



Escola d'Enginyeria de Telecomunicació i
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROYECTO FINAL DE CARRERA

Título: Interacción entre usuarios en aplicativos de Realidad Aumentada

Autor: Boris García Benítez

Director: Roc Meseguer

Fecha: 10 de diciembre de 2011

Título: Interacción entre usuarios en aplicativos de Realidad Aumentada

Autor: Boris García Benítez

Director: Roc Meseguer

Fecha: 10 de diciembre de 2011

Resumen

El presente PFC consiste en el diseño de una aplicación de Realidad Aumentada que posibilite la interacción entre usuarios. Esta aplicación consiste en la creación de una capa Layar que permita saber la localización de aquellas personas que el usuario tiene aceptados como amigos.

El proyecto está compuesto básicamente por tres partes. Una primera etapa en la que se hace un estudio de los posibles programas a utilizar en el desarrollo de la aplicación y la elección y justificación de Layar como el más idóneo.

Una segunda etapa que consiste en el diseño de la aplicación con todas sus funcionalidades. Entre las funciones más importantes destacan la creación de un perfil, la posibilidad de agregar y visualizar amigos, la de poder localizarlos en una ubicación GPS o incluso la poder interactuar con ellos a través de una llamada o un correo electrónico.

Por último se realiza una validación del funcionamiento del aplicativo de Realidad Aumentada, incluyendo una prueba de cada una de las funcionalidades mediante la utilización de dos smartphones.

Finalmente, se plantean las conclusiones obtenidas, los objetivos alcanzados, se proponen posibles mejoras y ampliaciones, así como una evaluación de la usabilidad de la aplicación.

Title: Interacción entre usuarios en aplicativos de Realidad Aumentada

Author: Boris García Benítez

Director: Roc Meseguer

Date: 10th December 2011

Overview

This PFC consists on the design of an Augmented Reality application. This application must allow interactions between users. It is the creation of a Laya layer that lets the user to know the location of his friends.

The project has three parts. A first stage that includes the comparison between different software applications for the development of the application. At the final of this chapter is chosen as the most suitable Laya.

The second stage involves the design of the application with all its functions. Among the most important functions are the creation of a profile, the ability to add and view friends and the possibility to locate them in a GPS. Even it offers the possibility to interact with them through a call or an email.

In the third part it presents a validation of the application of Augmented Reality. This phase includes a test of each of the functions through the use of two smartphones.

Finally, it presents the conclusions reached some improvements and extension of the application and the evaluation of the application.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN	1
1.1. Escenario y visión	1
1.2. Motivación	2
1.3. Objetivos	3
CAPÍTULO 2. REALIDAD AUMENTADA.....	5
2.1 Definición	5
2.2 Aplicaciones.....	6
2.3 Aplicaciones futuras	7
2.4 Tipos de Realidad Aumentada	8
2.4.1. Realidad Aumentada de Escritorio	8
2.4.2. Realidad Aumentada Móvil.....	9
2.4.3. Realidad Aumentada mediante reconocimiento de objetos	9
2.5 Tecnologías.....	9
2.5.1. Hardware	9
2.5.2. Software.....	10
2.6 Elección de Software	11
2.6.1. Comparativa	11
2.6.2. Layar, el Software escogido	12
2.6.3. Arquitectura de Layar	13
CAPÍTULO 3. DISEÑO DE LA APLICACIÓN	15
3.1 Funcionalidades de la aplicación	16
3.2 Creación de la capa.....	17
3.2.1 Creación de una cuenta Layar	17
3.2.2. Registro de la capa.....	18
3.3 Creación del hosting	22
3.3.1 Configuración del servidor.....	22
3.3.2 Creación de la Base de Datos (BBDD)	23
3.3.3 Creación de las páginas web	27

CAPÍTULO 4. DEMOSTRACIÓN Y EVALUACIÓN.....	31
4.1 Demostración.....	31
4.1.1 Visualización de amigos	31
4.1.2 Acciones en la capa	34
4.1.3 Acciones con amigos.....	38
4.2 Evaluación.....	39
4.2.1 Evaluación heurística	40
4.2.2 Evaluación mediante pruebas de usabilidad.....	40
4.2.3 Evaluación personal	42
 CAPÍTULO 5. CONCLUSIONES	 45
5.1 Conclusiones generales	45
5.2 Posibles mejoras	46
5.3 Impacto medioambiental	46
 BIBLIOGRAFÍA	 47
 ANEXO I BASE DE DATOS	 51
 ANEXO II PÁGINAS WEB	 53
 ANEXO III. TEST DE USABILIDAD	 81

CAPÍTULO 1. INTRODUCCIÓN

1.1. Escenario y visión

El sector de la tecnología es muy amplio, y está en constante evolución. Se debe tener presente que, en los últimos años la tecnología ha pasado de ser un mundo cerrado para unos pocos a estar presente en los diferentes ámbitos de la vida de los ciudadanos.

Prueba de ello es el mundo de los móviles, que en un primer momento eran utilizados por prestigiosos empresarios y hoy en día los usan desde niños de muy corta edad hasta personas jubiladas. Hemos llegado a un punto en el que nos es necesario saber dónde está la persona con la que hemos quedado, si nuestro hijo está bien o leer el correo electrónico en cualquier momento del día.

El hecho de esta dependencia informática en el mundo social hace que sea necesaria la constante creación y diseño de tecnologías más avanzadas para ser el primero en cubrir estas nuevas necesidades sociales. El presente proyecto pretende automatizar una aplicación que permita situar a diferentes usuarios en un lugar concreto por otro usuario sin necesidad de llamarle o enviarle un mensaje, sino a través de la Realidad Aumentada.

Como plataforma de desarrollo se utilizarán teléfonos móviles con Sistema Operativo Android, puesto que es libre y cualquier desarrollador puede bajarse su código fuente y crear aplicaciones. Además, está en constante evolución y crecimiento y cuenta con una gran comunidad de desarrolladores.

La realidad Aumentada hace posible que la información sobre el mundo real alrededor del usuario se convierta en interactiva y digital. La información puede ser almacenada y recuperada como una capa de información en la parte superior de la visión del mundo real.

Una de las últimas aplicaciones de la Realidad Aumentada es la publicidad, ámbito en el que muchas empresas están creando capas de información para que los consumidores puedan saber dónde está situado el establecimiento seleccionado, desde gasolineras hasta hoteles o restaurantes.

1.2. Motivación

Después de realizar mis estudios universitarios en Ingeniería Superior de Telecomunicaciones en la EPSC, hago balance de todas las asignaturas cursadas y me doy cuenta de que las que más me han interesado han sido las relacionadas con la programación y su posible y útil aplicación a la vida cotidiana. Es por esta razón que he intentado buscar un proyecto que se adapte a mis habilidades e inquietudes.

Además, de todos los proyectos que he hecho durante la carrera, los que más me han interesado y motivado han sido los de programación. Se debe al hecho de que me gusta poder diseñar e implementar algo que después produce un resultado operativo, que tenga interacción con un posible usuario. Lo prefiero a proyectos de investigación y diseño sobre el papel, que en ningún momento llegan a convertirse en algo concreto y utilizable.

Llegado el momento de desarrollar el proyecto final de carrera, decidí dirigirme a Roc Meseguer, uno de los profesores con los que cuenta la Universidad Politécnica de Cataluña. Fue entonces, cuando me propuso abarcar algún tema relacionado con la Realidad Aumentada, tecnología que actualmente se encuentra en un fuerte crecimiento.

Esta tecnología se encuentra en auge desde hace un par de años, tanto es así que un artículo de la revista TIME indica a principios de 2010 que la Realidad Aumentada es una de las tendencias tecnológicas a seguir en 2010 [16]. Este también indica que está en proceso de colocarse dentro de la cultura popular. Otro artículo de este mismo año se atreve a vaticinar que la RA podría salvar la prensa en papel, gracias a la mayor interactividad que nos podría ofrecer ésta al incluir Realidad Aumentada [17].

Destacar también que en el **Mobile World Congress 2011 de Barcelona** la Realidad Aumentada tuvo un importante papel [18]. Así como la mejora de varios puntos importantes necesarios para ésta, como el GPS y la brújula, claves para su precisión. No obstante varios artículos hacen referencia a que en la actualidad el interés más importante es que se desarrollen aplicaciones que cumplan las expectativas y los usuarios puedan disponer de cada vez más y mejores aplicaciones.

Tras barajar diversas posibilidades decidimos investigar sobre la creación de una aplicación de Realidad Aumentada que permitiera interacción entre usuarios.

1.3. Objetivos

El diseño de la aplicación está orientado a conseguir que la Realidad Aumentada no se vea como un dispositivo que ofrece información al usuario sino también como una tecnología que permite la comunicación entre diferentes usuarios, es decir, se pretende introducir al usuario dentro de esta realidad.

A partir de esta aplicación será posible saber la localización de aquellas personas que estén logadas como amigos, lo que permitirá por ejemplo, que si hemos quedado con una persona en un punto a una hora concreta podamos ver a tiempo real dónde se encuentra, si ha salido de casa o si ya ha cogido el metro.

El desarrollo del proyecto se divide en tres partes, que siguen el siguiente orden lógico y cronológico:

1. Elección del software con el que diseñar la aplicación. Se realizará la comparativa entre las diversas plataformas de desarrollo existentes o incluso la posibilidad de la creación de un software nuevo.
2. Desarrollo y diseño. Consiste en la programación a partir del software escogido, de forma que el usuario final pueda:
 - Ver una lista de los amigos agregados.
 - Añadir amigos nuevos.
 - Visualizar a los amigos que se encuentran próximos.
 - Interactuar con éstos, es decir, poder llamarlos o enviarles un correo electrónico desde el aplicativo.
3. Validación de la implementación. Esta fase incluye la revisión del cumplimiento de todos los objetivos prefijados. Para ello se utilizarán dos Smartphones y se irá realizando una prueba de cada una de las funcionalidades.
4. Evaluación del aplicativo. Por último se realizará un análisis de usabilidad del aplicativo mediante varias técnicas de evaluación.

CAPÍTULO 2. REALIDAD AUMENTADA

2.1 Definición

La **realidad aumentada** (RA) es el término que se usa para definir una visión del mundo real combinada con elementos virtuales para la creación de una realidad a tiempo real [1]. Consiste en un conjunto de dispositivos que añaden información virtual a la información física ya existente, es decir, añadir una parte sintética virtual a lo real. Ésta es la principal diferencia con la realidad virtual, puesto que no sustituye la realidad física, sino que superimprime los datos informáticos al mundo real.

Con la ayuda de la tecnología, la información sobre el mundo real alrededor del usuario se convierte en interactiva y digital. La información puede ser almacenada y recuperada como una capa de información en la parte superior de la visión del mundo real.

La definición de realidad aumentada, por tanto, debe incluir siempre tres características claves; la combinación de elementos reales y virtuales, la interacción a tiempo real y la visualización de imágenes en 3D.

La realidad aumentada puede ser usada en varios dispositivos, desde computadores hasta dispositivos móviles; HTC Android e Iphone son dos de los dispositivos que ya están implementando esta tecnología. La siguiente imagen muestra un ejemplo de ésta tecnología (**Figura 2.1**).



Figura 2.1 Ejemplo de Realidad Aumentada

2.2 Aplicaciones

La realidad aumentada ofrece infinidad de nuevas posibilidades de interacción, como son la arquitectura, el entretenimiento, la educación, el arte, la medicina o las comunidades virtuales.

Proyectos educativos: Actualmente la mayoría de aplicaciones de realidad aumentada para proyectos educativos se usan en museos, exhibiciones y parques de atracciones temáticos, puesto que su coste todavía no es suficientemente bajo para que puedan ser empleadas en el ámbito doméstico. Estos lugares aprovechan las conexiones wireless para mostrar información sobre objetos o lugares, así como imágenes virtuales, como por ejemplo ruinas reconstruidas o paisajes tal y como eran en el pasado.

Cirugía: La aplicación de realidad aumentada en operaciones permite al cirujano superponer datos visuales como por ejemplo termografías o la delimitación de los bordes limpios de un tumor, invisibles a simple vista, minimizando el impacto de la cirugía.

Entretenimiento: Teniendo en cuenta que el de los juegos es un mercado que millones de dólares al año, es comprensible que se esté apostando mucho por la realidad aumentada en este campo.

Una de las puestas en escena más representativas de la realidad aumentada es el "Can You See Me Now?", de Blast Theory. Es un juego on-line de persecución por las calles, donde los jugadores empiezan en localizaciones aleatorias de una ciudad, llevan un ordenador portátil y están conectados a un receptor de GPS. El objetivo del juego es procurar que otro corredor no llegue a menos de 5 metros de ellos, puesto que en este caso se les hace una foto y pierden el juego.

A pesar de estas aproximaciones, todavía es difícil obtener beneficios del mercado de los juegos puesto que el hardware es muy costoso y se necesitaría mucho tiempo de uso para amortizarlo.

Simulación: Se puede aplicar la realidad aumentada para simular vuelos y trayectos terrestres.

Servicios de emergencias y militares: En caso de emergencia la realidad aumentada puede servir para mostrar instrucciones de evacuación de un lugar. En el campo militar, puede mostrar información de mapas o la localización de los enemigos.

Arquitectura: La realidad aumentada es muy útil a la hora de resucitar virtualmente edificios históricos destruidos, así como proyectos de construcción que todavía están bajo plano.

Los dispositivos de navegación: La RA puede aumentar la eficacia de los dispositivos de navegación para una variedad de aplicaciones. Por ejemplo, las lunas delanteras de los automóviles pueden ser usadas como pantallas de visualización frontal para proporcionar indicaciones de navegación e información de tráfico.

Publicidad: Una de las últimas aplicaciones de la realidad aumentada es la publicidad. Hay diferentes campañas que utilizan este recurso para llamar la atención del usuario. Fiat ha lanzado una campaña en la que cualquier usuario puede crear su propio anuncio de televisión con el Fiat 500 como protagonista a través de la página web, el usuario solo necesita tener una webcam.

2.3 Aplicaciones futuras

En la realidad aumentada es muy importante determinar la **orientación y posición exacta** del usuario, sobre todo en las aplicaciones que así lo requieran. Uno de los retos más importantes que se tiene a la hora de desarrollar proyectos de realidad aumentada es que los elementos visuales estén coordinados a la perfección con los objetos reales, puesto que un pequeño error de orientación puede provocar un desalineamiento perceptible entre los objetos virtuales y físicos.

En zonas muy amplias los sensores de orientación usan magnetómetros, inclinómetros, sensores inerciales... que pueden verse afectados gravemente por campos magnéticos, y por lo tanto se ha de intentar reducir al máximo este efecto.

Como reto a largo plazo es posible sugerir el diseño de aplicaciones en los que la realidad aumentada fuera un poco más allá, lo que podemos llamar "realidad aumentada retroalimentada", esto es, que la "descoordinación" resultante del uso de sensores de posición/orientación, fuera corregida midiendo las desviaciones entre las medidas de los sensores y las del mundo real.

Es importante señalar que la realidad aumentada es un desarrollo costoso de la tecnología. Debido a esto, el futuro de la RA depende de si esos costes se pueden reducir de alguna manera. Si la tecnología de la RA se hace asequible, podría ser muy amplia, pero por ahora las grandes empresas son los únicos compradores que tienen la oportunidad de utilizar este recurso. En el futuro se podrían encontrar aplicaciones de este estilo:

- Aplicaciones de multimedia mejoradas como; pseudo pantallas holográficas virtuales, sonido envolvente virtual de cine, "holodecks" virtuales que permitan a imágenes generadas por ordenador interactuar con artistas en vivo y la audiencia.

- Sustitución de teléfonos móviles y pantallas de navegador de coche por la inserción de la información directamente en el medio ambiente. Por ejemplo, las líneas de guía directamente en la carretera.
- La mejora de la vida cotidiana a través de plantas virtuales, vistas panorámicas, obras de arte, decoración, iluminación, etc.
- Letreros virtuales, carteles, señales de tráfico, las decoraciones de Navidad y las torres de publicidad.
- Cualquier dispositivo físico que actualmente se produce para ayudar en tareas orientadas a datos (como el reloj, la radio, PC, fecha de llegada/salida de un vuelo, una cotización, etc.) podrían ser sustituidos por dispositivos virtuales.

2.4 Tipos de Realidad Aumentada

Es importante tener en cuenta que hay tres tipos diferentes de la realidad aumentada: la conocida como realidad aumentada móvil, la llamada realidad aumentada de escritorio y la RA mediante reconocimiento de objetos.

2.4.1. Realidad Aumentada de Escritorio

La Realidad Aumentada de Escritorio funciona con una cámara web. Sobre lo que captura la cámara, una aplicación muestra un objeto animado en 3D. La aplicación se activa cuando detecta un “marker”, algo parecido a un código QR. La **Figura 2.2** muestra un ejemplo de estos “markers”.

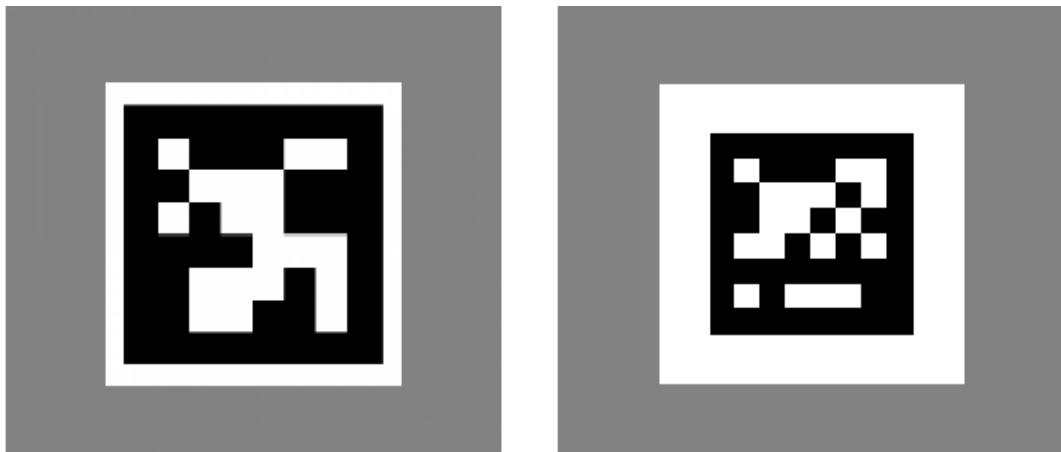


Figura 2.2 Ejemplo de Marker

Como podemos observar está compuesto por un marco y un símbolo en su interior, de tamaño y forma característica. El software se basa en el reconocimiento de estos objetos para determinar que objeto 3D se representa y su posición.

La dificultad que surge al utilizar markers es que es necesario tener la imagen impresa en alguna superficie para poder ver el objeto 3D, y si la imagen no se reconoce, ya sea porque esté oculta una parte o porque esté deteriorada, no se mostrará nada.

2.4.2. Realidad Aumentada Móvil

La Realidad Aumentada Móvil utiliza servicios basados en localización (GPS y similares) que nos dan la posición geográfica del dispositivo móvil. El problema de este modo es la poca precisión de los GPS existentes hoy en día, de manera que si nuestra posición varía, aunque sea un poco, podemos ver saltar los objetos de un punto a otro.

Usando una aplicación RA podemos saber, por ejemplo, dónde está la farmacia más cercana, qué casas están en venta o tener más información sobre el edificio histórico que tenemos en frente de nosotros.

2.4.3. Realidad Aumentada mediante reconocimiento de objetos

Esta opción se basa en el reconocimiento de objetos conocidos, por ejemplo, la forma de un edificio histórico o un objeto determinado. Al reconocerlo, nos informaría sobre su perfil, su historia y sus enlaces de interés.

El problema de este método es la dificultad de reconocer la forma del objeto, ya que en el proceso se tendrían que cotejar las formas de la imagen con una base de datos. Contra más objetos se quieran reconocer, más grande será la base de datos necesaria.

2.5 Tecnologías

El presente proyecto versa sobre la realidad aumentada móvil. A continuación se detallan las tecnologías necesarias para su composición.

2.5.1. Hardware

- Pantalla: instrumento donde se verá reflejada la suma de lo real y lo virtual que conforma la realidad aumentada.
- Cámara: es la encargada de capturar la realidad y transmitirla al software.
- GPS y brújula: son necesarios para que el software detecte la posición y dirección del usuario, es lo que se conoce como “dónde estoy y hacia dónde estoy mirando”.

Los smartphones son el ejemplo perfecto de hardware, ya que cumplen todos los requisitos necesarios para obtener una RA, son de pequeña dimensión y sus ventas están aumentando considerablemente en los últimos años.

2.5.2. Software

Programa que toma los datos reales y los transforma en realidad aumentada. Debe incluir una serie de marcadores, que básicamente son hojas de papel con símbolos que el software interpreta y de acuerdo a un marcador específico realiza una respuesta específica, como por ejemplo mostrar una imagen 3D, hacerle cambios de movimiento al objeto 3D que ya este creado con un marcador, etc.

A continuación, la **Figura 2.3** muestra una captura de ejemplo de realidad aumentada móvil. En ella se puede observar cómo el hardware es capaz de detectar la posición del usuario a través del GPS y transmitirla al software. Automáticamente aparecen los marcadores a partir de los cuales podemos obtener información sobre el monumento y cada una de las cosas que se quieren resaltar sobre él.



Figura 2.3 Ejemplo de Realidad Aumentada móvil

2.6 Elección de Software

2.6.1. Comparativa

Para la creación de la aplicación de interacción entre usuarios cabe la posibilidad de escoger entre diversas vías de diseño. La primera disyuntiva que se plantea es si escoger una aplicación ya existente que permita desarrollar la capa deseada o, por el contrario, diseñar un nuevo aplicativo.

Esta segunda opción es factible debido al hecho de que actualmente existen muchas librerías que permiten el desarrollo de aplicaciones de Realidad Aumentada. Hace unos años dichas librerías no estaban creadas y hubiera sido muy difícil el diseño de un aplicativo nuevo. Uno de los mejores *frameworks* para el diseño de un nuevo aplicativo hubiera sido PhoneGap, ya que es una plataforma que dispone de mucho soporte para el desarrollador.

No obstante, se ha optado por la primera opción debido a que, crear de cero el nuevo aplicativo supone un improductivo trabajo de programación para crear la base de éste. Además, de la comparativa de aplicaciones existentes se desprende que es posible el desarrollo de capas.

Centrándose pues en la opción de diseñar a partir de una aplicación que ya exista, es necesario buscar estos aplicativos y compararlos para escoger aquél que resulte más útil [2]. En la búsqueda realizada se han encontrado muchos aplicativos de Realidad Aumentada, no obstante, no todos eran aptos para crear la aplicación, pues algunos de ellos sólo están diseñados para ser utilizados en una zona geográfica concreta, como por ejemplo Sekaicamera que está pensado para ser ejecutado sólo en Japón.

Otros sólo permiten la creación de nodos, como por ejemplo ARViewer. Éste es bastante completo, ya que incluso permite la utilización de la altitud, pero no permite el desarrollo de capas, que es la característica que necesitamos para crear la interacción entre usuarios.

Por tanto, como aplicaciones que permitan el diseño del aplicativo se destacan: Wikitude [3], Junaio [4] y Layar [5].

En la **Tabla 2.1** se realiza una comparativa entre los tres:

	Wikitude	Junaio	Layar
Número de usuarios	500 K – 1M	100K – 500K	1M – 3M
Tutoriales	SI	SI	SI
Showcases	SI	SI	SI
API	SI	SI	SI
Soporte	BIEN	BIEN	MUY BIEN
Blogs	SI	NO	SI
Madurez	Refundada <1 año	1 año	2 años

Tabla 2.1 Comparativa plataformas para desarrollar en Realidad Aumentada

Las tres aplicaciones están en constante evolución y crecimiento, de hecho sus páginas web están en constante variación. Todas ellas permiten el desarrollo y la creación de capas, no obstante, tras el análisis de ellas por separado se ha decidido trabajar con Layar, debido a que tiene más madurez.

Esta mayor madurez es porque lleva mayor tiempo en el mercado, lo que hace que haya un mayor número de usuarios y de desarrolladores trabajando con esta aplicación. Además, el soporte es un poco mejor ya que los desarrolladores son más expertos.

Por tanto, la aplicación escogida para el diseño es Layar.

2.6.2. Layar, el Software escogido

Layar es como el Firefox de la Realidad Aumentada, creado como un navegador especial para esta tecnología. A partir de nuestro posicionamiento GPS, la brújula, la cámara y una conexión a Internet nos permite mostrar por pantalla las capturas del mundo real, añadiendo una capa virtual con información y posicionamiento de unos puntos llamados puntos de interés (POI). Es una aplicación disponible para móviles con sistema operativo Android y para Iphone.

Se pueden ir agregando layers (capas) que funcionan de una manera similar a los complementos de un navegador web normal. Cada capa agrega información y complejidad a la realidad aumentada. Cuando nos conectamos a Layar se debe elegir una capa a la que conectarse.

Una vez seleccionada la capa, la aplicación Layar obtiene la posición GPS del usuario y hace una petición de POI enviando al servidor de Layar diferentes campos, como el nombre de capa, posición GPS, etc. Cuando Layar recibe dicha información, comprueba si existe esa capa en su base de datos y de ser así, la base de datos (BD) de Layar ya tendrá asociada una URL de una página del autor que hace de proxy entre el servidor Layar y la BD del autor. Se reenvía la petición a dicha URL y ésta realiza la consulta en la BD que dispone y devuelve los puntos que están en el rango. Una vez recibidos los puntos la aplicación se encarga de posicionarlos en la pantalla según la orientación del usuario.

2.6.3. Arquitectura de Layar

Para el diseño de una capa es muy importante entender la arquitectura de la plataforma Layar. El usuario final no se dará cuenta al ejecutarla, pero detrás de ella existe toda una compleja estructura.

Se puede dividir el esquema de funcionamiento en tres grandes bloques: el navegador instalado en el dispositivo móvil del usuario, el servidor de Layar y el servidor que contiene la capa que estamos utilizando (*"Hosting"*). En la siguiente imagen se ven reflejados los tres bloques (**Figura 2.3**).

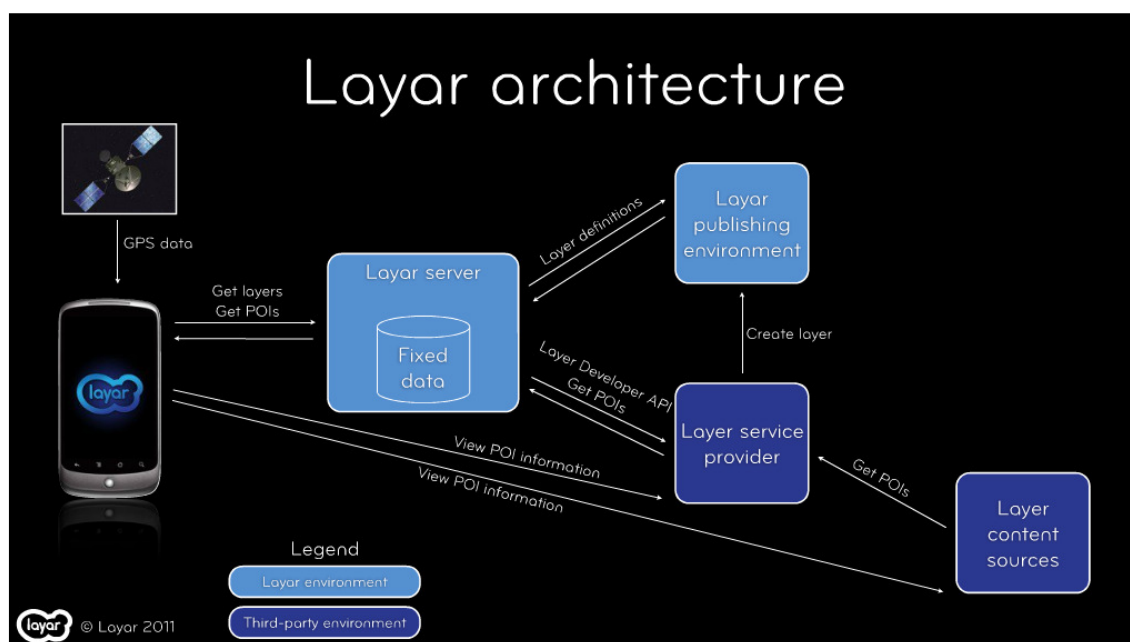


Figura 2.3 Arquitectura de Layar

Tal y como se ha indicado en el capítulo anterior, el tipo de Realidad Aumentada en que se centra el presente proyecto es la Realidad Aumentada Móvil, por lo que el primer gran bloque está formado por el dispositivo móvil y el

software instalado en dicho dispositivo. Layar dispone de un explorador ("*Browser*") que el usuario instala en su móvil. Además, el móvil debe disponer de la cámara, el GPS y demás elementos que se comentaron como indispensables al explicar este tipo de RA.

Como segundo gran bloque tenemos los servidores de Layar. El cliente entrará en el "*Browser*" de Layar y seleccionará la capa. Es en este momento cuando Layar recibe lo que el usuario está pidiendo y detecta en su Base de Datos si dicha capa existe. En caso de que exista, Layar redireccionará la petición al tercer gran bloque, comunicándole a éste cuál es la posición GPS y otros datos que puedan ser necesarios.

El tercer bloque está formado por un servidor Web y un servidor de Base de Datos. El primer servidor recibirá la petición que le está enviando Layar y realizará una consulta en la Base de Datos. Finalmente devolverá la respuesta en formato JSON [7], que será recibida en el aplicativo Layar instalado en el móvil del usuario. Esta respuesta son los llamados Puntos de Interés (POIs), que serán printados, es decir, mostrados en pantalla como objetos o en forma de lista, a elección del usuario.

Cabe señalar en este punto que la visualización de los POIs es la ventaja más importante de haber utilizado una aplicación existente, tal y como se explicó en el apartado de comparativa de aplicaciones. Esto es debido al hecho de que Layar ya dispone de elementos creados pero que deja configurar.

Centrándonos en el aplicativo que se pretende crear en el proyecto, la arquitectura que se seguiría sería la siguiente:

1. El usuario se conecta a Layar en su móvil y selecciona la capa "*Friends*".
2. Layar busca en su Base de Datos y detecta que ésta capa existe.
3. Layar redirige la petición a la url creada (<http://147.83.118.125/Friends.php>) y el servidor web hace la consulta en la Base de Datos diseñada.
4. A modo de respuesta se enviará al usuario aquellos amigos que están logados y situados dentro de una determinada distancia.

CAPÍTULO 3. DISEÑO DE LA APLICACIÓN

La aplicación diseñada debe permitir la localización de amigos situados en un radio de distancia cercano a la localización GPS del usuario que la ejecuta. Para que esto sea posible debe programarse un aplicativo que tenga algunas funciones que se incluyen en las redes sociales actuales, como por ejemplo, la opción de ver y editar mi perfil, la opción de añadir amigos o incluso de denegarlos.

Además, se pretende que sólo sea posible la visualización de aquellos amigos en el momento que ellos deseen ser vistos, es decir, debe existir una opción que permita al usuario ocultarse.

Por último, se ha fijado como objetivo que el usuario pueda interactuar con esos amigos que visualiza a través de una llamada o un correo electrónico. Se trata por tanto de una aplicación que introduce al usuario en el mundo de la Realidad Aumentada y hace posible la interacción entre amigos.

El presente capítulo está dividido en tres apartados. En la primera parte se explican las funcionalidades de la aplicación, es decir, se hace un breve resumen de las funciones que debe cumplir ésta.

En segundo lugar, se detalla el proceso de creación de una capa de Realidad Aumentada, y por último se explica la creación del *hosting*. En este último apartado podemos decir que se incluyen todos los pasos que se han llevado a cabo para la creación del aplicativo.

3.1 Funcionalidades de la aplicación

El diseño de la aplicación debe cumplir con las funcionalidades especificadas en el apartado de objetivos. La siguiente figura muestra de forma detallada todas las funcionalidades:

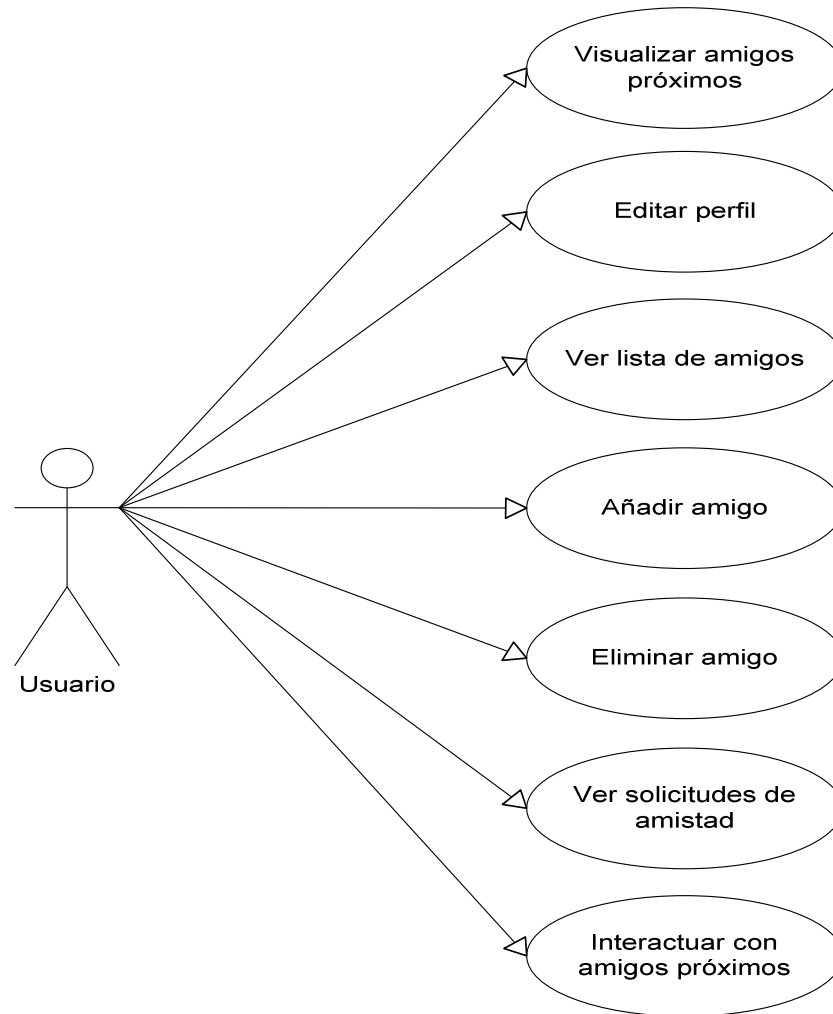


Figura 3.1 Diagrama de funcionalidades

Tal y como muestra la **Figura 3.1** se trata de una aplicación interactiva en la que existen algunas funcionalidades en las que el usuario recibe información, y otras, en las que es él el que realiza acciones.

3.2 Creación de la capa

3.2.1 Creación de una cuenta Layar

El primer paso para la creación de la capa es dar de alta una cuenta de desarrollador en Layar. Para ello se rellenarán los datos que se solicitan en la página web siguiente (**Figura 3.2**):

<https://www.layar.com/accounts/register/>

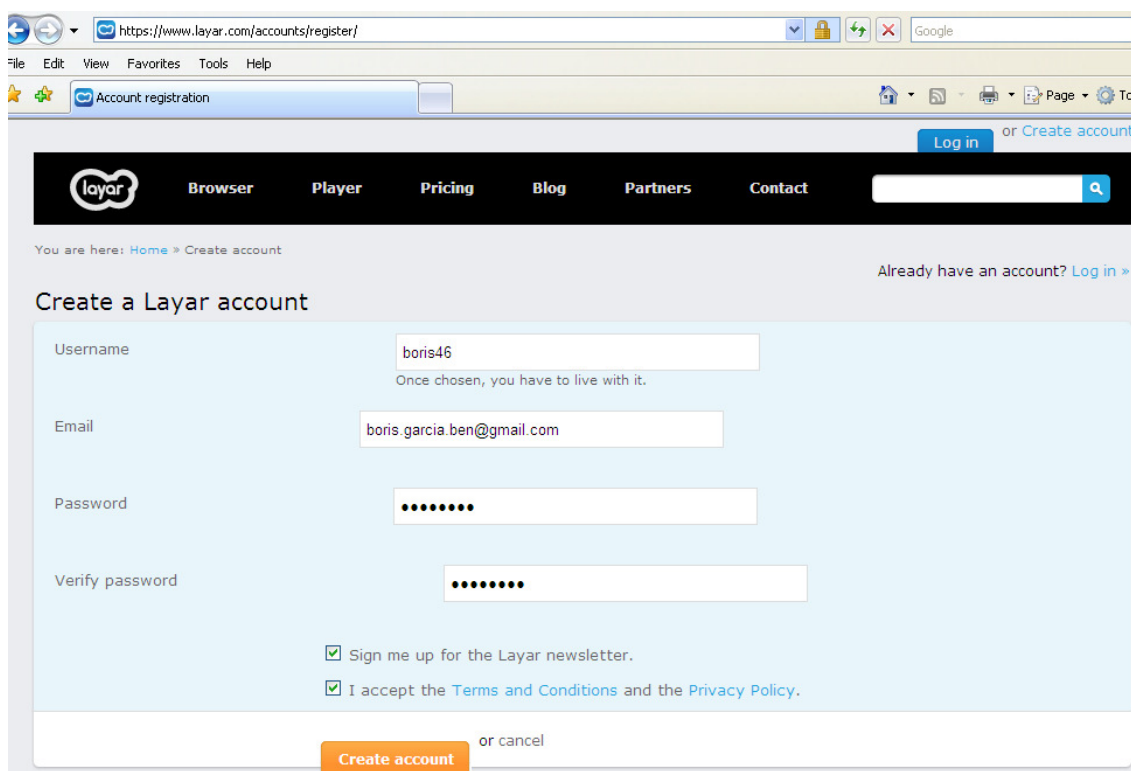
The image is a screenshot of a web browser displaying the Layar account registration page. The browser's address bar shows the URL 'https://www.layar.com/accounts/register/'. The page has a dark header with the Layar logo and navigation links: 'Browser', 'Player', 'Pricing', 'Blog', 'Partners', and 'Contact'. Below the header, there is a breadcrumb trail 'You are here: Home » Create account' and a link 'Already have an account? Log in »'. The main content area is titled 'Create a Layar account' and contains a registration form. The form has four input fields: 'Username' (containing 'boris46'), 'Email' (containing 'boris.garcia.ben@gmail.com'), 'Password' (masked with dots), and 'Verify password' (also masked with dots). Below the password fields, there are two checkboxes: 'Sign me up for the Layar newsletter.' and 'I accept the Terms and Conditions and the Privacy Policy.', both of which are checked. At the bottom of the form, there is an orange 'Create account' button and a link 'or cancel'.

Figura 3.2 Página de registro de Layar

Al cabo de unos días Layar envía un email al nuevo desarrollador indicándole que su usuario ya está activo y puede empezar a crear capas.

3.2.2. Registro de la capa

Una vez registrados, se accede a www.layar.com y se inicia sesión. Una vez iniciada la sesión, en la parte derecha superior se puede acceder a “My layers” para empezar a crear la nueva capa. Tal y como se muestra en la **Figura 3.3** aparecen las capas ya creadas y se ofrece la posibilidad de crear una nueva.

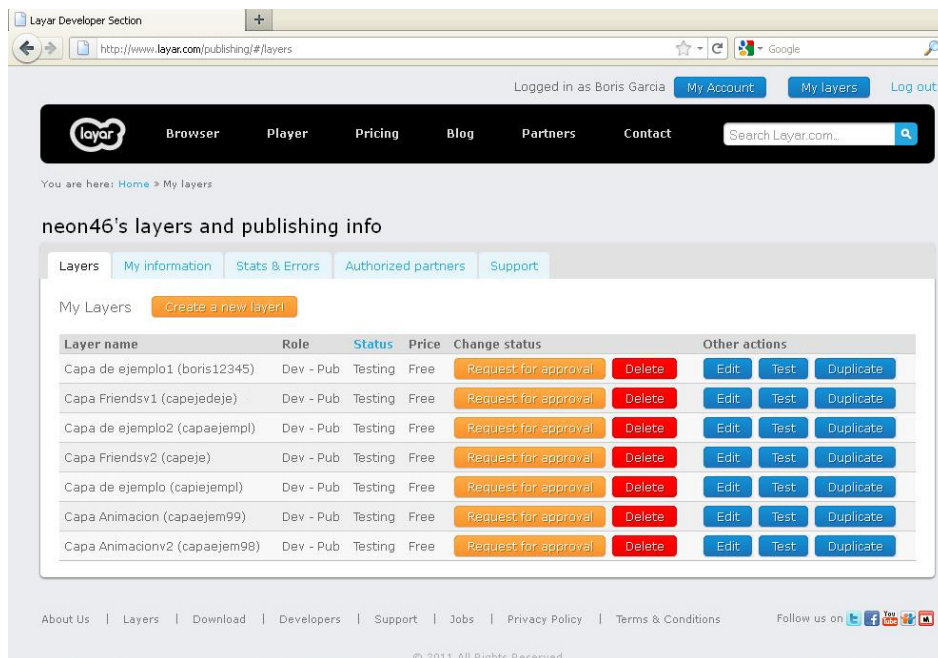


Figura 3.3 Apartado “My layers” de la página de Layar

A continuación se procede a la creación de la capa “Friends” clicando en la opción “Create a new layer!” y rellenando los campos del formulario que muestra la **Figura 3.4**:

X

Create a Layer

Layer name	<input type="text" value="friendship"/>
Title	<input type="text" value="Friends"/>
Short description	<div style="border: 1px solid #ccc; padding: 5px; min-height: 40px;">Utiliza esta capa para saber si tienes algún amigo cerca de dónde tú te encuentras.</div>
Publisher name	<input type="text" value="Boris"/>
API endpoint URL	<input type="text" value="http://147.83.118.125/Friends.php"/>
Layer type	<div style="border: 1px solid #ccc; padding: 2px 5px;">Generic (2D) ▼</div>
Layar Vision	<input type="checkbox"/> Enable <small>Enabling this option will allow you to create a layer with Layar Vision. Read more about Layar Vision.</small>

Create layer
Cancel

Figura 3.4 Create a Layer

Este formulario se utiliza para definir y crear una capa en el sitio de publicación. Todos estos campos son obligatorios. Todos los campos se pueden modificar o editar a posteriori excepto el nombre de la capa. A continuación se muestra una tabla con la descripción de cada uno de los campos:

Campo	Descripción
Layer name	Es el nombre de la capa. No es un nombre público, es decir, no será visible para el usuario final. Se utiliza para ir a buscar información dentro de la capa y sirve de referencia para la comunicación entre el desarrollador y el personal de Layar. Éste es único.
Title	Es el título de la capa que se mostrará en la lista de capas del teléfono.
Short description	Es la descripción que se muestra en la lista de capas del teléfono.
Publisher name	Nombre del editor de la capa que se muestra en la lista de capas.
API endpoint URL	Es la dirección URL del servicio web que proporciona los puntos de interés.
Layer type	Sirve para determinar cómo el cliente debe mostrar los puntos de interés: 1=genérico, 2 = 2d y 3d objetos en el espacio 3d.
Layar visión	Al activar esta opción se permite crear una capa con <i>Layar Vision</i> . <i>Layar Vision</i> es una nueva apartado de Layar que permite la creación de capas más potentes. Este servicio es de pago.

Tabla 3.1 Campos del formulario de creación de una capa.

Una vez creada la capa es posible editarla. A continuación se procede a definir brevemente cada una de las opciones que existen:

1. *General*: dentro de esta pestaña es posible editar la información básica de la capa, como por ejemplo, el email del publicador o ver el historial de cambios de la capa.
2. *API endpoint*: en esta ficha se puede modificar la URL del servicio web que devuelve los puntos de interés.
3. *Listing and indexing*: dentro de esta opción se puede modificar el logo que se muestra al usuario, el título de la capa e incluso se puede añadir una breve descripción de la capa.
4. *Look and feel*: se puede personalizar el aspecto de la vista de cámara, es decir, colores y formas de la visualización de los puntos de interés.
5. *Coverage*: en esta opción se puede delimitar el uso de la capa a un solo país o región.

6. *Filters*: se pueden configurar ajustes de filtro de la capa.
7. *Permissions*: en esta pestaña es posible modificar el editor de la capa.
8. *Additional settings*: se pueden configurar otros ajustes, como por ejemplo el uso de cookies.
9. *Pricing*: en esta ficha se puede definir y personalizar el precio de la capa.

Para adecuar la capa a nuestras necesidades y darle un diseño óptimo, se procede a editar algunas de las opciones señaladas anteriormente. La primera pestaña que se edita es *“listing and indexing”*, en la que se añade un logo personalizado y una breve descripción de la capa, tal y como se muestra en la **Figura 3.5**:

« Back to all layers Edit Layer: **Friends** (friendship)

General
API endpoint
Listing & indexing
Look & feel
Coverage
Filters
Permissions
Additional settings
Pricing

Layer Icon [Click here for an example](#)

Icon (128×128) [Replace](#)

Icon (96×96) [Replace](#)

Icon (64×64) [Replace](#)

Layer Screenshot

Image (480×320) [Upload](#) optional

Description and details

Title: Friends

Category: -----

Short description: Capa de amigos

Detail description: Utiliza esta capa para saber si tienes algún amigo cerca de dónde tú te encuentras.

Tags: amigos, friends

Figura 3.5 Icono de la capa Friends.

Además, es necesario seleccionar la versión 4.0 de Layar, ya que en la aplicación se desarrollan una serie de funcionalidades que requieren de esta versión.

Otra de las pestañas a personalizar es *“Look and feel”*, donde se modifican los colores. Inicialmente son todos en tonos grises y negros y se han cambiado por otros en tonos más vivos para dar una apariencia más juvenil a la capa. En la **Figura 3.6** se muestran dos capturas de pantalla, una primera donde se establecen los colores deseados y otra donde se puede apreciar el resultado:

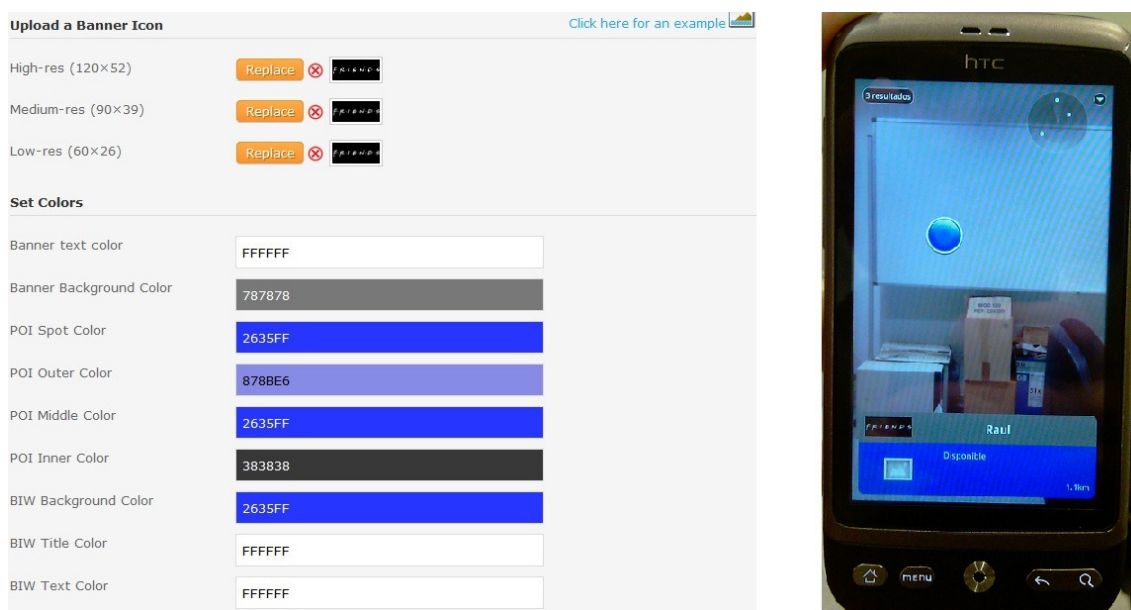


Figura 3.6 Personalización del Look & feel

3.3 Creación del hosting

3.3.1 Configuración del servidor

Como se ha explicado en el apartado de arquitectura de Layar, para poder ofrecer el servicio de una capa se necesita tener una página web accesible desde Internet que hará de proxy con las bases de datos.

Para la creación de este servidor web se ha utilizado un ordenador del laboratorio de investigación de la EPSC. Dicho ordenador dispone del sistema operativo de software libre Ubuntu [9]. A continuación, se exponen los pasos del tutorial [10] que se ha seguido para la configuración del servidor:

1. El primer requisito es disponer de un usuario con privilegios para poder instalar los paquetes necesarios.

2. Para hacer funcionar el servidor en el sistema se debe instalar el paquete **apache2**. Para hacerlo desde una consola se debe escribir:

```
$sudo apt-get install apache2
```

3. Instalar phpmyadmin y habilitar PHP para el servidor. Para que el servidor soporte páginas con formato php se debe instalar también el paquete php:

```
sudo aptitude install php5
```

A continuación se ha de resetear apache2 con el siguiente comando:

```
sudo /etc/init.d/apache2 restart
```

Además, se han de instalar algunos paquetes:

```
sudo aptitude install mysql-server  
sudo aptitude install libapache2-mod-auth-mysql  
sudo aptitude install php5-mysql
```

El siguiente paso es establecer una contraseña a phpmyadmin e instalarlo:

```
sudo mysqladmin -u root password contraseña  
sudo aptitude install phpmyadmin
```

Para comprobar el correcto funcionamiento se puede crear un fichero:

```
sudo gedit /var/www/index.php
```

En el fichero se escribe lo siguiente:

```
<?php phpinfo(); ?>
```


Se debe guardar el fichero editado y entrar en la dirección `http://localhost/index.php` en tu navegador. Se debería visualizar la información de php.

Se resetea apache para establecer los cambios:

```
sudo /etc/init.d/apache2 restart
```

Y finalmente se abre el navegador y se escribe `http://localhost/phpmyadmin` para poder administrar mysql.

3.3.2 Creación de la Base de Datos (BBDD)

En este apartado se expone la estructura de la Base de Datos, con la finalidad de dar una visión global y sencilla de entender de cómo funciona ésta. Además, facilitará la comprensión de cómo se van a programar las páginas web, apartado siguiente de la presente memoria.

Para configurar la Base de Datos, lo primero que se ha de hacer es acceder “`http://localhost/phpmyadmin`” como administrador. En la siguiente figura se muestra la página principal de MySQL.

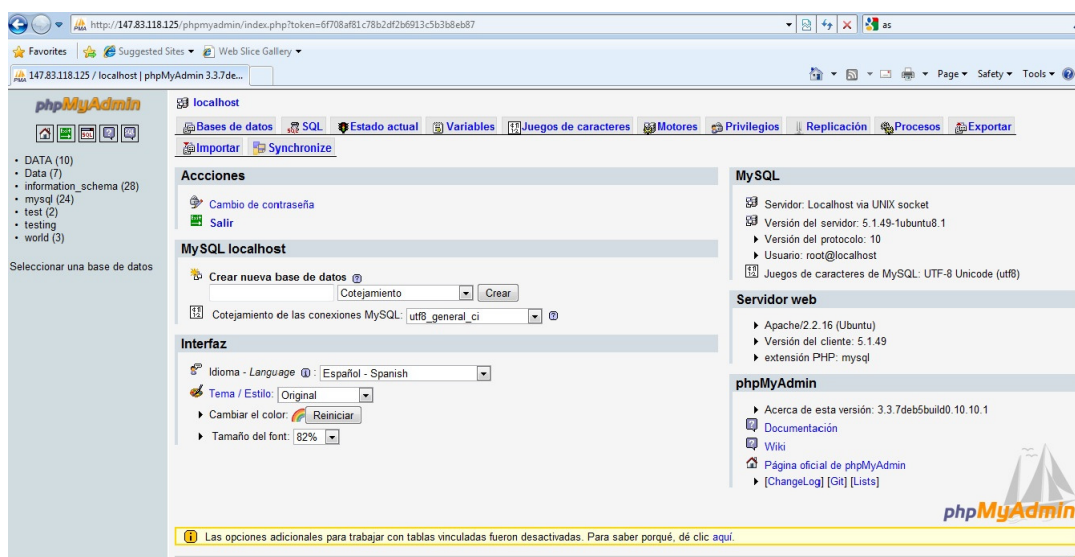


Figura 3.7 Página principal de MySQL

El primer paso es crear una Base de Datos y otorgarle un nombre, en nuestro caso se llamará “Data”. A continuación, se han de crear las tablas que la componen; la **Figura 3.8** muestra el diagrama de dicha composición.

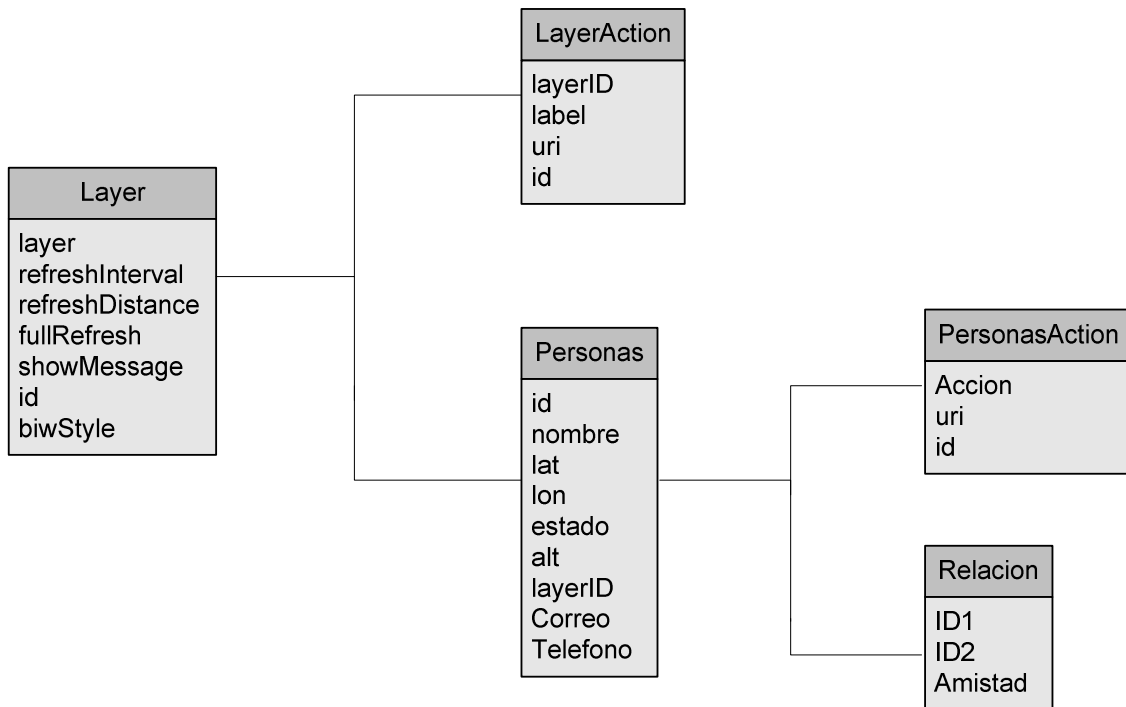


Figura 3.8 Diagrama de base de datos

Tal y como se observa en la figura, la base de datos está compuesta de cinco tablas. La primera de ellas se llama “**Layer**” y contiene la información básica de la capa. A continuación se describen cada uno de los campos que la forman:

- **Layer**: contiene el nombre único de la capa. En nuestro caso, “Friendship”.
- **RefreshInterval**: define cada cuantos segundos se refresca la información de los amigos que se encuentran próximos. Se ha establecido este valor en 10 segundos, que es el valor mínimo posible.
- **RefreshDistance**: establece un refresco obligatorio si existe algún movimiento, por parte del usuario que está ejecutando la aplicación, mayor a la distancia especificada. Se ha fijado 20 metros.
- **FullRefresh**: se trata de un operador booleano que da la posibilidad de que cuando se refresque se añada la información nueva a la anterior, o por el contrario, se sustituya la información nueva por la anterior. En la aplicación se ha escogido la segunda opción, esto es, sustituir la nueva información por la anterior.
- **ShowMessage**: este campo sirve para mostrar un mensaje como *pop-up* en la parte superior de la vista. Por defecto se establece como NULL pero está disponible por si es necesaria su utilización.
- **Id**: es el identificador de la capa, que se utiliza para generar congruencia en las consultas a la base de datos.

- BiwStyle: define el estilo de los puntos de interés. Se ha dejado el establecido por defecto, que es el clásico.

Dentro de la capa se definen dos tablas, “**LayerAction**” y “**Personas**”. La primera contiene todas las acciones que se pueden ejecutar dentro de la capa, así como los campos necesarios para ejecutarlas. Estos campos son los siguientes:

- LayerID: este campo sirve para establecer que esta acción está definida para la capa que tiene este mismo id.
- Label: es el nombre que se va a mostrar al usuario en el botón de la acción. Por ejemplo: Mi perfil.
- Uri: indica la url que se buscará cuando se clique sobre la acción.
- Id: es el identificador de la acción de la capa, que le identifica como único.

Por otro lado, y tal y como se ha dicho anteriormente, en esta tabla se definen las acciones de la capa. La siguiente tabla muestra las 5 acciones que se han creado:
















←T→	layerID	label	uri	id
  	1	Mi Perfil	http://147.83.118.125/Perfil.php?userId=	2
  	1	Mi estado	http://147.83.118.125/Estado.php?userId=	3
  	1	Añadir amigo	http://147.83.118.125/AddFriend.php?userId=	4
  	1	Solicitudes pendientes	http://147.83.118.125/Sol_pend.php?userId=	5
  	1	Amistades	http://147.83.118.125/Relations.php?userId=	6

Tabla 3.2 Acciones de la capa

La acción “Mi perfil” sirve para que el usuario edite su nombre, su correo electrónico y su teléfono. Además, visualiza su id de usuario para poder dárselo a otros y que éstos puedan añadirle como amigo.

“Mi estado” sirve para que el usuario pueda mostrarse a los demás como disponible, como ocupado o como no visible. Si el usuario ha definido que está ocupado, sus amigos no podrán ejecutar las acciones que se definen en la tabla “PersonasAction”. Si por el contrario, ha definido su estado como no visible, sus amigos no podrán visualizarle.

Por otro lado, existen las acciones de gestión de amigos, en las que se incluye la acción de poder añadir amigos, la acción de gestionar las solicitudes de amistad pendientes de aceptar o denegar y la acción amistades donde se pueden visualizar los amigos que tiene el usuario.

La tabla “**Personas**” está compuesta por todos los datos referentes a cada una de las personas que se encuentran en la capa. La información que se contiene es la siguiente:

- Id: es el identificador de la persona, que le identifica como único. Además, este campo se utilizará como clave para que otros usuarios puedan añadirle como amigo.
- Nombre: es el nombre que el usuario quiere mostrar en la aplicación y que será visto por sus amigos.
- Lat: es la coordenada de la latitud donde se encuentra la persona.
- Lon: es la coordenada de la longitud donde se encuentra la persona.
- Estado: el usuario puede configurar su estado como disponible, como ocupado o como no visible. Tal y como se ha dicho anteriormente, si fija el estado en ocupado no podrá recibir las acciones que se definen en la tabla “PersonasAction” y si lo fija como no visible no podrán visualizarle.
- Alt: es la coordenada de la altitud donde se encuentra la persona.
- LayerID: este campo sirve para establecer que esta persona está definida para la capa que tiene este mismo id.
- Correo: muestra el correo electrónico del usuario.
- Telefono: muestra el teléfono del usuario.

Finalmente, existen dos tablas más, “PersonasAction”, en la que se definen todas las acciones que se pueden ejecutar con un amigo y “Relacion”, donde se establecen las relaciones entre los usuarios. A continuación se definen los campos de “**PersonasAction**”:

- Accion: es el nombre que se va a mostrar al usuario en el botón de la acción. La tabla 3.3 muestra las dos posibles acciones que se han creado: llamar y enviar correo.








			accion	uri	id
			Llamar	tel:	2
			Enviar correo	mailto:	3

Tabla 3.3 Acciones con amigos

- Uri: indica la acción que va a ejecutar el dispositivo móvil cuando se clique sobre la acción llamar o enviar correo.

- Id: es el identificador de la acción, que le define como único.

Por otro lado, el significado de los campos de la tabla “**Relacion**” es el siguiente:

- ID1: es el usuario principal que está ejecutando la capa.
- ID2: es el amigo con el que está estableciendo la relación.
- Amistad: dentro de este campo se han definido tres valores: 0, 1 y 2. Cuando el valor es cero la solicitud se encuentra pendiente de contestación por parte del usuario, cuando el valor es uno se considera que la amistad ha sido aceptada y esa persona saldrá en la lista de amistades aceptadas y, finalmente, cuando el valor sea dos se considera que la amistad ha sido denegada y esa persona saldrá en dicha lista.

En el **Anexo I** se muestra la consulta para la creación de estas tablas, así como el tipo de carácter de cada campo.

3.3.3 Creación de las páginas web

Como último requisito para poder ofrecer el servicio de la capa se necesita tener páginas web accesibles desde Internet que harán de proxy con las Bases de Datos [12]. En este caso se han creado 17 páginas webs distintas, utilizando como lenguaje de programación PHP [14][15].

Estas páginas web se dividen en dos bloques diferenciados; las páginas que contienen la lógica de la capa y las páginas que contienen la lógica de las acciones de la capa. A continuación se detallan las funciones principales de cada una de ellas, así como su secuencia.

- Páginas que contienen la lógica de la capa

Este primer bloque se ha programado utilizando la estructura de los tutoriales de Layar, donde se dividen las funciones en diversas páginas web, de forma que la programación sea más legible que si se programan todas las funciones en una sola página.

- a) **Friends.php**: esta página es la principal de la capa. No tiene definida ninguna función, sino que se encarga de realizar, de forma ordenada, llamadas a las funciones establecidas en el resto de páginas definidas en este bloque. A continuación se van detallando cada una de estas funciones y las páginas que las contienen.

- b) **POI.php**: en esta página web se ha programado todo lo referente a los puntos de interés (POI). Por un lado se define la clase POI, con los parámetros que Layar necesita para mostrar los puntos de interés. Además, esta página incluye 3 funciones; `updateLocation`, `getHotspots` y `getPoiActions`. La primera función consulta en la tabla “**Personas**” de la Base de Datos si el usuario existe, y en caso de existir actualiza su posición GPS. Por el contrario, en caso de no existir, añade al usuario en la tabla “**Personas**”, introduciendo su id y su posición GPS.

La función `getHotspots` busca los amigos del usuario que cumplan dos condiciones: que se encuentren en un estado distinto a no visible y que se encuentren a una distancia inferior a la establecida en el filtro de distancia.

Por último, la función `getPoiActions` carga las acciones de la tabla “**PersonasAction**” para aquellos amigos que se encuentran con estado disponible.

- c) **Layer.php**: en esta página web se ha programado todo lo referente a la capa. Por un lado se define la clase Layer, con los parámetros que Layar necesita para cargar la capa. Además, esta página incluye 2 funciones; `getLayerDetails` y `getLayerActions`. La primera función consulta en la tabla “**Layer**” las características que se han establecido para la capa, como por ejemplo el intervalo de refresco o el id de la capa.

Por otro lado, la función `getLayerActions` carga las acciones que se han establecido en la tabla “**LayerAction**”.

- d) **Action.php**: esta página define la clase Action, con los parámetros que Layar necesita para mostrar las acciones.
- e) **CommonFuncs.php**: en esta página se han fijado algunas funciones comunes para las páginas anteriores, evitando así la repetición de código en varias páginas.
- f) **Abstract_class.php**: en esta página se define una clase abstracta para que los atributos sean heredados en clases hijas. Es la clase padre que se utiliza para las clases Action, Layer y POI.
- g) **Config.inc.php**: en esta página se establecen los parámetros de configuración para la conexión con la Base de Datos.

- Páginas que contienen la lógica de las acciones de la capa

En este segundo bloque se han programado dos páginas web para cada una de las acciones que se han explicado en la tabla “**LayerAction**” de la Base de Datos. Todas ellas reciben el parámetro id del usuario, tal y como se ha configurado en la página **Layer.php**. A continuación se describen cada una de las páginas creadas:

- a) **Perfil.php y Guardarperfil.php**: la primera página utiliza el id de usuario y busca en la tabla “**Personas**” de la Base de Datos su nombre, su teléfono y su correo electrónico. A continuación, muestra al usuario un formulario con título Perfil donde aparecen rellenos los campos que anteriormente ha buscado, es decir, ID usuario, nombre, teléfono y e-mail y le permite realizar cambios. No obstante, es necesario destacar que el id de usuario no es editable.

Además, aparece el botón guardar en la parte inferior de la página. Cuando el usuario lo clica, la página web hace una validación que consiste en comprobar que todos los campos estén rellenos correctamente. Se pueden destacar como ejemplos de validación que en el campo teléfono sólo existan números o en el e-mail exista una @. Por último, envía los datos a la página **Guardarperfil.php**.

Una vez realizados los cambios y la validación anterior, la página **Guardarperfil.php** realiza tres funciones; recoge los parámetros de la página **Perfil.php**, los actualiza en la tabla “**Personas**” e informa al usuario de que los cambios se han realizado correctamente.

- b) **Estado.php y Guardarestado.php**: la primera página utiliza el id de usuario y busca en la tabla “**Personas**” de la Base de Datos su estado. A continuación, muestra al usuario un formulario con título Estado, donde se le informa de su estado actual y otro campo en el que puede elegir de una lista desplegable su nuevo estado, eligiendo entre disponible, ocupado o no visible.

Además, aparece el botón guardar en la parte inferior de la página. Cuando el usuario lo clica, la página web envía los datos a la página **Guardarestado.php**, que realiza tres funciones; recoge los parámetros de la página **Estado.php**, actualiza el campo estado en la tabla “**Personas**” e informa al usuario de que el cambio de estado se ha realizado correctamente.

- c) **Addfriend.php y Addfriend2.php**: la primera página muestra al usuario un formulario con su id y otro campo que solicita el id del amigo a añadir. Además, aparece el botón añadir amigo en la parte inferior de la página, que al clicarlo valida que el id del amigo no esté vacío y, a continuación, envía los datos a la página **Addfriend2.php**.

La página **Addfriend2.php** realiza tres funciones; recoge los parámetros de la página **Addfriend.php**, añade la relación de amistad en la tabla “**Relacion**” de la Base de Datos e informa al usuario de que los cambios se han realizado correctamente. Si el id del amigo añadido no se encuentra en la Base de Datos de la capa el mensaje que se devuelve le informará de ello.

- d) **Sol_pend.php y Solicitud.php**: la primera página busca en la tabla “**Relacion**” si el id de usuario que hace la petición tiene solicitudes de amistad pendientes de contestar.

A continuación muestra una tabla al usuario listando todas aquellas que se encuentran pendientes de aceptar o denegar, organizándolas por filas, en las que se muestra el nombre, el id de usuario, la opción aceptar, la de denegar y el botón de guardar cambios.

El usuario puede ir aceptando o denegando cada una de las solicitudes, y al clicar el botón guardar, la página envía los datos de esa fila a la página **Solicitud.php**. Ésta página actualiza la tabla “**Relacion**” y devuelve al usuario un mensaje de que los cambios se han realizado correctamente.

Finalmente, a los tres segundos, la página **Solicitud.php** devuelve al usuario a la página **Sol_pend.php** para que éste pueda seguir aceptando o denegando amigos. Destacar que, una vez que el usuario vuelve a ver la tabla de solicitudes pendientes ésta se ha actualizado, y ya no se muestran las filas sobre las que se acaban de realizar cambios.

Añadir, que en caso de que el usuario no tenga solicitudes de amistad pendientes la página **Sol_pend.php** informa de ello al usuario.

e) Relations.php y Relations_change.php: la primera página busca en la tabla “**Relacion**” las amistades aceptadas y denegadas del usuario y muestra dos tablas, listando por un lado las amistades aceptadas y por otro las denegadas. Si no existen datos para alguna de las dos, la página muestra un mensaje diciendo que no existen amistades de ese tipo.

Cada tabla muestra el nombre del amigo, su id y una columna que permite clicar un botón para cambiar el estado de esa amistad. El usuario puede ir aceptando o denegando cada una de las amistades, y al clicar el botón la página envía los datos de esa fila a la página **Relations_change.php**. Ésta página actualiza la tabla “**Relacion**” y devuelve al usuario un mensaje de que los cambios se han realizado correctamente.

Finalmente, a los tres segundos, la página **Relations_change.php** devuelve al usuario a la página **Relations.php** para que éste pueda seguir viendo sus amigos. Destacar que, una vez que el usuario vuelve a ver las tablas de amigos éstas se habrán actualizado.

En el **Anexo II** puede verse el código completo de cada una de las páginas web.

CAPÍTULO 4. DEMOSTRACIÓN Y EVALUACIÓN

4.1 Demostración

En este apartado se realiza una demostración de las funcionalidades de la aplicación. Se incluyen fotografías de los diferentes escenarios, así como la explicación de cada una de las acciones que se pueden realizar en cada una de ellas.

4.1.1 Visualización de amigos

En el apartado de objetivos, una de las principales funcionalidades deseada era la visualización de amigos que se encuentran cerca del usuario que ejecuta la capa. Una vez programada la aplicación se puede afirmar que ésta posibilita la visualización de amigos de tres formas distintas.

Una primera forma de verlos es a través de la cámara. Esta vista consiste en un radar que muestra todos los amigos y los sitúa alrededor del usuario. En la **Figura 4.1** puede observarse esta vista.

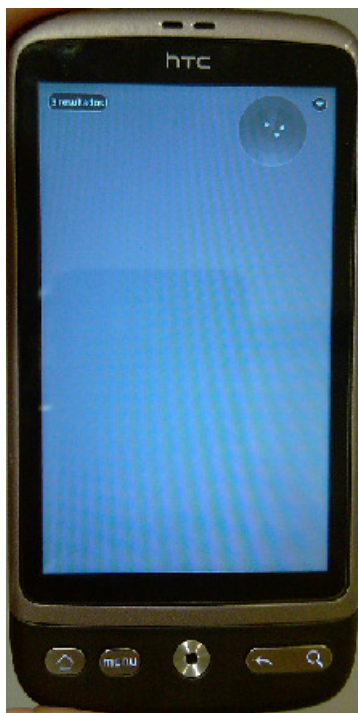


Figura 4.1 Vista general de cámara

Si se apunta hacia un punto en concreto con la cámara, la vista cambia y muestra a la persona concreta que se encuentra en ese punto, es decir, su nombre. Además, informa sobre su estado, distinguiendo entre disponible u ocupado. Es preciso recordar en este momento que si el estado de ese amigo fuera no visible, directamente no aparecería. En la **Figura 4.2** aparece una foto de la vista explicada anteriormente.

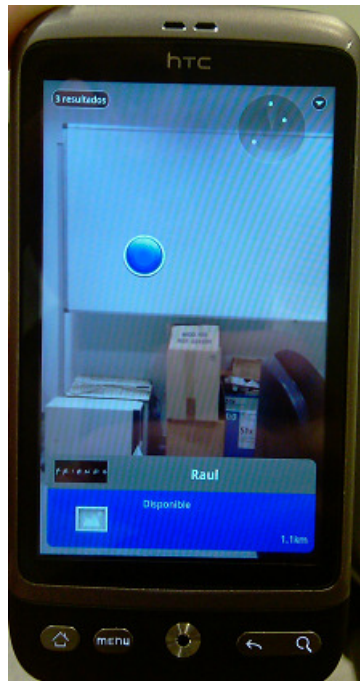


Figura 4.2 Visualización de amigos en vista de cámara

Además, gracias a la elección de Layar como software, es posible la visualización de dos formas añadidas. Por un lado, en forma de listado, en la que es posible ver el nombre del amigo, su estado y la distancia a la que se encuentra. No obstante, esta forma tiene un inconveniente, no informa sobre la dirección en la que se encuentra, es decir, no sabemos si se encuentra a 100 metros al norte o al sur. A continuación, la **Figura 4.3** ofrece una foto de esta posible vista.

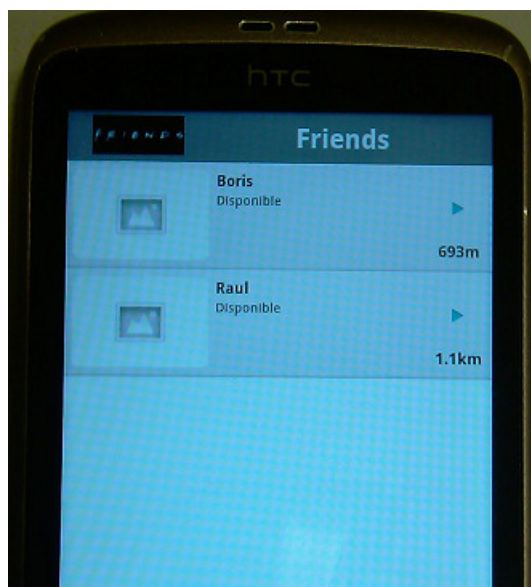


Figura 4.3 Visualización de amigos en forma de lista

Por último es posible la vista en forma de mapa sobre “Google Maps”, en la que se sitúa al usuario y a sus amigos en la calle exacta en la que se encuentran. En la **Figura 4.4** puede observarse un ejemplo de esta vista.

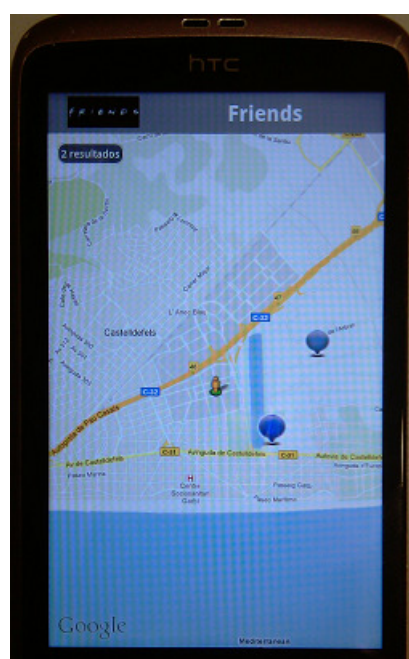


Figura 4.4 Visualización de amigos sobre mapa

Además, la aplicación conseguida es capaz de mostrar los amigos que se encuentran en un radio de distancia desde 100 metros hasta 5 kilómetros, pudiendo cambiar el rango de distancia deseada en cualquier momento desde dentro de la capa. En la siguiente figura se muestra el filtro que puede cambiar el usuario.

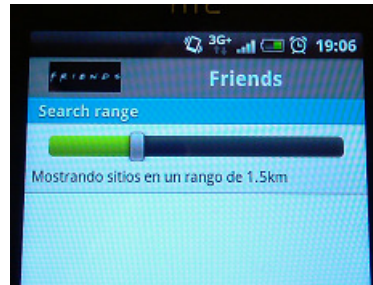


Figura 4.5 Filtro de distancia

4.1.2 Acciones en la capa

En el menú de la capa es posible realizar las acciones que se muestran en la siguiente **Figura 4.6**.

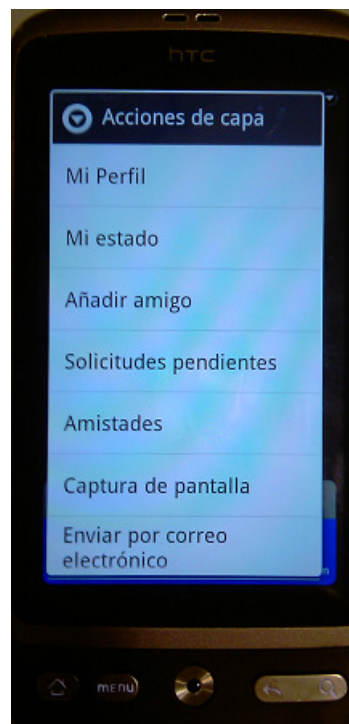


Figura 4.6 Acciones en la capa

La acción “Mi perfil” sirve para que el usuario edite su nombre, su correo electrónico y su teléfono. Además, visualiza su id de usuario para poder dárselo a otros y que éstos puedan añadirle como amigo. La Figura 4.7 muestra la página de la acción “Mi perfil”.

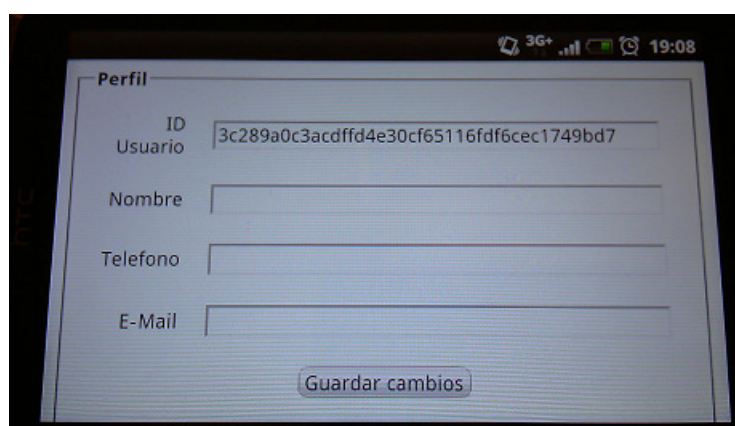


Figura 4.7 Acción “Mi perfil”

La acción “Mi estado” informa al usuario de su estado actual, además de ofrecer la posibilidad de cambiarlo a disponible, ocupado o no visible. La Figura 4.8 muestra la página de la acción “Mi estado”.

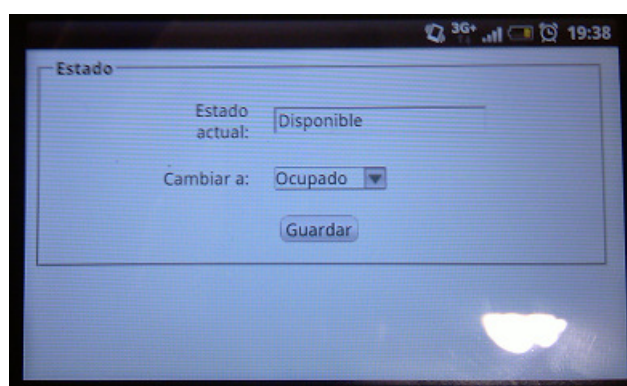


Figura 4.8 Acción “Mi estado”

Tal y como se ha comentado anteriormente en el apartado de creación de Bases de Datos, el estado disponible permite que sus amigos pueden ejecutar las acciones con amigos, mientras que el estado ocupado oculta estas acciones. La siguiente figura muestra estas diferencias.

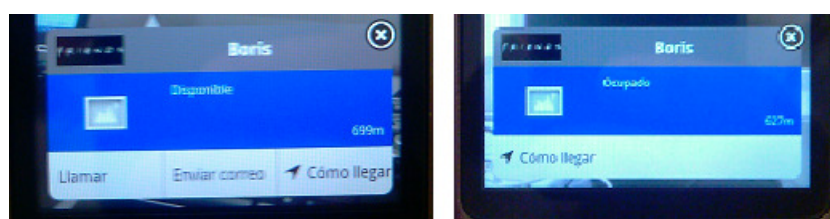


Figura 4.9 Acciones con amigos según estado

Por otro lado, existen las acciones de gestión de amigos, en las que se incluye la acción de poder añadir amigos, la acción de gestionar las solicitudes de amistad pendientes de aceptar o denegar y la acción amistades donde se pueden visualizar los amigos que tiene el usuario.

La acción “Añadir amigo” muestra una página donde se ha de introducir el id del amigo a añadir. La **Figura 4.10** muestra una foto de la página de esta acción.

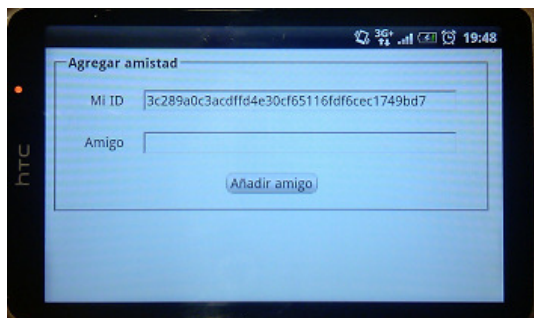


Figura 4.10 Acción “Añadir amigo”

La acción de gestionar las solicitudes de amistad permite al usuario aceptar o denegar amigos. La **Figura 4.11** un ejemplo de la página con solicitudes pendientes.

Solicitudes pendientes				
Nombre	ID	Aceptar	Denegar	Guardar
Boris	3c289a0c3acdffd4e30cf65116fdf6cec1749bd7	<input type="radio"/>	<input type="radio"/>	Guardar
Raul	123jkkhj1g231dasfadsad65a:	<input type="radio"/>	<input type="radio"/>	Guardar

Figura 4.11 Gestión de solicitudes de amistad

La acción “Amistades” permite ver al usuario dos listas; una de amigos aceptados y una de amigos denegados. Además, ofrece la posibilidad de realizar cambios entre las listas anteriores, es decir, pasar un amigo a denegado o por el contrario, aceptar a un amigo denegado. La **Figura 4.12** muestra un ejemplo de las listas citadas anteriormente.

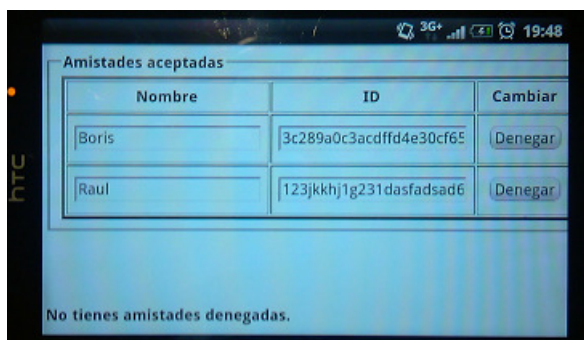


Figura 4.12 Lista de Amistades

Además, existen otras acciones no programadas específicamente para la aplicación, pero que son estándar de Layar. Éstas son: captura de pantalla, enviar por correo electrónico, compartir y agregar a favoritos.

La captura de pantalla sirve para hacer una foto de lo que se está visualizando en ese momento. Por otro lado, enviar correo electrónico y compartir sirven para comunicar a otras personas que se está usando la capa. Por último, agregar a favoritos permite tener la capa más accesible al usuario dentro del menú de Layar.

Finalmente, comentar que una vez que se realiza algún cambio en alguna de las acciones anteriores, el aplicativo muestra un mensaje de que el cambio se ha realizado correctamente o por el contrario, del error sucedido. La **Figura 4.13** muestra una captura de este mensaje.

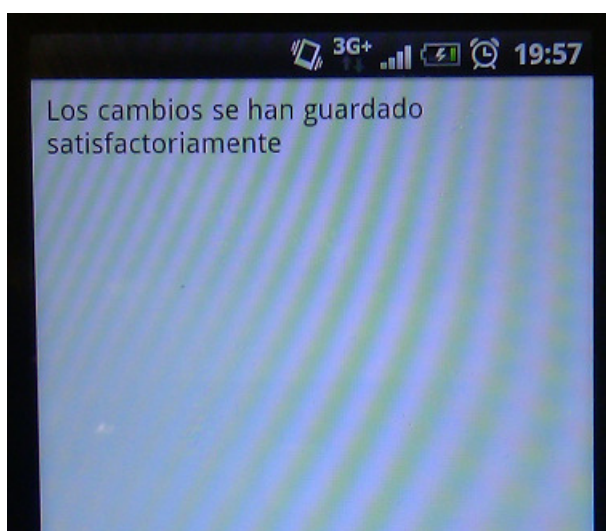


Figura 4.13 Mensaje de validación de cambios

4.1.3 Acciones con amigos

Cuando se visualiza un amigo en estado disponible la aplicación permite ejecutar dos acciones con ellos; llamar y enviar un correo electrónico. Si se pulsa sobre la opción llamar la aplicación marca el número de teléfono, de forma que este se muestra en pantalla a falta de darle al botón de llamar.

Por otro lado, si se clicla sobre la opción enviar correo electrónico la aplicación redirige hacia la aplicación que hay instalada en el dispositivo móvil de correo electrónico y escribe en el “para” el correo electrónico de ese amigo.

Por tanto, se puede afirmar que la aplicación no sólo cumple una función de visualización y localización de amigos, sino que también proporciona al usuario que está ejecutando la capa la opción de interactuar con ese amigo de una forma mucho más rápida que si tuviera que buscar su número de teléfono en la agenda o escribir el correo electrónico en la aplicación de correo.

Recordar, por último, que la ejecución de estas acciones con amigos sólo será posible si el usuario se muestra como disponible, ya que si se muestra como ocupado la aplicación no da la opción de ejecutarlas.

Finalmente añadir que, tal y como sucede en el apartado de acciones en la capa, existen otra acción no programada específicamente para la aplicación, pero que es estándar de Layar. En este caso se trata de la acción “cómo llegar”, en la que es posible recibir indicaciones de cómo llegar al lugar donde se encuentra al amigo visualizado a partir de Google Maps.

4.2 Evaluación

Este apartado pretende hacer una evaluación sobre la aplicación desde el punto de vista de su **usabilidad**, es decir, cómo de fácil es su uso para los usuarios a los que va dirigida.

La evaluación es un aspecto crítico necesario en el proceso de desarrollo de cualquier aplicación o sistema. Ésta permite crear productos con menos errores y alcanzar mejor las necesidades de los usuarios.

Existen muchos y muy diversos métodos de evaluación de usabilidad. No obstante, los más conocidos pueden clasificarse en: evaluación heurística, test de usabilidad y *focus group* [19]. Para poder desarrollar dichas evaluaciones resulta necesario tener diseñado un prototipo de la aplicación final; de esta forma se podrán extraer conclusiones a partir de su uso y análisis.

La evaluación heurística consiste en que un cierto número de especialistas juzguen si cada elemento de la interfaz de usuario cumple con un conjunto de normas aceptadas de la facilidad de uso. Este tipo de evaluación fue originalmente desarrollada por Nielsen y Molich en 1990 y posteriormente refinada por Nielsen en 1994.

Por otro lado, la técnica de test de usabilidad es una medida empírica de la usabilidad de una herramienta, sitio o aplicación, tomada a partir de la observación sistemática de usuarios llevando a cabo tareas reales. El test de usabilidad nos permitirá verificar la existencia de posibles problemas de usabilidad así como encontrar posibles soluciones para los problemas encontrados.

Los *focus group* o grupos de debate, permiten preguntar a los usuarios acerca de sus experiencias, sentimientos y preferencias respecto de la aplicación diseñada. Existen dos formas diferentes de aplicar el método. El primero consiste en pedir a los usuarios que comenten lo que sienten mientras usan la aplicación, y por el contrario, el segundo consiste en preguntar a posteriori sobre esas experiencias.

En el presente proyecto se ha llevado a cabo la evaluación a partir de las dos primeras técnicas, es decir, la evaluación heurística y la evaluación mediante test de usabilidad.

4.2.1 Evaluación heurística

Esta evaluación detecta aproximadamente el 42% de los problemas graves de diseño y el 32% de los problemas menores, dependiendo del número de evaluadores que revisen la aplicación. La teoría sobre evaluación heurística recomienda su implementación con 2 o 3 evaluadores. No obstante, en el presente proyecto se ha implementado a partir de las opiniones de un solo experto; el tutor del proyecto.

Cuando la aplicación se encontraba en desarrollo y a medida que se realizaban reuniones con él fueron surgiendo posibles retoques en las funcionalidades de la aplicación. Cabría destacar que una de las más importantes aportaciones sobre evaluación de usabilidad de la interfaz fue la necesidad de añadir a la aplicación que el usuario pudiera seleccionar un estado.

El experto detectó que debía existir un estado que permitiera al usuario la posibilidad de que sus amigos no pudieran interactuar con él a través de las “acciones con amigos” si él no lo deseaba. De esta forma, se creó el concepto estado, diferenciando entre estado disponible y estado ocupado.

Destacar como principal ventaja de este método su bajo coste. Otra ventaja es que en esta evaluación es posible interrogar a los evaluadores, profundizar en determinadas cuestiones de interés y ayudarles cuando tienen problemas. En los test de usuario por el contrario, los usuarios no deben disponer de más información que la necesaria para permitir su comportamiento espontáneo.

La principal desventaja se deriva del hecho de que el usuario final no forma parte de la evaluación. Esto significa que se ignora si los problemas identificados por los expertos son relevantes para los usuarios finales de la aplicación, es decir, la evaluación heurística, por sí sola no es capaz de confirmar si el prototipo satisface las necesidades de los usuarios.

4.2.2 Evaluación mediante pruebas de usabilidad

Este método puede hacerse de varias maneras, más o menos informal. Es posible que en ciertos casos, cuando los objetivos o los problemas a detectar parecen estar muy claros, se puede considerar que no hay necesidad de un documento de plan de pruebas formal.

El número de usuarios óptimo a utilizar en la evaluación puede variar. No obstante, las observaciones realizadas por Virzi en 1990 confirman que cinco participantes cubren el 80% de los problemas, mientras que diez participantes cubren el 90% de los problemas.

La prueba de usabilidad ha consistido en informar a los usuarios elegidos del funcionamiento básico del sistema, planteándoles a continuación una prueba del mismo. Las pruebas se han realizado con tres parejas, dos formadas por personas de unos 20 años y otra por personas de unos 40 años.

Durante la prueba se les indicó que podían ir expresando aquello que sentían mientras ejecutaban la aplicación. Posteriormente, se les pasó un test para evaluar la usabilidad de las principales funcionalidades de la aplicación.

Por tanto, las pruebas de usabilidad han consistido en sensaciones de los usuarios en el momento de ejecutar la aplicación y en la realización de un pequeño test a posteriori.

La **prueba del funcionamiento** de la aplicación ha consistido en qué un miembro de la pareja añadiera al otro como amigo y el otro a su vez aceptara dicha solicitud de amistad. Además, debían ejecutar la visualización de su amigo en las 3 vistas posibles.

Otra de las pruebas que debían realizar era cambiar su estado e interactuar entre ellos a través de las acciones con amigos en función de su estado y del estado de su amigo en este instante. Finalmente, debían denegar la amistad que tenían aceptada de un amigo y visualizar su perfil para modificar alguno de sus datos personales.

Una de las principales conclusiones que se extraen de estas pruebas es que un usuario se quejó de la dificultad de facilitar a sus amigos el id de usuario para que éstos le agregaran. La idea inicial era que el id se copiara y pegara en un correo electrónico o en un chat y se enviara al amigo. Esto no pareció ser obvio para los usuarios, sobre todo para la pareja de mayor edad.

Es por esto que el problema se plantea como posible mejora de usabilidad de la aplicación en las conclusiones del proyecto; que el id sea más corto y fácil. No obstante, ya existe la posibilidad de copiar el id manteniéndolo pulsado durante algunos segundos, aunque sería recomendable, quizás, añadir un literal al lado indicando “mantener pulsado para copiar”.

Otra de las apreciaciones de usabilidad detectada durante las pruebas fue que una de las personas valoró mucho la opción del estado no visible. Añadir que, este estado surgió como crítica durante el desarrollo del proyecto al explicar a persona conocida en qué consistía la aplicación. Ésta apreció que debía existir la posibilidad de que los amigos no vieran dónde se encontraba si él no lo deseaba en ese momento.

Como parte final de la evaluación se realizó un **test de satisfacción** con la aplicación a los usuarios. En el **Anexo III** se muestra el cuestionario completo de dicho test.

En general, las seis personas estuvieron satisfechas con el uso del aplicativo y con la facilidad de realización de las pruebas. Como punto positivo de usabilidad destacaron que era muy útil que los botones del dispositivo móvil pudieran ejecutarse dentro de la aplicación, es decir que por ejemplo, el botón de menú fuera el menú del aplicativo.

Sin embargo, en la pregunta sobre la interfaz del aplicativo dos personas comentaron que las tablas eran bastante sencillas, que les hubiera gustado que hubiera más colorido o fueran más modernas.

El tiempo de realización de las pruebas fue el correcto y estuvieron muy de acuerdo en que la información del aplicativo era fácil de encontrar y de entender. Por último, cuando se les preguntó sobre las funciones del aplicativo tres de ellos no echaron en falta ninguna funcionalidad, mientras que una de las personas jóvenes comentó que dentro de las acciones con amigos le hubiera gustado que existiera la opción de chatear con su amigo.

4.2.3 Evaluación personal

Como primer punto podría destacarse que se trata de una aplicación relacionada con las necesidades sociales actuales; cada vez más tenemos la necesidad de saber dónde se encuentran nuestros amigos y de estar en continuo contacto con ellos. Es por esto que era necesario que la aplicación estuviera al alcance de cualquiera y poder ejecutarse desde cualquier punto.

Hoy en día prácticamente todos los jóvenes y gente de mediana edad tienen un móvil con Internet por lo que desde el punto de vista de **facilidad de acceso** la aplicación es accesible.

Por otro lado, es necesario hablar de la **facilidad de visualización** de los amigos. Desde este punto de vista puede destacarse que si bien es cierto que la visualización de amigos es fácil de entender y ejecutar, la vista de cámara no es del todo precisa, ya que sabemos que ese amigo está cerca pero no exactamente dónde está. Sin embargo, cuando visualizamos a los amigos en la opción de mapa, ejecutando Google Maps, sí que se pueda saber exactamente en qué calle se encuentra esa persona.

Además, otro punto negativo de la vista de cámara es que si varios amigos se encuentran en la misma dirección, es prácticamente imposible diferenciar entre el indicador de uno u otro amigo para seleccionarlo. En este caso, la visualización en forma de lista es mucho más útil.

Por último, sería necesario hablar de la **rapidez en el acceso a la capa** y en el **tiempo que tardan en cargarse los puntos de interés** cuando el usuario está ejecutando la aplicación. En ese sentido, se puede afirmar que los procesos son muy rápidos, tardando sólo alrededor de un segundo en cargarse. No obstante, la respuesta también va a depender de las características del acceso a Internet que tenga el dispositivo móvil del usuario, no dependiendo directamente su resultado de la aplicación diseñada.

Se trata por tanto de una aplicación fácil de usar, al alcance de todos en la actualidad y que resulta útil en el momento social en el que nos encontramos, donde las redes sociales cada vez más van adquiriendo importancia en todos los ámbitos, incluso en campañas políticas.

CAPÍTULO 5. CONCLUSIONES

5.1 Conclusiones generales

Llegados a este punto se hace necesario mirar atrás y recordar cuáles eran los objetivos fijados para el Proyecto Final de Carrera. En primer lugar, se pretendía el diseño de una aplicación que cambiara la percepción de la Realidad Aumentada, es decir, que ésta no se viera como un dispositivo que ofrece información al usuario sino también como una tecnología que permite introducir al usuario dentro de esta Realidad.

Por otro lado, se fijaron los objetivos relacionados con las funcionalidades de la aplicación a diseñar. Ésta debía permitir la localización de amigos situados en un radio de distancia cercano a la localización GPS del usuario que la ejecutara.

Además, se pretendía que sólo fuera posible la visualización de aquellos amigos en el momento que ellos desearan ser vistos, es decir, debía existir una opción que permitiera al usuario ocultarse.

Por último, se había fijado como objetivo que el usuario pudiera interactuar con esos amigos que visualiza a través de una llamada o un correo electrónico.

Una vez realizado el diseño de ésta y comprobado su funcionamiento, se puede afirmar que la aplicación no sólo cumple una función de visualización y localización de amigos, sino que también proporciona al usuario que está ejecutando la capa la opción de interactuar con ese amigo de una forma mucho más rápida que si tuviera que buscar su número de teléfono en la agenda o escribir el correo electrónico en la aplicación de correo.

Hasta ahora esto no era posible, ya que el mundo de la Realidad Aumentada estaba desarrollado para recibir información sobre puntos de interés estáticos en una localización GPS. Ahora, en cambio, se ha conseguido que los puntos de interés sean los amigos, encontrándose éstos en constante movimiento.

Se puede concluir por tanto, que la aplicación cumple con los objetivos deseados.

Finalmente, me gustaría añadir como opinión personal que durante la realización del presente proyecto he puesto en práctica los conocimientos de programación adquiridos a lo largo de la carrera, pero a la vez he aprendido conocimientos nuevos de Realidad Aumentada y del lenguaje de programación PHP, lenguaje que desconocía.

5.2 Posibles mejoras

Siempre existen posibles mejoras y ampliaciones de los proyectos en investigaciones realizadas. En este caso, se mencionan las siguientes:

1. Como posible mejora se plantea la opción de que el id de usuario pueda cambiarse y éste pueda introducir un nombre más corto y fácil de proporcionar a los amigos que quieren añadirle. Actualmente se trata de un id único pero demasiado largo.
2. Por otro lado, y como posible ampliación, se propone la posibilidad de que dentro de las acciones con amigos exista la opción de establecer un chat entre usuarios, tipo whatsapp o Blackberry Messenger.
3. Finalmente, sería interesante que cuando el usuario no tenga abierta la aplicación, bien porque tenga apagado el dispositivo móvil o bien porque no tenga cobertura, su estado cambiara automáticamente a no visible.

5.3 Impacto medioambiental

Todo proyecto debe tener en consideración el impacto medioambiental que genera su puesta en marcha.

En este caso concreto el impacto se materializa en un mayor consumo de energía, ya que el dispositivo móvil que ejecuta la capa gasta más batería, lo que hace necesario que el móvil deba cargarse más a menudo.

Además, es necesario un servidor funcionando las 24 horas del día, lo que también repercute en este aumento de consumo de energía.

BIBLIOGRAFÍA

- [1] Realidad Aumentada, URL: <http://es.wikipedia.org/wiki/Realidad_aumentada>
- [2] Aplicativos de Realidad Aumentada, URL: <<http://alt1040.com/2010/01/las-5-mejores-aplicaciones-de-realidad-aumentada-para-celulares>>
- [3] Wikitude, Web oficial, URL: <<http://www.wikitude.com/>>
- [4] Junaio, Web oficial, URL: <<http://www.junaio.com/>>
- [5] Layar, Web oficial, URL: <<http://www.layar.com/>>
- [6] Android Market, URL: <<https://market.android.com/>>
- [7] JSON, JavaScript Object Notation, URL: <<http://en.wikipedia.org/wiki/JSON>>
- [8] Google Maps, URL: <<http://maps.google.es/>>
- [9] How to en Ubuntu, URL: <<http://www.howtoforge.com/howtos/linux/ubuntu>>
- [10] Guía de Ubuntu, URL: <<http://www.guia-ubuntu.org/index.php>>
- [11] W3schools.com, URL: <<http://www.w3schools.com/sql/default.asp>>
- [12] Tutorial de SQL, URL: <<http://www.desarrolloweb.com>>
- [13] Manual HTML, URL: <<http://www.webestilo.com/html/>>
- [14] PHP, URL: <<http://www.php.net/>>
- [15] Manual PHP, URL: <<http://www.programacionphp.net/>>
- [16] Artículo en TIME sobre RA, URL:<<http://www.ticbeat.com/general/realidad-aumentada-tecnologia-revista-time/>>
- [17] Artículo sobre RA, URL: <<http://www.ticbeat.com/novedades/realidad-aumentada-podria-salvar-prensa-papel>>
- [18] Artículo sobre el World Mobile Congress 2011, URL: <<http://www.pcworld.com.mx/Articulos/11832.htm>>
- [19] Página de revista especializada en diseño y usabilidad, URL: <<http://www.nosolousabilidad.com> >



Escola d'Enginyeria de Telecomunicació i
Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANEXO

Título: Interacción entre usuarios en aplicativos de Realidad Aumentada

Autor: Boris García Benítez

Director: Roc Meseguer

Fecha: 10 de diciembre de 2011

ANEXO I BASE DE DATOS

```

-SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Table structure for table `Layer`
--
CREATE TABLE IF NOT EXISTS `Layer` (
  `layer` varchar(255) NOT NULL,
  `refreshInterval` int(10) DEFAULT '300',
  `refreshDistance` int(10) DEFAULT '100',
  `fullRefresh` tinyint(1) DEFAULT '1',
  `showMessage` varchar(255) DEFAULT NULL,
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `biwStyle` enum('classic','collapsed') DEFAULT 'classic',
  PRIMARY KEY (`id`),
  UNIQUE KEY `layer` (`layer`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=6 ;

--
-- Dumping data for table `Layer`
--
INSERT INTO `Layer` (`friends`, `refreshInterval`, `refreshDistance`,
  `fullRefresh`, `showMessage`, `id`, `biwStyle`) VALUES
('layername', 10, 20, 1, NULL, 1, 'classic');

--
-- Table structure for table `Personas`
--
CREATE TABLE IF NOT EXISTS `Personas` (
  `id` varchar(255) NOT NULL,
  `nombre` varchar(50) NOT NULL,
  `lat` decimal(13,10) NOT NULL,
  `lon` decimal(13,10) NOT NULL,
  `estado` varchar(11) DEFAULT NULL,
  `alt` int(10) DEFAULT NULL,
  `layerID` int(11) NOT NULL,
  `Correo` varchar(50) DEFAULT NULL,
  `Telefono` int(20) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `layerID` (`layerID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Table structure for table `Relacion`
--
CREATE TABLE IF NOT EXISTS `Relacion` (
  `ID1` varchar(255) NOT NULL,
  `ID2` varchar(255) NOT NULL,
  `Amistad` int(10) NOT NULL,
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-----
--
-- Table structure for table `LayerAction`
--

```

```

CREATE TABLE IF NOT EXISTS `LayerAction` (
  `layerID` int(11) NOT NULL,
  `label` varchar(30) NOT NULL,
  `uri` varchar(255) NOT NULL,
  `id` int(10) NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`),
KEY `layerID` (`layerID`)
)ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=3 ;

--
-- Dumping data for table `LayerAction`
--
INSERT INTO `LayerAction` (`layerID`,`label`,`uri`,`id`) VALUES
(1,'Mi Perfil','http://147.83.118.125/Perfil.php?userId=',2),
(1,'Mi estado','http://147.83.118.125/Estado.php?userId=',3),
(1,'Añadir amigo','http://147.83.118.125/Addfriend.php?userId=',4),
(1,'Solicitudes pendientes',
'http://147.83.118.125/Sol_pend.php?userId=',5),
(1,'Amistades','http://147.83.118.125/Relations.php?userId=',6);

--
--
-- Table structure for table `PersonasAction`
--
CREATE TABLE IF NOT EXISTS `PersonasAction` (
  `accion` varchar(30) NOT NULL,
  `uri` varchar(255) NOT NULL,
  `id` int(11) NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`),
)ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=7 ;
--
-- Dumping data for table `PersonasAction`
--
INSERT INTO `PersonasAction` (`accion`,`uri`,`id`) VALUES
('Llamar','tel:',2),
('Enviar correo','mailto:',3);

--
-- Constraints for dumped tables
--
--
-- Constraints for table `Personas`
--
ALTER TABLE `Personas`
ADD CONSTRAINT `Personas_ibfk_8` FOREIGN KEY (`layerID`) REFERENCES
`Layer` (`id`);
--
-- Constraints for table `LayerAction`
--
ALTER TABLE `LayerAction`
ADD CONSTRAINT `LayerAction_ibfk_1` FOREIGN KEY (`layerID`) REFERENCES
`Layer` (`id`);

```

ANEXO II PÁGINAS WEB

<Perfil.php>

```

<?php
// Include database credentials.
// database configuration.
require_once('config.inc.php');
// Include POI.php
include 'POI.php';
// Include Layer.php
include 'Layer.php';
/** Main entry point */
// Put needed parameter names from GetPOI request in an array called
$keys.
$keys = array('layerName', 'lat', 'lon', 'radius', 'userId');
// Initialize an empty associative array.
$requestParams = array();
// Call function getRequestParams()
$requestParams = getRequestParams($keys);
/* Connect to MySQL server. We use PDO which is a PHP extension to
formalise database
connection.
For more information regarding PDO, please see
http://php.net/manual/en/book.pdo.php.
*/
// Connect to predefined MySQL database.
$db = connectDb();
/* Construct the response into an associative array.*/
// Create an empty array named response.
$response = array();
// Update location of user or create user if not exist.
$response = updateLocation($db, $requestParams);
// Assign corresponding values to mandatory JSON response keys.
$response = getLayerDetails($db, $requestParams['layerName'],
$requestParams['userId']);
// Use Gethotspots() function to retrieve Friends with in the search
range.
$response['hotspots'] = getHotspots($db, $requestParams);
// if there is no Friends found, return a custom error message.
if (!$response['hotspots']) {
$response['errorCode'] = 20;
$response['errorString'] = 'No Friends found. Please adjust the
range.';
}
else {
$response['errorCode'] = 0;
$response['errorString'] = 'ok';
}
/* All data is in $response, print it into JSON format.*/
// Put the JSON representation of $response into $jsonresponse.
$jsonresponse = json_encode($response);
// Declare the correct content type in HTTP response header.
header('Content-type: application/json; charset=utf-8');
// Print out Json response.
echo $jsonresponse;
?>

```

<POI.php>

```

<?php
// Include Action.php
include_once 'Action.php';
// Define child class POI.

class POI extends Parameter {
// Define the default values of optional parameters in POI object.
static $defaults = array (
    "imageUrl" => NULL,
    "doNotIndex" => FALSE,
    "inFocus" => FALSE,
    "showSmallBiw" => TRUE,
    "showBiwOnClick" => TRUE,
    "biwStyle" => "classic",
    "text" => array(
    "status" => NULL,
    "footnote" => NULL
    ),
    "anchor" => array("geolocation" => array("alt" => NULL)),
    "icon" => array("url" => NULL, "type" => 0),
    "object" => array("reducedURL" => NULL),
    "transform" => array(
    "rotate" => array(
    "rel" => FALSE,
    "angle" => 0.0,
    "axis" => array("x" => 0.0, "y" => 0.0, "z" => 1.0)
    ),
    "translate" => array("x" => 0.0, "y" => 0.0, "z" => 0.0),
    "scale" => 1.0
    ),
    "actions" => array()
    );
} //POI
// Construct anchor object based on poiType of each POI.
// Arguments:
// rawPoi, array ; An associative array which contains a POI object.
//
// Returns:
// array ; An array which contains anchor dictionary information for a
// Geo POI.
function getAnchor($rawPoi) {
    $anchor = array();
    $anchor['geolocation']['lat'] = changetoFloat($rawPoi['lat']);
    $anchor['geolocation']['lon'] = changetoFloat($rawPoi['lon']);
    $anchor['geolocation']['alt'] = changetoInt($rawPoi['alt']);
    return $anchor;
} //getAnchor
// Construct text object information
//
// Arguments:
// rawPoi, array ; An associative array which contains a POI object.
//
// Returns:
// array ; An array which contains text dictionary information.
function getTextDetail($rawPoi) {
    $text = array();
    $text['title'] = $rawPoi['nombre'];
    $text['description'] = $rawPoi['estado'];
    // $text['footnote'] = $rawPoi['footnote'];
    return $text;
} //getTtext

// Put fetched actions for each Friend into an associative array.

```



```

//
// Arguments:
// db ; The database connection handler.
// poi ; The Friend array.
//
// Returns:
// array ; An associative array of received actions for this
POI.Otherwise,
// return an empty array.
//
function getPoiActions($db , $poi) {
// Define an empty $actionArray array.
$actionArray = array();
// A new table called 'POIAction' is created to store actions
$sql_actions = $db->prepare('
SELECT accion,
uri,
estado,
Correo,
Telefono
FROM POIAction, Personas
WHERE Personas.estado = "Disponible"
AND Personas.id=:id ');
// Binds the named parameter marker ':id' to the specified parameter
value
$sql_actions->bindParam(':id', $poi['id'], PDO::PARAM_STR);
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql_actions->execute();
// Iterator for the $actionArray array.
$count = 0;
// Fetch all the poi actions.
$actions = $sql_actions->fetchAll(PDO::FETCH_ASSOC);
/* Process the $actions result */
// if $actions array is not empty.
if ($actions) {
$actionObject = new Action();
// Put each action information into $actionArray array.
foreach ($actions as $action) {
if($action['accion']=='Llamar' && $action['Telefono']!=NULL) {
$actionObject->add('label' , $action['accion']);
$actionObject->add('uri' , $action['uri'] . $action['Telefono']);
// Assign each action to $actionArray array.
$actionArray[$count] = $actionObject->getFiltered();
// filter each action array to remove default values of optional
}///if
else if ($action['accion']=='Enviar correo' && $action['Correo']!=NULL) {
$actionObject->add('label' , $action['accion']);
$actionObject->add('uri' , $action['uri'] . $action['Correo']);
// Assign each action to $actionArray array.
$actionArray[$count] = $actionObject->getFiltered();
// filter each action array to remove default values of optional
}///else if
$count++;
}/// foreach
}///if
return $actionArray;
}///getPoiActions

// Put received Friends into an associative array. The returned values
are
// assigned to $reponse['hotspots'].
//
// Arguments:
// db ; The handler of the database.
// value , array ; An array which contains all the needed parameters
// retrieved from GetPOI request.
//
// Returns:

```

```

// array ; An array of received Friends.
//
function getHotspots( $db, $value ) {
// Define an empty $hotspots array.
$hotspots = array();
/* Create the SQL query to retrieve Friends within the 'radius'
returned from
GetPOI request.
The first 50 returned Friends are selected.
The distance is caculated based on the Haversine formula. Note: this
way of calculation is not scalable for querying large database.
*/
// Use PDO::prepare() to prepare SQL statement. This statement is used
due to
// security reasons and will help prevent general SQL injection
attacks.
// ':lat1', ':lat2', ':long', ':userId' and ':radius' are named
parameter markers for
// which real values will be substituted when the statement is
executed.

// $sql is returned as a PDO statement object.
$sql = $db->prepare('
SELECT Personas.id,
nombre,
estado,
lat,
lon,
alt,(((acos(sin((:lat1 * pi() / 180)) * sin((lat * pi() / 180)) +
cos((:lat2 * pi() / 180)) * cos((lat * pi() / 180)) *
cos((:long - lon) * pi() / 180))
) * 180 / pi()
) * 60 * 1.1515 * 1.609344 * 1000
) as distance,
Correo,
Telefono
FROM Personas, Layer, Relacion
WHERE Personas.layerID = Layer.id AND
Layer.layer = :layerName AND Personas.estado<>"No visible" AND
Relacion.ID1=Personas.id AND Relacion.ID2=:userId AND
Relacion.Amistad=1
HAVING distance < :radius
ORDER BY distance ASC
LIMIT 0, 50');

// PDOStatement::bindParam() binds the named parameter markers to the
// specified parameter values.
$sql->bindParam( ':lat1', $value['lat'], PDO::PARAM_STR );
$sql->bindParam( ':lat2', $value['lat'], PDO::PARAM_STR );
$sql->bindParam( ':long', $value['lon'], PDO::PARAM_STR );
$sql->bindParam( ':radius', $value['radius'], PDO::PARAM_INT );
$sql->bindParam( ':layerName', $value['layerName'], PDO::PARAM_STR );
$sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
// Use PDO::execute() to execute the prepared statement $sql.
$sql->execute();
// Iterator for the response array.
$i = 0;
// Use fetchAll to return an array containing all of the remaining
rows in
// the result set.
// Use PDO::FETCH_ASSOC to fetch $sql query results and return each
row as an
// array indexed by column name.
$rowsPois = $sql->fetchAll(PDO::FETCH_ASSOC);
/* Process the $pois result */
// if $rowsPois array is not empty
if ($rowsPois) {
$myPoiParameters = new POI();
// Put each POI information into $hotspots array.

```

```

foreach ( $rawPois as $rawPoi ) {
    $myPoiParameters->add('id', $rawPoi['id']);
    // Get anchor object information
    $myPoiParameters->add('anchor', getAnchor($rawPoi));
    // Get text object information
    $myPoiParameters->add('text', getTextDetail($rawPoi));
    // Get POI action array
    $myPoiParameters->add('actions' , getPoiActions($db , $rawPoi));
    // Put the filtered poi parameters into the $hotspots array.
    $hotspots[$i] = $myPoiParameters->getFiltered();
    $i++;
} //foreach
} //if
return $hotspots;
} //getHotspots
function updateLocation($db, $value) {
    // Define an empty $hotspots array.
    $hotspots = array();
    $sql = $db->prepare('
    SELECT Personas.id,
    Layer.layer
    FROM Personas, Layer
    WHERE Personas.id= :userId AND
    Personas.layerID = Layer.id AND
    Layer.layer = :layerName');
    // PDOStatement::bindParam() binds the named parameter markers to the
    // specified parameter values.
    $sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
    $sql->bindParam( ':layerName', $value['layerName'], PDO::PARAM_STR );
    // Use PDO::execute() to execute the prepared statement $sql.
    $sql->execute();
    // Use fetchAll to return an array containing all of the remaining
    rows in
    // the result set.
    // Use PDO::FETCH_ASSOC to fetch $sql query results and return each
    row as an
    // array indexed by column name.
    $rawPer = $sql->fetchAll(PDO::FETCH_ASSOC);
    if($rawPer){
        // $sql is returned as a PDO statement object.
        $sql = $db->prepare('
        UPDATE Personas
        SET lat=:lat, lon=:long
        WHERE id=:userId ');
        // Binds the named parameter
        $sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
        $sql->bindParam( ':lat', $value['lat'], PDO::PARAM_STR );
        $sql->bindParam( ':long', $value['lon'], PDO::PARAM_STR );
        // Use PDO::execute() to execute the prepared statement $sql_actions.
        $sql->execute();
    } //if
    else{
        $sql = $db->prepare('
        INSERT INTO Personas (id, nombre, lat, lon, estado, alt, layerID,
        Correo, Telefono)
        VALUES (:userId, "", :lat, :long, "Disponible", NULL, 1, NULL,
        NULL) ');
        // PDOStatement::bindParam() binds the named parameter markers to the
        // specified parameter values.
        $sql->bindParam( ':lat', $value['lat'], PDO::PARAM_STR );
        $sql->bindParam( ':long', $value['lon'], PDO::PARAM_STR );
        $sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
        // Use PDO::execute() to execute the prepared statement $sql.
        $sql->execute();
    } //else
    return $hotspots;
} //updateLocation
?>

```

<Layer.php>

```

<?php
include_once('Action.php');
// Define child class Layer.
class Layer extends Parameter {
// Define the default values of optional parameters in Layer object.
static $defaults = array (
    "refreshInterval" => 300,
    "refreshDistance" => 100,
    "fullRefresh" => TRUE,
    "showMessage" => NULL,
    "biwStyle" => "classic",
    "deletedHotspots" => array(),
    "actions" => array();
}
// Put fetched actions for this layer into an associative array.
// Arguments:
// db ; The database connection handler.
// layerName, string ; The layer name.
// userId, string ; The user Id.
//
// Returns:
// array ; An associative array of received actions for this layer.
// Otherwise, return an empty array.
//
function getLayerActions($db , $layerName, $userId) {
// Define an empty $actionArray array.
$actionArray = array();
// A new table called 'LayerAction' is created to store actions, each
action
// has a field called 'layerName' which indicates the layer that this
action
// belongs to. The SQL statement returns actions which have the same
// layerName as $layerName.
$sql_actions = $db->prepare('
SELECT label,
uri
FROM LayerAction , Layer
WHERE layerID = Layer.id
AND Layer.layer = :layerName ');
// Binds the named parameter markers ':layerName' to the specified
parameter
// value '$layerName'.
$sql_actions->bindParam(':layerName', $layerName, PDO::PARAM_STR);
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql_actions->execute();
// Iterator for the $actionArray array.
$count = 0;
// Fetch all the layer actions.
$actions = $sql_actions->fetchAll(PDO::FETCH_ASSOC);
/* Process the $actions result */
// if $actions array is not empty.
if ($actions) {
$actionObject = new Action();
// Put each action information into $actionArray array.
foreach ($actions as $action) {
$actionObject->add('label' , $action['label']);
$actionObject->add('uri' , $action['uri'] . $userId);
// Assign each action to $actionArray array.
$actionArray[$count] = $actionObject->getFiltered();
$count++;
} // foreach
} // if
return $actionArray;
} // getLayerActions

```

```

// Put retrieved layer level parameters into an associative array.
//
// Arguments:
// db ; The database handler.
// layerName, string ; The name of the layer in the getPOI request.
// userId, string ; The ID of the user in the getPOI request.
// Return:
// array ; An associative array which contains parameters defined on
// layer
// level.
function getLayerDetails($db, $layerName, $userId){
// Define an empty $layer array.
$layer = array();
// A new table called 'Layer' is created to store general layer level
// parameters.
// 'layer' is the name of this layer.
// The SQL statement returns layer which has the same name as the
// $layerName passed in getPOI request.

$sql = $db->prepare( '
SELECT layer,
refreshInterval,
refreshDistance,
fullRefresh,
showMessage,
biwStyle
FROM Layer
WHERE layer = :layerName ');
// Binds the named parameter marker ':layerName' to the specified
// parameter
// value $layerName
$sql->bindParam(':layerName', $layerName, PDO::PARAM_STR);
// Use PDO::execute() to execute the prepared statement $sql.
$sql->execute();
// Retrieve layer parameters
$layerValue = $sql->fetch(PDO::FETCH_ASSOC);

// If $layerName is not found in the database, throw an exception.
try{
if(!$layerValue)
throw new Exception('layer:' . $layerName . 'no se encuentra en la Base
de datos.');
```

```

else {
$layerDetails = new Layer();
$layerDetails->add('layer' , $layerValue['layer']);
$layerDetails->add('refreshInterval',
changoToInt($layerValue['refreshInterval']));
$layerDetails->add('refreshDistance',
changoToInt($layerValue['refreshDistance']));
-2-
D:\users\garciabs\Desktop\www\Layer.php lunes, 28 de noviembre de 2011 16:04
$layerDetails->add('fullRefresh',
changoToBool($layerValue['fullRefresh']));
$layerDetails->add('showMessage', $layerValue['showMessage']);
$layerDetails->add('biwStyle', $layerValue['biwStyle']);
// Get Layer level actions
$layerDetails->add('actions' , getLayerActions($db , $layerName,
$userId));
// Filter out optional default values
$layer = $layerDetails-> getFiltered();
}
return $layer;
}
catch(Exception $e){
echo 'Error: ' . $e->getMessage();
}
} //getlayerDetails
?>

```

<Action.php>

```

<?php
// Include abstract_class.php where the parent class 'Parameter' is
defined.
require_once('abstract_class.php');
// Include common functions defined in commonFuncs.php
include_once('commonFuncs.php');
// Define child class Action based on parent class Parameter
class Action extends Parameter {
// Define the default values of optional parameters in Action object.
static $defaults = array (
"contentType" => "application/vnd.layar.internal",
"method" => "GET",
"activityType" => NULL,
"params" => array(),
"closeBiw" => FALSE,
"showActivity" => TRUE,
"activityMessage" => NULL,
"autoTriggerRange" => NULL,
"autoTriggerOnly" => FALSE,
"autoTrigger" => FALSE
);
}
?>

```

<CommonFuncs.php>

```

<?php
/** Custom Common functions */
// Put needed getPOI request parameters and their values in an
associative array
//
// Arguments:
// array ; An array of needed parameters passed in getPOI request
//
// Returns:
// array ; An associative array which contains the request parameters
and
// their values.
function getRequestParams($keys) {
$paramsArray = array();
try {
// Retrieve parameter values using $_GET and put them in $value array
with
// parameter name as key.
foreach( $keys as $key ) {
if ( isset($_GET[$key]) )
$paramsArray[$key] = $_GET[$key];
else
throw new Exception($key .' parameter is not passed in GetPOI
request. ');
}
return $paramsArray;
}
catch(Exception $e) {
echo 'Message: ' . $e->getMessage();
}
}
//getRequestParams
// Connect to the database, configuration information is stored in
// config.inc.php file
function connectDb() {
try {

```

```

$dbconn = 'mysql:host=' . DBHOST . ';dbname=' . DBDATA ;
$db = new PDO($dbconn , DBUSER , DBPASS ,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET
NAMES utf8'));
// set the error mode to exceptions
$db->setAttribute(PDO::ATTR_ERRMODE , PDO::ERRMODE_EXCEPTION);
return $db;
} // try
catch(PDOException $e) {
error_log('message:' . $e->getMessage());
} // catch
} // connectDb
// Change a string value to integer.
//
// Arguments:
// string ; A string value.
//
// Returns:
// Int ; If the string is empty, return NULL.
//
function changetoInt($string) {
if (strlen(trim($string)) != 0)
return (int)$string;
return NULL;
} //changetoInt
// Change a string value to float
//
// Arguments:
// string ; A string value.
//
// Returns:
// float ; If the string is empty, return NULL.
//
function changetoFloat($string) {
if (strlen(trim($string)) != 0)
return (float)$string;
return NULL;
} //changetoFloat
// Convert a TinyInt value to a boolean value TRUE or FALSE
//
// Arguments:
// int value_Tinyint ; The Tinyint value (0 or 1) of a key in the
database.
//
// Returns:
// boolean ; The boolean value, return 'TRUE' when Tinyint is 1.
Return
// 'FALSE' when Tinyint is 0.
//
function changetoBool($value_Tinyint) {
if (strlen(trim($value_Tinyint)) != 0) {
if ($value_Tinyint == 0)
return FALSE;
else
return TRUE;
}
return NULL;
} //changetoBool
// Convert a string into an array.
//
// Arguments:
// string ; The input string
// separator, string ; The boundary string used to separate the input
string
//
// Returns:
// array ; An array of strings. Otherwise, return an empty array.

```

```

function changetoArray($string, $separator){
    $newArray = array();
    if($string) {
        if (substr_count($string,$separator)) {
            $newArray= array_map('trim' , explode($separator, $string));
        }//if
        else
            $newArray[0] = trim($string);
        return $newArray;
    }
    }//changetoArray
?>

```

<Abstract_class.php>

```

<?php
// An abstract class which can be inherited by its child class. It is
the parent class of
Layer, POI and Action
abstract class Parameter {
    // Defines the default values for optional parameters of the object.
    private $defaults;
    // Define the object itself.
    private $parameters;
    // Filter out optional parameters which have default values.
    private function filter(&$myParameters, $myDefaults) {
        foreach($myDefaults as $key => $value)
            if(array_key_exists($key , $myParameters)){
                if(is_array($value) && !empty($value))
                    self::filter($myParameters[$key], $value);
                if($myParameters[$key] == $value || empty($myParameters[$key]) &&
                    !is_bool(
                        $myParameters[$key]))
                    unset($myParameters[$key]);
            }
        }//filter
    // construction function
    function __construct() {
        $childDefaults = get_called_class();
        $this->defaults = $childDefaults::$defaults;
    }//__construct
    // Add a parameter to the object.
    function add($name, $value) {
        $this->parameters[$name] = $value;
    }//add
    // Get the object
    function get() {
        return $this->parameters;
    }//get
    // Get clean object with non-default optional parameters.
    function getFiltered() {
        $myParameters = $this->parameters;
        self::filter($myParameters, $this->defaults);
        return $myParameters;
    }//getFiltered
    }//Parameter
?>

```


<Config.inc.php>

```
<?php
/* Pre-define connection to the MySQL database.*/
define('DBHOST', 'localhost');
define('DBDATA', 'Data');
define('DBUSER', 'root');
define('DBPASS', 'boris476');
?>
```

<Perfil.php>

```
<html>
<head>
<Title>Mi perfil</Title>
</head>
<body>
<script type="text/javascript">
function validateForm()
{
var y=document.forms["formulario"]["nombre"].value;
var z=document.forms["formulario"]["telefono"].value;
var x=document.forms["formulario"]["correo"].value;
var atpos=x.indexOf("@");
var dotpos=x.lastIndexOf(".");
if(y==null || y=="")
{
alert("El nombre no puede estar vacío");
return false;
}
else if(z==null || z=="")
{
alert("El número de teléfono no puede estar vacío");
return false;
}
else if(isNaN(z))
{
alert("El número de teléfono es incorrecto");
return false;
}
else if(atpos<1 || dotpos<atpos+2 || dotpos+2>=x.length)
{
alert("Formato incorrecto de e-mail");
return false;
}
}
</script>
<?php
// Include database credentials.
// database configuration.
require_once('config.inc.php');
function getRequestParams($keys) {
$paramsArray = array();
try {
// Retrieve parameter values using $_GET and put them in $value array
with
// parameter name as key.
foreach( $keys as $key ) {
if ( isset($_GET[$key]) ){
```

```

$paramsArray[$key] = $_GET[$key];
}
else
throw new Exception($key . ' parameter is not passed in GetPOI
request.');
```

```

}
return $paramsArray;
}
catch(Exception $e) {
echo 'Error: ' . $e->getMessage();
}
} //getRequestParams
function connectDb() {
try {
$dbconn = 'mysql:host=' . DBHOST . ';dbname=' . DBDATA ;
$db = new PDO($dbconn , DBUSER , DBPASS ,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET
NAMES utf8')));
// set the error mode to exceptions
$db->setAttribute(PDO::ATTR_ERRMODE , PDO::ERRMODE_EXCEPTION);
return $db;
} // try
catch(PDOException $e) {
error_log('Error: ' . $e->getMessage());
} // catch
} // connectDb
function viewStatus($db, $value ) {
// Define an empty $hotspots array.
$status = array();
$sql = $db->prepare('
Select nombre,Correo, Telefono
FROM Personas
WHERE id=:userId ');
// Binds the named parameter marker ':userId' to the specified
parameter
value
$sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql->execute();
$status= $sql->fetchAll(PDO::FETCH_ASSOC);
return $status['0'];
} //viewStatus
/** Main entry point */
$keys = array('userId');
// Initialize an empty associative array.
$requestParams = array();
// Call funtion getRequestParams()
$requestParams = getRequestParams($keys);
/* Connect to MySQL server. We use PDO which is a PHP extension to
formalise database
connection.
For more information regarding PDO, please see
http://php.net/manual/en/book.pdo.php.
*/
// Connect to predefined MySQL database.
$db = connectDb();
// Create an empty array named response.
$response = array();
// Use viewStatus() to retrieve user information.
$response = viewStatus($db, $requestParams);
?>
<form name="formulario" onsubmit="return validateForm();"
action="Guardarperfil.php" method=
"POST">
<fieldset>
<legend> <b>Perfil</b></legend>

```

```

<table width="70%" border="0" align="center" cellpadding="5"
cellspacing="10">
<tr>
<td width="20%" align="right"><span class="style5">ID Usuario</span></td>
<td width="50%"><span class="style7"><span id="sprytextfield1">
<label>
<input name="userId" type="text" id="IDuser" size="40" value="<?php echo
$_GET[
'userId'] ?>" readonly="readonly" >
</label>
</td>
</tr>
<tr>
<td width="20%" align="right"><span class="style5">Nombre</span></td>
<td width="50%"><span class="style7"><span id="sprytextfield1">
<label>
<input name="nombre" type="text" id="name" value="<?php echo
$response['nombre'
] ?>" size="40">
</label>
</td>
</tr>
<tr>
<td width="20%" align="right"><span class="style5">Telefono</span></td>
<td width="50%"><span class="style7"><span id="sprytextfield4">
<label>
<input name="telefono" type="text" id="phone_business" value="<?php echo
$response['Telefono'] ?>" size="40">
</label>
</td>
</tr>
<tr>
<td width="20%" align="right"><span class="style5">E-Mail</span></td>
<td width="50%"><span class="style7"><span id="sprytextfield5">
<label>
<input name="correo" type="text" id="email" value="<?php echo $response[
'Correo'] ?>" size="40">
</td>
</tr>
</table>
<div align=center>
<input type="submit" value="Guardar cambios"/>
</div>
</fieldset>
</form>
</body>
</html>

```

<Guardarperfil.php>

```

<html>
<head>
<Title>Guardar perfil</Title>
</head>
<body>
<?php
// Include database credentials.
// database configuration.
require_once('config.inc.php');
function getRequestParams($keys) {
$paramsArray = array();
try {

```

```

// Retrieve parameter values using $_POST and put them in $value array
with
// parameter name as key.
foreach( $keys as $key ) {
if ( isset($_POST[$key]) ){
$paramsArray[$key] = $_POST[$key];
}
else
throw new Exception($key .' parameter is not passed in GetPOI
request.');
```

```

echo 'Los cambios se han guardado satisfactoriamente';
?>
</body>
</html>

```

<Estado.php>

```

<html><head>
<Title>Mi estado</Title>
</head>
<body>
<?php
// Include database credentials
// database configuration.
require_once('config.inc.php');
function getRequestParams($keys) {
$paramsArray = array();
try {
// Retrieve parameter values using $_GET and put them in $value array
with
// parameter name as key.
foreach( $keys as $key ) {
if ( isset($_GET[$key]) ){
$paramsArray[$key] = $_GET[$key];
}
else
throw new Exception($key .' parameter is not passed in GetPOI
request. ');
}
return $paramsArray;
}
catch(Exception $e) {
echo 'Error: ' . $e->getMessage();
}
}
//getRequestParams
function connectDb() {
try {
$dbconn = 'mysql:host=' . DBHOST . ';dbname=' . DBDATA ;
$db = new PDO($dbconn , DBUSER , DBPASS ,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET
NAMES utf8'));
// set the error mode to exceptions
$db->setAttribute(PDO::ATTR_ERRMODE , PDO::ERRMODE_EXCEPTION);
return $db;
}
}
catch(PDOException $e) {
error_log('Error:' . $e->getMessage());
}
}
}
}
function viewStatus($db, $value ) {
// Define an empty $status array.
$status = array();
$sql = $db->prepare('
Select Estado
FROM Personas
WHERE id=:userId ');
// Binds the named parameter marker ':userId' to the specified
parameter value
$sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql->execute();
$status= $sql->fetchAll(PDO::FETCH_ASSOC);
return $status['0']['Estado'];
}

```

```

} //viewStatus
/* Put parameter userId into an associative array named $requestParams
*/
// Put needed parameter userId in an array called $keys.
$keys = array('userId');
// Initialize an empty associative array.
$requestParams = array();
// Call funtion getRequestParams()
$requestParams = getRequestParams($keys);
/* Connect to MySQL server. We use PDO which is a PHP extension to
formalise database
connection.
For more information regarding PDO, please see
http://php.net/manual/en/book.pdo.php.
*/
// Connect to predefined MySQL database.
$db = connectDb();
// Create an empty array named response.
$response = array();
// Use viewStatus() function to know status user.
$response['status'] = viewStatus($db, $requestParams);
?>
<form name="formulario" action="Guardarestado.php" method="POST">
<fieldset>
<legend> <b>Estado</b></legend>
<table width="70%" border="0" align="center" cellpadding="5"
cellspacing="10">
<tr>
<td width="50%" align="right"><span class="style5">Estado actual:</span></td>
<td width="50%"><span class="style7"><span id="sprytextfield1">
<label>
<input name="userId" type="hidden" id="userId" readonly="readonly"
value="<?php
echo $_GET['userId'] ?>">
<input name="estado1" type="text" id="estado1" value="<?php echo $response[
'status'] ?>" readonly="readonly">
</label>
</td>
</td>
</tr>
<tr>
<td width="50%" align="right"><span class="style5">Cambiar a:</span></td>
<td width="50%"><span class="style7"><span id="sprytextfield5">
<select name="estado">
<option value="Disponible">Disponible</option>
<option value="Ocupado">Ocupado</option>
<option value="No visible">No visible</option>
</select>
</td>
</tr>
</table>
<div align="center">
<input type="submit" value="Guardar"/>
</div>
</fieldset>
</form>
</body>

```

<Guardarestado.php>

```

<html>
<head>
<Title>Guardar estado</Title>
</head>
<body>
<?php
// Include database credentials
// database configuration.
require_once('config.inc.php');
function getRequestParams($keys) {
$paramsArray = array();
try {
// Retrieve parameter values using $_POST and put them in $value array
with
// parameter name as key.
foreach( $keys as $key ) {
if ( isset($_POST[$key]) ){
$paramsArray[$key] = $_POST[$key];
}
else
throw new Exception($key .' parameter is not passed in GetPOI
request. ');
}
return $paramsArray;
}
catch(Exception $e) {
echo 'Error: ' . $e->getMessage();
}
} //getRequestParams
function connectDb() {
try {
$dbconn = 'mysql:host=' . DBHOST . ';dbname=' . DBDATA ;
$db = new PDO($dbconn , DBUSER , DBPASS ,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET
NAMES utf8'));
// set the error mode to exceptions
$db->setAttribute(PDO::ATTR_ERRMODE , PDO::ERRMODE_EXCEPTION);
return $db;
} // try
catch(PDOException $e) {
error_log('Error:' . $e->getMessage());
} // catch
} // connectDb
function updateEstado($db, $value ) {
// Define an empty $status array.
$status = array();
$sql = $db->prepare('
UPDATE Personas
SET estado=:estado
WHERE id=:userId ');
// Binds the needed parameter to the specified parameter value
$sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
$sql->bindParam( ':estado', $value['estado'], PDO::PARAM_STR );
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql->execute();
return $status;
} //updateEstado
/** Main entry point */
/* Put parameters into an associative array named $requestParams */
// Put needed parameter in an array called $keys.
$keys = array( 'userId', 'estado');
// Initialize an empty associative array.
$requestParams = array();

```

```
// Call funtion getRequestParams()
$requestParams = getRequestParams($keys);
/* Connect to MySQL server. We use PDO which is a PHP extension to
formalise database
connection.
For more information regarding PDO, please see
http://php.net/manual/en/book.pdo.php.
*/
// Connect to predefined MySQL database.
$db = connectDb();
// Create an empty array named response.
$response = array();
// Use updateEstado() to change user's status.
$response['hotspots'] = updateEstado($db, $requestParams);
echo 'Los cambios se han guardado satisfactoriamente';
?>
</body>
</html>
```

<Addfriend.php>

```
<html>
<head>
<Title>Agregar amistad</Title>
</head>
<body>
<script type="text/javascript">
function validateForm()
{
var y=document.forms["formulario"]["friendId"].value;
if(y==null || y=="")
{
alert("El ID del amigo a añadir no puede estar vacío");
return false;
}
}
</script>
<form name="formulario" onsubmit="return validateForm();"
action="AddFriend2.php" method=
"POST">
<fieldset>
<legend> <b>Agregar amistad</b></legend>
<table width="70%" border="0" align="center" cellpadding="5"
cellspacing="10">
<tr>
<td width="20%" align="right"><span class="style5">Mi ID</span></td>
<td width="50%"><span class="style7"><span id="sprytextfield1">
<label>
<input name="userId" type="text" id="IDuser" size="40" value="<?php echo
$_GET[
'userId'] ?>" readonly="readonly" >
</label>
</td>
</tr>
<tr>
<td width="20%" align="right"><span class="style5">Amigo</span></td>
<td width="50%"><span class="style7"><span id="sprytextfield1">
<label>
<input name="friendId" type="text" id="name" size="40">
</label>
</td>
</tr>
```



```

</table>
<div align=center>
<input type="submit" value="Añadir amigo"/>
</div>
</fieldset>
</form>
</body>
</html>

```

<Addfriend2.php>

```

<html>
<head>
<Title>Agrega amistad</Title>
</head>
<body>

<?php
// Include database credentials.
// database configuration.
require_once('config.inc.php');
function getRequestParams($keys) {
$paramsArray = array();
try {
// Retrieve parameter values using $_POST and put them in $value array
with
// parameter name as key.
foreach( $keys as $key ) {
if ( isset($_POST[$key]) ){
$paramsArray[$key] = $_POST[$key];
}
else
throw new Exception($key .' parameter is not passed in GetPOI
request. ');
}
return $paramsArray;
}
catch(Exception $e) {
echo 'Error: ' . $e->getMessage();
}
} //getRequestParams
function connectDb() {
try {
$dbconn = 'mysql:host=' . DBHOST . ';dbname=' . DBDATA ;
$db = new PDO($dbconn , DBUSER , DBPASS ,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET
NAMES utf8'));
// set the error mode to exceptions
$db->setAttribute(PDO::ATTR_ERRMODE , PDO::ERRMODE_EXCEPTION);
return $db;
} // try
catch(PDOException $e) {
error_log('Error:' . $e->getMessage());
} // catch
} // connectDb
function addFriend($db, $value ) {
// Define an empty $hotspots array.
$status = array();
$sql = $db->prepare('
SELECT id
FROM Personas
WHERE id=:friendId' );

```



```

// Include database configuration
require_once('config.inc.php');
function getRequestParams($keys) {
$paramsArray = array();
try {
// Retrieve parameter values using $_GET and put them in $value array
with
// parameter name as key.
foreach( $keys as $key ) {
if ( isset($_GET[$key]) ){
$paramsArray[$key] = $_GET[$key];
}
else
throw new Exception($key .' parameter is not passed in GetPOI
request. ');
}
return $paramsArray;
}
catch(Exception $e) {
echo 'Error: ' . $e->getMessage();
}
}
}
function connectDb() {
try {
$dbconn = 'mysql:host=' . DBHOST . ';dbname=' . DBDATA ;
$db = new PDO($dbconn , DBUSER , DBPASS ,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET
NAMES utf8'));
// set the error mode to exceptions
$db->setAttribute(PDO::ATTR_ERRMODE , PDO::ERRMODE_EXCEPTION);
return $db;
}
}
}
catch(PDOException $e) {
error_log('Error: ' . $e->getMessage());
}
}
}
function friendsPending($db, $value ) {
// Define an empty $hotspots array.
$status = array();
$sql = $db->prepare('
SELECT ID2, nombre
FROM Relacion, Personas
WHERE Relacion.ID1=:userId AND
Relacion.ID2=Personas.id AND Relacion.Amistad=0' );
// Binds the named parameter marker ':userId' to the specified
parameter value
$sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql->execute();
// array indexed by column name.
$rowsPers = $sql->fetchAll(PDO::FETCH_ASSOC);
echo '<br><br>';
if($rowsPers){
echo '<fieldset>';
echo '<legend> <b>Solicitudes pendientes</b></legend>';
//create the table
echo '<TABLE Border=3 CellPadding=5><TR>';
//create headsets of the table
echo '<th bgcolor=WHITE>Nombre</th><th bgcolor=White>ID</th><th
bgcolor=WHITE>Aceptar</th><th bgcolor=WHITE>Denegar</th><th
bgcolor=WHITE>Guardar</th></TR>';
$i=0;
foreach ($rowsPers as $rowPer) {
echo '<tr>';
$nombre = 'nombre'. $i;

```

```

$userId = 'userId' . $i;
echo '<form name="form" action="Solicitud.php" method="POST">';
echo '<td><input name="userId" type="hidden" id="userId"
readonly="readonly"
value="'. $value['userId'] . '>';
echo '<td><input name="nombre" type="text" id="nombre"
readonly="readonly"
value="'. $rawPer['nombre'] . '></label></td>';
echo '<td><input name="friendId" type="text" id="friendId"
readonly="readonly" value="'. $rawPer['ID2'] . '></label></td>';
echo '<td><input type="radio" name="aceptar"
value="1"/></label></td>';
echo '<td><input type="radio" name="aceptar"
value="2"/></label></td>';
echo '<td><div align=center><input type="submit"
value="Guardar"/></div></td></form></tr>';
$i++;
} //foreach
echo "</table>";
echo "</fieldset>";
} //if
else{
echo '<br><br><b>No tienes solicitudes pendientes. </b>';
} //else
return $status;
} //friendsPending
/** Main entry point */
/* Put parameters into an associative array named $requestParams */
// Put needed parameter names in an array called $keys.
$keys = array('userId');
// Initialize an empty associative array.
$requestParams = array();
// Call function getRequestParams()
$requestParams = getRequestParams($keys);
/* Connect to MySQL server. We use PDO which is a PHP extension to
formalise database
connection.*/
// Connect to predefined MySQL database.
$db = connectDb();
// Create an empty array named response.
$response = array();
// Use friendsPending() function to create a table with relations
pendings of answer.
$response['hotspots'] = friendsPending($db, $requestParams);
?>
</body></html>
<Solicitud.php>

<html>
<head>
<Title>Solicitudes pendientes</Title>
<meta http-equiv="Refresh"
content="2;url=http://147.83.118.125/Sol_pend.php?userId=<?php
echo $_POST['userId'] ?>">
</head>
<body>
<?php
// Include database credentials.
// database configuration.
require_once('config.inc.php');
function getRequestParams($keys) {
$paramsArray = array();
try {
// Retrieve parameter values using $_POST and put them in $value array
with

```

```

// parameter name as key.
foreach ( $keys as $key ) {
if ( isset($_POST[$key]) ){
$paramsArray[$key] = $_POST[$key];
}
else
throw new Exception($key .' parameter is not passed in GetPOI
request.');
```

```

}
return $paramsArray;
}
catch(Exception $e) {
echo 'Error: ' . $e->getMessage();
}
} //getRequestParams
function connectDb() {
try {
$dbconn = 'mysql:host=' . DBHOST . ';dbname=' . DBDATA ;
$db = new PDO($dbconn , DBUSER , DBPASS ,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET
NAMES utf8'));
// set the error mode to exceptions
$db->setAttribute(PDO::ATTR_ERRMODE , PDO::ERRMODE_EXCEPTION);
return $db;
} // try
catch(PDOException $e) {
error_log('Error:' . $e->getMessage());
} // catch
} // connectDb
function updateFriendship($db, $value ) {
// Define an empty $status array.
$status = array();
$sql = $db->prepare('
UPDATE Relacion
SET Amistad=:aceptar
WHERE ID1=:userId AND ID2=:friendId');
// Binds the named parameter marker ':userId' to the specified
parameter value
$sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
$sql->bindParam( ':friendId', $value['friendId'], PDO::PARAM_STR );
$sql->bindParam( ':aceptar', $value['aceptar'], PDO::PARAM_STR );
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql->execute();
return $status;
} //updateFriendship
/** Main entry point */
/* Put parameters into an associative array named $requestParams */
/* Put needed parameter names in an array called $keys.
$keys = array( 'userId', 'friendId', 'aceptar' );
// Initialize an empty associative array.
$requestParams = array();
// Call funtion getRequestParams()
$requestParams = getRequestParams($keys);
/* Connect to MySQL server. We use PDO which is a PHP extension to
formalise database
connection.
For more information regarding PDO, please see
http://php.net/manual/en/book.pdo.php.
*/
// Connect to predefined MySQL database.
$db = connectDb();
// Create an empty array named response.
$response = array();
// Use updateFriendship() function to change the relation of row
clicked.
$response['hotspots'] = updateFriendship($db, $requestParams);
echo '</br></br><b>Los cambios se han guardado satisfactoriamente</b>';

```

```
?>
</body>
</html>
```

<Relations.php>

```
<html>
<head>
<Title>Amistades</Title>
</head>
<body>
<?php
// Include database credentials. Please customize these fields with
your own
// database configuration.
require_once('config.inc.php');
function getRequestParams($keys) {
$paramsArray = array();
try {
// Retrieve parameter values using $_GET and put them in $value array
with
// parameter name as key.
foreach( $keys as $key ) {
if ( isset($_GET[$key]) ){
$paramsArray[$key] = $_GET[$key];
}
else
throw new Exception($key .' parameter is not passed in GetPOI
request.');
```

```

$status = array();
$sql = $db->prepare('
SELECT ID2, nombre
FROM Relacion, Personas
WHERE Relacion.ID1=:userId AND
Relacion.ID2=Personas.id AND Relacion.Amistad=1' );
// Binds the named parameter marker ':userId' to the specified
parameter value
// '$poiID.
$sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql->execute();
// array indexed by column name.
$rawPers = $sql->fetchAll(PDO::FETCH_ASSOC);
echo '<br><br>';
if($rawPers){
echo '<fieldset>';
echo '<legend> <b>Amistades aceptadas</b></legend>';
//create the table
echo '<table border=3 CellPadding=5><tr>';
// create the headsets of table
echo '<th bgcolor=WHITE>Nombre</th><th bgcolor=White>ID</th><th
bgcolor=WHITE>Cambiar</th></tr>';
$i=0;
foreach ($rawPers as $rawPer){
echo '<tr>';
echo '<form name="form" action="Relations_change.php" method="POST">';
echo '<td><input name="userId" type="hidden" id="userId"
readonly="readonly" value=' .
$value['userId'] . '>';
echo '<label> <input name="nombre" type="text" id="nombre"
readonly="readonly" value=' .
$rawPer['nombre'] . '></label></td>';
echo '<td><label> <input name="friendId" type="text" id="friendId"
readonly="readonly"
value=' . $rawPer['ID2'] . '></label></td>';
echo '<td><input name="aceptar" type="hidden" id="aceptar"
readonly="readonly"
value="2">';
echo '<div align=center><input type="submit"
value="Denegar"></div></td></form></tr>';
$i++;
} //foreach
echo '</table>';
echo '</fieldset>';
} //if
else{
echo '<br><br><b>No tienes amistades aceptadas. </b> </br></br>';
} //else
$sql = $db->prepare('
SELECT ID2, nombre
FROM Relacion, Personas
WHERE Relacion.ID1=:userId AND
Relacion.ID2=Personas.id AND Relacion.Amistad=2' );

// Binds the named parameter marker ':userId' to the specified
parameter value
// '$poiID.
$sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql->execute();
// array indexed by column name.
$rawPers = $sql->fetchAll(PDO::FETCH_ASSOC);
echo '<br><br>';
if($rawPers){
echo '<fieldset>';

```

```

echo '<legend> <b>Amistades denegadas</b></legend>';
//create the table
echo '<TABLE Border=3 CellPadding=5><TR>';
// create the headsets of table
echo '<th bgcolor=WHITE>Nombre</th><th bgcolor=White>ID</th><th
bgcolor=WHITE>Cambiar</th></TR>';
$i=0;

foreach ($rawPers as $rawPer) {
echo "<tr>";
echo '<form name="form" action="Relations_change.php" method="POST">';
echo '<td><input name="userId" type="hidden" id="userId"
readonly="readonly" value=' .
$value['userId'] . '>';
echo '<label> <input name="nombre" type="text" id="nombre"
readonly="readonly" value=' .
$rawPer['nombre'] . '></label></td>';
echo '<td><label> <input name="friendId" type="text" id="friendId"
readonly="readonly"
value=' . $rawPer['ID2'] . '></label></td>';
echo '<td><input name="aceptar" type="hidden" id="aceptar"
readonly="readonly"
value="1">';
echo '<div align=center><input type="submit"
value="Aceptar"/></div></td></form></tr>';
$i++;
} //foreach
echo "</table>";
echo "</fieldset>";
} //if
else {
echo '</br></br><b>No tienes amistades denegadas. </b> </br></br>';
} //else
return $status;
} //friends
/** Main entry point */
/* Put parameters into an associative array named $requestParams */
// Put needed parameter names in an array called $keys.
$keys = array('userId');
// Initialize an empty associative array.
$requestParams = array();
// Call funtion getRequestParams()
$requestParams = getRequestParams($keys);

/* Connect to MySQL server. We use PDO which is a PHP extension to
formalise database
connection.
For more information regarding PDO, please see
http://php.net/manual/en/book.pdo.php.
*/
// Connect to predefined MySQL database.
$db = connectDb();
// Create an empty array named response.
$response = array();
// Use friends() function to retrieve the list of friends.
$response['hotspots'] = friends($db, $requestParams);
?>
</body>
</html>

```

<Relations_change.php>

```
<html>
```



```

<head>
<Title>Amistades</Title>
<meta http-equiv="Refresh"
content="2;url=http://147.83.118.125/Relations.php?userId=
<?php echo $_POST['userId'] ?>">
</head>
<body>
<?php
// Include database credentials. Please customize these fields with
your own
// database configuration.
require_once('config.inc.php');
function getRequestParams($keys) {
$paramsArray = array();
try {
// Retrieve parameter values using $_POST and put them in $value array
with
// parameter name as key.
foreach( $keys as $key ) {
if ( isset($_POST[$key]) ){
$paramsArray[$key] = $_POST[$key];
}
else
throw new Exception($key .' parameter is not passed in GetPOI
request. ');
}
return $paramsArray;
}
catch(Exception $e) {
echo 'Error: ' . $e->getMessage();
}
} //getRequestParams
function connectDb() {
try {
$dbconn = 'mysql:host=' . DBHOST . ';dbname=' . DBDATA ;
$db = new PDO($dbconn , DBUSER , DBPASS ,
array(PDO::MYSQL_ATTR_INIT_COMMAND => 'SET
NAMES utf8'));
// set the error mode to exceptions
$db->setAttribute(PDO::ATTR_ERRMODE , PDO::ERRMODE_EXCEPTION);
return $db;
} // try
catch(PDOException $e) {
error_log('Error:' . $e->getMessage());
} // catch
} // connectDb

function updateFriendship($db, $value ) {
// Define an empty $status array.
$status = array();
$sql = $db->prepare('
UPDATE Relacion
SET Amistad=:acceptar
WHERE ID1=:userId AND ID2=:friendId');
// Binds the needed parameters
$sql->bindParam( ':userId', $value['userId'], PDO::PARAM_STR );
$sql->bindParam( ':friendId', $value['friendId'], PDO::PARAM_STR );
$sql->bindParam( ':acceptar', $value['acceptar'], PDO::PARAM_STR );
// Use PDO::execute() to execute the prepared statement $sql_actions.
$sql->execute();
return $status;
} //updateFriendship
/** Main entry point */
/* Put parameters into an associative array named $requestParams */
/* Put needed parameter names in an array called $keys.
$keys = array( 'userId', 'friendId', 'acceptar' );

```

```
// Initialize an empty associative array.
$requestParams = array();
// Call function getRequestParams()
$requestParams = getRequestParams($keys);

/* Connect to MySQL server. We use PDO which is a PHP extension to
formalise database
connection.
For more information regarding PDO, please see
http://php.net/manual/en/book.pdo.php.
*/
// Connect to predefined MySQL database.
$db = connectDb();
// Create an empty array named response.
$response = array();
// Use updateFriendship() function to change the relation of row
clicked.
$response['hotspots'] = updateFriendship($db, $requestParams);
echo '</br></br><b>Los cambios se han guardado satisfactoriamente</b>';
?>
</body>
</html>
```

ANEXO III. TEST DE USABILIDAD

1. En general, estoy satisfecho con lo fácil que es de utilizar el sistema.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios:

2. Yo pude finalizar las tareas que se me propusieron en la prueba.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios:

3. Me sentí cómodo utilizando esta aplicación.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios:

4. Siempre que cometí un error pude recuperar fácilmente la pantalla anterior.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios:

5. Fue fácil encontrar la información que necesitaba.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios:

6. La información proporcionada por el sistema era fácil de entender.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios:

7. La organización de la información en las pantallas era evidente.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios:

8. La interfaz (botones, pantalla, imágenes, tablas) de la aplicación era agradable.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios:

9. Este aplicativo tiene todas las funciones que esperaba que tuviera.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios:

10. En general, estoy satisfecho con el aplicativo.

DESACUERDO 1 2 3 4 5 MUY DE ACUERDO

Comentarios: