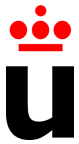

Universidad
Rey Juan Carlos

Escuela Técnica Superior de Ingeniería de Telecomunicación

Ingeniería de Telecomunicación - LADE

**PROYECTO FIN DE CARRERA**

# Diseño, implementación y despliegue de un servicio web basado en hiperficciones explorativas con soporte Android.

Autor : Guillermo Méndez de Vigo Texidor

Tutor : Gregorio Robles Martínez

Curso Académico 2012/2013

II

*Dedicado a*

*mi familia*

*Caminante no hay camino,*
*se hace camino al andar.*

*Antonio Machado*

# Agradecimientos

Como dijo Johann Wolfang Goethe, *todo comienzo tiene su encanto*. Hace unos cuantos años dejé el colegio, para empezar una nueva etapa: la universidad. Con estas páginas acabo esta etapa. Pero todo final es un comienzo de algo nuevo. *La vida es como una caja de bombones, nunca sabes lo que te va a tocar*, decía la sabia madre de *Forrest Gump*.

Nunca creí que fuera hacer una aplicación web como PFC. Pero estas líneas que escribo tienen un culpable, Gregorio Robles Martínez, mi tutor. Él hizo que esto de la programación hasta me acabase gustando un poquito. Además, él ha sido el propulsor de esta idea. Por tanto, quiero mostrar mi más profundo agradecimiento hacia él por varias razones: ayudarme con el proyecto, por la sabiduría transferida y por hacerme *un friki* (como dice él).

En estos últimos meses, muchas personas han ayudado. Quiero agradecer también a Jorge su inestimable ayuda. Así como todos aquellos que probaron la aplicación y con sus opiniones ayudaron a mejorarla, en especial a Vicky.

Tampoco me puedo olvidar de mis compañeros durante estos años, en especial de Cristina y Javier, por su ayuda y cooperación, sobretodo en estos últimos meses con el proyecto. Y a todos los profesores (buenos y malos) porque algo de ellos ha quedado en mí.

Por supuesto, agradecer a mi familia la paciencia durante estos años, que no ha debido de ser fácil, en especial a mi madre, ya que sin ella no creo que fuera quien soy hoy. También a mi hermana Marta, por su cantidad de horas de tutoría impagadas. Y a Juan, que con sus críticas constructivas siempre intenta hacer un mejor hombre de mí.

Por último, agradecerte a ti, al lector, que pierdas un poco de tu tiempo en compartir el esfuerzo que este proyecto ha conllevado, aunque creo que es difícil demostrarlo en unas cuantas páginas.

# Resumen

Los avances tecnológicos han permitido cada vez más posibilidades de aprendizaje y ocio a nuestra sociedad. Actualmente los *smartphones* son de común uso, y se pueden utilizar para algo más que chatear con el *WhatsApp*.

Es por ello que nos propusimos hacer una aplicación web, con soporte *Android*, que implementara lo que se conoce como *hiperficción explorativa*. Este nombre tan raro esconde un tipo de narrativa, como la de aquellos libros de *Elige tu propia aventura*, donde un autor escribe, pero el lector, a través de sus decisiones cambia el argumento de la historia.

En este Proyecto Fin de Carrera se ha procedido a diseñar, implementar y probar la hiperficción explorativa. Para ello, se ha implementado un servidor, una interfaz web y una interfaz *Android*, basándose en una arquitectura *Cliente-Servidor*.

A través de la interfaz web un usuario podrá crear actividades y controlarlas (hace el papel de escritor). Otro usuario será el que, a través de sus decisiones, da forma a esta actividad (hará el papel de lector). Para todo ello se utilizarán interacciones HTTP, en formato HTML o JSON (para dispositivos *Android*).

Para destacar brevemente algunas características, el usuario que crea la actividad, podrá ver en forma de árbol las relaciones de las pruebas o preguntas (usando *Canvas*), y controlar la actividad de los jugadores de forma remota. El jugador, podrá participar a través de la web o a través de un dispositivo móvil *Android*. En éste último caso, se utilizará GPS.

Por tanto, el PFC consiste en coger un elemento narrativo 'clásico' e implementarlo en forma de servicio *web*, aumentando su funcionalidad, ya que ofrece movilidad.

Esta aplicación ha sido probada en varias ocasiones, tanto de forma privada como pública, y el *feedback* de los usuarios es lo que ha hecho que la aplicación sea lo que es hoy.

# Índice general

# Índice de figuras

# Capítulo 1

# Introducción

*El hombre está dispuesto siempre*
*a negar lo que no comprende.*
*Blaise Pascal (1623-1662 AC)*

## 1.1. Motivación

Entre los años 1979 y 1998, se publicaron bajo el sello de *Bantam Books* unas serie de *libros juego* que en España conocimos como *Elige tu propia aventura*, pertenecientes al genero de la hiperficción explorativa. Actualmente se han reeditado en varios países algunos de estos títulos.

Los libros tenían este lema: *Las posibilidades son múltiples; algunas elecciones son sencillas, otras sensatas, unas temerarias... y algunas peligrosas. Eres tú quien debe tomar las decisiones. Puedes leer este libro muchas veces y obtener resultados diferentes. Recuerda que tú decides la aventura, que tú eres la aventura. Si tomas una decisión imprudente, vuelve al principio y empieza de nuevo. No hay opciones acertadas o erróneas, sino muchas elecciones posibles.*

Bajo este lema también hemos desarrollado esta aplicación, adaptándolo a las nuevas tecnologías, lo que permite realizar cosas que los autores de los libros no habrían imaginado, como que el lector no sólo lea, sino que de verdad viva la historia.

De esta forma, los potenciales usos de una aplicación como esta son infinitos, pudiendo ser usada en el ámbito académico o en una fiesta, e incluso para pasar el rato.

Además, creemos que este tipo de aplicaciones puede permitir acercar la tecnología a niños, ya que pueden aprender jugando.

Por ello, nos lanzamos a implementar estos *libros juego*, y para ello hay que darle un nombre: *Explohyperfiction*.

## 1.2. Objetivos

El principal objetivo de este proyecto es el desarrollo de una aplicación que permita la creación de actividades en forma de árbol, donde cada usuario con sus decisiones podrá tomar distintos caminos.

Para conseguir este objetivo final, necesitamos primero cumplir los siguientes objetivos generales:

1. Disponer de una base de datos para el almacenamiento estructurado de toda la información que se manejará en el sistema.
2. Disponer de una aplicación web para la creación de las actividades y su posterior monitorización.
3. Disponer de una aplicación web para participar en las actividades
4. Disponer de una aplicación *Android* para participar en las actividad.
5. Integrar la aplicación dentro de *LibreGeoSocial*.
6. Comprobar el correcto funcionamiento del sistema en un escenario real y atender los comentarios de los usuarios

Además, dentro de cada uno de estos objetivos se han definido otros objetivos específicos para las diferentes interfaces, que comentamos a continuación:

- Objetivos específicos en la aplicación web para la creación de actividades:
    1. Sencilla e intuitiva.
    2. Creación de códigos QR para las preguntas que se solicite.
    3. Mostrar una figura con las relaciones entre las preguntas y respuestas.

- Objetivos específicos en la aplicación *web* para participar en actividades:

  1. Mostrar los resultados una vez finalizada una actividad.

- Objetivos específicos en la aplicación *Android*:

  1. Interpretar códigos QR.
  2. Geolocalización.

Además, y considerando su posible uso dentro del ámbito académico, se marca como objetivo el poder dar soporte a grupos de participantes. También, como objetivos relacionados y necesarios:

- Aprender a programar en *Android*
- Aprender a escribir en LaTeX

En el apartado de conclusiones, analizaremos el cumplimiento de los objetivos.

## 1.3. Estimación del trabajo realizado

En este apartado vamos a comentar como se ha realizado el proyecto de manera breve, para que el lector se haga una idea del trabajo realizado.

Al decidir comenzar con este proyecto y con los objetivos marcados, era el momento de empezar a trabajar. Comenzamos a trabajar en el mes de Febrero. Debido a clases, exámenes y otros, el trabajo ha sido discontinuo hasta el mes de agosto. A partir de entonces se ha dedicado una gran cantidad de horas a este trabajo. Pero vamos a resumirlo a continuación.

Entre febrero y marzo se realiza una primera aplicación web, que permitía la creación de actividades y su participación, que fue la base para todo el desarrollo posterior. Durante estos meses, se estima el tiempo dedicado en unas 100 horas. Esta aplicación no incluía soporte a códigos QR ni la figura de relaciones.

En abril y mayo se mejoró dicha aplicación, añadiendo el soporte a grupos. El tiempo dedicado a esta mejora fue de aproximadamente 50 horas. Además, durante este periodo

de tiempo, se aprende a programar en *Android*. El tiempo estimado al aprendizaje es de unas 50 horas.

Entre junio y julio se integran los códigos QR a la aplicación así como la figura de las relaciones entre las preguntas (usando Canvas). También se desarrolla una primera aplicación *Android* que permite la comunicación con el servidor, pero que todavía no tenía el servicio de GPS. El tiempo dedicado a esto es de unas 100 horas.

A finales de Agosto se decide el nombre de la aplicación: *Explohyperfiction*. Además nos ponemos a trabajar con la integración en LibreGeoSocial, lo cual nos acarreó muchos problemas, debido a diferentes incompatibilidades de versiones. A mediados de Septiembre se solucionan los problemas y a finales se tiene una aplicación lista para la prueba, con la geolocalización y monitorización incluida en web y *Android*. Durante este tiempo se dedican unas 250 horas.

Por último, se propone la realización de dos pruebas (1 y 15 de octubre), mientras que se arreglan pequeños detalles y se escribe esta memoria. El tiempo estimado utilizado es de unas 75 horas.

Por tanto, el tiempo estimado a la realización de este proyecto ha sido de unas 625 horas. Se ha intentado ser sincero con este cálculo para mostrar al lector el esfuerzo que ha llevado este trabajo.

## 1.4. Antecedentes

El proyecto que hoy presentamos se podría considerar una evolución de otro Proyecto Fin de Carrera presentado por Jorge Férnandez: *Diseño, implementación y despliegue de un servicio de telecomunicación para M-Learning en Móviles Android: Gymkhanas de nueva generación*[6]. Este trabajo ha estado muy presente en todo el desarrollo de todo el servicio.

La principal diferencia entre aquel proyecto y el que hoy presentamos es la estructura de los eventos. En las *gymkhana* teníamos una estructura circular, donde de una prueba pasabas a otra, y así todo el rato. En este proyecto se modifica esto, y pasamos a una estructura de árbol, y donde se pueden dar incluso situaciones de bucles infinitos, impensables en la *gymkhana*.

Pero si queremos encontrar algo parecido en el mercado, lo más similar que podemos encontrar es el juego *¿Dónde se esconde Carmen San Diego*, donde cada respuesta que damos modifica el hilo del juego. Sin embargo, creemos que este proyecto se parece mucho más a los libros juego de *Elige tu propia aventura*. Además, ni uno ni otro ofrecen movilidad, cosa que nosotros sí.

## 1.5. Estructura de la memoria

Con la intención de facilitar la lectura, creemos imprescindible explicar brevemente cada uno de los apartados de la memoria.

El primer capítulo es la *Introducción*, donde presentamos la motivación de este proyecto y los objetivos marcados. También hemos considerado oportuno mostrar un pequeño informe sobre el trabajo realizado. Por último, se comentan trabajos relacionados.

El segundo capítulo es el *Estado de la ciencia*[1]. En este capítulo comentamos brevemente las bases tecnológicas de este proyecto. También intentamos justificar la elección de estas tecnologías y no otras.

El tercer capítulo es el de *Diseño e Implementación*. Este capítulo es el núcleo de la memoria. Aquí presentamos y explicamos todo el desarrollo de la aplicación con la intención de que el lector pueda comprender cada paso dado, intentando explicar también cada problema surgido.

En el cuarto capítulo se presentan los *Resultados* que se analizarán en el quinto capítulo *Conclusiones*, donde además se presentan una autoevaluación de los objetivos y futuras líneas de trabajo en base a este proyecto.

Además, se presenta una bibliografía con las principales referencias utilizadas, para que el lector pueda acceder a ellas si lo considerase necesario.

En el cd que se adjunta, se puede ver el código, así como una copia digital de está memoria.

[1]Es de uso común utilizar Estado del Arte, pero según la RAE su uso es incorrecto, ya que es un calco del inglés

# Capítulo 2

# Estado de la Ciencia

*No se puede desatar un nudo*
*sin saber cómo está hecho.*

*Aristóteles (384 AC-322 AC)*

En este capítulo se presentan las bases tecnológicas del proyecto. Se trata de un breve resumen de las características básicas, por lo que en caso de que el lector necesitara más información, se indicará al final de cada apartado dónde hallar más información. Asimismo, se justificará la elección de cada una de las tecnologías presentadas.

## 2.1. Aplicaciones Cliente-Servidor

La aplicación que presentamos sigue una arquitectura cliente servidor, por lo se debe conocer este concepto. Es este apartado lo explicamos brevemente.

### 2.1.1. Conceptos

Una aplicación Cliente-Servidor es un tipo de programación distribuida. La programación distribuida, de uso muy extendido, pretende aprovechar la potencia y los recursos de los ordenadores, que son interconectados para llevar una tarea de forma cooperativa. Por lo tanto, una aplicación distribuida está formada por varios programas que se ejecutan en ordenadores diferentes y que se comunican por medio de una red.

En este tipo de programación, cada programa, por sí sólo, no hace nada. Es necesaria la colaboración para obtener resultados, por lo que se hace imprescindible la existencia

de protocolos de comunicación. En la mayoría de los casos, estos protocolos siguen un estándar, lo que permite además la escalabilidad de las aplicaciones distribuidas.

Existen muchos tipos de aplicaciones distribuidas, pero sin duda, el modelo más exitoso es de Cliente-Servidor. En este modelo, la aplicación se divide en dos partes, con roles claramente diferenciados:

- Servidor
- Cliente

En este tipo de aplicaciones, el cliente es el que inicia la comunicación a través de peticiones, ya que el servidor espera pasivamente por ellas. Al recibir una petición, el servidor la procesa y envía una respuesta al cliente. El ejemplo más clásico de un modelo Cliente-Servidor es Internet, donde a través del navegador (que es el cliente), solicitamos diferentes recursos a un servidor Web, recibiendo como respuesta un mensaje, que será la página HTML solicitada.

La principal ventaja de este tipo de aplicaciones es su escalabilidad, ya que cliente y servidor son elementos independientes y podemos añadir más elementos sin que tenga que afectar al sistema (aunque hay que vigilar la capacidad). Además, la existencia del servidor como elemento de procesamiento, permite un mayor control de acceso, así como asegurar la integridad de los datos.

Las desventajas son también considerables, en especial en lo que se refiere a la robustez del sistema. Si el servidor estuviera caído o recibiera demasiadas peticiones, algunas de ellas podrían no ser satisfechas. Esto se puede solucionar parcialmente con hardware y software específico y/o duplicando el servidor.

### 2.1.2. ¿Por qué una aplicación Cliente-Servidor?

Se ha elegido una aplicación Cliente-Servidor, principalmente porque nos ha permitido separar el desarrollo de la aplicación. Por un lado se ha realizado el servidor, y por otro, el cliente. En nuestro caso, y como veremos posteriormente, el cliente puede ser un navegador Web o un dispositivo Android, lo que nos muestra la escalabilidad de este tipo de arquitectura.

Además, pensando en un posible uso académico, este tipo de arquitectura nos permite tener todos los datos sensibles de la aplicación en el servidor, y nunca en manos del cliente. Un cliente nunca podrá acceder a ciertos recursos sin ciertos permisos, lo que nos permite, por ejemplo, que un jugador de la aplicación no tenga acceso a todas las preguntas, si no simplemente a las que le correspondan durante el juego.

### 2.1.3. Más información

La información que podemos encontrar acerca de sistemas distribuidos y aplicaciones Cliente-Servidor es muy extensa. En caso de estar interesado en este tema, se recomienda acudir a bibliografía especializada, como el libro de Tanenbaum[11] o Kurose-Ross [8]. Una lectura más ligera y amena sería el libro la Universitat Oberta de Catalunya [9] , que se recomienda efusivamente para aquellos con pocos o nulos conocimientos en redes de computadores, ya explican detalles que en otros libros más avanzados no podemos encontrar.

## 2.2. HTML5 y CSS

Antes de pasar a ver aspectos específicos de la aplicación, consideramos necesario hablar brevemente de la evolución del desarrollo WEB. Esta evolución está muy relacionada con el modelo Cliente-Servidor que vimos en el apartado anterior, ya que al servidor cada vez se le exigen cosas más complejas, mientras que el cliente debe ser capaz de procesar las respuestas, cada vez más complejas.

Queda claro que la evolución del desarrollo WEB se debe principalmente a la demanda de los usuarios. El crecimiento de usuarios y sitios activos ha crecido de manera exponencial los últimos años [10]. En los comienzos, estos usuarios utilizaban principalmente Netscape, donde se nos presentaba un HTML básico, con más diseño que funcionalidad. En esta época los usuarios no generaban contenido, en contraposición a lo que ahora ocurre.

Por lo tanto, hemos pasado de aplicaciones WEB que sólo servían contenido, a unas aplicaciones que interactúan con el usuario, lo que se conoce como Web 2.0, Esta evolución fue paulatina, pasando del Netscape al Internet Explorer, que ya incluía frames, CSS

y plugins, así como lenguaje de Script. Los navegadores de la siguiente generación presentaron nuevas tecnologías como AJAX o FLASH, lo que permitió el generar contenido dinámico. Todos estos hechos, provocaron que el primitivo HTML utilizado en 1991, fuera desarrollándose hasta llegar a la nueva versión HTML5, así como también se desarrollaron las hojas de estilo CSS.

### 2.2.1. CSS

Comenzamos hablando de la situación actual del CSS. La idea que sustenta su existencia es el separar la estructura del documento HTML de su presentación. La primera especificación fue recomendada por la W3C [1] en 1996. Esta versión fue desarrollándose hasta llegar al actual CSS 2.1, recomendada oficialmente en 2011. Actualmente se trabaja en CSS 3, que se caracteriza por la modularización del contenido, y partes de este desarrollo ya fueron incluidos en la última recomendación oficial [12].

A pesar de que CSS3 no es todavía un estándar, la mayoría de los navegadores lo acepta. Solamente tendremos problemas con CSS (también en la versión CSS 2.1) al utilizar Internet Explorer.

### 2.2.2. HTML5

La versión 5 de HTML se encuentra actualmente en modo experimental, pero la mayoría de los navegadores ya lo aceptan. El desarrollo está regulado por la W3C. Además es la primera vez que se ha desarrollado en paralelo al XHTML [2].

Su completa integración con todos los navegadores nos llevará hasta el año 2020, pero su potencial es enorme, y su uso cada vez más extendido.

Esta nueva versión incluye nuevos elementos como audio y vídeo, que antes sólo conseguíamos a través de complementos. Pero una de las mejoras más impresionantes ha sido el elemento *canvas*, que nos permite graficar en dos y tres dimensiones. Además, se supone que la integración de este lenguaje con CSS3, hará que dominen el desarrollo de

[1] Worl Wide Web Consurtium es un consorcio internacional que produce recomendaciones. Destacamos su trabajo en las recomendaciones en HTML y CSS. Sitio web: `http://www.w3c.org`

[2] XHTML es básicamente HTML expresado como un XML válido

aplicaciones en dispositivos móviles, dejando de lado el desarrollo en el código de cada uno de ellos.

### 2.2.3. ¿Por qué HTML5 y CSS?

En el desarrollo de este proyecto hemos optado por utilizar HTML5 y CSS. La elección de las hojas de estilo queda justificada en que la separación del diseño de la estructura y del contenido facilitaba mucho el trabajo.

La elección de utilizar HTML5 se debe principalmente a la necesidad de la aplicación de generar gráficos dinámicos. Se estuvo planteando utilizar otras opciones, como FLASH o crear una imagen temporal en el servidor. Pero se considero oportuno la utilización de esta potente herramienta, que además deja el trabajo de realizar la imagen al cliente (la realiza el navegador).

Pero estas ventajas no pueden ocultar que la elección de HTML5 y CSS pueden provocar ciertas incompatibilidades en algunos navegadores. Tras sopesar las ventajas y desventajas de esta elección se consideró que los beneficios obtenidos con estas tecnologías superan los problemas. Aun así, se intentó que la mayor parte del código sensible a estas incompatibilidades sea el más reducido posible, y que el acceso sea bastante restringido (en el apartado de diseño de la aplicación explicaremos donde se utiliza HTML5, pero podemos ir adelantando que no todos los usuarios podrán acceder a estos contenidos).

### 2.2.4. Más información

La principal fuente de información sobre HTML5 y CSS la podemos encontrar en la página oficial del organismo W3C. A parte la documentación oficial existente, en caso de estar interesado en HTML5 y sus nuevas funcionalidades se recomienda el libro de Kleinfield [7], que se puede obtener de forma gratuita en O'Reilly.

## 2.3. Android

Tras conocer el modelo que utilizaremos en la aplicación (Cliente-Servidor), hemos visto en el apartado anterior con que herramientas realizamos la interfaz web. Pero además

de una interfaz web, y con el objetivo de darle una mayor usabilidad a la aplicación, se ha desarrollado una interfaz para dispositivos móviles, concretamente dispositivos *Android*. En este apartado conoceremos los conceptos (muy) básicos de *Android*.

### 2.3.1. Conceptos

*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*. Actualmente lo desarrolla la OHA (*Open Handset Alliance*). Se distribuye bajo una licencia *Apache 2.0* y *GPLv2*, por lo que se trata de software libre.

El desarrollo de aplicaciones para esta plataforma se recomienda hacerlo a través del *IDE Eclipse*, para el cual se proporciona un *plug-in* para facilitar el trabajo. Junto la *SDK (Software Development Kit)* y otras APIs, se pueden construir aplicaciones utilizando el lenguaje de programación *Java*.

El hecho de que se utilice el lenguaje de programación *Java* para desarrollar aplicaciones en *Android*, ha hecho que muchos programadores experimentados pudieran desarrollar sus aplicaciones, siendo sólo necesario ciertos conceptos propios de Android. Con el SDK, disponemos además de un emulador donde podremos probar nuestras aplicaciones.

Actualmente, los dispositivos *Android* están muy extendidos. Desde las primeras versiones del sistema, lanzadas en 2008, ha evolucionado mucho. Actualmente la mayor parte de los dispositivos llevan una versión 2.3 o superior. Una cosa importante es que la mayoría de las aplicaciones son compatibles con versiones superiores, aunque siempre es recomendable su revisión. En este momento, se está trabajando en la versión 4.2, siendo la última versión la 4.1. Este cambio de versiones del sistema operativo ha afectado también al desarrollo de la *SDK*. Actualmente se trabaja con la API 16.

El futuro, sin embargo, se habla de desarrollos de aplicaciones en *Android* utilizando HTML5, cosa que ya es posible gracias a APIs especializadas, aunque su impacto aún es pequeño, el desarrollo de este tipo de aplicaciones sería muy similar para todos los sistemas operativos.

### 2.3.2. ¿Por qué Android?

Actualmente existen cuatro grandes sistemas operativos para dispositivos móviles:

- iOS de Apple
- Android
- BlackBerry OS
- Windows Phone

Además de otros sistemas, actualmente minoritarios. Para desarrollar la interfaz móvil del cliente se ha optado por *Android*, debido principalmente a que se trata de software libre y código abierto. Esto nos ha permitido poder instalar la aplicación en los móviles sin que *Google* revisase el contenido. Esto con otros sistemas no ocurre. Además, teníamos conocimientos del lenguaje de programación *Java*, lo cuál facilitaba el desarrollo de la aplicación.

En este momento, el proyecto sólo se hace para *Android*, pero se podría llevar a otros sistemas operativos sin problemas.

### 2.3.3. Más información

La cantidad de recursos disponibles sobre *Android* es considerable. Se recomienda acudir al sitio oficial de *Android*[1], para cualquier consulta, ya que podemos encontrar documentación, tutoriales, ...

## 2.4. Python y Django

Tras comentar brevemente el modelo Cliente-Servidor que hemos utilizado, y conociendo las tecnologías HTML y CSS, así como Android, utilizadas para la interfaz web del cliente, pasamos a presentar las tecnologías que hemos utilizado en el lado del servidor.

### 2.4.1. Python

*Python* es un lenguaje de programación interpretado, que sigue una filosofía[3] que hace hincapié en una sintaxis limpia y que favorezca un código legible.

Creado por Guido van Rossum a finales de los años 80 y distribuido bajo una licencia de código abierto (*Python Software Foundation License*), *Python* es un lenguaje de programación multiparadigma, que permite desde una programación funcional, hasta una programación orientada a objetos.

Entre sus características, podemos destacar que es usa un tipado dinámico, y es multiplataforma.

Actualmente, se encuentra desarrollándose la versión 3.0, y la última versión estable es la 2.7. Una de las principales ventajas que encontramos es la gran cantidad de APIs que existen, lo que permite trabajar con muchos y diversos elementos.

### 2.4.2. Django

*Django* es un framework de desarrollo web de código abierto, escrito en *Python*. *Django* se centra en la reutilización, la conectividad y extensibilidad de los componentes que forman un sitio web, en general complejo.

*Django* sigue el paradigma Modelo Vista Controlador (MVC). El paradigma Modelo Vista Controlador es un modelos de abstracción de desarrollo de Software que separa los datos de una aplicación (Modelo), la interfaz de usuario (Vista) y la lógica de negocio (Controlador) en tres componentes distintos.

Las funcionalidades más importantes de *Django* como entorno para el desarrollo de aplicaciones web son las siguientes:

- Herramientas para la gestión de la aplicación
- *Framework* de capa de presentación
- Manipulación de la base de datos mediante el mapeo de objetos de la aplicación a entradas relacionales

[3]El *Zen de Python* establece una filosofía a la hora de programar en *Python*. Fue escrito por Tim Peters

- Seguridad

Pero lo más destacable de *Django*, hagamos uso o no de todas sus funcionalidades, es que nos permite desarrollar sitios web complejos y dinámicos, de una forma rápida y sencilla, configurando un entorno integrado y completo (incluye un servidor web, que es perfectamente válido para la fase de depuración, aunque no recomendado para producción). También incluye una base de datos SQLite3 (muy sencilla de utilizar en la fase de desarrollo), aunque se recomienda el uso de *PostgreSQL*[4].

Actualmente se encuentra en la versión 1.4.1. El único requerimiento para poder utilizar *Django* es tener instalado una versión adecuada de *Python*.

### 2.4.3. ¿Por qué Python y Django?

Existen numerosos lenguaje de programación aptos para programar la lógica de un servidor. Se ha optado por utilizar *Python* por su gran cantidad de APIs y por el conocimiento previo del mismo. También podríamos haber utilizado otros lenguajes como Java o Ruby.

Se prefirió trabajar con un framework en lugar de hacerlo directamente (aunque *Python* nos lo habría permitido), para reducir el tiempo de desarrollo. Entre los frameworks existentes para *Python* se optó por *Django*, principalmente porque se tenían conocimientos previos de él. Además, como veremos a continuación, el servidor de LibreGeoSocial está realizado en *Django*.

Lo que debemos tener claro es que ha sido una decisión tomada principalmente en base a los conocimientos previos de ambas herramientas, pero que todo el diseño se podría haber implementado de manera similar con otros lenguajes y frameworks.

### 2.4.4. Más información

Si el lector estuviera interesado en más información acerca de *Python* o *Django* se recomienda acudir a la documentación oficial ([5] y [2]). Además, existen muchos libros acerca de estos temas, pero que en principio sólo recomendamos si los conocimientos sobre programación son escasos, ya que la documentación de ambas herramientas es bastante extensa y concisa.

## 2.5. LibreGeoSocial

*LibreGeoSocial*[4] es un proyecto desarrollado dentro del grupo *LibreSoft* del Departamento *GSYC (Grupo de Sistemas y Comunicaciones)*, perteneciente a la *Escuela Técnica Superior de Ingeniería de Telecomunicación* de la *Universidad Rey Juan Carlos*. Este proyecto consiste en una red social móvil y contando con una interfaz de realidad aumentada, contando con geolocalización.

En cuanto a la estructura interna del sistema, se destacan dos componentes. La primera de ellas es un *framework* desarrollado en *Python* y *Django* que facilita la creación de redes sociales. La segunda componente es una aplicación *Android* que explota la funcionalidad de *LibreGeoSocial*

En el desarrollo de esta aplicación, se han utilizado algunos de estos componentes, como indicaremos posteriormente en el apartado de *Diseño e implementación*.

Si desea más información sobre este tema, puede dirigirse a la página del proyecto.

[4]Sitio web: `http://libregeosocial.morfeo-project.org`

# Capítulo 3

# Diseño del sistema

*Es genial trabajar con ordenadores. No discuten, lo recuerdan todo y no se beben tu cervexa.*

*Paul Leary*

En este capítulo presentamos los aspectos más relevantes del diseño e implementación del sistema realizado. Se trata del núcleo de la memoria. En todo momento se intentará que el lector comprenda cada uno de los pasos (aunque no siempre se logrará), explicando todos los problemas surgidos. Para el diseño se ha utilizado un patrón cliente-servidor.

Se comenzará con un breve análisis genérico del sistema. Continuaremos con el análisis del servidor, para finalizar analizando los diferentes clientes.

## 3.1. Introducción

Para abordar el problema presentado se procederá a la elaboración de varios componentes que operando entre sí formarán nuestro sistema. En la Figura 3.1 se puede observar un esquema general de estos componentes.

Como se puede observar en la figura, el sistema está compuesto por tres módulos fundamentales:

- Una máquina que operando a modo de servidor, pondrá a disposición de cualquier cliente un servicio web a través del cual interoperar con la base de datos que sea manipulada convenientemente por este servidor.

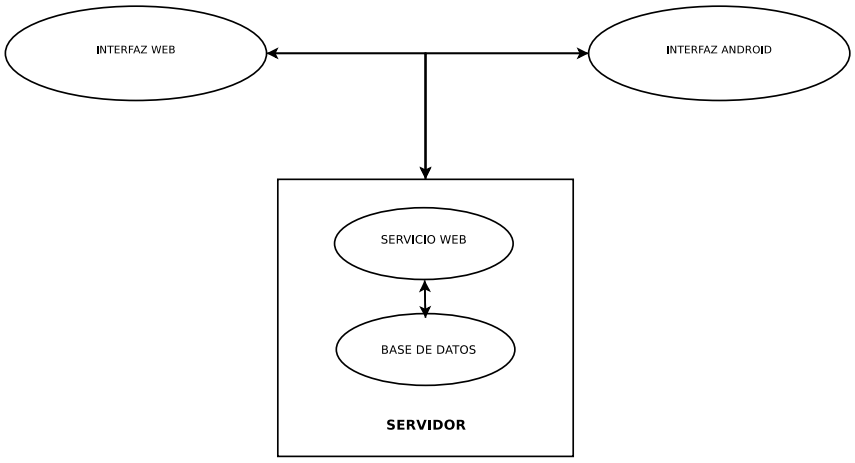


Figura 3.1: Esquema general de la aplicación web

- Una aplicación web que podrá ser utilizada por los creadores de las actividades o los jugadores. Servirá como interfaz entre los creadores/jugadores y el servidor.

- Una aplicación *Android* que se podrá instalar en los dispositivos móviles que usen esta plataforma. Sólo la podrán usar los jugadores de las actividades y será la interfaz con la que se comunican con el servidor.

Con esta breve visión general, pasaremos a estudiar cada uno de los módulos señalados. Veremos dos apartados para el cliente diferenciados: cliente web y cliente *Android*, ya que sus diferencias son considerables y aunque tengan en muchos casos funciones similares, se ha considerado oportuno el dividirlo.

## 3.2. Diseño e implementación del servidor

En el diseño del servidor, podemos diferenciar dos partes diferentes. Por un lado, necesitamos un modelo de datos y, por otro, las funciones propias del servidor. Comenzaremos explicando el modelo de datos utilizado, para pasar a estudiar el diseño del servicio web.

### 3.2.1. Modelo de datos

#### Antecedentes

A la hora de diseñar el modelo de datos, partíamos con dos modelos ya realizados que nos debían servir de referencia:

- El modelo de datos de *LibreGeoSocial*[3].
- El modelo de datos de la *Gymkhana* de *LibreGeoSocial*[6].

La utilización del primer modelo como referencia queda justificado en la intención de conseguir una integración con el proyecto de *LibreGeoSocial*. El segundo modelo se tomo como referencia debido a la multitud de similitudes existente entre un proyecto y otro.

#### Nuestro modelo

Con los modelos de referencia presentados en el apartado anterior, se procedió a diseñar nuestro propio modelo de datos, que se puede observar en la Figura 3.2. En esta figura se puede observar que hacemos uso de uno de los modelos definido en *LibreGeoSocial*, concretamente *Person*. Este modelo lo utilizamos como base a la hora de definir *Player*, que además contendrá diferentes permisos y los grupos (*Group*) a los que pertenece.

Nos centraremos un momento en *Player*. Cualquier miembro de *LibreGeoSocial* puede utilizar esta aplicación mediante su inscripción. El inscribirse le da directamente permisos de *jugador*, con lo que podrá participar en actividades y unirse a grupos (luego explicamos con más detalle los grupos). Además de ser jugador, el usuario podría crear grupos y eventos si fuera *mánager*, para lo que debe enviar un petición a un *superusuario*, que aceptará o rechazará la misma. Por lo tanto, el sistema requiere un *superusuario* para su correcto funcionamiento.

Cualquier usuario con permisos de *mánager* puede crear grupos. Estos grupos pueden ser abiertos o cerrados[1]. Cualquier usuario podrá solicitar unirse a un grupo privado a través de las *PetitionGroup*. Los *SystemMessages* son utilizados para notificar a los usuarios de las diferentes respuestas de las peticiones enviadas.

[1] A la hora de diseñar los grupos se tomo como referencia los cursos del *moodle* de la ETSIT, donde algunos son abiertos y otros requieren autorización

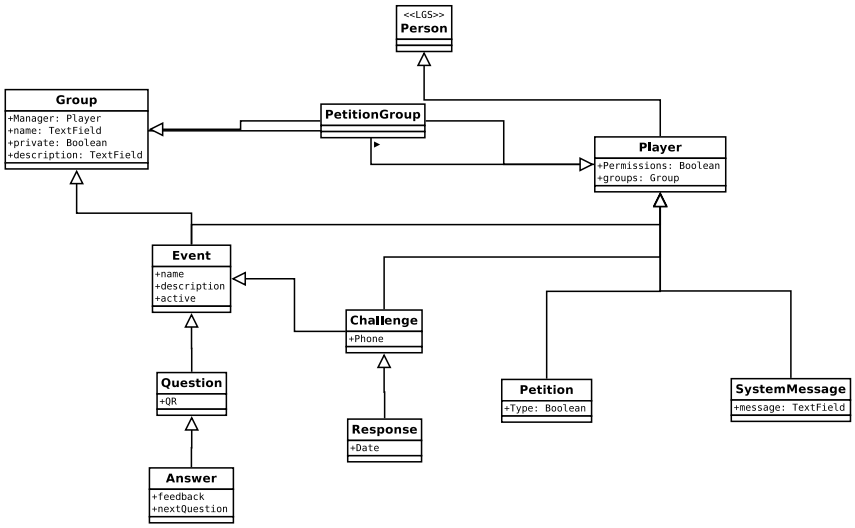


Figura 3.2: Diagrama del modelo de datos

Todos los modelos anteriores se usan para darle una mayor funcionalidad a la aplicación, y ahora nos centraremos en los específicos para poder realizar la aplicación web deseada. El modelo *Event* puede estar presente en uno o más grupos, y debe tener un *mánager*. Las preguntas (*Question*) formarán parte de estos eventos, con sus posibles respuestas (*Answer*). Cabe destacar que en las respuestas no se indica si son correctas o incorrectas, sino que llevan un *feedback* para el participante en la actividad, que verá cuando la haya finalizado.

Una vez definidos los eventos o actividades, hace falta ciertos modelos para que los jugadores puedan participar en ellos. Por un lado definimos el *Challenge* y por otro las *Responses* donde se almacenará toda lo realizado por el usuario en el transcurso de la actividad.

### Implementación

La implementación del modelo de datos se realizó usando el gestor de bases de datos *PostgreSQL* debido a que es el utilizado en *LibreGeoSocial*, además de ser el recomendado por *Django*.

*PostgreSQL* es software libre y da soporte a la geolocalización, con lo que su uso queda más que justificado.

#### Problemas

En el diseño e implementación del modelo de datos han surgido diferentes problemas, que consideramos oportuno comentarlos, con el objetivo de que el lector comprenda mejor el trabajo realizado.

En primer lugar, a la hora de diseñar el modelo de datos, y mientras se iba desarrollando la aplicación, nos fuimos dando cuenta que había que añadir nuevos elementos, lo que hizo que constantemente hubiera cambios en el modelo.

Pero los mayores problemas surgieron al utilizar *PostgreSQL*. A pesar de ser recomendado como gestor de base de datos, su utilización requiere horas de preparación. En primer lugar, se deben instalar multitud de paquetes y programas relacionados. Por otro lado, se debe configurar *Django* para su utilización. Esto consumió horas de trabajo y, en momentos, provocó desesperación, hasta el punto de llegar a plantearnos la utilización de *SQLite3*.

Finalmente los problemas surgidos fueron resueltos. Sin embargo, consideramos que el modelo de datos puede ser mejorado y lograr una mayor integración con *LibreGeoSocial*.

### 3.2.2. Servicio Web

#### Integración en *LibreGeoSocial*

En primer lugar queremos remarcar que desde el principio del proyecto se tuvo en cuenta que la aplicación podría formar parte algún día de *LibreGeoSocial*, por lo que se intento seguir las mismas pautas de diseño que en el utilizan.

La integración se ha logrado a modo de *plug-in*, de manera que en cualquier momento se puede incluir esta funcionalidad dentro de *LibreGeoSocial*, con tan sólo copiar una serie de ficheros en el directorio adecuado del árbol de directorios del proyecto. Con esto conseguimos una integración limpia y no renunciamos a la reutilización de las funcionalidades ya realizadas y que pudieran ser útiles en nuestro desarrollo. Esto lo hemos visto en el modelo de datos, por ejemplo.

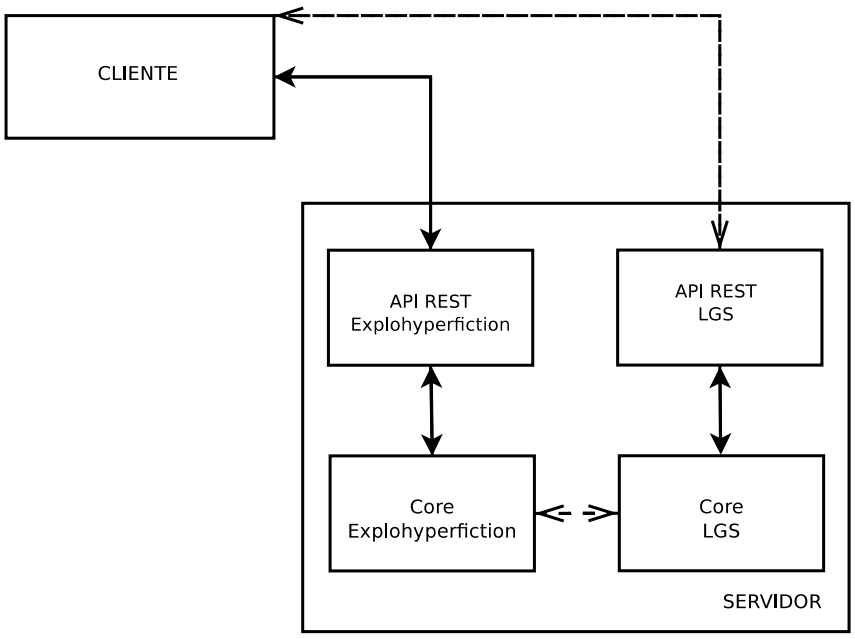


Figura 3.3: Diagrama de interacción entre cliente y servidor

### Patrón de diseño MVC

Como ya comentamos anteriormente, el *framework* de desarrollo utilizado *Django* sigue un patrón de diseño conocido como *MVC* (*Modelo-Vista-Controlador*, aunque con ciertas particularidades. Para que estas particularidades no nos afectasen, se ha seguido el modelo de desarrollo utilizado en *LibreGeoSocial*. De esta forma se ha desarrollado por una parte una *API REST* y por otra parte una librería que denominamos *core*.

En el *core* se realizarán las operaciones fundamentales de procesado, creación, modificaciones y destrucción sobre el modelo de elementos en la base de datos, mientras que el *API REST* actuará como interfaz de cara al cliente, por lo que quedará encargada de recibir y procesar las peticiones, y en caso de que estas sean correctas, utilizará el *core* para las operaciones que sean necesarias. La respuesta se proporcionarán en forma de documento HTML que será generado a través de plantillas destinadas a ello, o, en el caso de JSON, directamente a través de la librería de *Python simplejson* (ya volveremos a este punto posteriormente).

Si consideramos todo lo dicho, el esquema de funcionamiento del servidor sería el que presenta la Figura 3.3, donde observamos que el cliente sólo conecta con las *APIs* y el el dentro del servidor donde se utiliza el *core* tanto de nuestra aplicación como de

*LibreGeoSocial*.

Por lo tanto, las *API REST* interaccionara con sus respectivos *core* siempre que la petición recibida lo requiera. Lo habitual en nuestra aplicación será que las peticiones se realicen sobre el *API REST* de *Explohyperfiction*, pero algunas operaciones como registrar usuarios o el *login* se hacen utilizando elementos de *LibreGeoSocial*.

Entonces el diseño realizado en el servidor sigue el patrón *Modelo-Vista-Controlador*, de la siguiente manera:

- El modelo de datos presentado en la sección anterior seguirá siendo la representación a modo de objetos de todas las tablas de la base de datos, y por lo tanto, la base fundamental de operación para el funcionamiento del servicio y está definido en el archivo *models.py*, existente en cualquier proyecto *Django*.

- La vista estará compuesta por el *API REST* y las plantillas, actuando siempre como interfaz con el cliente, tanto para la recepción de peticiones como para la generación de las respuestas.

- El controlador se corresponderá con el *core* de la aplicación, que se encarga de la manipulación del modelo de datos de acuerdo con el procesamiento que se indicó desde la vista, y cuya respuesta servirá como base para la generación de las respuestas al cliente.

Al usar este patrón *MVC*, se consigue desagregar las diferentes funciones a desempeñar en diversos componentes independientes. Además, podemos destacar las siguientes ventajas[6]:

- Un mismo modelo podría tener múltiples vistas o controladores.

- Las vistas son automáticamente notificadas de cambios en el modelo.

- Los modelos no tienen que sufrir modificación alguna para adaptarse a nuevos tipos de vista o controladores.

- La desagregación de los tres componentes hace que el diseño y/o implementación de cada uno de ellos se pueda realizar en paralelo con el resto.

- Permite una mejor escalabilidad del sistema.

En contrapartida, también se pueden citar algunos inconvenientes[6]:

- La complejidad del sistema aumenta al separar sus distintos componentes.
- La cantidad de ficheros a desarrollar y mantener aumenta de forma considerable.

### URLs y arquitectura REST

*REST* (*R*Epresentation State Transfer) es un estilo arquitectural ampliamente extendido en servidores web estáticos, pero no tanto en servidores dinámicos. *REST* está basado en un conjunto de reglas o normas que comentamos brevemente a continuación:

1. Cada recurso debe contar con su propia *URL*, entendiéndose por recurso cualquier tipo de información que se quiera hacer visible.
2. La comunicación de debe ser sin estado, de modo que una petición del cliente debe contener toda la información necesaria para que el servidor pueda procesarla, puesto que el servidor no tiene información previa de la comunicación. Está información puede enviarse bien a través de la propia URL y/o pasando valores como parámetros de la petición.
3. Los recursos deben tener hiperenlaces a otros recursos. También es de importancia contar con uniformidad en la presentación o interfaz del estado de los recursos.
4. Existe un conjunto estándar de métodos, que pese a ser bastante reducido, permite la realización de todas y cada una de las operaciones que se necesiten realizar sobre un recurso del sistema. Estos métodos son:
    - GET: para obtener información, sin cambiar estado y siendo un operación idempotente.
    - POST: para crear un recursos dada la URL de un constructor de recursos, cambiando el estado y sin ser una operación idempotente.
    - PUT: Destinado a la actualización o creación de un recurso conocida su URL, cambiando el estado y siendo una operación idempotente.

- DELETE: Permite eliminar un recurso conocida su URL, cambiando el estado y siendo una operación idempotente.

De todos los métodos comentados, nosotros sólo haremos uso de los métodos GET y POST.

### Implementación

Como se ha comentado a lo largo de la memoria, la implementación del servicio web se ha realizado utilizando *Python* y *Django*, con una base de datos *PostgreSQL* e integrándolo dentro de *LibreGeoSocial*.

Para lograr esto, se creo un proyecto *Django* dentro del proyecto *LibreGeoSocial* donde se añadieron los archivos necesarios para el funcionamiento de este servicio web. La puesta en marcha en del servidor se realizó a través de una máquina virtual facilitada por el tutor, y se usó el servidor de prueba de *Django*.

El desarrollo del servicio se hizo en paralelo al desarrollo de los clientes, por lo que hablaremos de la funcionalidad en apartados posteriores.

### Problemas

A la hora de implementar el servicio web fue donde mayores problemas tuvimos. Aparte de los ya comentados problemas a la hora de usar *PostgreSQL*, la integración con *LibreGeoSocial* tampoco fue fácil.

El principal problema fue por compatibilidad, ya que la versión facilitada de *LibreGeoSocial* está realizada sobre *Django 1.1* y los cambios desde entonces han sido considerables, mientras que siempre estuvimos trabajando con una versión *Django 1.4*.

Estas incompatibilidades (nuevas funciones, funciones obsoletas, nuevas rutas de archivos) convirtió la integración en un autentico infierno. Se dedicaron muchas horas a identificar y resolver el problema, tomando finalmente la decisión de adaptar parcialmente *LibreGeoSocial* a nuestro intereses.

Por lo tanto, en el proyecto que presentamos no toda la funcionalidad de *LibreGeoSocial* está disponible, pero sí la que usa nuestra aplicación.

Además, debido a estos problemas, se optó por no utilizar un servidor *Apache* y utilizar

en su lugar el servidor de *Django*, ya que la configuración del servidor era compleja y, aunque se intentó, todo eran problemas.

Al menos, todo esto ha servido para recordar y aprender cosas de *shell*, así como para leer mucha documentación de programas relacionados.

### 3.2.3. Características de nuestro servidor

Ya hemos visto como se diseñó el modelo de datos y el servicio web que componen el servidor. Pero además, nuestro servidor realizará cierta funciones que merecen un apartado aparte, debido a su importancia dentro del proyecto.

Las funciones que destacaremos serán la creación de los códigos QR, y el multiformato soportado.

#### Códigos QR

Un código QR (*Quick Response code*) es un módulo para almenar información en forma de códigos bidimensionales.

La utilización de estos códigos en nuestra aplicación se restringe a la codificación de las respuestas, de tal forma que los clientes *Android* pudieran responder a sus preguntas por medio de dichos códigos.

Además, la utilización de estos códigos nos permitiría controlar la ubicación geográfica de la actividad gracias a la colocación de los mismos.

El objetivo era poder crear los códigos QR de las respuestas de manera dinámica y permitir a un usuario a través de la interfaz web acceder a ellos en forma de documento *pdf*.

La generación de los códigos y del *pdf* se hacen de forma dinámica en el servidor. Cuando un usuario solicita los códigos el servidor los crea a partir del texto de las respuestas almacenadas en la base de datos (usando el paquete de *Python qrencode*), y utilizando el paquete de *Python Reportlab* se genera el documento a descargar. Para lograr esto, se guardan las imágenes que corresponden a los códigos de forma temporal en una carpeta.

Este recurso será utilizado por los clientes cuando usen la interfaz *Android*, de ahí su importancia en el proyecto.

**Multiformato**

Nuestro servidor está preparado para soportar dos tipos de formato estándar:

- HTML
- JSON

Para soportar el formato HTML se hace uso del sistema de plantillas de *Django*.

Para el caso de JSON debemos profundizar un poco más. El servidor habilita ciertas URLs para que los clientes pueden acceder por este medio. Solicitarán información en JSON tanto los dispositivos *Android* como la interfaz web (lo usa para la monitorización).

La generación del contenido JSON se hace a través del paquete de *Python simplejson*, que nos permite crear documentos HTTP con el formato JSON. Se ha optado por esta opción y no por el uso de plantillas debido a que consideramos necesario indicarle que tipo de codificación íbamos a utilizar, y esto no se logró con las plantillas.

## 3.3. Diseño e implementación de cliente WEB

Tras haber estudiado como se realizó el diseño e implementación del servidor, pasamos a estudiar lo propio con los clientes. Comenzaremos con el cliente web, es decir, aquel en la que el usuario utilizará como interfaz de comunicación un navegador web. En el siguiente apartado estudiaremos el diseño y la implementación del cliente *Android*.

### 3.3.1. Introducción

La existencia de una interfaz web que posibilite la comunicación con el servidor se justifica en que, a pesar de los grandes avances realizados en los dispositivos móviles, hay cosas que todavía no podemos realizar, aunque en un futuro próximo, con la integración de HTML5, sí será posible.

En las próximas hojas mostraremos como se diseñó e implementó este cliente web. La parte de diseño será breve y concisa, mientras que en la implementación nos entretendremos más explicando aspectos básicos de la aplicación.

Para comenzar, es importante señalar que la aplciación podrá ser usada por tres tipos de usuarios diferentes:

- Superusuario: Se encarga del mantenimiento y control de todo el sistema.
- Mánager: Se encarga de crear y gestionar grupos y actividades.
- Jugador: Usuario que puede participar en eventos.

Posteriormente estudiaremos cada una de sus funciones de forma detallada.

### 3.3.2. Diseño de la interfaz Web

A la hora de diseñar la interfaz web se planteó que debería ser accesible desde la mayor parte de los navegadores, y debería ser sencilla e intuitiva para facilitar el uso a sus usuarios.

Esta interfaz web consistirá en una serie de páginas HTML por medio de las cuales el usuario, tras previa autenticación, podrá realizar las tareas para las cuales tenga permiso (las acciones que puede realizar cada tipo usuario son diferentes).

Los objetivos mínimos que debía cumplir esta interfaz web los enumeramos a continuación:

- Que un usuario *mánager* pueda crear y monitorizar eventos.
- Que un jugador pueda acceder a las actividades y ver sus resultados.

A partir de estos objetivos básicos se realiza toda la aplicación web, que veremos en detalle en el siguiente subapartado.

### 3.3.3. Implementación

En esta sección estudiaremos de manera detenida la implementación del servidor, comentando todas sus particularidades. Especial atención requerirá la creación y gestión de las actividadades, ya que es uno de los puntos fuertes de la aplicación.

Comenzaremos con una base de datos limpias, sin usuarios, y explicaremos en detalle las funciones de los tres tipos de jugador: *superusuario*, *mánager*, y jugador.

Aunque no se diga explícitamente, todos los formularios son verificados con *Javascript* y las acciones que afectan a la base de datos siempre requerirán confirmación, también por medio de *Javascript*.


Figura 3.4: Página web de inicio de *Explohyperfiction*

### Creación de un superusuario

En primer lugar presentamos la página de inicio de la aplicación en la Figura 3.4, siempre y cuando no estemos autenticados en el sistema de *LibreGeoSocial*

La aplicación nos ofrece la opción de registrarnos en *LibreGeoSocial* y también autenticarnos. No nos meteremos en el tema de registro y autenticación, ya que las operaciones las realiza *LibreGeoSocial* y no nuestra propia aplicación. Al acceder al *home* tras autenticarnos nos encontramos la Figura 3.5, donde podemos observar que se nos da la opción de unirnos a *Explohyperfiction*, momento en el cual, crearemos nuestro perfil en la aplicación. Si no nos unimos no podremos hacer uso de las funcionalidades del servicio. En la Figura 3.6 vemos la página de inicio para los miembros, donde observamos:

- Existe un menú horizontal que nos permite el acceso a las características básicas del sistema. Este menú contiene a su vez menús despegables.
- Existe un mecanismo para cambiar la vista del usuario.

Posteriormente estudiaremos el cambio de vista y ese menú en cada uno de los casos existentes.

Cuando nos unimos a *Explohyperfiction* comenzaremos por defecto con una única función de jugador. Pero siempre que no haya un *superusuario* en el podremos solicitar


Figura 3.5: Página web de inicio de *Explohyperfiction* como usuario autenticado


Figura 3.6: Página web de inicio de *Explohyperfiction* como un usuario miembro


Figura 3.7: Mecanismo para crear el superusuario

al servidor serlo, tal y como se muestra en la Figura 3.7, y de esta manera tendremos permiso para acceder al sistema utilizando cualquiera de las tres vistas disponibles.

La creación del superusuario es necesaria para el funcionamiento correcto del sistema, ya que él se encarga de administrar los usuarios y grupos existentes.

### Cambio de vista

Partiremos de un superusuario con permiso para entrar al sistema como *superusuario*, *mánager* o jugador. Tal y como se mostró en la Figura 3.6, proporcionamos un mecanismo para hacer este cambio de forma rápida, sencilla e intuitiva. En la Figura 3.8 se muestra de manera específica este mecanismo.

Con este mecanismo seleccionamos la vista a la que deseamos acceder de una manera simple [2]. Cuando se cambia de vista se envía un POST al servidor, que realizará las opraciones deseadas, devolviendo la nueva vista. Este mecanismo está presente en todas las páginas de la aplicación web.

[2]Se tomó como referencia a la hora de crear este mecanismo la forma de selección de idioma que existe en el *moodle* de la ETSIT.


Figura 3.8: Mecanismo para cambiar de vista

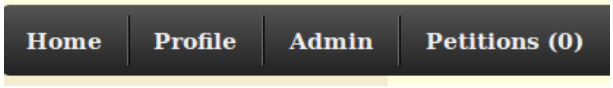


Figura 3.9: Menú disponible para un *superusuario*

### Vista de superusuario

Ahora nos pondremos a analizar la vista de *superusuario*. Esta vista sirve para administrar usuarios y grupos, pudiendo ser eliminados si el *superusuario* lo considerara oportuno. Además, se encarga de responder las peticiones existentes de los usuarios para ser *mánager* o *superusuario*.

En la Figura 3.9 se muestra el menú de la página de inicio con esta vista, donde observamos las siguientes opciones:

- Home: Da la posibilidad de volver a la página de inicio.
- Profile: Vista al perfil del usuario. Está disponible en todas las vistas, cambiando las operaciones disponibles.
- Admin: Menú despegable que nos da la opción de administrar usuarios o grupos.
- Petitions: Recurso que nos permite responder a las peticiones de los usuarios para ser *mánager* o *superusuario*.

En la Figura 3.10 se muestra el perfil para un *superusuario*. Sea cual sea la vista por la cual acceda a este recurso, sólo dispondrá de la opción de dejar de ser superusuario. Esta limitación de opciones se debe a la importancia del *superusuario* dentro del sistema.

Además, en la misma figura se puede ver los permisos que tenemos, así como nuestro nombres y apellidos. En nuestra aplicación sólo podremos modificar los permisos, ya

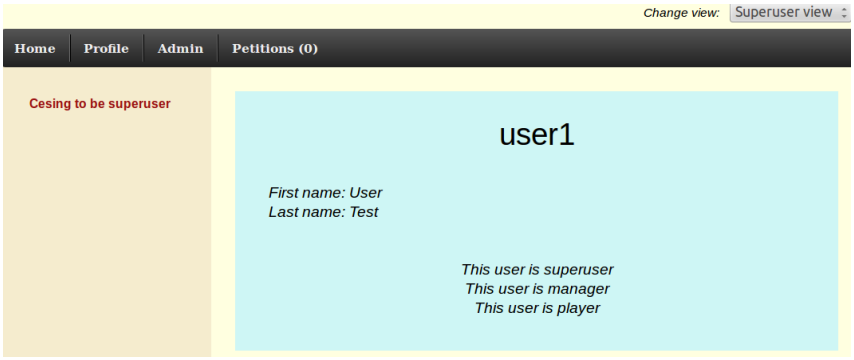


Figura 3.10: Profile para un *superusuario*.

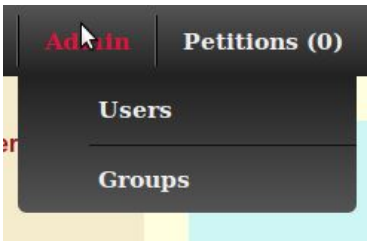


Figura 3.11: Detalle del menú despegable *Admin*

que los datos personales y otros datos son parte de *LibreGeoSocial* y hemos considerado adecuado que no se puedan modificar desde nuestra aplicación.

En la Figura 3.11 observamos el menú despegable que aparece al pasar el ratón por *Admin*. La aplicación de al *superusuario* la posibilidad de administrar grupos y usuarios, pudiéndolos eliminar.

Al acceder a estos recursos, se mostrará una lista de usuarios de la aplicación o de los grupos existentes. A modo de ejemplo, se muestra en la Figura 3.12 la lista de usuarios, donde el sistema nos permite acceder a su perfil, hacerle *mánager* o eliminarle del sistema.

Cabe destacar que un *superusuario* nunca podrá eliminar a otro igual, y que en cualquier momento puede quitar los privilegios de *mánager* a cualquiera. Cualquiera de estas acciones, debido a su importancia, requieren confirmación, implementada por medio de Javascript.

No mostraremos la lista de grupos, pero hemos de señalar que el *superusuario* podría borrar cualquier grupo existente (confirmándolo) excepto un grupo especial que crea el servidor, denominado *Free Group*, y que siempre debe estar presente en el sistema, y todos


Figura 3.12: Mecanismo para administrar los usuarios.

los usuarios son miembros de este grupo.

La última opción disponible para un *superusuario* es la gestión de peticiones. En el menú se le mostrará el número de peticiones pendientes que tiene para contestar, y podrá acceder a ellas para responderlas. Estas peticiones son creadas por los demás usuarios de la aplicación, y en caso de ser aceptadas estos usuarios pasarán a convertirse en *mánager* o *superusuario*, según el tipo de petición. En la Figura 3.13 se muestra el formato de estas peticiones y se ve como dos botones (creados usando *CSS*) permiten aceptar o rechazar las mismas.

La creación de peticiones por parte de los usuarios se verá posteriormente las respectivas vistas.

### Vista de *mánager*

En esta sección abordaremos brevemente las funciones básicas de un *mánager*, pero no comentaremos todo, ya que algunas funciones, debido a su importancia, merecen un apartado propio.

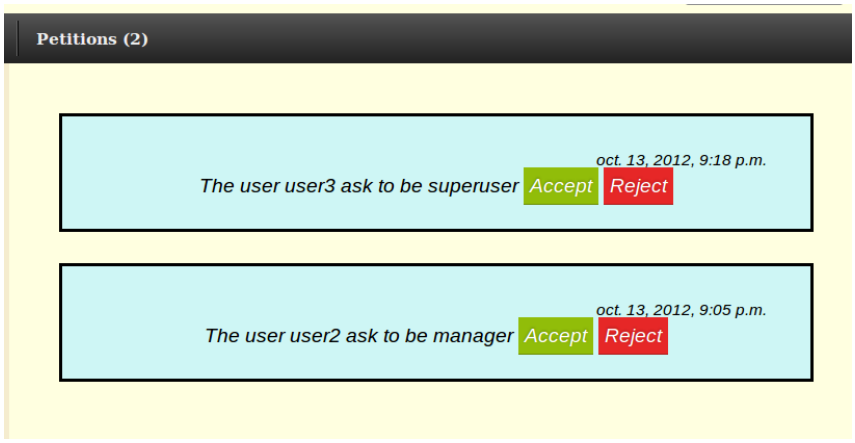


Figura 3.13: Mecanismo para gestionar las peticiones.


Figura 3.14: Menú en la vista de *mánager*.

En la Figura 3.14 podemos observar el menú que se ofrece a un usuario al acceder a la vista de *mánager*, que ofrece las siguientes opciones:

- Home: Para volver a la página de inicio.
- Profile: Para ver el perfil de usuario.
- Groups: Menú despegable que permite la creación y gestión de los grupos.
- Events: Menú despegable para la creación y gestión de actividades.
- Results: Menú despegable para ver los resultados de actividades finalizadas, pudiendo ser ordenadas por grupos.
- Monitoring: Herramienta de monitorización para actividades en curso.
- Notices: Mensajes de notificación del sistema.

En la Figura 3.15 podemos observar ciertas diferencias en el recurso *Profile* que presentamos para un *superusuario*. En primer lugar es la opción de solicitar ser *superusuario*, que generará un petición que recibirá un *superusuario*. Una vez creada esta petición, el

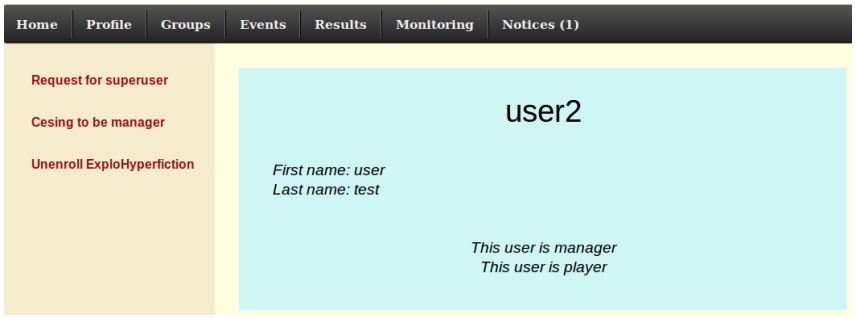


Figura 3.15: Perfil de un *mánager*.

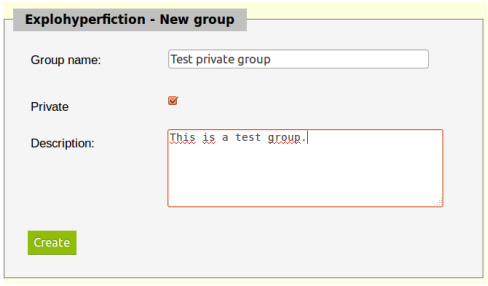


Figura 3.16: Formulario para crear un grupo.

usuario que la haya creado podrá eliminarla en cualquier momento. Además, se permite al *mánager* la opción de borrarse del sistema, cosa que no se permite a los *superusuarios*.

Ahora pasamos a tratar brevemente los grupos. Cualquier *mánager*, puede crear y gestionar grupos. Esto se hace con el propósito de aumentar los usos de la aplicación, ya que podría ser utilizado, por ejemplo, para dar soporte a diferentes asignaturas. En la Figura 3.16, se muestra el formulario a partir del cual se crea un grupo. Debemos indicar que el nombre será el elemento que identifique al grupo y que el elemento *private* indica si para formar parte del grupo se requiere autorización o no (de esta forma se crea un sistema de peticiones para grupos, similar al del sistema).

Tras haber creado un grupo, podremos acceder al perfil del grupo. En este recurso se nos permite ver todos los *mánager* del grupo y los usuarios que pertenecen a él. Además podremos gestionar los usuarios (haciéndolos *mánager* de nuestro grupo o expulsándolos) de manera similar a como hace un *superusuario* pero sólo sobre nuestro grupo. En el caso de que el grupo sea privado, podremos gestionar las peticiones de los usuarios, aceptándolas o rechazándolas. El perfil del grupo se muestra en la Figura 3.17.

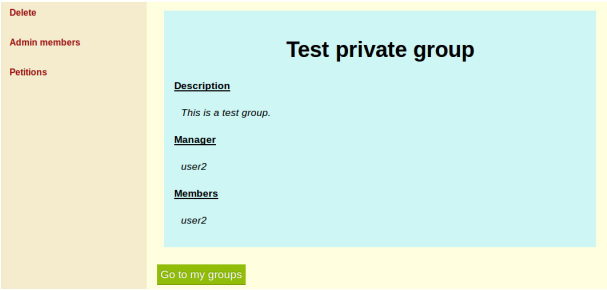


Figura 3.17: Perfil de un grupo privado.

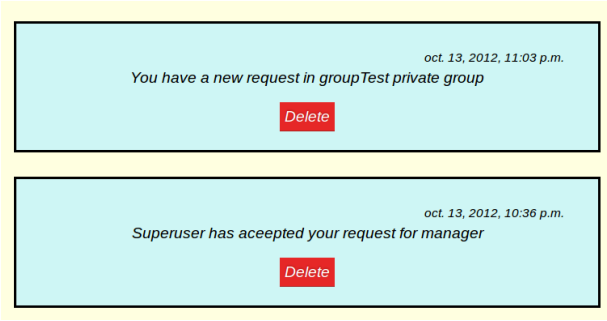


Figura 3.18: Lista de notificaciones del sistema.

La creación y gestión de eventos como los resultados y lo monitorización y resultados de los mismos irán en apartados posteriores, ya que por su importancia, consideramos que deben ser tratados de manera independiente.

Por último, nos centraremos en *Notices* que es un sistema de notificaciones del sistema y está presente en las vistas *mánager* y jugador. Cada vez que un *superusuario* haga algo relacionado con el usuario, o se responda a una de sus peticiones, el servidor generará una notificación que podrá ver el usuario explicando que es lo que ha pasado. Además, en el caso del *mánager*, se recibirán notificaciones siempre que se reciban peticiones de algún usuario para pertenecer a un grupo. En la Figura 3.18 se muestra esta lista de notificaciones del sistema, que pueden ser borradas por el usuario.

### Creación y gestión de actividades

En esta subsección comentaremos detalladamente el mecanismo implementado para crear y gestionar los eventos, lo cual es uno de los puntos fuertes del proyecto.

En primer lugar hemos de señalar que la creación y monitorización de actividades está

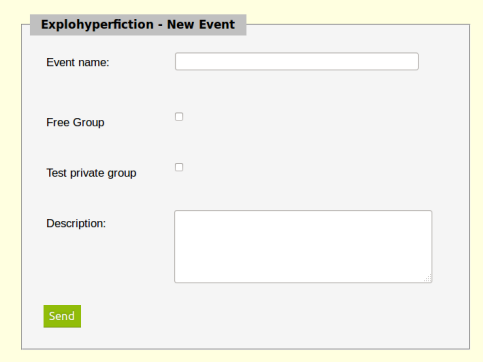


Figura 3.19: Formulario para la creación de formularios.

restringido a la vista de *mánager*, lo que requiere que el usuario del sistema tenga dichos permisos. De esta manera se evita que jugadores puedan acceder a estos recursos.

La creación de una actividad se realiza a través del formulario que se presenta en la Figura 3.19, donde observamos que un evento necesita un nombre, una descripción e indicar en que grupos está disponible. En la lista de grupos aparecerá solo los grupos en los que el usuario es *mánager* (a estos efectos se considera que cualquier *mánager* gestiona el *Free Group*), y el usuario seleccionará los que desee, pudiendo ser editado en cualquier momento.

Tras la creación del evento, el usuario es redirigido a la página principal del evento, que se muestra en la Figura 3.20. Estudiaremos a continuación las opciones que aparecen en el menú izquierdo:

- *Active/Deactive*: Permite al usuario activar o desactivar la actividad, es decir, define la visibilidad de la actividad. Si el evento está desactivado, lo jugadores no podrán acceder a él.

- *View questions*: Permite acceder a todas las preguntas creadas que pertenecen a esta actividad. Desde ahí se podrán editar las preguntas.

- *Print QR codes*: Nos ofrece la opción de obtener en un sólo documento todos los códigos QR de la actividad. Esta opción sólo está disponible en aquellos eventos con al menos un pregunta para códigos QR y se complementa con la opción de poder obtener los códigos QR de cada pregunta de forma aislada.


Figura 3.20: Página principal de una actividad

- *Add question*: Permite crear nuevas preguntas.
- *View map*: Mapa que permite ver la relación entre las preguntas y respuesta de la actividad.
- *Results*: Resultados de la participación de los jugadores en esta actividad.
- *Edit*: Permite editar los datos básicos de la actividad: nombre, descripción y grupos.
- *Delete*: Permite eliminar la actividad.

Tras haber presentado las opciones básicas pasaremos a estudiar detalladamente la creación de las preguntas. Para crear las preguntas se nos ofrece un formulario HTML, que se presenta en la Figura 3.21. En primer lugar indicaremos cual es la pregunta anterior, en el campo *before*, realizado con una etiqueta OPTION de HTML, donde nos aparecen todas las respuestas existentes del sistema. Esto nos permite asignar un nivel a la pregunta. Es decir, si su respuesta es *START* se le asigna el nivel 1, y si fuera otra, se le asignaría un nivel más que el que tuviera la pregunta anterior. Esto permitirá realizar el mapa lo más sencillo posible.

El problema de asignar de esta forma los niveles, es que una vez guardada una pregunta, el nivel no podrá ser editado.

Luego, se nos pide el texto de la pregunta y si la pregunta consiste en un código QR (esto sólo afecta en la interfaz *Android*). La pregunta admitirá un máximo de cinco respuestas (se limita por comodidad para el usuario, ya que a medida que aumentas las opciones los posibles caminos crecen exponencialmente y el diseño de las actividades sería

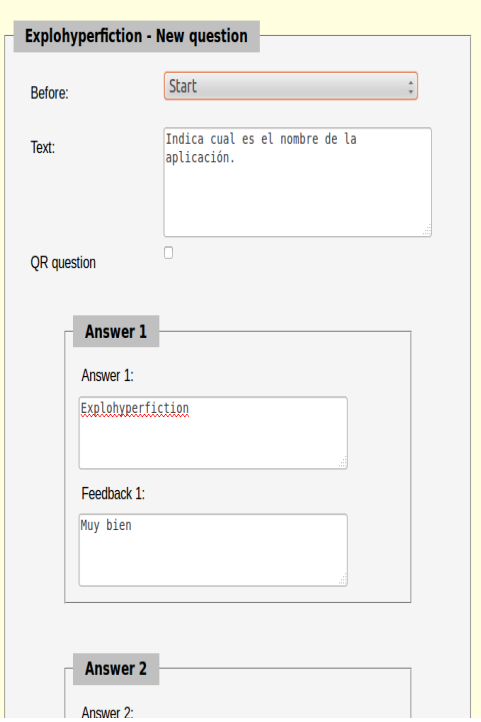


Figura 3.21: Formulario para la creación de preguntas

muy tedioso) con unos *feedback* asociado a cada respuesta. Este *feedback* es el mensaje que el creador quiere que aparezca en los resultados, es decir, es la única manera de indicar si una respuesta es mejor que otra.

La edición de la pregunta es similar a la creación, pero podremos modificar cual es la siguiente pregunta a una respuesta de la misma. Cuando se crea una pregunta, todas las respuestas son almacenadas considerando que no tienen siguiente pregunta. Esto se modifica cuando se crea una pregunta con esa respuesta como anterior o cuando desde la edición le indicamos que esa respuesta nos lleva a otra pregunta.

La creación de las preguntas y las respuestas son importantes, pero lo que caracteriza está aplicación son la relación existente entre ellas. Por ello, se ofrece al creador de una aplicación el ver un mapa de esas relaciones. Este mapa se realiza teniendo en cuanta el nivel al que corresponde cada pregunta. Para dibujarlo se ha utilizado el elemento *canvas* presente en *HTML5*. En la Figura 3.22 podemos observar un ejemplo de este mapa.

Este mapa se ofrece para ayudar al diseño de una actividad. Como vemos, la implementación de la misma es sencilla e intuitiva, pero el diseño de una actividad es algo tan

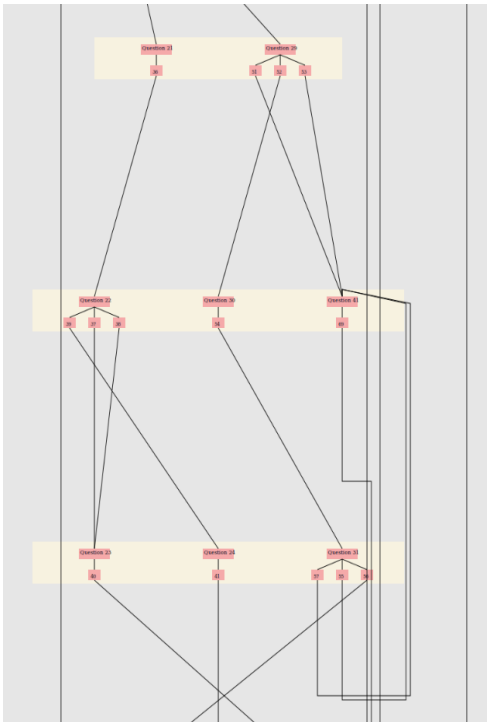

Figura 3.22: Ejemplo de mapa de relaciones entre preguntas

complejo commo escribir un libro. Al disponer de un mapa, podemos saber si hay algunas preguntas o respuestas con relaciones incorrectas con un rápido vistazo.

Con todo esto y añadiendo la generación de los códigos QR, ya estudiada en la implementación del servidor, tenemos los mecanismos suficientes para crear actividades. Pero además de crearlas es necesario poder acceder a resultados de ellas, y si es posible, monitorizarlas, lo que veremos en el siguiente apartado.

### Resultados y monitorización de actividades

Es esta sección mostramos como desde la interfaz web, un *mánager* puede ver los resultados de sus actividades finalizadas y monitorizar las actividades en curso. Como en la creación de actividades, estos recursos están restringidos a la vista de *mánager*.

A través de la opción *Results* del menú principal de la vista de *Mánager*, se puede acceder a los resultados de las actividades finalizadas. En la Figura 3.23 se muestra un ejemplo de estos resultados. Posteriormente en el estudio de la vista de jugador se estudiará más detalladamente la creación de este resumen, pero debemos decir en este punto que el

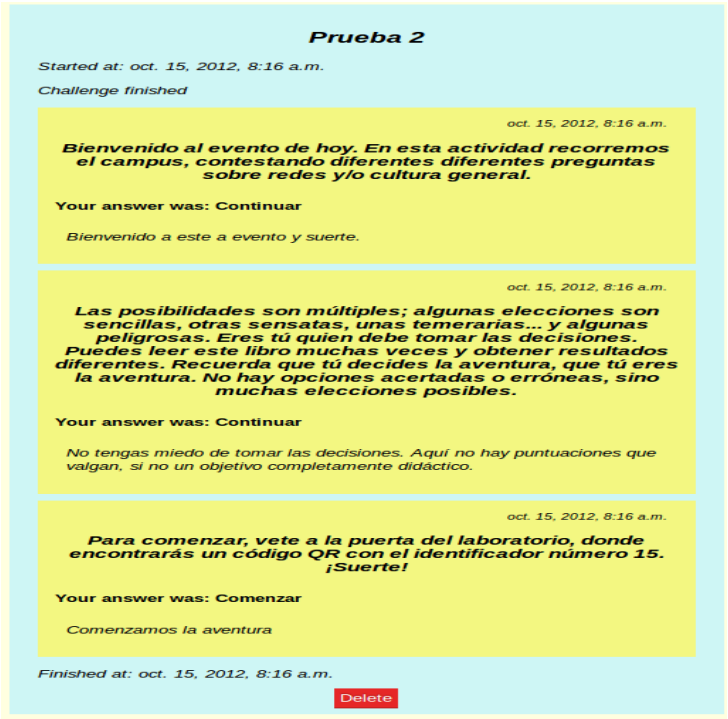


Figura 3.23: Resumen de resultados en la vista *mánager*

*mánager* es el único que puede borrar los resultados.

No es ámbito de esta sección el explicar como funciona la interfaz *Android* (esto se hará en el apartado específico), sino que nos centraremos en explicar como conseguir que el servidor muestre los datos necesarios en la interfaz *web*. Por lo tanto, supondremos a tales efectos que el servidor conoce la posición de los dispositivos móviles que estén participando en la actividad.

Respecto la monitorización de las actividades, podrán ser monitorizadas todas aquellas actividades que sean realizadas a través de la interfaz *Android*. El sistema sabe cuando una actividad es *Android* a través de un *flag* que se activa siempre que la actividad sea iniciada desde este tipo de dispositivos.

Cuando un *mánager* accede a la pestaña de monitorización se le mostrará una lista de todos las actividades activas. Al acceder a alguna de ellas, se le mostrara una primera monitorización con todos los participantes dentro de un mapa como se muestra en la Figura 3.24. Este mapa se ha realizado con la ayuda de *Google Maps*, donde a través de *AJAX* el cliente solicita periódicamente la nueva posición


Figura 3.24: Monitorización de todos los usuarios.

Además, se ofrece un listado de las últimas respuestas de todos los usuarios que estén participando, que también se va actualizando por medio de *AJAX*. Similar es la situación si seleccionamos un usuario en concreto, pero esta vez se nos mostrara todo el progreso del usuario en la actividad. Estas opciones no estarán disponibles en todos los navegadores, debido a que es necesaria la compatibilidad con *AJAX*.

### Vista de jugador

A continuación indicaremos brevemente las funciones básicas de la vista de jugador. Aunque se ofrece interfaz web para el *jugador*, el proyecto pretende que esta sea auxiliar y para realizar funciones que no sean el jugar (también se puede participar en actividades), proponiendo que se utilice la interfaz *Android* para ello, ya que de esta forma se podrán monitorizar los eventos y damos *vida* a la actividad.

Al entrar con la vista de jugador dentro de la aplicación obtenemos un menú como el que se observa en la Figure 3.25. En ella tenemos opciones similares a las de otras vistas. Expliquemos brevemente los recursos accesibles y posteriormente nos centraremos en algunos de ellos:

- Home: Para volver a la página de inicio

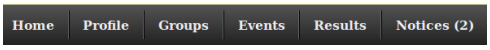


Figura 3.25: Menú en la vista de jugador.

- Profile: Para ver el perfil de usuario
- Groups: Menú despegable ver y unirnos a grupos
- Events: Menú despegable para ver los eventos disponibles, pudiendo ser ordenados por grupos.
- Results: Menú despegable para ver los resultados de actividades finalizadas.
- Notices: Mensajes de notificación del sistema

Comenzamos por comentar lo que es similar al resto de las vistas. El perfil será similar al mostrado en la Figura 3.15, pero ahora podremos mandar peticiones para ser *mánager* en vez de a *superusuario*. El resto es igual.

El sistema de notificaciones no cambia nada, ya que son generadas por el servidor cada vez que ocurre un hecho reseñable para el usuario, independientemente de su vista.

El primer cambio reseñable son las opciones que tenemos en el menú *Groups*. Por motivos de seguridad, a un usuario no se le permite crear grupos, por lo que está opción no la tendrán disponible. En su lugar, el jugador podrá ver los grupos a los que pertenece y una lista de todos los grupos existentes en el sistema, pudiendo, si desea, enviar una petición para unirse al grupo si éste es privado o unirse al grupo si éste no es privado, y desapuntarse del grupo en el caso de que el usuario ya pertenezca al mismo. En la Figura 3.26 se pueden observar algunas de estas opciones.

Si el grupo fuera privado el sistema genera un petición, y el jugador recibirá una notificación en el momento en que el mánager del grupo haya respondido a la misma. En caso contrario nos unimos al grupo. Todas las acciones anteriores requieren confirmación.

Por otro lado, todos los usuarios deben pertenecer al *Free Group*, por lo que de este grupo no podrán desapuntarse. Como ya se dijo, este es un grupo especial, creado por el sistema, con el objetivo de proveer a cualquier usuario de, al menos, un grupo al que pertenecer.

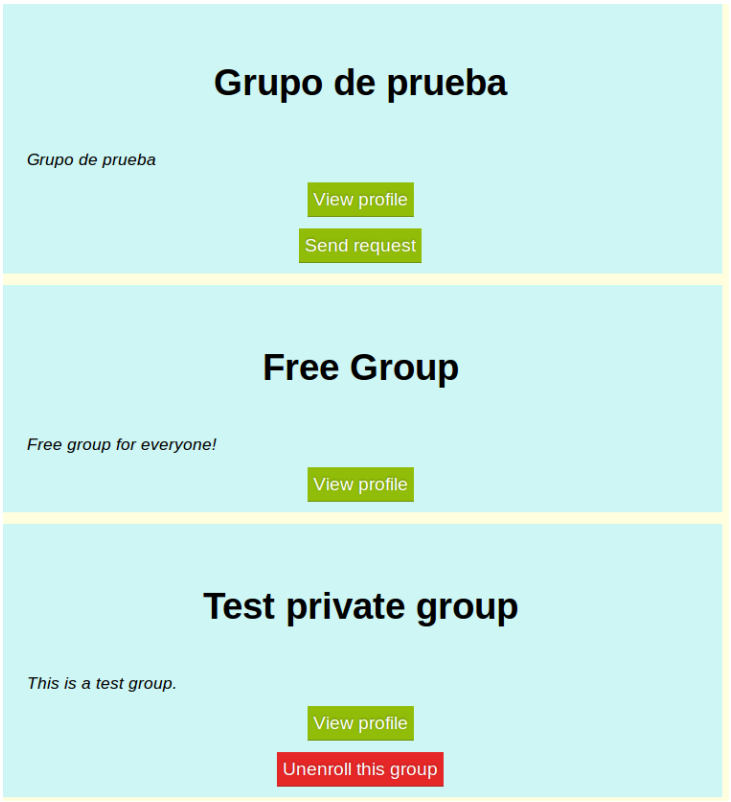


Figura 3.26: Lista de grupos para un jugador.

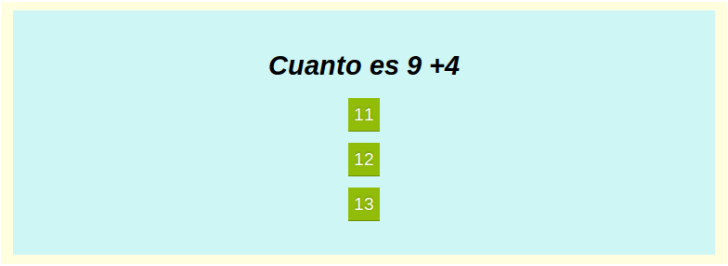


Figura 3.27: Preguntas en la interfaz web.

También un usuario puede participar en actividades a través de la interfaz *web*, pero este tipo de actividades no permitirán la geolocalización. La existencia de esta opción se justifica en que permite darle una mayor flexibilidad al sistema. Al seleccionar un evento el usuario puede comenzar a participar en él. Una pregunta se presentará con el texto y una serie de botones donde aparecen las respuestas posibles (los botones son realizados con CSS), que al *clickearlos* mandan la respuesta al servidor obteniendo la siguiente pregunta. En la Figura 3.27 se ilustra esto.

En el aspecto técnico, al seleccionar una de las opciones, enviamos al servidor nuestro respuesta. Cada respuesta tiene asociada la siguiente pregunta (ver en detalle en la sección *Creación y gestión de actividades*) que el servidor nos devuelve. En ese tiempo, el servidor crea un objeto *Response* que tiene asociada la pregunta y la respuesta. Así cuando seleccionamos una respuesta que ya no tiene siguiente pregunta, el sistema recopila todas las respuestas y nos muestra un resumen de nuestra actividad, tal y como se muestra en la Figura 3.28, donde observamos que además de tener la pregunta y la respuesta, hay un mensaje. Este mensaje es el feedback que proporciona el creador de la actividad al usuario.

En el menú *Results* el usuario puede acceder a los resultados de las actividades realizadas, siempre y cuando no hayan sido eliminadas por el *mánager* de dicha actividad. Esta información es exactamente igual a la que se muestra al usuario al finalizar una actividad y que se mostraba en la Figura 3.28.

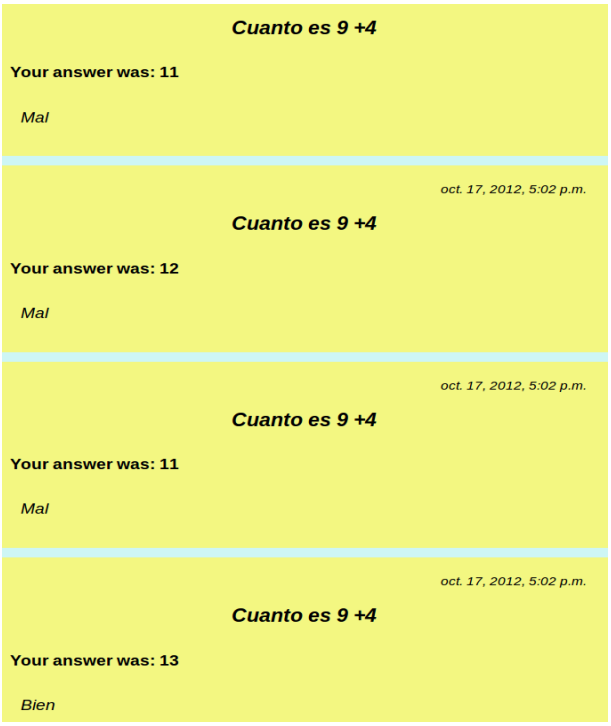


Figura 3.28: Ejemplo de resumen de actividad para el usuario.

### 3.3.4. Problemas

Tal vez, la parte que acabamos de presentar la parte en la que se ha dedicado más tiempo. El tener una interfaz básica para crear y gestionar grupo y eventos no fue muy complejo, pero había que añadir el resto de la funcionalidad.

Los mayores problemas estuvieron concentrados en el mapa de relaciones. Cuando nos lo planteamos, no sabíamos que tecnologías tendríamos disponibles para hacerlo. Investigamos sobre varias hasta que descubrimos que HTML5 nos podía permitir realizar el mapa, así que optamos por ella, ya que no era complejo y se trataba de algo nuevo para el uso de *canvas*.

La implementación no es compleja, pero el diseño sí, ya que hubo que plantearse todas las opciones, los bucles, las flechas que van a la izquierda, las de la derecha ... Pero al final, lo conseguimos.

El otro gran problema fue con la monitorización. Como se ha dicho en muchas ocasiones siempre se intento seguir como referencia la *gymkhana*[6], pero estaba implementada con una *API* desactualizada, por lo que hubo que rehacerlo completamente. Es por ello, que

una y otra no se parecen en nada, aunque en un primer momento era el objetivo. Sin embargo, hemos de decir que la comunicación con el servidor utilizando *AJAX* sí es muy similar.

## 3.4. Diseño e implementación de cliente *Android*

En apartados anteriores hemos estudiado como se diseñó e implementó el servidor y el cliente *web*. El tercer componente fundamental dentro de nuestra aplicación es la interfaz *Android*, que permite que dispositivos que usen este sistema operativo, puedan acceder al servicio a través de una *app*. El diseño y la implementación de esta *app* es lo que estudiaremos en este apartado.

### 3.4.1. Diseño de la aplicación *Android*

El diseño de la aplicación para dispositivos móviles que usen el sistema operativo *Android* se realizó bajo la principal premisa de que debería ser compatible con la mayor parte de los dichos dispositivos.

Además, la aplicación debería proveer al menos de los siguientes mecanismos, siendo a la vez una aplicación de uso sencillo:

- La aplicación debe ser capaz de conectarse al servidor, realizar peticiones y entender las respuestas, usando el formato JSON.
- La aplicación debe proveer de un mecanismo estable para mandar al servidor la localización del dispositivo.
- La aplicación debe ser capaz de trabajar con códigos QR.

Estos debían ser los servicios mínimos que ofreciera nuestra aplicación al usuario. Por ello, se diseña una aplicación simple pero que cumple todas especificaciones. En el siguiente apartado se estudiará como se ha implementado la aplicación de forma breve, pero explicando todos los detalles que sean necesarios para entender el proceso.

### 3.4.2. Implementación de la aplicación *Android*

Para la implementación de la aplicación *Android* se utilizó el *IDE Eclipse*, tal y como se comentó anteriormente. Bajo la premisa de que la aplicación fuera compatible con la mayor parte de los dispositivos *Android* se utiliza la API 1 de *SDK*, lo cual nos asegura que todos los dispositivos [3] puedan ejecutar nuestra aplicación.

Para explicar la implementación, nos apoyaremos en diferentes figuras que obtendremos del emulador que nos proporciona *SDK*, e iremos explicando todo aquello que se considere oportuno.

El objetivo de esta implementación es que a través de la misma podamos acceder a la mayoría de los servicios que tenemos en la interfaz *web*, en concreto, el que cualquier usuario pueda participar en actividades con su dispositivo móvil.

Al iniciar la aplicación, se nos permite autenticarnos en el sistema. Este paso es imprescindible, ya que el servidor realiza la gestión de sesiones a través de *cookies*, tal y como debe hacer un servidor que siga una filosofía *REST*, por lo que en cada petición del cliente debe existir algo para identificar su sesión. Por ello, al autenticarnos, igual que pasa con la interfaz *web* recibimos una *cookie* que hará posible que exista el concepto de sesión.

Tras autenticarnos llegamos al menú principal, que se muestra en la Figura 3.29. Ahí observamos que tenemos tres opciones:

- *Events*: Nos permite acceder a otro menú donde tendremos acceso a nuestro eventos disponibles.

- *Results*: Nos muestra una lista de resultados disponibles, de manera similar a como ocurre en la interfaz *web*.

- *Exit*: Salir de la aplicación de manera limpia

El menú *Events* nos llevará a otro menú, donde se nos da la opción de acceder a e acceder a una lista de todos los eventos disponibles o la de acceder a los eventos que ya hayamos comenzado y no hayamos finalizado (está opción se crea por si la aplicación

[3] Al menos en teoría todos los dispositivos deberían ser capaces de ejecutar la aplicación.

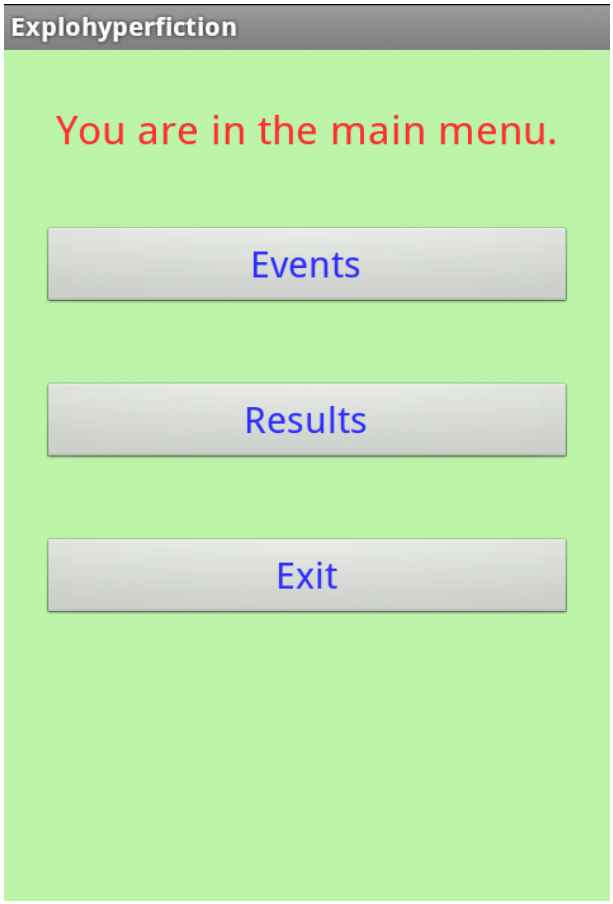


Figura 3.29: Menú principal de la aplicación *Android*

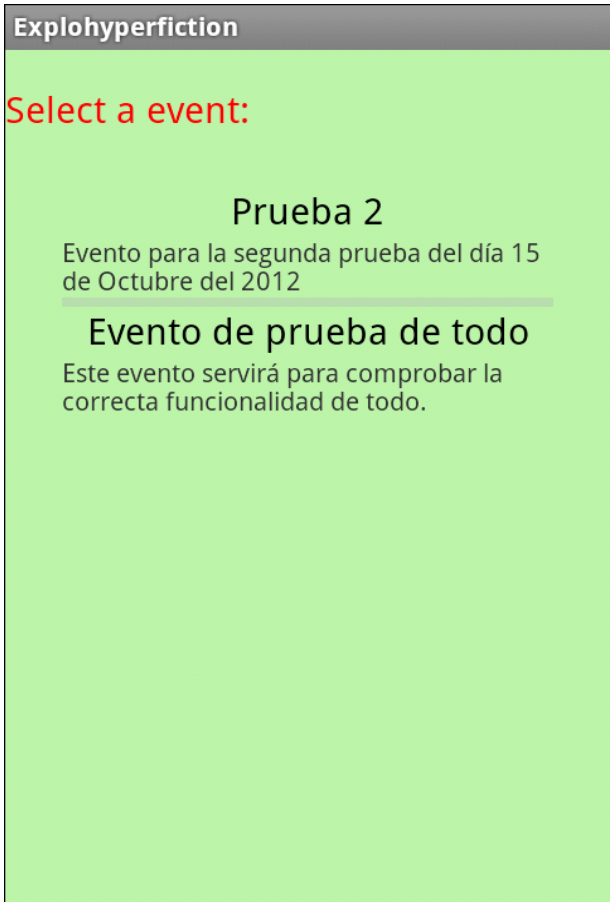


Figura 3.30: Lista de eventos disponibles en aplicación *Android*.

sufriera cualquier problema el usuario pudiera continuar por donde estuviera de la misma actividad).

Pulsemos la opción que pulsemos, la aplicación se conectará con el servidor para pedirle los eventos que debe mostrar (ya sean todos o los ya comenzados), y recibirá dichos eventos en formato JSON. Este información se convierte en *objetos* dentro de la aplicación, y se muestra por pantalla el nombre de cada actividad y su descripción. Esto se muestra en la Figura 3.30.

Tras confirmar que queremos comenzar ese evento, el cliente vuelve a conectarse con el servidor. Lo hace en dos ocasiones. La primera es para comunicar al servidor que va a comenzar esa actividad. A esta primera petición el servidor le responde con la pregunta que debe pedir y además, en ese momento arranca el servicio de geolocalización (que estudiaremos posteriormente). La segunda interacción es para pedirle dicha pregunta. En

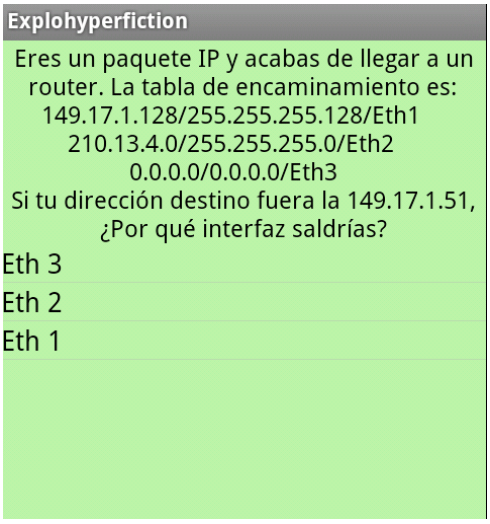


Figura 3.31: Pregunta en dispositivo *Android*

caso de que el que estemos continuando el evento el proceso es análogo, sólo que en vez de que el cliente comunique al servidor que va a comenzar un nuevo evento, le pide que le permita continuar uno ya comenzado, por lo que el servidor le devuelve la pregunta que corresponda.

Por tanto, el cliente pide la pregunta (ya sea la primera o la última, el proceso es igual) y recibe los datos en formato *JSON*, por lo que de nuevo debemos transformarlo y convertirlo en objetos. Como ya se dijo en la creación de actividades, las preguntas pueden ser para códigos QR o no. Primero centrémonos en el caso de preguntas sin QR, se muestran al cliente como se ilustra en la Figura 3.31.

Al seleccionar una respuesta, la aplicación la envía al servidor, y la respuesta consiste en indicarle la siguiente pregunta, y en caso de que ya no haya más comunicárselo. Realmente el proceso con códigos QR es el mismo y sólo cambia la forma de obtener el texto de la respuesta, ya que en este caso está en una imagen. El procedimiento para conseguir esto se explica en un apartado posterior, pero en este momento nos basta con saber que a partir de esa cadena, la aplicación ya puede enviar la respuesta.

Cuando el servidor indica que ya no hay más preguntas, el cliente solicitará el resumen, de manera similar a como lo hace en la interfaz *web*, sólo que en esta ocasión todo vendrá en formato JSON. Este formato se trata, y se muestra en pantalla tal y como vemos en la Figura 3.32. En este momento se acaba el servicio de geolocalización.

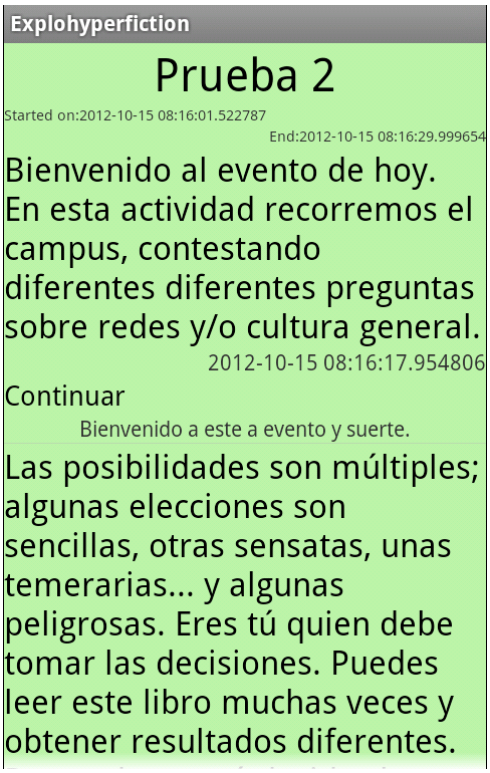


Figura 3.32: Ejemplo de resumen de actividad en *Android*.

La Figura 3.32 representa también lo que veremos cuando intentamos acceder a nuestros resultados a través del menú *Results*.

Por último, hemos de señalar que en todo momento se controla la acción que ocurre cuando se pulsa la tecla *atrás*, con el objetivo de que la salida del programa siempre sea lo más limpia posible.

Con esto, hemos visto como funciona la aplicación *Android*, pero hay dos elementos que merecen ser estudiados con un poco más de detalle: como leer códigos QR y como se realiza la geolocalización.

### Lectura de códigos QR

Como ya hemos visto, el cliente *Android* debe ser capaz de responder preguntas a través de la lectura de un código QR.

Para trabajar con códigos QR en aplicaciones *Android* podemos utilizar un aplicación externa o incluir en nuestra aplicación esa misma aplicación. En nuestro desarrollo, hemos optado por la primera opción, por lo que para la correcta ejecución de la aplicación es

necesario tener instalado el programa *Barcode*, propiedad de *XZing* y que está disponible de manera gratuita.

Al recibir una pregunta donde necesitemos leer un código QR, tendremos un botón para comenzar a ejecutar la aplicación externa. Esta aplicación, leerá el código QR y devolverá la cadena de caracteres que haya leído. De esta manera, ya se puede enviar al servidor la respuesta.

Por lo tanto, vemos que la lectura de los códigos QR utilizando una aplicación externa se realiza de una forma sencilla y eficaz, con el inconveniente de que el correcto funcionamiento de nuestra aplicación depende de otra.

### Servicio de geolocalización

El servicio de geolocalización consiste en que el dispositivo móvil enviará periódicamente su posición GPS al servidor, para poder ser usada en la monitorización de los eventos.

La geolocalización se ha implementado como un servicio, es decir, como una aplicación que está ejecutándose en segundo plano siempre que así se lo indiquemos.

*SDK* proporciona herramientas suficientes para realizar esta labor, por lo que nuestro trabajo consistió en encontrar un equilibrio de actualización y envío de mensajes para que este servicio no bloqueará todo la aplicación.

Por ello se optó porque el servicio de localización estuviera sólo disponible durante la realización de la prueba, por lo que se detendría al acabar la misma. Además, para no sobrecargar el dispositivo, cada vez que se accediera a la aplicación externa encarga de leer los códigos QR, se detiene este servicio, reiniciándolo cuando la otra actividad haya finalizado. De esta forma se evita que nuestra aplicación este en constante comunicación con otras.

El envío de la posición se hace a través de un mensaje POST, en el que se incluye latitud, longitud y altura, datos que se encargará de procesar el servidor para almacenarlos de forma adecuada.

### 3.4.3. Problemas

El desarrollo de la interfaz *Android* ha sido un verdadero quebradero de cabeza, ya que era la primera vez que se programaba en dispositivos móviles, y los conocimientos de *Java* eran muy básicos.

Se dedicaron muchas horas a conseguir un nivel adecuado de conocimiento de este tipo de programación hasta que comenzamos. Y al empezar, se decidió utilizar la *API 8*, lo que asegura la compatibilidad con cualquier dispositivo que lleve *Android 2.3* o superior (es decir, casi todo el mercado). Pero para mi sorpresa, los móviles que se nos facilitaron para pruebas llevaba un versión inferior, y hubo un salto considerable de una *API* a otra, por lo que se tuvo que repetir todo el desarrollo, esta vez utilizando la *API 1*.

Además la geolocalización fue un auténtico problema, porque bloquea constantemente el dispositivo sino se controla adecuadamente las actualizaciones de la posición. Hasta que nos dimos cuanta de ello, pensábamos que era la aplicación la que estaba fallando, lo cual nos hizo cambiar cosas y cosas. Finalmente, se encontró el fallo y se optó por una frecuencia de actualización de la posición en el servidor de 30 segundos.

En fin, hemos presentado una aplicación sencilla, que cumple lo que buscamos, pero somos conscientes de que en este aspecto la aplicación puede ser muy mejorada.

## 3.5. Resumen y conclusiones del diseño e implementación

A lo largo de las últimas páginas hemos estudiado como se ha diseñado e implementado nuestro servicio *web*. Comenzamos con una breve explicación del servidor, entrando un poco más en detalle en la generación de los códigos QR y en el soporte multiformato que ofrecemos. Este servidor debía dar soporte tanto a una interfáz cliente *web* como a una interfaz cliente *Android*.

Conociendo como trabaja el servidor, procedimos a explicar el diseño del cliente *web*. En él vimos que había tres vistas disponibles: *superusuario*, *mánager* y *jugador*.

De las tres vistas citadas la más importante era la de *mánager*, ya que es donde se crean las actividades y donde se encuentra la mayor parte de la funcionalidad específica de

la aplicación. Dentro de toda la funcionalidad comentada en el apartado correspondiente, debemos destacar:

- El usuario puede crear las actividades de una forma sencilla e intuitiva
- El usuario puede ver las relaciones entre preguntas y respuestas a través de un mapa, implementado a través de *canvas*.
- El usuario puede monitorizar sus actividades

Estas actividades que se crean pueden ser solicitadas por jugadores desde una interfaz *web* o una interfaz *Android*, siendo esta última de mayor importancia, ya que es la que permite usar tanto la lectura de los códigos QR, como la geolocalización.

A lo largo de estas páginas hemos intentando explicar de una manera sencilla y clara, como se ha diseñado e implementado todo.

Todo ha llevado muchas horas de desarrollo y estamos bastante orgullosos de lo conseguido, pero también hay que hacer autocrítica en esta parte. Podemos considerar la interfaz *web* como una herramienta completa, que permite la creación, gestión y monitorización de actividades de una manera sencilla. Sin embargo, la interfaz Android es bastante simple, y puede ser fácilmente mejorada, pero al menos cumple con los requisitos que se esperaba de ella. Por ello, queremos concluir esta parte de la memoria, destacando las bondades de la interfaz web, y esperando que el lector haya podido seguir sin problemas todo el desarrollo aquí planteado.

# Capítulo 4

# Resultados

*El mundo exige resultados. No les cuentes a otros tus dolores del parto, muéstrales al niño*

*Indira Ghandhi*

En este capítulo se presentan brevemente los resultados obtenidos en las pruebas realizadas con la aplicación.

Realmente, a lo largo de estos meses de trabajo se ha probado multitud de veces todas las funcionalidades del sistema, pero las pruebas de las que vamos a hablar son las que se han realizado con personas ajenas a mi familia o al proyecto, es decir, gente voluntaria que se ofreció a probar *Explohyperfiction*.

Se realizaron dos pruebas de esta forma y en las próximas líneas se comentarán brevemente los resultados en ellas obtenidos.

## 4.1. Primera prueba

La primera prueba se realizó el 1 de octubre de 2012. El objetivo era comprobar el correcto funcionamiento del servicio en un escenario real.

Para llevarla a cabo se contó con la ayuda de 2 voluntarios, cada uno equipado con un dispositivo móvil. La prueba consistía en la realización de un evento en el campus de Fuenlabrada, sencilla y corta para probar, sobre todo, la siguiente funcionalidad:

- Correcto funcionamiento de la lectura de códigos QR.

- Correcto envío de la posición por parte de los dispositivos al servidor.

Además, el otro objetivo, aunque los voluntarios no lo comprobasen, era ver que el mecanismo para crear las actividades era el adecuado.

### 4.1.1. Desarrollo de la primera prueba

La primera prueba se tuvo que detener al poco tiempo de comenzar por un problema en la lectura de los códigos QR. La aplicación leía sin problemas el primer código, pero se bloqueaba al intentar leer el segundo. Se debía a un problema en el código, ya que faltaba una palabra. Se solucionó el problema y los voluntarios volvieron a empezar.

A lo largo de toda la prueba se bloquea la aplicación de manera constante debido a que el servicio de geolocalización envía demasiados mensajes al servidor. Este problema no puede arreglarse *in situ*, por lo que queda pendiente.

### 4.1.2. Comentarios de los voluntarios

El voluntario 1 dice que la idea está bien, pero que la aplicación era muy lenta y si se bloqueaba y se cerraba tenía que volver a empezar de cero.

El voluntario 2 se queja sobre la velocidad de la aplicación, y el tamaño de la letra.

### 4.1.3. Conclusiones de la primera prueba

Esta primera prueba nos sirvió para comprobar que la aplicación permitía la creación de actividades correctamente, pero que el cliente *Android* todavía necesitaba arreglos. Se deciden realizar las siguientes acciones:

- Se decide modificar el servicio de geolocalización, disminuyendo de forma considerable el envío de mensajes al servidor.
- Se procede a introducir un mecanismo para poder continuar un evento ya empezado.
- Se intenta mejorar la interfaz gráfica para que su uso sea más cómodo para el usuario.

Además, al comprobarse que lo monitorización funciona, se deja como está.

Por tanto, esta prueba fue muy productiva, y nos permitió mejorar el servicio de forma considerable.

## 4.2. Segunda prueba

La segunda prueba se realizó el día 15 de octubre de 2012. Para la búsqueda de voluntarios se puso un mensaje en el *moodle* de la ETSIT, pero por problemas desconocidos, este mensaje no generó el correo que habitualmente llega a todos los alumnos cuando se crea un mensaje en el foro, por lo que la convocatoria no surtió mucho efecto.

Finalmente contamos con cinco voluntarios, y disponíamos de tres dispositivos móviles[1]. Se dividieron en dos grupos de dos personas y otra persona más iba sola.

La actividad consistía en responder a preguntas sobre redes y/o cultura general, y en caso de equivocarse, se *castigaba* al usuario con desplazamientos largos. Con esto se pretendía probar la función didáctica de la aplicación.

Además de la función didáctica esta prueba tenía como objetivo probar los cambios introducidos así como ver si los usuarios disfrutaban de la prueba.

### 4.2.1. Desarrollo de la segunda prueba

Durante todo el desarrollo de la prueba no hubo ningún problema técnico. El único problema que hubo fue que uno de los dispositivos se quedo sin batería justo antes de acabar, pero este hecho no oculta el éxito de la aplicación.

Los cambios introducidos en el servicio de GPS no crean ningún problema a los usuarios.

Durante el desarrollo de la prueba, y durante todo aquel día, hay problemas de red en el campus de Fuenlabrada, esto provoca que tengamos que trabajar con un ordenador del laboratorio, con una versión de *Firefox* que no permitía la correcta monitorización del sistema, ya que no mostraba el progreso de los participantes. Esto no se considera un problema, ya que fue algo eventual provocado por lo problemas de red y la desactualización del equipo utilizado.

### 4.2.2. Opiniones de los voluntarios

El voluntario 1 y el voluntario 2, que ya habían participado en la primera prueba, me felicitan por la mejora de la aplicación. Preguntados por si habían aprendido algo,

[1] También hubo problemas a la hora de conseguir los dispositivos

contestan que creían de antemano que se habían equivocado, ya que habían dado mucha vuelta. A pesar de indicar la mejora en la manejabilidad de la aplicación, dicen que podría ser algo más grande la letra.

El voluntario 3, pregunta interesado acerca de la idea, que considera muy buena. Dice que se ha equivocado bastante, porque en un momento le ha tocado repetir preguntas. Al leer sus resultados confirmó esa suposición.

Los voluntarios 4 y 5 dicen que se lo han pasado muy bien, que es muy divertido. No muestran ninguna queja acerca del funcionamiento

### 4.2.3. Conclusiones de la segunda prueba

La conclusión principal de la segunda prueba es que se comprueba que la aplicación está lista. Siempre se puede mejorar, pero cumple sobradamente los objetivos inicialmente marcados.

Se comprueba que puede ser usada didácticamente, ya que los usuarios reconocieron que después de la *caminata* que les había tocado dar por su error, ya no olvidarían las respuestas.

Además, nos alegramos al conocer que la gente había disfrutado con la actividad, ya que además se consigue que la aplicación pueda ser usada para actividades ociosas.

Por tanto, a pesar de que hay mucho margen de mejora, tras esta segunda prueba se decide parar con el desarrollo, dejando lo hecho como una primera versión, sin saber si habrá una segunda.

# Capítulo 5

# Conclusiones

*Capacidad experimental, honestidad en la publicación de los resultados e inteligencia para interpretarlos.*

*Indira Ghandhi*

Este apartado tiene como objetivo presentar al lector un resumen de todo el trabajo realizado. Para ello, en primer lugar analizaremos el cumplimiento de los objetivos marcados al inicio de esta memoria. Luego realizaremos un análisis subjetivo de todo el trabajo realizado, seguido de una autocrítica del proyecto, para continuar con un presentación de posibles líneas futuras de desarrollo. En el último lugar, presentamos una recapitulación a modo de despedida.

## 5.1. Análisis de objetivos

En la introducción presentamos de manera esquemática una serie de objetivos que debía cumplir este proyecto. En primer lugar presentamos unos objetivos generales, a saber:

1. Disponer de una base de datos para el almacenamiento estructurado de toda la información que se manejará en el sistema.

2. Disponer de una aplicación web para la creación de las actividades y su posterior monitorización.

3. Disponer de una aplicación web para participar en las actividades

4. Disponer de una aplicación *Android* para participar en las actividad.

5. Integrar la aplicación dentro de *LibreGeoSocial*.

6. Comprobar el correcto funcionamiento del sistema en un escenario real y atender los comentarios de los usuarios

A lo largo de la presente memoria hemos tratado de mostrar al lector como se han logrado cada uno de estos objetivos. Como hemos estudiado, todos estos objetivos se han conseguido en mayor o menos grado, y analizaremos a continuación algunos detalles que no se deben pasar por alto.

En primer lugar, hemos de considerar que disponemos de la base de datos y de las aplicaciones, y a través de estas herramientas conseguimos que el sistema sea accesible a todos los usuarios que lo deseen. De esta manera damos por cumplidos los cuatro primeros objetivos.

El quinto objetivo, la *integración de la aplicación dentro de LibreGeoSocial*, consideramos que no se ha logrado plenamente. A pesar de utilizar elementos de *LibreGeoSocial*, no hemos sido capaces de lograr una plena integración, debido sobre todo a que *LibreGeoSocial* (o al menos la versión con la que trabajamos) nos dio muchos problemas por la versión de desarrollo de *Django* utilizada. Este hecho ha provocado que, en muchos casos, no utilicemos las bondades que este proyecto ofrece, y hayamos tenido que implementar nosotros mismos cosas que ya estaban hechas, lo que conllevó más tiempo de desarrollo.

El sexto objetivo se considera cumplido, tal y como se presentó en el capítulo de *Resultados*. Ahí pudimos ver que se realizaron dos pruebas, y en base a los comentarios de los usuarios se procedió a mejorar la aplicación.

De esta manera podemos considerar que el servicio que presentamos cumple los objetivos generales que pedíamos casi al cien por cien. Pero aparte de estos objetivos generales pedíamos otros específicos. En concreto, para la aplicación de creación de actividades se pedía:

1. Sencilla e intuitiva.

2. Creación de códigos QR para las preguntas que se solicite.

3. Mostrar una figura con las relaciones entre las preguntas y respuestas.

El primer objetivo creo que lo hemos conseguido, pero eso dependerá de cada usuario. Para ello se hizo uso de las tecnologías HTML5 y CSS, creando una interfaz *web* que ya estudiamos en detalle en Capítulo 3.

El segundo objetivo se logró en el servidor, utilizando la librería de *Python qrencode*. De esta forma se permite que los usuarios puedan acceder a un documento *pdf* donde aparecen todas las preguntas con sus respectivos códigos QR, facilitando así la creación de actividades.

Por último, se cumple que la aplicación muestre al usuario un mapa de de las relaciones entre preguntas y respuesta, facilitando el diseño de las actividades. Este mapa se implementa utilizando el elemento *canvas* de HTML5.

También presentábamos algunos objetivos en la jugabilidad de la aplicación, que podemos resumir en:

1. Mostrar los resultados una vez finalizada una actividad.

2. Interpretar códigos QR en la interfaz *Android*.

3. Geolocalización en la interfaz *Android*, permitiendo la monitorización.

Como hemos visto a la largo de estas páginas todos estos objetivos se han cumplido. Mención aparte requiere el segundo objetivo, ya que para lograrlo se ha utilizado un aplicación externa, por lo que realmente nuestra aplicación no interpreta por ella misma los códigos QR.

Para finalizar los objetivos, considerábamos, que aparte de los objetivos generales y específicos, existían dos objetivos relacionados que debían ser cumplidos para la correcta realización del proyecto:

- Aprender a programar en *Android*

- Aprender a escribir en LaTeX

El aprendizaje de la programación en *Android* se ha realizado de forma autodidacta, lo que explica la sencillez de la interfaz presentada. Se intentó realizar un curso especializado, pero el mismo fue suspendido por falta de participantes, lo que provocó que tuviera

que dedicar muchas horas a entender *Android*. Al menos, podemos concluir que algo he aprendido, por lo que se da este objetivo como logrado.

Con respecto al aprendizaje de LaTeX, estas líneas que ahora mismo lee han sido escritas utilizando este programa, por lo que queda demostrado que se ha aprendido a utilizar LaTeX.

En resumen, en la *Introducción* se presentaron una serie de objetivos que este proyecto debía cumplir. Durante todo el Capítulo 3 se intento explicar de la manera más clara posible como se intento llegar a los mismo. Y en este apartado comprobamos que todos ellos se han cumplido (al menos parcialmente). Que hayamos cumplido estos objetivos no quiere decir que la aplicación sea perfecta, simplemente significa que la aplicación hace lo que debe hacer.

## 5.2. Análisis subjetivo del proyecto.

Tras hacer un análisis objetivo del proyecto en la sección anterior, en esta analizaremos subjetivamente todo lo realizado. Esto quiere decir que se trata de un análisis personal de las bondades del servicio (posteriormente ya criticaremos el sistema).

Tras comprobar que hemos satisfecho los objetivos marcados al inicio del proyecto, y tras haber probado el mismo, creemos que se puede afirmar que conseguimos, a través del servicio diseñado, *dar vida* a las *hiperficciones exploravitas*, permitiendo que se pueda aplicar el sistema a diversos y variados ámbitos, desde la educación hasta el ocio. Es por este potencial de la aplicación por lo que se recomienda encarecidamente seguir con el desarrollo de la misma.

Es por ello, que nos gustaría destacar la originalidad del proyecto, siempre teniendo en cuenta los antecedentes del mismo, es decir la *gymkhana* y los libre de *Elige tu propia aventura*. Queda patente este hecho por la ausencia de aplicaciones *software* o *web* similares, ya que a lo largo de la memoria se ha podido comprobar las diferencias tanto con la *gymkhana* (la estructura de una actividad es completamente diferente), como de los libros (damos *vida* a los libros).

No nos podemos olvidar de la innovación presente en el proyecto, sobre todo por el uso de HTML5 para crear los mapas dinámicamente. Antes de tomar esta opción, se investigó

(y mucho) acerca de las maneras posibles de implementarlo, y no se encontró ninguna factible de hacerlo de manera dinámica. Estáticamente era posible, incluso utilizando CSS, pero esto no era ni adecuado ni eficiente. Otras tecnologías utilizadas como *Android*, *Django* o *CSS* también se pueden considerar innovadoras, pero no en el mismo grado que HTML5, ya que ésta última se halla actualmente en desarrollo, mientras que del resto es muy fácil encontrar documentación y foros donde la mayor parte de los problemas están resueltos.

También, hemos de señalar que el proyecto ha intentado seguir todos los etapas que tendría un proyecto en el mundo empresarial. Desde el diseño hasta el despliegue (aunque este último sólo se realizó en pruebas). En el mismo sentido, indicar que todo el desarrollo se realizó buscando la satisfacción de un potencial usuario, algo imprescindible en cualquier aplicación empresarial.

Por último, nos gustaría recalcar que todo el proyecto se ha realizado enfocado a su sencillo uso, ocultando a los usuarios toda la complejidad existente, elemento que consideramos clave si queremos que el servicio sea utilizado en el futuro.

## 5.3. Autocrítica

Tras presentar las bondades del proyecto, creemos adecuado también hacer un poco de autocrítica. Al partir de otro proyecto ya existente (*gymkhana*[6]), estábamos convencidos de que su desarrollo sería rápido y sencillo. Pero al final, como se ha demostrado, se parece poco a la *gymkhana*, por lo que casi todo el desarrollo ha sido desde cero y con muchos problemas por el camino. Además, cuando comienzas a trabajar en algo más serio que una práctica de clase, te das cuenta de la cantidad de acciones que debes controlar, y la cantidad de problemas que eso conlleva.

En este sentido, la aplicación se ha diseñado e implementado, teniendo en mente en todo momento, que se debería poder manejar cualquier acción posible. Es por ello que se ha limitado el acceso a los recursos, y además se ha intentado evitar cualquier error en el servidor o en las aplicaciones. Sin embargo, no es malo reconocer que alguno se nos ha podido pasar, como puede ser un enlace con una dirección mal puesta, o un recurso al que puede acceder un usuario que no debería poder. Además, en la interfaz *Android*, se

intento de manera análoga, controlando cualquier excepción que pudiera ocurrir. Aun con el tiempo y esfuerzo dedicado a que el usuario no notase posibles errores en el servicio, podría haber quedado alguno.

En relación con la aplicación *Android*, nos gustaría criticar su sencillez. La aplicación sólo permite las operaciones básicas para su funcionamiento, y la interfaz que se proporciona al usuario es sencilla y algunos dirían que hasta *prehistórica*. Sin embargo, cumple sus objetivos básicos, que era lo que pretendíamos. Este hecho no debe ocultar que nos habría gustado presentar una mejor aplicación *Android*, pero los conocimientos que tenemos y el tiempo no nos han permitido más.

Pero igual que criticamos la interfaz *Android*, debemos alabar la interfaz *web*. En este sentido creemos que cumple los objetivos marcados para ella sobradamente y que su uso es sencillo. Especialmente orgullosos estamos del mapa de relaciones entre preguntas y respuestas. Pero también hemos de decir que es una pena que no lo hayamos desplegado de manera definitiva, de tal forma que cualquiera pudiera acceder ahora mismo al servicio.

Por lo tanto, podemos concluir que los puntos fuertes del servicio que hoy presentamos son los siguientes:

- Interfaz *web* potente y completa.
- Facilidad de uso.
- Gran ámbito de aplicación.

En contra, debemos señalar los siguientes aspectos:

- Aplicación *Android* muy simple.
- Posibles fallos no detectados.

Y teniendo en cuenta los puntos fuertes y débiles del sistema, creo que se puede concluir que el servicio implementado es útil, atractivo, y con un gran potencial de crecimiento y desarrollo.

## 5.4. Futuras líneas de trabajo

En esta sección nos gustaría proponer diferentes líneas de desarrollo con el objetivo de mejorar esta aplicación.

En primer lugar, y considerando la relación de este servicio con el de la *gymkhana*, creemos que se podrían integrar ambos servicios en uno mismo, utilizando las bondades de cada uno de ellos, lo que daría lugar a una aplicación más que interesante.

Si nos centramos en nuestro servicio, dejando de lado la *gymkhana*, creemos que en el lado del servidor, los trabajos futuros deberían encaminarse a lograr un plena integración con *LibreGeoSocial*, para lo cual también consideramos adecuado que este proyecto se actualice, lo que facilitaría enormemente el trabajo.

En el lado del cliente, concretamente en la interfaz *web* creo que los avances que pudieran hacerse deberían ir orientados en dos direcciones:

- Tener herramientas automatizadas de análisis de resultados.
- Proveer de un sistema de comunicación en tiempo real con los dispositivos móviles durante la monitorización.

Además, también se podría trabajar en la optimización del código realizado, y añadir pequeñas mejoras a la aplicación como nuevos tipos de preguntas, nuevas formas de obtener el feedback, o meter incertidumbre dentro de las pruebas, lo que generaría actividades totalmente inesperadas.

Respecto a la aplicación *Android*, creemos que es donde más trabajo se puede hacer, ya que la aplicación presentada es bastante simple. En este aspecto se propone desarrollar y mejorar dicha aplicación, permitiendo que la misma sea lo mas parecida posible a la interfaz *web*. También se propone desarrollar la aplicación móvil en HTML5 (que actualmente podría ser posible) y así poder migrar a otros tipos de dispositivos de una manera sencilla.

De todas formas, estas son sólo algunas de las líneas de desarrollo, ya que debido al potencial del servicio, las posibles implementaciones desarrolladas son infinitas, por lo que se invita al lector a crear algo a partir del servicio que proponemos.

## 5.5. Recopilación y despedida

Llegados a este punto creemos que es el momento de preparar el final. A lo largo de estás hojas hemos visto como hemos diseñado, implementado y desplegado un servicio *web*, que trata de recrear las *hiperficciones exploratívas*.

De esta forma, tras presentar las tecnologías que utilizamos, procedimos a mostrar como se ha realizado el trabajo. Y este trabajo tiene un resultado claro, el servicio *web*. Pero a parte de este resultado tangible, hay otras consecuencias que no son tan fáciles de mostrar en las hojas. En especial queremos comentar lo que hemos aprendido:

- Nos hemos dado cuenta que realizar un servicio *web* no es algo trivial. Hay que intentar no dejar nada al azar y esto requiere mucho tiempo, por muy sencillo que pudiera ser el servicio.
- Hemos aprendido a desarrollar aplicaciones *Android*.
- Tras todo el esfuerzo dedicado, comenzamos a disfrutar del trabajo, especialmente durante las pruebas, donde ves como todo aquello por lo que has estado trabajando funciona, y la gente disfruta con ello.

Por ello, y llegado este momento, tengo que decir que he disfrutado haciendo este proyecto, pero que también he sufrido. Podemos ilustrar este sufrimiento con la pérdida de peso, que ha sido de casi diez kilos en los últimos dos meses.

Ahora que llega el final, queremos agradecer al lector su tiempo. Se ha intentado no ser demasiado técnico siempre que no fuera necesario, permitiendo que cualquier lector pudiera entender lo escrito, sin necesidad de una fuerte base tecnológica. Aun así, sabemos que para algún lector todas estas páginas pudieran resultar tediosas y, en momentos, incomprensibles. Sin embargo, para el lector con conocimientos en la materia, la lectura ha podido resultar ligera, y el contenido poco profundo. La búsqueda de este compromiso buscando hacer este texto lo más accesible posible, es lo que ha provocado esto, pero siempre hemos intentando que todos los detalles importantes quedasen lo suficientemente claros para retratar el esfuerzo dedicado.

Espero sinceramente que el vocabulario y la escritura que han sido utilizados no oculten la labor realizada, ya que en ocasiones puede parecer que algunas cosas son triviales, pero

detrás de cada una de las lineas de código realizadas hay un gran esfuerzo.

No quisiera extenderme mucho más. Así que espero que el lector haya disfrutado leyendo esta memoria igual que yo he disfrutado escribiéndola.

# Bibliografía


[1] Android developer. `http://developer.android.com`.

[2] Django documentation. `http://www.djangoproject.com`.

[3] Libregeosocial. `http://libregeosocial.morfeo-project.org`.

[4] Postgresql documentation. `http://www.prostresql.org`.

[5] Python documentation. `http://www.python.org`.

[6] Fernández González, J. *Diseño, implementación y despliegue de un servicio de telecomunicación para M-Learning en móviles Android: Gymkhanas de nueva generación.* ETSIT URJC, 2010.

[7] Klinfeld, S. *HTML5 for Publishers.* O'Reilly Media, 2011. ISBN: 1-4493-1460-0.

[8] Kurose, J. F., and Ross, K. W. *Redes de computadoras: Un enfoque descendente 5ª Edición.* Addison Wesley, 2010. ISBN: 978-84-7829-119-9.

[9] Megías Jiménez, D., Mas i Hernández, J., Peig Olivé, E., Barceló Ordinas, J. M., Iñigo Griera, J., Martí Escudé, R., and Perramon Tornil, X. *Redes de computadores.* Universitat Oberta de Catalunya, 2004.

[10] Sáenz Albanés, A. J. Evolución del desarrollo web. In *Lenguajes y Sistemas Informáticos. Universidad de Sevilla* (2008).

[11] Tanenbaum, A. S. *Redes de Computadoras 4ª Edición.* Pearson, 2003. ISBN: 970-26-0162-2.

[12] W3C. *Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification*, June 2011.