificación del Proyecto

En este apéndice se muestra cada una de las fases que componen la planificación del proyecto llevado a cabo en esta memoria para su realización y la consecución de los objetivos marcados. Cada fase se desglosará en cada una de sus tareas y subtareas.

C.1 Diagrama de Gantt

En la Figura C.1 se presenta el Diagrama de *Gantt* en el que se reflejan cada una de las fases que se han desarrollado en el proyecto para poder conseguir superar todos los objetivos marcados al comienzo de esta memoria. Se pueden observar cada una de las tareas y subtareas que componen el proyecto, la fecha de inicio y finalización de cada una de ellas y cada uno de los hitos que componen el proyecto y que se deben ir cumpliendo para poder pasar a la siguiente fase del mismo. Este es uno de los diagramas más importantes a la hora de desarrollar cualquier proyecto en el ámbito profesional.



Nombre	Fecha de in...	Fecha de fin
INICIO DEL PROYECTO	3/01/11	3/01/11
Arranque	4/01/11	28/02/11
Estudio y elección de las tecnologías	4/01/11	31/01/11
Instalación y configuración del software necesario	1/02/11	14/02/11
Primera Fase de la Memoria del Proyecto	15/02/11	28/02/11
INICIO DEL DISEÑO	1/03/11	1/03/11
Diseño del Sistema	2/03/11	7/04/11
Diseño del servidor	2/03/11	15/03/11
Diseño de la aplicación Android	16/03/11	24/03/11
Segunda Fase de la Memoria del Proyecto	25/03/11	7/04/11
INICIO DE LA IMPLEMENTACIÓN	8/04/11	8/04/11
Implementación	11/04/11	4/07/11
Implementación del servidor	11/04/11	10/06/11
Implementación de la interfaz web	13/06/11	28/06/11
Implementación de la aplicación Android	11/04/11	13/05/11
Documentación del código fuente	29/06/11	4/07/11
Depuración de errores	5/07/11	8/08/11
Tercera Fase de la Memoria del Proyecto	9/08/11	5/09/11
INICIO DE PRUEBAS EXPERIMENTALES	6/09/11	6/09/11
Pruebas con usuarios ajenos al proyecto	7/09/11	25/11/11
Instalación y configuración del servidor	7/09/11	12/09/11
Puesta a punto de cliente y servidor	13/09/11	14/09/11
Primera Fase de pruebas	15/09/11	15/09/11
Análisis de los resultados	16/09/11	21/09/11
Mejora y depuración del cliente y servidor	22/09/11	13/10/11
Segunda fase de pruebas	14/10/11	7/11/11
Análisis de los resultados	8/11/11	11/11/11
Mejora y depuración del cliente y servidor	14/11/11	25/11/11
Cuarta Fase de la Memoria del Proyecto	28/11/11	15/12/11
FIN DEL PROYECTO	16/12/11	16/12/11

Figura C.1: Diagrama de Gantt

Como se puede observar en el diagrama, las dos fases que más tiempo han consumido han sido la fase de implementación y la fase de experimentación y pruebas. En la primera fase es trivial que se emplee muchísimo tiempo porque es la fase más lenta y en la que hay que dedicar mucho esfuerzo para plasmar el diseño de cada uno de los elementos en código interpretable. Sin embargo, la duración de la fase de pruebas puede a priori resultar sorprendente, pero cabe indicar que es también muy importante debido a que gracias a los análisis de los resultados obtenidos tanto en el aspecto técnico como a través de las opiniones de los voluntarios que han procedido a la prueba del sistema, es necesario llevar a cabo una subtarea de mejora y depuración de los elementos que componen tanto el cliente como el servidor, así como añadir nuevos aspectos técnicos a la aplicación que permitan aumentar su valor añadido de cara a una futura implementación.

Resaltar igualmente la importancia de la redacción al cierre de cada gran etapa del documento técnico a modo de memoria del proyecto, recogiendo los principales aspectos de la fase y evitándose así la pérdida de información valiosa durante su transcurso.

Bibliografía

[Adrian Holovaty(2009)] Jacob Kaplan-Moss Adrian Holovaty. *The Django Book*. GNU Free Document License, second edition, 2009. URL `http://www.djangobook.com/en/2.0/`.

[Dana Moore(2008)] William Wright Dana Moore, Raymond Budd. *Professional Python frameworks: Web 2.0 programming with Django and TurboGears*. Wiley Publishing, Inc., second edition, 2008.

[DiMarzio(2008)] Jerome F. DiMarzio. *Android: a programmer's guide*. McGraw Hill, 2008.

[González(2010)] Jorge Fernández González. Diseño, implementación y despliegue de un servicio de telecomunicación param-learning en móviles android: Gymkhanas de nueva generación. Master's thesis, ETSIT, Universidad Rey Juan Carlos, 2010.

[Hiroko Kato(2010)] Douglas Chai Hiroko Kato, Keng T. Tan. *Barcodes for Mobile Devices*. Cambridge University Press, first edition, 2010.

[Meier(2008)] R. Meier. *Professional Android Application Development*. Wrox, 2008.

[Márquez-García(1996)] Francisco M. Márquez-García. *UNIX. Programación Avanzada*. Editorial Ra-Ma, 1996.

[R. Fielding(1999)] J. Gettys J. Mogul H. Frystyk L. Masinter P. Leach T. Berners-Lee R. Fielding, UC Irvine. Rfc2616 - hypertext transfer protocol: Http/1.1. Technical report, 1999.

[Reenskaug(1979)] Trygve Reenskaug. Thing-model-view-editor: an example from a planningsystem. Technical report, Xerox PARC, 1979.

[Sonia Jaramillo Valbuena(2008)] Dumar Antonio Villa Zapata Sonia Jaramillo Valbuena, Sergio Augusto Cardona Torres. *Programación Avanzada en Java*. Ediciones Elizcom, 2008.

[Vlist(2007)] Eric Van Der Vlist. *Programación Web 2.0*. Anaya Multimedia, 2007.

Apéndice D

Glosario

AJAX Acrónimo de *Asynchronous JavaScript and XML*. No configura una tecnología en sí misma, sino que se trata de la unión de varias (*XHTML* y *CSS*; *DOM*; *XML*, *XSLT* y *JSON* ; *XMLHttpRequest* ; *JavaScript*). Esto permite contar con una nueva técnica de desarrollo *web* para crear aplicaciones interactivas, semejantes a las propias aplicaciones de escritorio. Estas nuevas aplicaciones se ejecutan en el cliente *web* del usuario (el navegador), al mismo tiempo que se mantiene una comunicación asíncrona con el servidor en segundo plano (normalmente intercambio de datos vía *XML* o *JSON*, nunca páginas *HTML* completas) sin que el usuario lo perciba. De este modo, se consiguen realizar cambios (actualizaciones) sobre la página que está visitando el usuario en ese momento, sin necesidad de recargarla nuevamente y por completo, lo que supone un aumento de la interactividad, de la velocidad y de los usos de la aplicación.

Android *Android* es una plataforma software que incluye un sistema operativo destinado a dispositivos móviles y cuyo kernel está basado en los sistemas *Linux*. Inicialmente fue desarrollado por Google tras adquirir en julio de 2005 una pequeña empresa de California llamada *Android, Inc.*, dedicada al desarrollo de software para teléfonos móviles.

Android Market *Android Market* es una tienda de *software* en línea desarrollada por *Google* para los dispositivos *Android*. La propia *Android Market* no es más que una *app* (aplicación) llamada *Market* que está preinstalada en la mayoría de los dispositivos *Android* y que permite a los usuarios buscar y descargar aplicaciones publicadas por desarrolladores terceros.

Apache El servidor *HTTP Apache* es un servidor *web HTTP* de código abierto, para plataformas *Unix*, *Microsoft Windows* o *Macintosh* entre otras, que implementa el protocolo *HTTP/1.12* y la noción de sitio virtual. Apache es usado principalmente para enviar páginas *webs* estáticas y dinámicas en la *WWW* (*World Wide Web*). Muchas aplicaciones *web* están diseñadas asumiendo como ambiente de implantación a *Apache*, o al menos con que utilizarán características propias de este servidor *web*.

API Un Interfaz para Programación de Aplicaciones (*Application Programming Interface*, *API*) es un conjunto de clases que el desarrollador de software podrá emplear a modo de librerías o módulos, sin tener que invertir tiempo en implementar los métodos, funciones o procedimientos que se utilizan habitualmente en programación y que ya son ofrecidos por el fabricante.

Aplicación Cliente-Servidor Arquitectura de comunicaciones en la que conuyen un conjunto de aplicaciones basadas en dos categorías que llevan a cabo funciones distintas (un cliente que solicita servicios y un servidor que los ofrece), pero que al mismo tiempo pueden desarrollar actividades tanto de manera conjunta como de forma independiente.

Aplicación web En la ingeniería de *software* se denomina aplicación *web* a aquellas aplicaciones que los usuarios pueden utilizar accediendo a un servidor *web* a través de Internet o de una intranet mediante un navegador. En otras palabras, es una aplicación *software* que

se codifica en un lenguaje soportado por los navegadores *web* en la que se confía la ejecución al navegador.

Base de Datos Partiendo de que los datos son hechos conocidos que pueden registrarse y que cuentan con un signicado implícito (susceptible de ser modelado), una base de datos será una colección coherente de datos con signicado inherente. Es decir, se trata de un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior procesado y uso.

Check-in En aplicaciones de carácter social, procedimiento por el cuál una aplicación basada en geolocalización registra la posición de un cliente, almacenando la hora a la que ha sido realizada y otra información relevante para el administrador del sistema y para otros usuarios que puedan acceder a su contenido.

Clase En Programación Orientada a Objetos, se corresponde con el modelo o abstracción de un tipo determinado de objetos, que presentan determinadas características o propiedades comunes a todos ellos, y cuya diferencia de valores hará que cada objeto sea una instancia única de la clase.

Cliente Se trata de un proceso ejecutándose en una determinada máquina cuya función será solicitar un determinado servicio ofrecido por un servidor. Para ello, deberá establecer un canal de comunicación con dicho servidor para lo cual necesitará conocer su ubicación (nombre de máquina o dirección *IP*, habitualmente), proceder a la realización de la petición de servicio manteniéndose a la espera del resultado de dicho servicio (esperar una respuesta), para nalizar cerrando el canal de comunicación y ofreciendo al usuario el resultado fruto del servicio solicitado, terminándose posteriormente su ejecución.

Código QR Un código QR (*Quick Response Barcode*) es un sistema de código abierto utilizado para almacenar cualquier tipo de informa-

ción en una matriz de puntos o un código de barras bidimensional creado por la compañía japonesa *Denso-Wave* en 1994.

Cookie El tráfico *web* se realiza sobre el protocolo de comunicaciones *HTTP*. Dicho protocolo no mantiene por sí mismo información de estado sobre la comunicación del cliente con el servidor. Para poder conservar la información entre una página visitada y la siguiente (ejemplos de login de usuario, productos almacenados en el carro en un sistema de compra on-line), es necesario almacenarla. Los mecanismos que tradicionalmente se han empleado para conservar esa información de estado han sido la reescritura de *URLs*, el uso de campos ocultos en formularios y las propias *cookies* (mecanismo recogido en la RFC 2965). Así, se puede entender una cookie como un fragmento de información que el servidor envía al cliente para que éste lo almacene en su disco duro, y vuelva a enviárselo al servidor en una petición *HTTP* posterior.

CSS Acrónimo de *Cascading Style Sheets*. En *HTML* existen elementos y atributos que permiten especificar el estilo de los documentos (tipos y tamaños de fuente, colores de texto, colores de fondo, ancho de los elementos, etc.). Pero el uso de estas características de estilo de *HTML* presenta una serie de problemas como por ejemplo la dicultad para homogeneizar las características estéticas al estar incluida la información de estilo dentro del propio documento. Los ficheros denominados hojas de estilo *CSS* son por tanto un mecanismo para solventar este tipo de problemas, al configurar un lenguaje para definir la presentación de un documento estructurado asociado y escrito en *HTML* o *XML* de manera externa a los documentos que lo incorporen.

DalvikVM Máquina virtual desarrollada para *Android*, similar a las máquinas virtuales de *Java* (*JVM, Java Virtual Machine*). Incluye las librerías fundacionales de *Java* y por lo tanto, las funcionalidades básicas del lenguaje.

Django *Framework* de desarrollo de aplicaciones *web*. Escrito en lenguaje *Python* y de código abierto, cumple en cierta medida con el patrón de diseño MVC (Modelo-Vista-Controlador). En un principio, fue concebido y desarrollado para la gestión de varias páginas orientadas a noticias de la World Company de Lawrence, Kansas, y fue liberado al público bajo licencia BSD en julio de 2005.

Eclipse Entorno de Desarrollo Integrado (*Integrated Development Environment, IDE*) de aplicaciones muy potente diseñado originalmente para *Java*, de código abierto y multiplataforma para el desarrollo de proyectos *software*, desarrollado por la Fundación Eclipse. Es una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

Framework Se trata de un conglomerado de herramientas que da soporte a la organización y desarrollo de otro proyecto, incluyendo para ello compiladores, enlazadores, *API*s y bibliotecas, bases de datos y servidores (poco recomendables cuando el proyecto pasa a modo de producción), etc., que faciliten el desarrollo, unión, prueba, ejecución, etc., de los diferentes componentes del proyecto.

GUI Acrónimo de *Graphical User Interface*. La interfaz gráfica de usuario (*GUI*) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

GPS El Sistema de Posicionamiento Global (*Global Positioning System, GPS*) es un sistema global de navegación por satélite que permite determinar la posición de un ente (objeto, persona, vehículo, nave) en el planeta con una precisión bastante buena, siendo el error habitual de unos pocos metros.

Hardware El *hardware* corresponde a todas las partes tangibles de un sistema informático. Sus componentes son: eléctricos, electrónicos, electromecánicos y mecánicos. Sus cables, gabinetes o cajas, periféricos de todo tipo y cualquier otro elemento físico involucrado. Por el contrario, el soporte lógico es intangible y es el llamado *software*.

HTML Acrónimo de *HyperText Markup Language*. Es el lenguaje de marcado predominante para la elaboración de páginas *web*. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. *HTML* se escribe en forma de «etiquetas», rodeadas por corchetes angulares (<,>). *HTML* también puede describir, hasta un cierto punto, la apariencia de un documento, y puede incluir un *script* (por ejemplo *JavaScript*), el cual puede afectar el comportamiento de navegadores *web* y otros procesadores de *HTML*.

HTTP Acrónimo de *HiperText Transfer Protocol*. Definido en la *RFC 2616 (HTTP/1.1)*, *HTTP* es un protocolo transaccional de comunicaciones a nivel de aplicación (puerto 80 por defecto, sobre el protocolo de nivel de transporte *TCP*) de petición-respuesta que se utiliza para construir sistemas distribuidos y colaborativos de información hipermedia. Se trata de un protocolo de comunicaciones sin estado, utilizado desde 1990 por *WWW* (*World Wide Web*). Su funcionamiento básico será la conexión de un cliente a un servidor al que envía su petición y a la cual el servidor envía una respuesta. El formato de los mensajes de petición y respuesta queda especificado por el propio protocolo *HTTP*, pudiendo aparecer entre cliente y servidor diversos intermediadores tales como un proxy (modica petición y respuesta), una pasarela (traduce entre distintos protocolos) o un túnel (reenvía la petición para atravesar un *firewall*).

IDE Un Entorno de Desarrollo Integrado (*Integrated Development En-*

vironment, IDE) es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (*GUI*). Los *IDE*s pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

Java Lenguaje de Programación Orientada a Objetos (POO) desarrollado por *Sun Microsystems, Inc.* a principios de los años 90, teniendo entre sus principales objetivos la portabilidad entre plataformas independientemente de la arquitectura *hardware* y/o *software*, cosa que se consigue gracias a la compilación de las aplicaciones en bytecodes que posteriormente son interpretados por una Máquina Virtual de Java (*Java Virtual Machine, JVM*), siendo también posible su compilación a código nativo para la ejecución. Cuenta con un modelo de objetos sencillo y elimina herramientas de bajo nivel que suelen inducir a errores en el desarrollador, otorgándole abstracción con aspectos tales como la manipulación de punteros y el acceso a memoria.

JavaScript Lenguaje de *scripting* basado en objetos. Se utiliza principalmente integrado en un navegador *web* (aunque existe también *JavaScript* para aplicaciones servidor) permitiendo el desarrollo de interfaces de usuario mejoradas y páginas *web* dinámicas. El código *JavaScript* es enviado por el servidor incrustado en una página *HTML* solicitada por el cliente, o bien en un fichero aparte como resultado de una petición *web*. El cliente ejecutará dicho código, consiguiendo efectos llamativos y de gran interés para el usuario cliente sin necesidad de transmitir ningún dato por la red.

JSON Acrónimo de *JavaScript Object Notation*. Recogido en la *RFC 4627*, es un lenguaje de marcado que ofrece un formato ligero para el intercambio de datos a través de una red de comunicaciones, representando estructuras de datos. *JSON* es un subconjunto de la notación literal de objetos de *JavaScript* que no requiere el uso

de *XML*. Su simplicidad ha llevado a la generalización de su uso como alternativa a *XML* en *AJAX* (para el cual fue especícamente pensado), especialmente debido a la sencillez de su parseo o análisis semántico por medio de *JavaScript*.

JVM Una Máquina Virtual de Java (*Java Virtual Machine, JVM*) es un programa nativo (es decir, ejecutable en una plataforma especíca) capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (los *bytecodes* de *Java*), generado por el compilador del lenguaje *Java*. La JVM es una de las piezas fundamentales de la plataforma Java. Básicamente se sitúa en un nivel superior al *hardware* del sistema sobre el que se pretende ejecutar la aplicación, y éste actúa como un puente que entiende tanto el *bytecode*, como el sistema sobre el que se pretende ejecutar. Así, cuando se escribe una aplicación *Java*, se hace pensando que será ejecutada en una máquina virtual *Java* en concreto, siendo ésta la que en última instancia convierte de código *bytecode* a código nativo del dispositivo final.

Middleware Capa de *software* (normalmente distribuida) que se suele situar entre la capa de aplicaciones y las capas inferiores (sistema operativo, e incluso, red). El *middleware* proporciona un modelo de programación (un *API*) conveniente para los programadores de aplicaciones distribuidas (facilitando su programación y manejo de aplicaciones), abstrayendo o enmascarando la complejidad y heterogeneidad de plataformas (sistemas operativos, lenguajes de programación, redes de comunicación subyacentes, etc.).

MVC El patrón de diseño MVC (Modelo-Vista-Controlador) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos. El patrón MVC se encuentra frecuentemente en aplicaciones *web*, donde la vista es la página *HTML* y el código que provee de datos dinámicos a la página.

MySQL Sistema de Gestión de Bases de Datos (*SGBD*) relacionales orientadas a objetos. Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario distribuido en código abierto, pudiendo ser utilizado por cualquier persona y modificado adaptándolo a las necesidades del desarrollador. MySQL es muy utilizado en aplicaciones *web*, en plataformas (*Linux*/*Windows*, *Apache*) y en herramientas de seguimiento de errores.

Objeto En Programación Orientada a Objetos, instancia única de una clase que mantiene la estructura y comportamiento denidos por dicha clase. Se diferenciará de sus homónimos a partir de los distintos valores que tomen sus atributos o propiedades.

Plug-in Pequeña aplicación que a modo de complemento de otra aplicación de mayor entidad, le aporta una nueva funcionalidad y por lo general, muy específica. Esta aplicación será ejecutada por la aplicación principal, interactuando ambas por medio de una *API*. Son famosos los *plug-in* para navegadores *web* o para los *IDE* como *Eclipse*, ofreciendo soportes a nuevos lenguajes de programación, sistemas, etc., no soportados en su versión de serie.

PostgreSQL Sistema de Gestión de Bases de Datos (*SGBD*) relacionales orientadas a objetos, distribuido bajo una licencia de *software* libre *BSD*. Con un potente motor de bases de datos, ofrece prestaciones y funcionalidades equivalentes a muchos *SGBD* comerciales.

Programación Orientada a Objetos Paradigma de programación que define los programas en términos de clases de objetos, siendo los objetos entidades que combinan estado (los datos), comportamiento (métodos, procedimientos o funciones) e identidad (propiedad del objeto que lo distingue del resto de objetos de su misma clase). La Programación Orientada a Objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para

llevar a cabo una tarea, con lo que se consigue programar de manera más modular y fácil, siendo el resultado final más sencillo de mantener y reutilizar.

Python Lenguaje de programación interpretado distribuido bajo una licencia de código abierto (*Python Software Foundation License*) compatible con la licencia *GPL*, *Python*, ofrece ventajas muy parecidas a las de los lenguajes propios de *scripting*, como puede ser un rápido desarrollo, además de buscar una mayor facilidad tanto de lectura del propio código, como en el diseño.

RFID Acrónimo de *Radio Frequency IDentification*. Sistema de almacenamiento y recuperación de datos remoto que usa dispositivos denominados etiquetas, tarjetas, transpondedores o *tags RFID*. El propósito fundamental de la tecnología *RFID* es transmitir la identidad de un objeto (similar a un número de serie único) mediante ondas de radio. Las etiquetas *RFID* son unos dispositivos pequeños, similares a una pegatina, que pueden ser adheridas o incorporadas a un dispositivo (generalmente, tarjetas). Contienen antenas para permitirles recibir y responder a peticiones por radiofrecuencia desde un emisor-receptor *RFID*.

SDK Un *Software Development Kit* (*SDK*) es un conjunto de herramientas destinadas a hacer más sencillo al programador el desarrollo de aplicaciones. Entre las herramientas que normalmente se incluyen destaca el compilador, el enlazador, el depurador y el conjunto de librerías y su documentación, acerca del lenguaje de programación.

Servidor Un servidor es un proceso ejecutándose en una determinada máquina de la red. Su cometido será gestionar diversos servicios así como el acceso a los recursos capaces de ofrecer dichos servicios. Para hacer posible esta gestión, el servidor estará en continua espera de una petición de servicio proveniente de un cliente. En el

momento en que esa petición se produzca, el servidor comenzará a realizar las acciones necesarias para atender dicha petición, volviendo al estado de espera una vez que se haya terminado de ofrecer el servicio solicitado.

SGBD o Sistema de Gestión de Bases de Datos Tipo de software muy especíco que facilita la definición, construcción y manipulación de bases de datos, sirviendo a modo de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

Sistema Operativo *Software* de sistema que controla de manera eficiente todos los recursos de una computadora y que establece la base sobre la que puede escribirse el *software* de aplicación. Actúa por tanto de intermediario entre el usuario de un computador y su hardware, siendo sus principales objetivos el control y ejecución de programas, el almacenamiento de datos de usuarios y la facilitación del uso del sistema, todo ello mediante una gestión y administración eficiente del *hardware* (recursos) disponible.

Smartphone El teléfono inteligente (*smartphone*) es un término comercial para denominar a un teléfono móvil que ofrece más funciones que un teléfono móvil tradicional que se empleaba únicamente para llamar y mandar mensajes de texto. Los teléfonos inteligentes son dispositivos que soportan completamente un cliente de correo electrónico con la funcionalidad completa de un organizador personal. Además, permiten la instalación de programas para incrementar el procesamiento de datos y la conectividad.

Software Se conoce como *software* al equipamiento lógico o soporte lógico de un sistema informático. Comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados *hardware*. Los componentes lógicos incluyen, entre muchos otros, las aplicaciones informáticas o el sistema operativo.

SQLite *SQLite* es un sistema de gestión de bases de datos relacional. *SQLite* es un proyecto de dominio público creado por D. Richard Hipp. A diferencia de los sistema de gestión de bases de datos cliente-servidor, el motor de *SQLite* no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca *SQLite* se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de *SQLite* a través de llamadas simples a subrutinas y funciones, reduciendo la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. Además, el conjunto de la base de datos (definiciones, tablas, índices y los propios datos), son guardados como un sólo fichero estándar.

Tablet Una *tablet* (en español tableta) es un tipo de computadora portátil con la que se puede interactuar a través de una pantalla táctil o multitáctil; el usuario puede manejarla mediante sus los dedos, sin necesidad de teclado físico ni ratón. La *tablet* funciona como un ordenador personal, solo que más orientado a la multimedia, lectura de contenidos, entretenimiento y a la navegación *web* que a usos profesionales.

UML Lenguaje Unicado de Modelado (*Unified Modeling Language, UML*). Es el lenguaje de modelado de sistemas software más conocido y utilizado en la actualidad. Permite de manera gráfica visualizar, especificar, construir y documentar un sistema *software*.

URI Acrónimo de *Uniform Resource Identifier*. Es una cadena de caracteres corta que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc.). Normalmente estos recursos son accesibles en una red o sistema. Un *URI* se diferencia de un *URL* en que permite incluir en la dirección una subdirección, es decir, un *URL* no es más que una parte de la *URI* de un recurso.

URL Acrónimo de *Uniform Resource Locator*. Es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su acceso, localización o identificación, como por ejemplo documentos textuales, imágenes, vídeos, presentaciones, presentaciones digitales, etc. El *URL* de un recurso de información es su dirección en Internet, la cual permite que el navegador la encuentre y la muestre de forma adecuada. Por ello el *URL* combina el nombre del ordenador que proporciona la información, el directorio donde se encuentra, el nombre del archivo, y el protocolo a usar para recuperar los datos.

Web 2.0 El término *Web 2.0* (también llamada *web* social) fue empleado por primera vez en el año 2004 por Tim O'Reilly para referirse a una nueva etapa en la historia del desarrollo de tecnología *web* basada en comunidades de usuarios y una gama especial de servicios, como las redes sociales, los *blogs* o los *wikis*, que fomentan la colaboración y el intercambio ágil y eficaz de información entre los usuarios de una comunidad o red social. Desde un punto de vista meramente técnico, la *Web 2.0* supone la evolución del *web* de colección de sitios a plataforma informática completa. Proporciona aplicaciones *web* a los usuarios finales, muchas de las cuales podrán sustituir a otras aplicaciones de escritorio. Además, explota los llamados efectos de red, por ejemplo con las redes sociales (dada la arquitectura de participación).

Webkit *WebKit* es una plataforma para aplicaciones que funciona como base para ciertos navegadores web, entre los que destacan *Safari*, *Google Chrome*, *Epiphany*, *Maxthon* o *Midori* entre otros. La *API* de *WebKit* posibilita interactuar con un servidor web para recuperar y renderizar páginas web, descargar archivos, y administrar *plug-ins*. *Webkit* incluye dos *frameworks* de más bajo nivel: *WebCore*, un analizador sintáctico y motor de renderizado *HTML* y *JavaScriptCore*, un intérprete de *JavaScript*.

XML Acrónimo de *eXtensible Markup Language*. Es un metalenguaje extensible de marcado o de etiquetas, desarrollado por el *W3C* (*World Wide Web Consortium*). Se trata realmente de un dialecto simplicado y adaptado del *SGML* (*Standard Generalized Markup Language*) y permite definir la gramática de lenguajes específicos, pretendiendo ser razonablemente simple. Por tanto, *XML* no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades, siendo el ejemplo más utilizado en la actualidad *RSS*.