Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Aplicación Web y Móvil para el seguimiento de autobuses escolares

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Eduardo Yago Marco

Tutor: José Vicente Busquets Mataix

2014/2015

Resumen

El objetivo de este proyecto es la creación y despliegue de una plataforma web, mediante el uso del *framework* Cakephp, la cual permite gestionar un servicio de geolocalización de flotas por GPS.

Esta plataforma se centra en ofrecer, mediante *Push Notifications* o email, un servicio de alertas sobre la proximidad de buses escolares a una zona definida, es decir, a las paradas dentro de una ruta. Este servicio está pensado para informar a los padres de la llegada de sus hijos a su parada habitual antes de que ésta tenga lugar. La plataforma se compone de varios módulos PHP ya que por una parte ofrecemos la posibilidad de realizar labores de administración; p.ej., creación de usuarios, asignación de dispositivos… y por otra parte un *frontend* a los usuarios registrados en nuestro sistema.

También se ha desarrollado una aplicación móvil, utilizando para ello el framework Phonegap, tanto para recibir las notificaciones mencionadas anteriormente como para ofrecer de una forma rápida información en tiempo real sobre la localización del dispositivo GPS.

Cabe destacar que se ha utilizado un *template*, basado en Bootstrap 3 y que incluye directrices en AngularJS, para dotar de diseño a la interfaz web.

Palabras clave: Plataforma Web, Geolocalización, CakePHP, HTML, CSS, Bootstrap, JavaScript/jQuery, Mysql, Phonegap, GPS, AngularJS.

Abstract

The objective of this project is the creation and deployment of a web platform, using the CakePHP framework, which allows you to manage a fleet geolocation service by GPS.

This platform focuses on providing, through push notifications or email, an alert service on the proximity of school buses to a defined area, i.e. the stops within a route. This service is designed to inform parents of their children's arrival at his usual stop before it takes place. The platform consists of several PHP modules that on the one hand offer the possibility to perform management tasks; e.g. creating users, device mapping ... and on the other hand offer a frontend to registered users in our system.

It has also developed a mobile application, using the framework Phonegap, both to receive notifications as mentioned above and to provide fast real-time information on the location of the GPS device.

Note that we used a template, based on Bootstrap 3, which includes AngularJS directives to provide a nice design to the web interface.

Keywords: Web Platform, Geolocation, CakePHP, HTML, CSS, Bootstrap, JavaScript/jQuery, Mysql, Phonegap, GPS, AngularJS.

Índice de Contenidos

1. Introducción 10
2. Especificación de Requisitos 11
2.1 Requisitos Funcionales 11
2.1.1 Aplicación Web 11
2.1.2 Aplicación Móvil 11
2.2 Requisitos No Funcionales 12
2.2.1 Aplicación Web 12
2.2.2 Aplicación Móvil 13
2.3 Requisitos Técnicos 13
2.3.1 Aplicación Web 13
2.3.2 Aplicación Móvil 14
3. Planificación 14
4. Análisis 15
4.1 Casos de Uso 15
4.2 Diagrama de Clases 19
4.3 Diagrama de Base de Datos 19
5. Diseño 22
5.1 Diseño Aplicación Web 22
5.1.1 (MVC) Modelo-Vista-Controlador 22
5.1.2 Prototipo de las vistas 23
5.2 Diseño Aplicación Móvil 25
5.2.1 Prototipo de la aplicación móvil 25
6. Implementación 27
6.1 Dispositivo GPS 27
6.2 CakePHP 29
6.2.1 Historia y Características 29
6.2.2 Licencia 29
6.2.3 Descarga 29
6.2.4 ¿Por qué CakePHP? 30
6.3 Node.js 30
6.3.1 Historia y Características 30
6.3.2 Licencia 30
6.3.3 Descarga 30
6.3.4 ¿Por qué Node.js? 31
6.4 Phonegap 31

6.4.1 Historia y Características 31
6.4.2 Licencia 32
6.4.3 Descarga 32
6.4.4 ¿Por qué Phonegap? 32
6.5 Aplicación Web 33
6.5.1 Configuración 33
6.5.2 Modelos 34
6.5.3 Controladores 34
6.5.4 Vistas 36
6.5.5 Añadir Elemento 39
6.5.6 Editar Elemento 41
6.5.7 Borrar Elemento 42
6.5.8 Login de Usuarios y Autorización 42
6.5.9 Detalles de Funcionalidad 47
6.5.10 Recepción de Tramas 50
6.6 Aplicación Móvil 52
6.6.1 Desarrollo 52
6.6.2 Notificaciones 54
6.6.3 Autenticación y envío de parámetros. 62
6.6.4 Recepción de geolocalización 62
6.6.5 Aspecto Final 64
7. Test y Pruebas 66
7.1 Aplicación Web 66
7.2 Aplicación Móvil 67
8. Prospección de Futuro 68
9. Conclusión 68
10. Bibliografía 69

Índice de Ilustraciones

Ilustración 1: Caso de uso web: Usuario → Iniciar sesión 15
Ilustración 2: Caso de uso web: Administrador → Casos de uso 15
Ilustración 3: Caso de uso web: Administrador → Dar alta usuario 16
Ilustración 4: Caso de uso web: Administrador → Definir nueva ruta 16
Ilustración 5: Caso de uso web: Usuario estándar y alarma → Casos de uso 17
Ilustración 6: Caso de uso web: Usuario especial → Casos de uso 17
Ilustración 7: Caso de uso móvil: Usuario móvil → Iniciar sesión 18
Ilustración 8: Casos de uso móvil: Todos los usuarios → Casos de uso 18
Ilustración 9: Diagrama de clases 19
Ilustración 10: Diagrama de base de datos 1 20
Ilustración 11: Diagrama de base de datos 2 21
Ilustración 12: Esquema resumen Modelo-Vista-Controlador [5] 22
Ilustración 13: Index principal (Primera Versión) 23
Ilustración 14: Index Rutas (Primera Versión) 24
Ilustración 15: Index principal (Versión Final) 24
Ilustración 16: Index Rutas (Versión Final) 25
Ilustración 17: Aplicación Móvil (Prototipo) 26
Ilustración 18: Dispositivo GPS 27
Ilustración 19: Características accesibles en Phonegap 32
Ilustración 20: Jerarquía de carpetas de CakePHP 33
Ilustración 21: Aclaración funcionamiento de layouts 36
Ilustración 22: Ejemplo de relación layout + vista 39
Ilustración 23: Vista añadir dispositivo GPS 48
Ilustración 24: Vista añadir geozona 48
Ilustración 25: Vista añadir ruta 48
Ilustración 26: Vista añadir usuario estándar 49
Ilustración 27: Vista selección ruta y geozona del usuario 49
Ilustración 28: Jerarquía de carpetas de un proyecto Phonegap 52
Ilustración 29: Arquitectura GCM [2] 55
Ilustración 30: Arquitectura APNS [3] 57

1. Introducción

El proyecto redactado a continuación se trata de un trabajo realizado en la empresa Tecnoprotel-Elian S.L., externa a la Universidad Politécnica de Valencia, durante un periodo de 5 meses aproximadamente.

La plataforma descrita en este trabajo pretende solventar la problemática existente a la hora de efectuar la recogida, principalmente por parte de los padres, de un alumno que usa transporte escolar. Podemos resumir esta problemática en dos puntos clave:

- Esperar en la parada un tiempo indeterminado la llegada del autobús, que puede sufrir los lógicos retrasos de las rutas.
- Falta de información (y su consiguiente preocupación) sobre la llegada de los alumnos al colegio o parada.

Una vez marcados los objetivos de la plataforma se llevó a cabo la elección de tecnologías a utilizar. Estas herramientas se encuentran descritas en apartados posteriores de este trabajo.

Cabe informar de que todo el despliegue de la aplicación web se ha realizado sobre un servidor externo a la empresa y por tanto no se ha tenido acceso directo a éste, sin embargo la instalación y mantenimiento de los módulos necesarios se realizaron de manera rápida y eficaz.

Durante los meses de trabajo se realizaron en primer lugar labores de aprendizaje, ya que las tecnologías comentadas en este proyecto fueron seleccionadas por la citada empresa, siempre se realizaba una toma de contacto con ellas antes de comenzar el desarrollo de la plataforma final; posteriormente se comenzó por abordar el desarrollo de la aplicación web, dejando el desarrollo de la aplicación móvil como tarea posterior cuando la aplicación web estuviese en funcionamiento; una vez que la aplicación web, incluyendo la recepción de tramas de los dispositivos GPS, fue accesible se comenzó a realizar pruebas sobre la recepción y almacenamiento de tramas de los dispositivos GPS; al garantizar que todas las operaciones anteriores se realizaban correctamente se comenzó con el desarrollo de la aplicación móvil; por último se cambió el aspecto de la aplicación web por un diseño más moderno, atractivo y funcional para terminar de maquetar la aplicación final.

2. Especificación de Requisitos

En la siguiente sección vamos a numerar los diferentes requisitos necesarios tanto para la parte de aplicación web como para la parte de aplicación móvil del proyecto, distinguiremos entre: 1) requisitos funcionales, que son aquellos que establecen cómo debe funcionar nuestra aplicación web, 2) requisitos no funcionales "Describen aspectos del comportamiento de un sistema, capturando las propiedades y restricciones bajo las cuales un sistema debe operar" [1] y 3) requisitos técnicos, que enumeran de una manera más específica las tecnologías a usar.

2.1 Requisitos Funcionales

2.1.1 Aplicación Web

La aplicación web debe permitir realizar las funciones CRUD[1] en la base de datos y por tanto visualizar en tiempo real las modificaciones que éstas ejecutan. En nuestra aplicación distinguiremos entre tres tipos de usuarios: usuario estándar (con rol de usuario o administrador), usuario alarma y usuario especial. Ambos tipos de usuario deben estar dados de alta en nuestro servicio previamente, es decir, no se permite el acceso a la aplicación web a usuarios anónimos o usuarios sin registro.

Los usuarios estándar con rol usuario y los usuarios alarma, tras la identificación, verán en la aplicación web la localización actual del dispositivo asignado, así como las últimas 5 posiciones que se han registrado de éste, podrán ver la ruta que ha efectuado éste filtrando por día. Cabe destacar que los usuarios alarma tendrán un apartado dedicado a las posiciones que se han recibido permitiendo filtrar los datos.

Los usuarios estándar que tengan el rol de administrador tienen acceso total y por tanto esto les permite tanto añadir, editar y eliminar como visualizar todos los contenidos de la aplicación web.

Los usuarios especiales tienen la peculiaridad de que pueden tener asignados más de un dispositivo y por tanto esta característica nos obliga a añadir la funcionalidad de filtrar por dispositivo.

Un aspecto importante de la aplicación es que debemos registrar correctamente la información que recibamos de los dispositivos y almacenarla en la base de datos.

Como requisito final y al tratarse de una aplicación que podrá administrar otra persona en el futuro debemos dar importancia a realizar un diseño simple, limpio y funcional para así disminuir en la medida de lo posible los errores cometidos por el usuario final.

2.1.2 Aplicación Móvil

La aplicación móvil será una aplicación de acceso gratuito, sin embargo, únicamente podrá utilizarse si previamente hemos obtenido un usuario y contraseña válidos. Distinguiremos entre tres tipos de usuarios, que coinciden con los comentados en el apartado anterior: usuario estándar, usuario especial y usuario alarma. Cada tipo de usuario obtendrá una vista personalizada una vez identificado en la aplicación.

[1] CRUD: (Create, Read, Update and Delete) Se refiere a las operaciones básicas que pueden realizarse en una base de datos.

Los usuarios estándar, en este caso no existen diferencias entre un rol u otro, observarán en la aplicación un mapa en el cual mediante dos *markers*[2] nos muestra la situación actual del dispositivo, incluyendo en una ventana el nombre de la calle, la ciudad y la región, y la posición en la que estaba éste 10 posiciones atrás. Mediante la unión de todas las posiciones obtenidas entre la posición actual y la posición más atrasada presentaremos en el mapa la ruta que ha seguido el autobús.

Los usuarios especiales son capaces de en un mismo mapa estar observando simultáneamente varios dispositivos. El funcionamiento es similar al descrito en el usuario estándar, se generan dos *markers* por dispositivo y posteriormente con el uso de las posiciones mostramos la ruta entre ellos, en este caso no incluimos una ventana con información de la ubicación hasta que el usuario pulsa sobre el *marker*. Cabe destacar que los usuarios especiales tendrán un botón especial en la parte superior del mapa que les permitirá reiniciar el nivel del zoom de éste para así poder ver todos los dispositivos en pantalla.

Los usuarios alarma simplemente obtendrán la posición actual del dispositivo y una ventana de información con la información de esa ubicación. Al tratarse de usuarios que simplemente quieren tener constancia de la ubicación en tiempo real no se requiere realizar ningún tipo de ruta.

2.2 Requisitos No Funcionales

2.2.1 Aplicación Web

Basándonos en el esquema de Mamani [4], [6] podemos separar los requisitos no funcionales en tres grandes grupos: calidad, alcance y operación. Apoyándonos en esto podemos concluir que los requisitos no funcionales primordiales para nuestra aplicación web son:

-Calidad-

Precisión: Ya que estamos registrando información de geolocalización, tenemos que ofrecer, en la medida de lo posible, datos fiables y minimizar los errores de posición que generan los dispositivos tanto como sea posible.

Seguridad: Debemos garantizar que cada tipo de usuario disponga de sus privilegios y que, por tanto, únicamente tenga acceso a los recursos propios de ese tipo de usuario.

Rendimiento: Al ser una aplicación web que gestiona los datos recibidos por los dispositivos necesitamos que esta información sea almacenada rápidamente y así ser capaces de ofrecer un servicio en tiempo real.

Usabilidad: Como la aplicación está destinada a ser accedida por variedad de usuarios debemos ofrecer una página usable, para así reducir el tiempo de reacción y de ese modo la frustración de éstos.

-Alcance-

Transferencia de Información: Nuestra aplicación web será la encargada del envío y recepción de información a la aplicación móvil, que se describirá posteriormente, por esta razón es muy importante que esta transferencia se realice de manera correcta entre ambas partes.

[2] Markers: Marcador característico de Google Maps que identifica una ubicación en un mapa.

-Operación-

Automatización: Se realizará una operación cron[3] (funcionalidad disponible en nuestro servidor) cada minuto, por consiguiente debemos ser conscientes de que esta tarea de automatización se realice correctamente en el servidor y de que no cause un error que impida su correcto funcionamiento.

2.2.2 Aplicación Móvil

Los requisitos no funcionales fundamentales para nuestra aplicación móvil son los siguientes:

-Calidad-

Seguridad: Como comentamos anteriormente, aunque se trate de una aplicación gratuita y disponible para su descarga, únicamente serán capaces de acceder a ella los usuarios que previamente posean un nombre de usuario y una contraseña, por esto es muy importante contar con sistemas de validación de usuarios y que el envío de información entre web-móvil y móvil-web se realice de forma segura.

Interoperabilidad: Al tratarse de una aplicación móvil que basa su funcionalidad en recibir datos desde la aplicación web y mostrarlos en tiempo real, la interoperabilidad entre sistemas es un requisito fundamental.

Usabilidad: Pilar fundamental de las aplicaciones móviles, nuestra aplicación debe ser: sencilla, ya que el principal usuario de la aplicación serán adultos y no todos tienen conocimientos avanzados en aplicaciones y gestos; y rápida, tratándose de este tipo de aplicación debemos informar a los usuarios en el momento en el que se produzca cualquier cambio en los dispositivos.

Mantenimiento: Como toda aplicación móvil el objetivo es conseguir *feedback*[4] de los consumidores y por tanto ofrecer un mantenimiento constante a la aplicación y así seguir mejorando ésta basándonos en las opiniones que genera.

Portabilidad: Otro requisito muy importante ya que queremos que la aplicación se pueda ejecutar en la mayoría de terminales para abarcar el mayor número de usuarios.

-Alcance-

Transferencia de Información: Igual que en la aplicación web se produce tanto envío como recepción de información por parte del dispositivo móvil. Se debe realizar de manera correcta y sin comprometer el rendimiento de la aplicación.

2.3 Requisitos Técnicos

2.3.1 Aplicación Web

Para la correcta implementación y funcionamiento de la aplicación web necesitamos un servidor donde hospedar nuestra web, a su vez en este servidor debemos tener instaladas las herramientas que necesitaremos para realizar todas las funciones de la aplicación, estas herramientas son:

- Servidor web http
- Php
- CakePHP
- Node.js
- Motor de base de datos
- Conexión y dirección IP fija

[3] Cron: Administrador de procesos en segundo plano que ejecuta operaciones a intervalos regulares (p.ej: una vez por minuto, una vez cada cinco minutos,…).

[4] Feedback: Retroalimentación en la que, en nuestro caso, los usuarios solicitan mediante comentarios cambios en la aplicación, estos comentarios deben ser revisados para así seguir actualizando la aplicación en consecuencia.

2.3.2 Aplicación Móvil

A su vez para realizar la aplicación móvil necesitamos disponer de los siguientes elementos:

- Java JDK y JRE
- Android SDK
- Phonegap
- Ant
- Un dispositivo con OS X
- iOS SDK

3. Planificación

La planificación de este proyecto se divide en varias fases, las cuales podríamos catalogar según su contenido en: 1) apartado funcional y aspecto de ambas aplicaciones, se tratan estos aspectos en los apartados de análisis y diseño, 2) configuración y apartado técnico, que se describen en la sección de implementación, siendo ésta la fase más extensa del proyecto, 3) verificación y camino a seguir, a esta última fase se le da cobertura en los apartados test y pruebas y prospección de futuro.

Cabe destacar que cada uno de los apartados anteriormente descritos se divide en dos secciones, una destinada a la aplicación web y otra a la aplicación móvil.

4. Análisis

4.1 Casos de Uso

Partiendo de los requisitos descritos en el apartado 2 podemos formar los siguientes casos de uso:

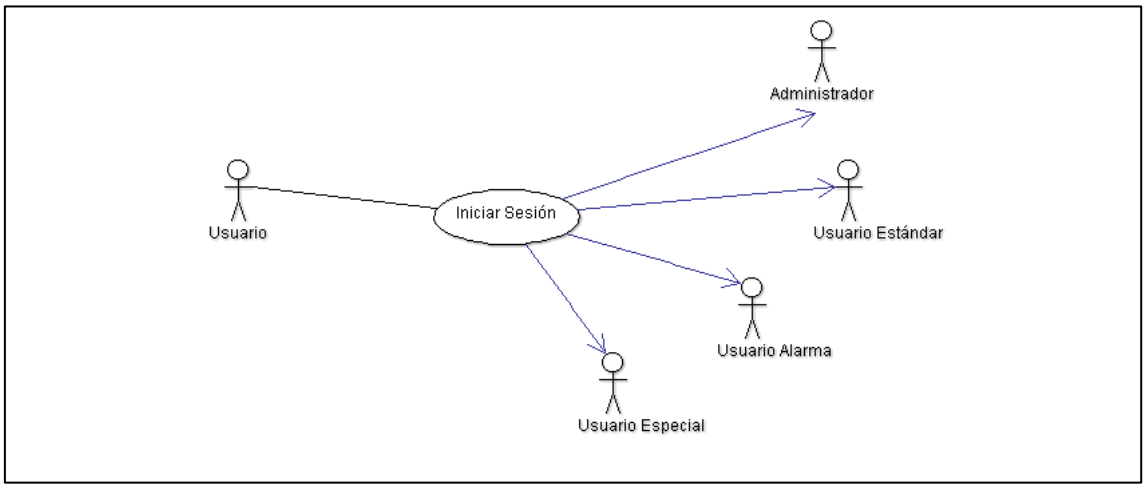


Ilustración 1: Caso de uso web: Usuario → Iniciar sesión

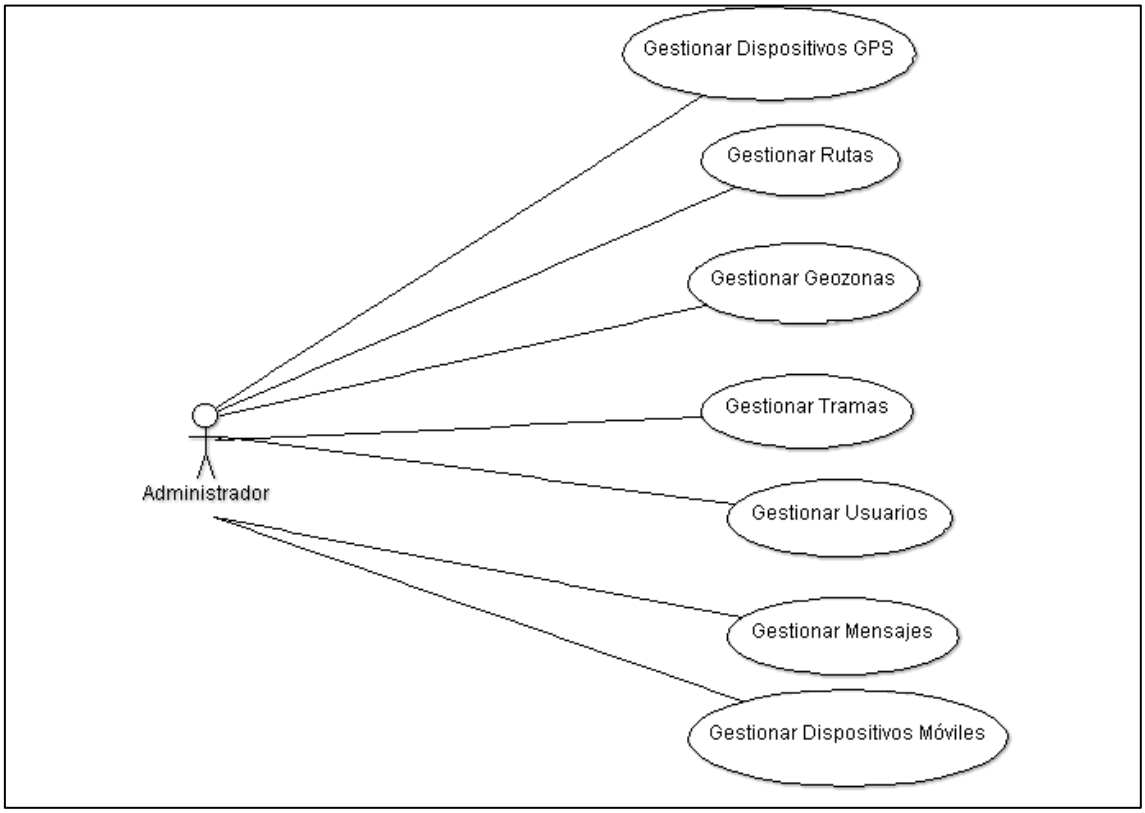


Ilustración 2: Caso de uso web: Administrador → Casos de uso

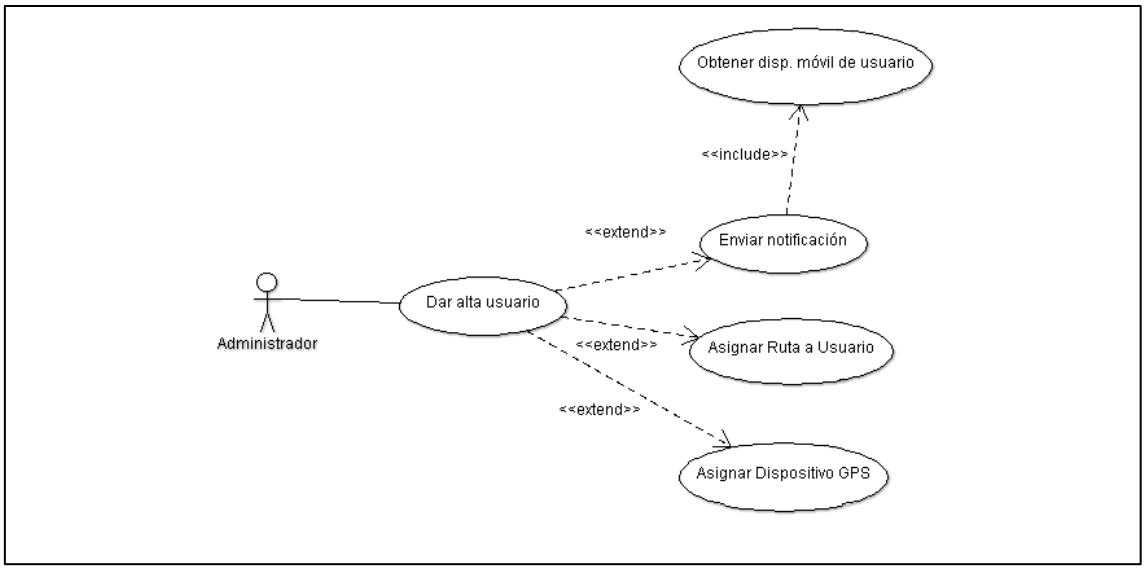


Ilustración 3: Caso de uso web: Administrador → Dar alta usuario

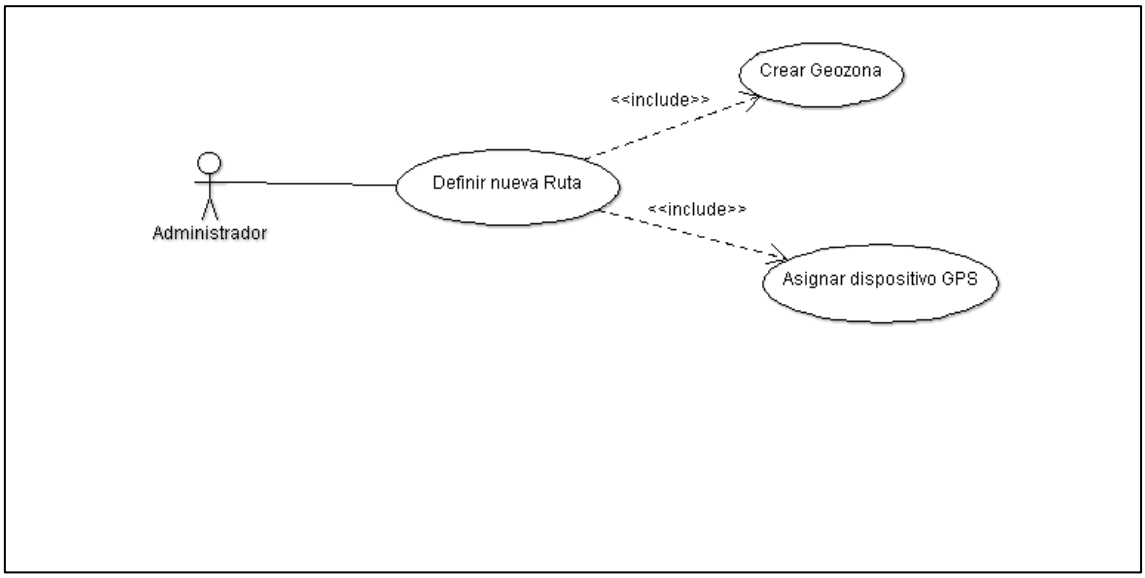


Ilustración 4: Caso de uso web: Administrador → Definir nueva ruta

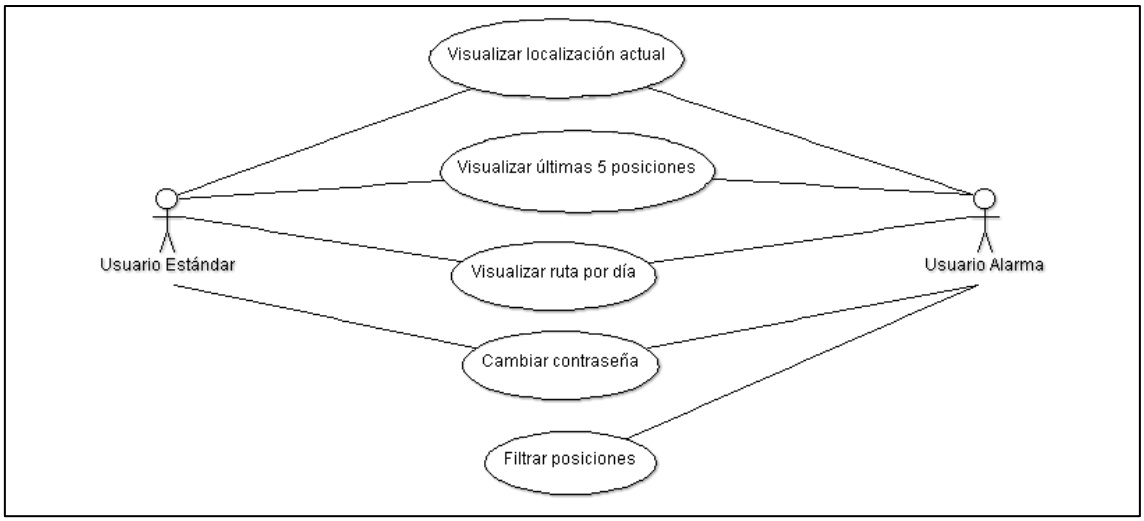


Ilustración 5: Caso de uso web: Usuario estándar y alarma → Casos de uso

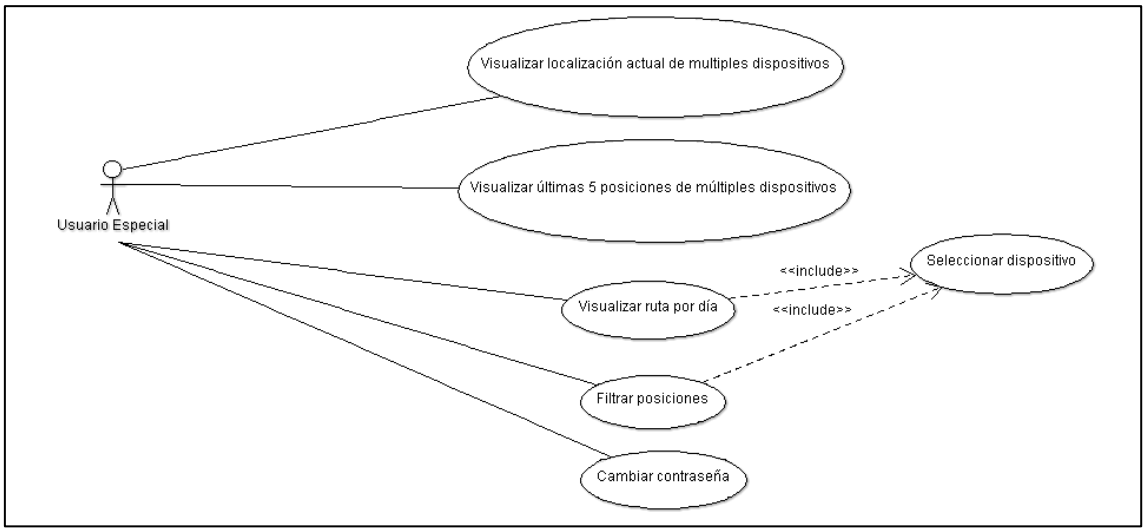


Ilustración 6: Caso de uso web: Usuario especial → Casos de uso

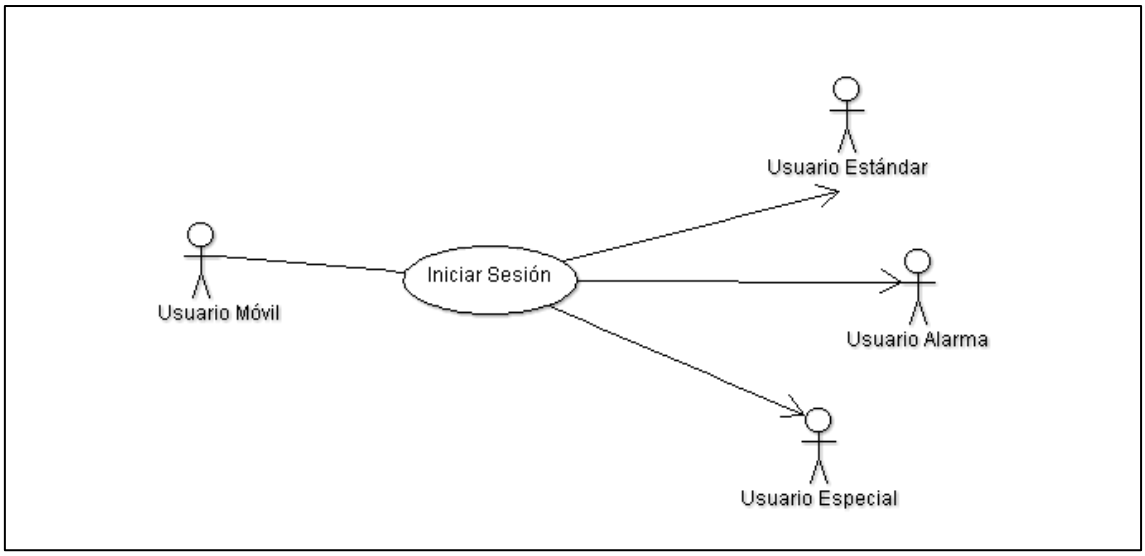


Ilustración 7: Caso de uso móvil: Usuario móvil → Iniciar sesión

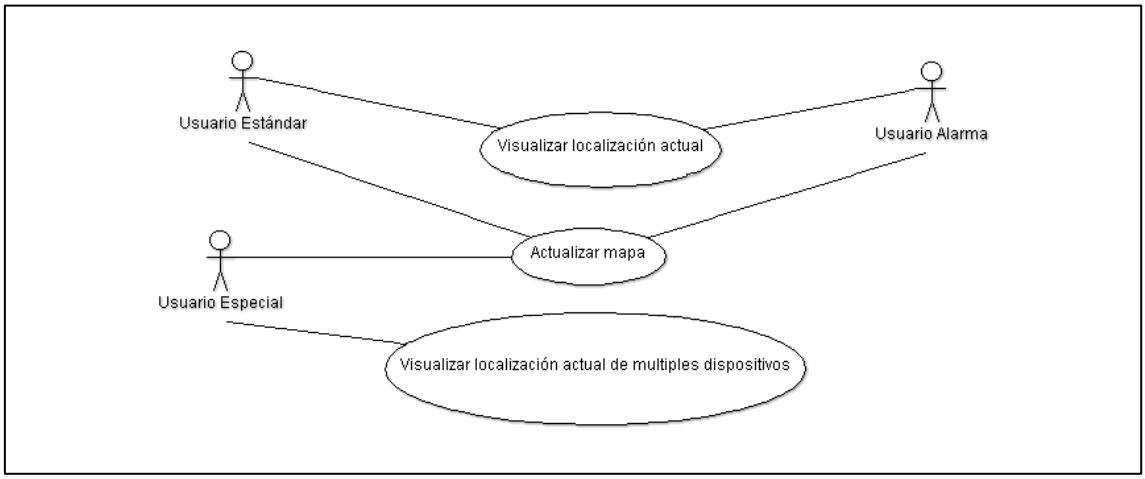


Ilustración 8: Casos de uso móvil: Todos los usuarios → Casos de uso

4.2 Diagrama de Clases

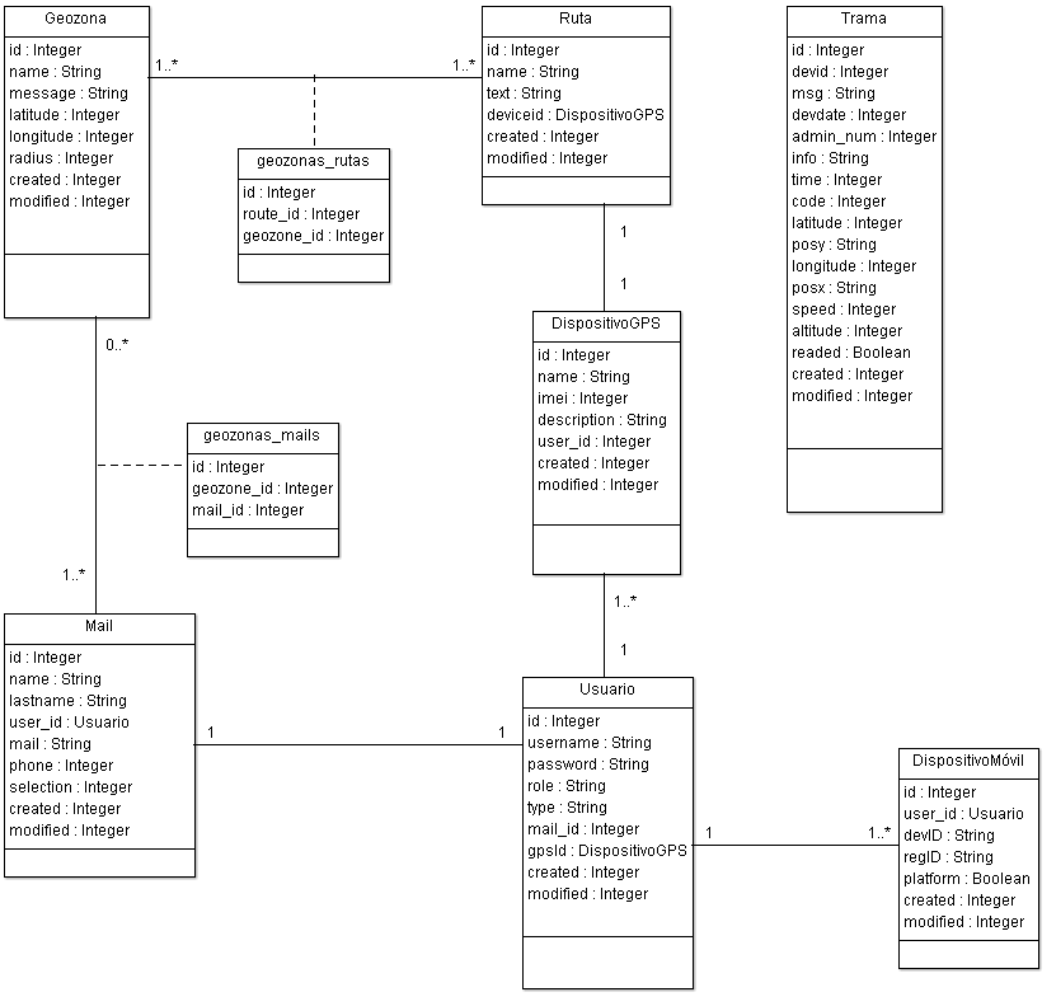


Ilustración 9: Diagrama de clases

4.3 Diagrama de Base de Datos

Para mayor comodidad visual hemos dividido el siguiente diagrama en varios fragmentos.

Hemos utilizado MySQL Workbench para su creación ya que esta aplicación nos permite obtener el diagrama de base de datos de una base de datos previamente creada mediante la opción de ingeniería inversa.

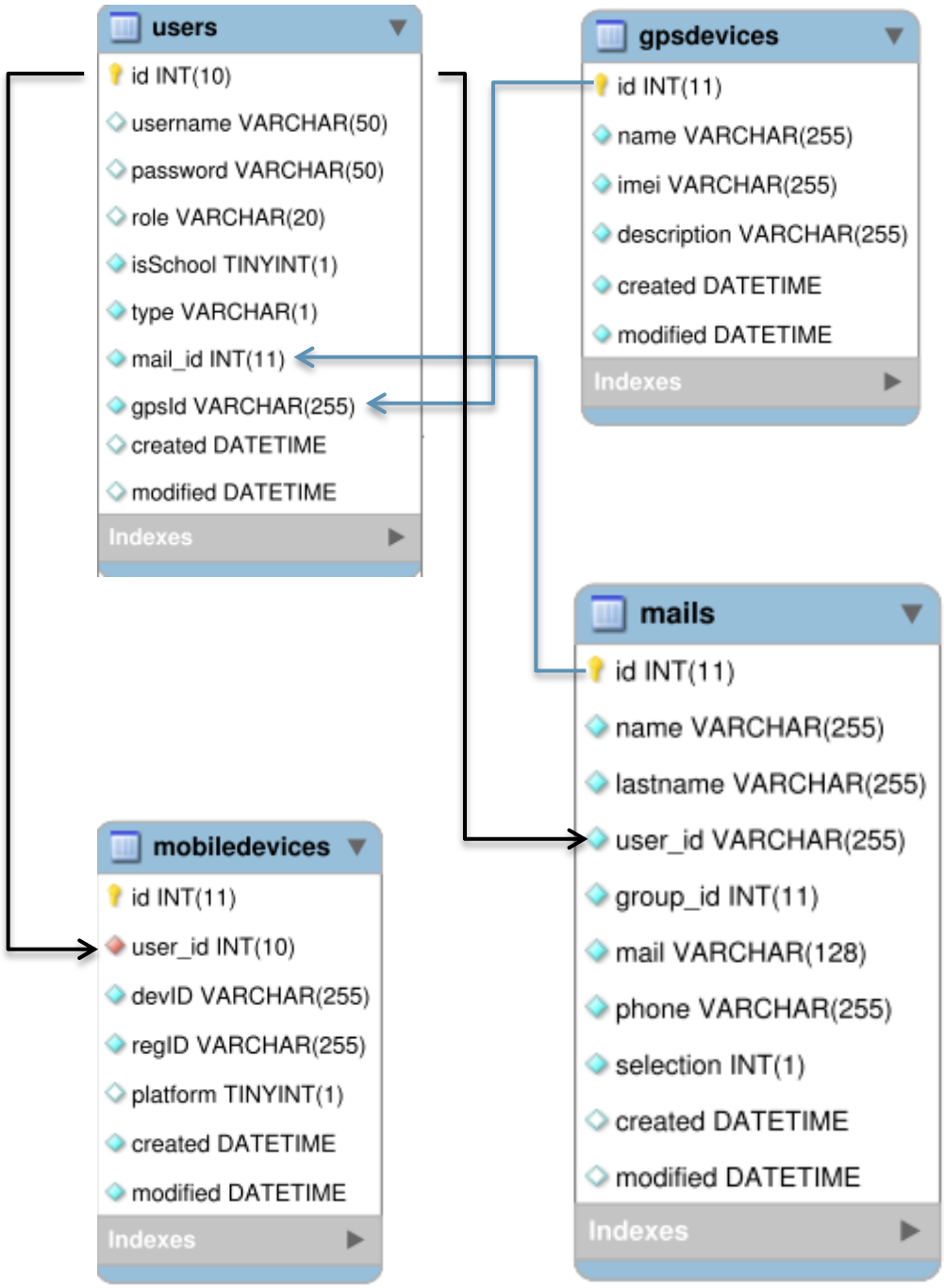


Ilustración 10: Diagrama de base de datos 1

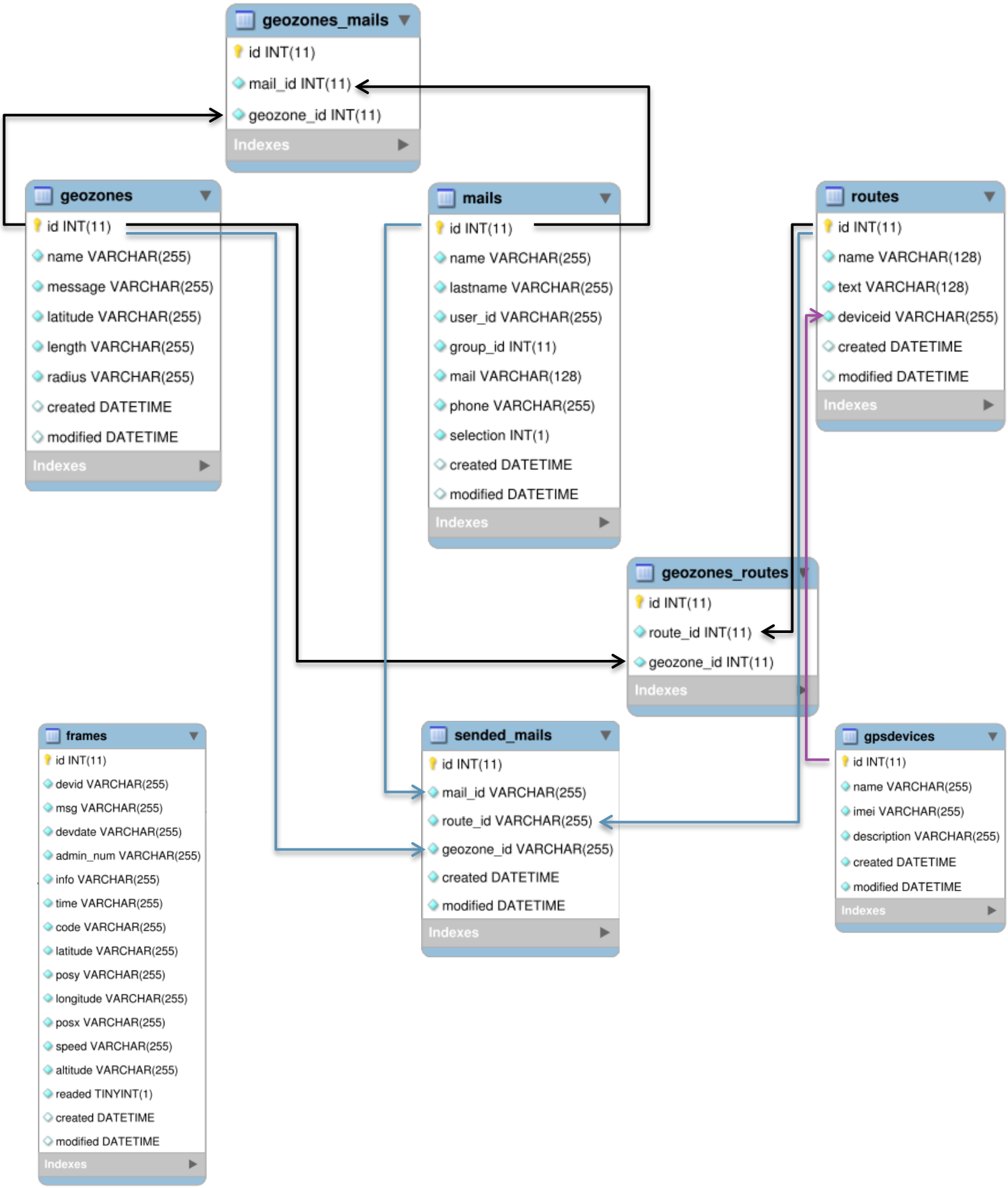


Ilustración 11: Diagrama de base de datos 2

5. Diseño

5.1 Diseño Aplicación Web

A continuación debemos describir cómo funciona el principal "motor" de nuestra aplicación web, CakePHP. Según su web el *framework* CakePHP nos permite crear aplicaciones rápidamente y escribiendo menos código que con los procedimientos habituales. CakePHP se basa en la arquitectura de software MVC (*Model-View-Controller*) y es a partir de este patrón y de las *conventions*[5] que incluye lo que hace que el desarrollo de aplicaciones web y su posterior mantenimiento sea mucho más sencillo.

5.1.1 (MVC) Modelo-Vista-Controlador

Este paradigma separa las partes características de una aplicación (lógica de negocio y datos, interfaz del usuario y comunicadores) en tres componentes distintos llamados modelo, vista y controlador.

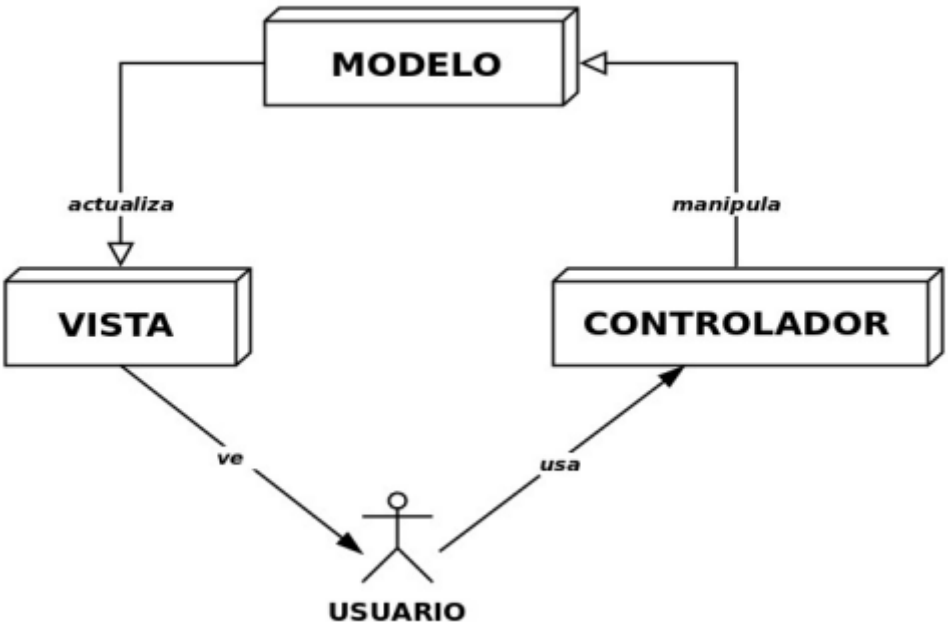


Ilustración 12: Esquema resumen Modelo-Vista-Controlador [5]

El Modelo es el componente con el cual se realiza la lógica de la aplicación. Es el encargado de gestionar cualquier tarea vinculada a la manipulación de datos (p.ej. crear reglas de validación que se comprobarán antes de insertar datos en una base de datos) y su función consiste en proporcionar a la vista la información que en cada momento ésta nos solicita.

[5] Conventions: Serie de reglas que pueden o no seguirse durante el desarrollo de la aplicación web, lo cual nos permite establecer un marco común de trabajo y por tanto estandarizar el proceso de despliegue de ésta.

El Controlador se ocupa de responder a las interacciones que realiza el usuario sobre la vista mediante el manejo de los datos del modelo. Podríamos afirmar que el controlador hace de enlace entre la vista y el modelo.

La Vista muestra la información del modelo tras su demanda y es presentada al usuario en un formato adecuado para que éste interactúe con ella.

5.1.2 Prototipo de las vistas

En el siguiente punto vamos a mostrar varios prototipos sobre el aspecto que deberán tener las vistas de nuestra aplicación web, cabe mencionar que en primer lugar se muestra el boceto de la primera versión de la aplicación web, más tarde se modificó el diseño convirtiéndose éste en el boceto final. Para realizar este apartado se ha utilizado la herramienta Pencil ya que está disponible en todas las plataformas y es *open-source*[6].

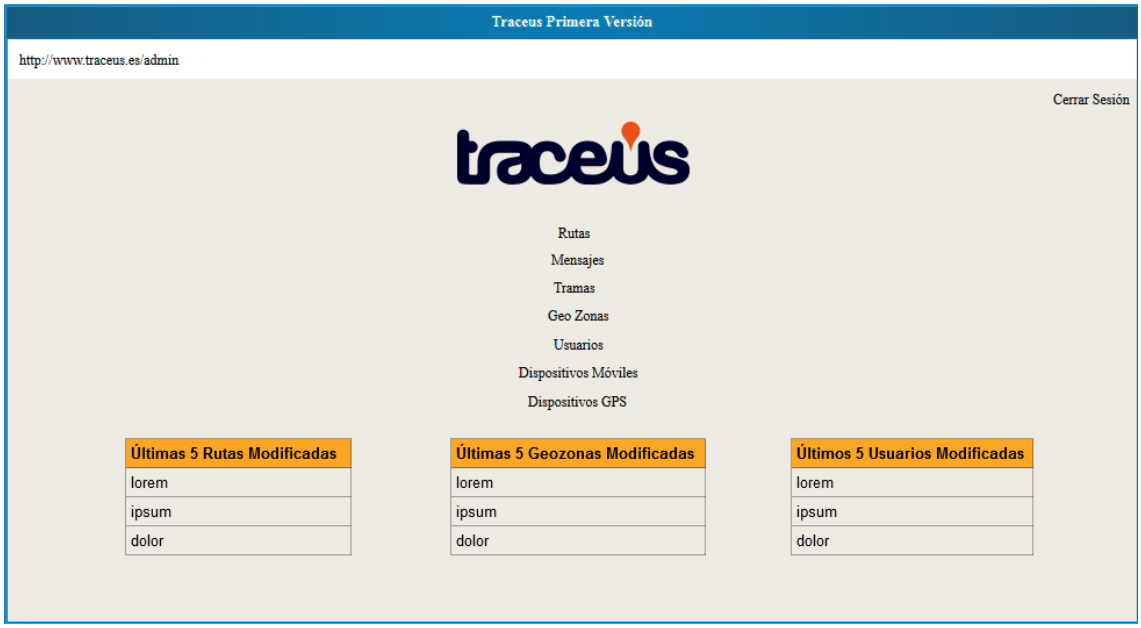


Ilustración 13: Index principal (Primera Versión)

[6] Open-Source: Software o Hardware distribuido libremente.

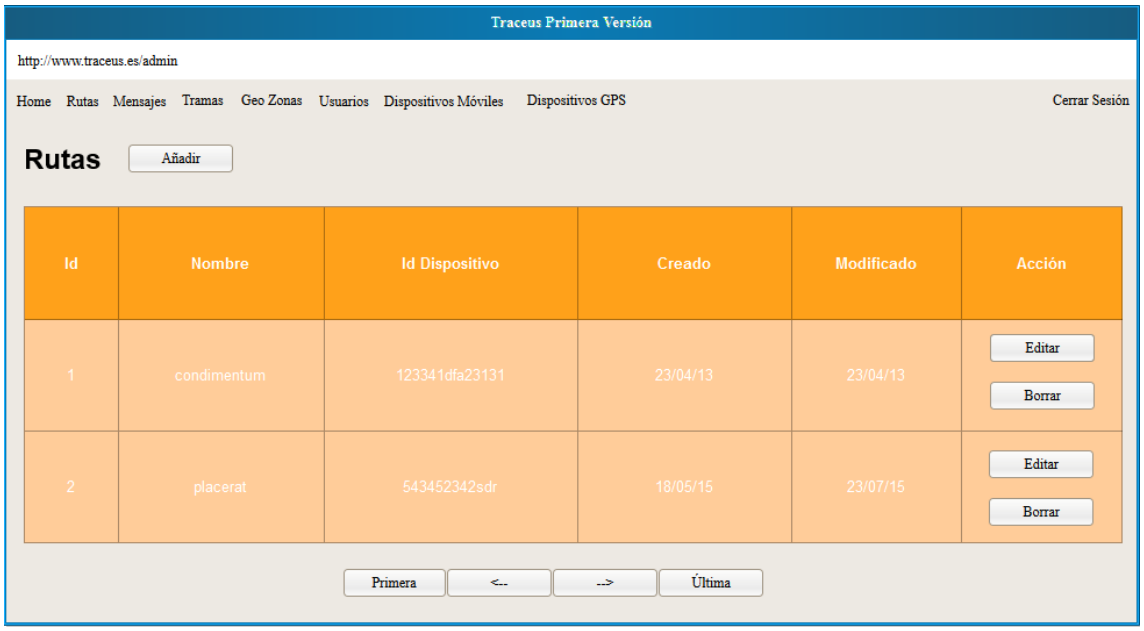


Ilustración 14: Index Rutas (Primera Versión)

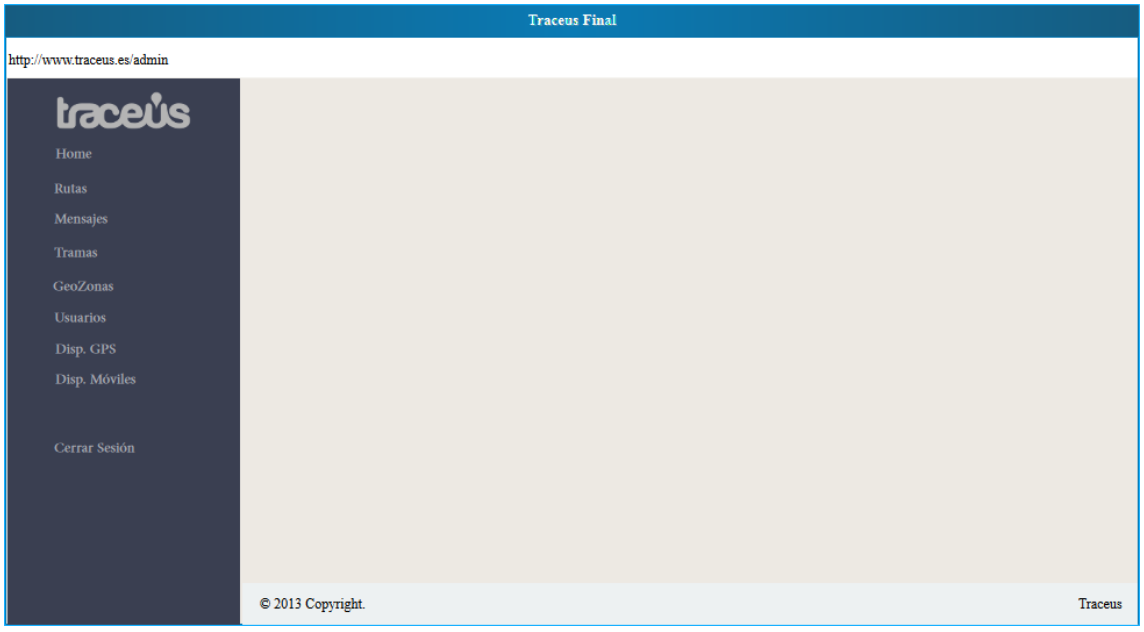


Ilustración 15: Index principal (Versión Final)

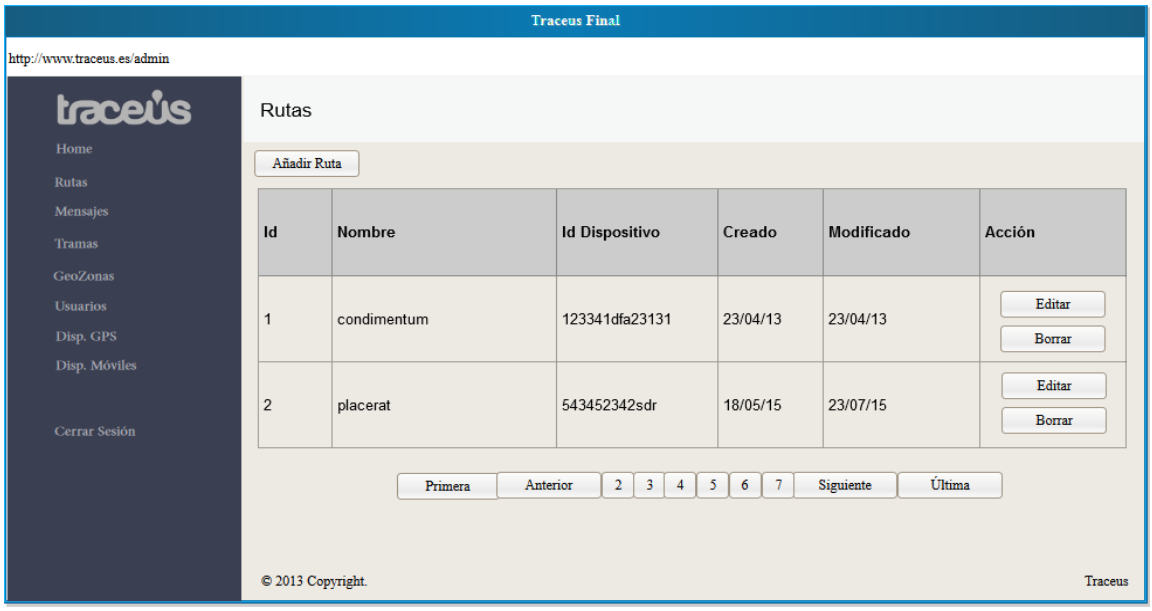


Ilustración 16: Index Rutas (Versión Final)

Debemos aclarar que solo mostramos los diseños de la página principal y de la vista index de Rutas a modo de ejemplo ilustrativo.

5.2 Diseño Aplicación Móvil

Para el desarrollo de la aplicación móvil nos hemos basado en la utilización de la herramienta Phonegap, la cual permite crear aplicaciones para cualquier dispositivo móvil (Amazon-fireOS, Android, Blackberry10, iOS,…) utilizando para ello herramientas centradas en el desarrollo web como son JavaScript, HTML5 y CSS3. Ya que estamos desarrollando la aplicación como si se tratase de una aplicación web hace que su depuración sea mucho más simple, siendo capaces de ejecutar la aplicación en un navegador web que permita el uso de las tecnologías anteriormente señaladas.

5.2.1 Prototipo de la aplicación móvil

Como ya ocurría en el caso anterior para la realización de los bocetos de la aplicación móvil se ha utilizado el software Pencil.

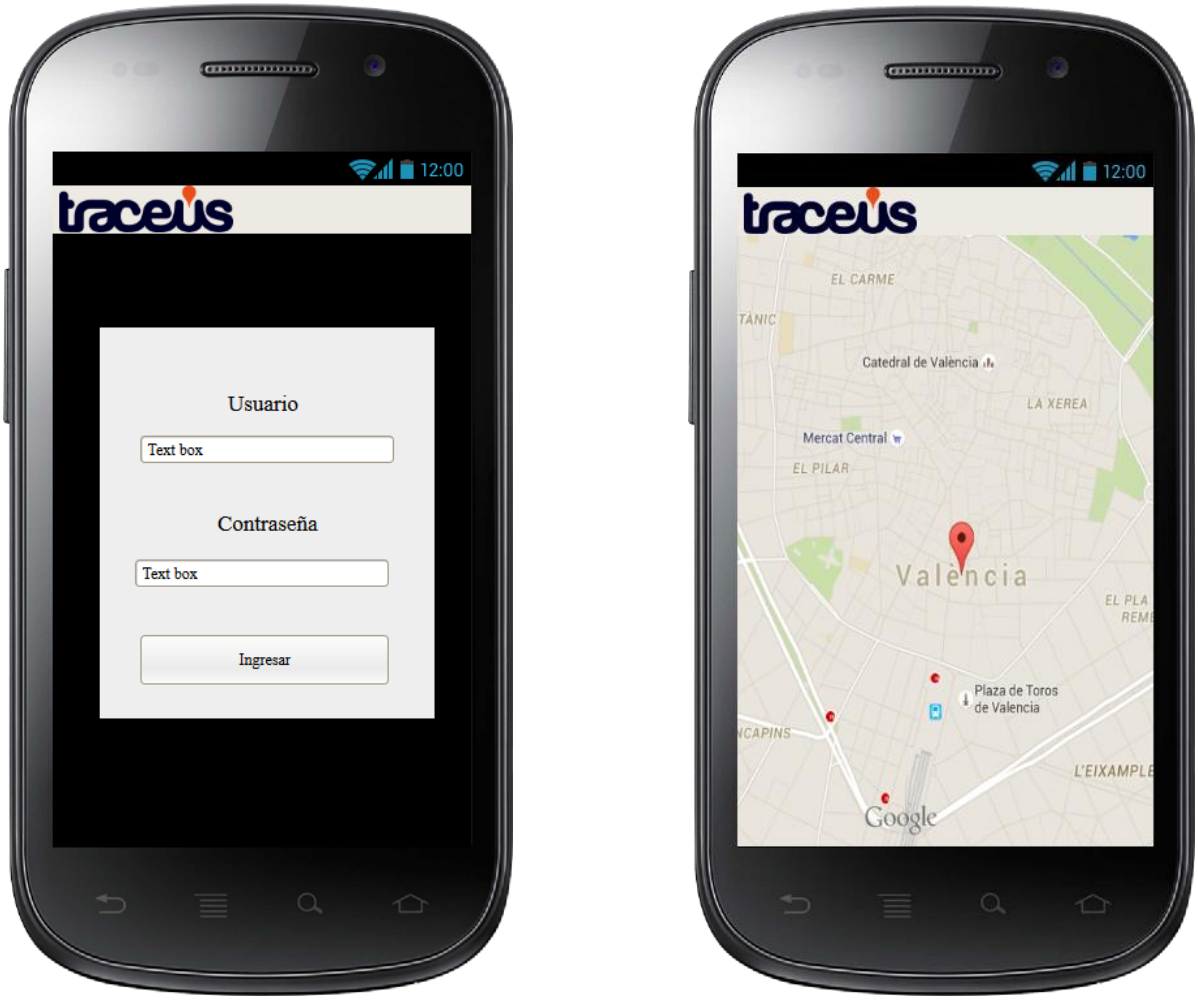


Ilustración 17: Aplicación Móvil (Prototipo)

Como se puede observar la aplicación móvil se trata fundamentalmente de dos apartados. El primer apartado está dedicado al *login* mediante un nombre de usuario y una contraseña y el segundo apartado se trata de la página principal de la aplicación que se describirá posteriormente.

6. Implementación

Antes de pasar a la explicación del código de ambas aplicaciones vamos a describir la configuración y características del hardware de localización y de todo el software utilizado en el proyecto.

6.1 Dispositivo GPS

Ilustración 18: Dispositivo GPS

-Especificaciones-

Dimensiones	64mm x 46mm x 17mm (1.8” x 2.5” x 0.65”)
Peso	50g
Red	GSM/GPRS
Bandas	850/1800/1900Mhz o 900/1800/1900Mhz
Chip GPS	SIRF3 chip
Módulo GSM/GPRS	Siemens MC55/Siemens MC56 o Simcom300/Simcom 340
Sensibilidad GPS	-159dBm
Exactitud GPS	5m
Tiempo hasta definir la primera posición	Frío (primera vez que se utiliza) 45s Caliente (no se ha utilizado en las últimas 4h) 35s Muy Caliente (se ha utilizado en las últimas 4h) 1s
Cargador de Coche	12 – 24V entrada 5V salida
Cargador de Pared	110 – 220V entrada 5V salida
Batería	Batería recargable Li-ion 3.7V 800mAh
Tiempo de Uso en Modo Reposo	48horas
Temperatura de Almacenamiento	De -40°C a +85°C
Temperatura de Trabajo	De -20°C a +55°C
Humedad	5% -- 95% sin condensación

*Datos obtenidos del *User Manual* [7]

Para conseguir el seguimiento de los autobuses o vehículos que tengamos en la flota es necesario el uso de un hardware adiccional, en este caso se ha utilizado el tracker tk102-2. Es sencillo de utilizar, ofrece un muy buen posicionamiento y para su funcionamiento únicamente

se requiere el uso de una tarjeta SIM y una tarjeta microSD (Opcional) , en caso de quedarnos sin cobertura de señal GPS podremos recuperar y procesar posteriormente la información través de nuestro ordenador.

El dispositivo presenta únicamente dos botones en la parte derecha que nos permiten realizar las siguientes acciones:

- Encender y apagar el dispositivo mediante una pulsación prolongada.
- Mandar una señal de SOS especial, que se tratará posteriormente.

Para configurar el dispositivo se utiliza el envío de SMS al número de teléfono que pertenezca la anteriormente citada tarjeta SIM. Por ejemplo para programar la dirección IP donde el dispositivo transmitirá las tramas de datos se utilizan los siguientes mensajes SMS:

Inicialización:	[begin+password[7]]
Crear usuario del operador:	[up+password+space[8]+username+space+passwordForUsername]
Configurar IP y Puerto destino:	[adminip+password+space+IP+space+Port]
Configurar APN del operador:	[apn+password+space+APNdelOperador]
Activar modo GPRS:	[gprs+password]
Activar mensaje de batería baja:	[lowbattery+password+space+on]
Activar alarma de movimiento:	[nostockade+password]
Sensibilidad de la alarma de movimiento:	[sensitivity+password+space+level[9]]
Fijar el intervalo de rastreo:	[fix045s***n+password]
Deshabilitar el rastreo al estar parado:	[less+space+gprs+password+space+on]

Una vez configurado correctamente el dispositivo enviará datos en un intervalo de 45s y dejará de enviar dichos datos al no moverse durante 5 minutos.

[7] Password: El password por defecto del dispositivo es: 123456.
[8] Space: Esto nos indica que en el SMS debemos dejar un espacio en blanco.
[9] Level: Existen 3 distintos:

1. (La alarma de movimiento se dispara en cuanto el dispositivo recibe una pequeña vibración).
2. (La alarma de movimiento se dispara cuando el dispositivo recibe 8 secuencias de vibración en 2 segundos).
3. (La alarma de movimiento se dispara cuando el dispositivo recibe 25 secuencias de vibración en 5 segundos).

6.2 CakePHP

6.2.1 Historia y Características

CakePHP es un *framework* o marco de desarrollo basado en PHP, creado por Michal Tatarynowicz en Abril de 2005 como una RAD[10] para PHP. Esto llevó a que en Diciembre de ese mismo año se fundase la "Cake Software Foundation" para promover el desarrollo relacionado con CakePHP. La versión 1.0 de CakePHP fue lanzada en Mayo de 2006, en la actualidad este *framework* se encuentra en la versión 3.0.

CakePHP se sitúa en la undécima posición en el ranking de los mejores *frameworks* de PHP de este año [8], esto nos permite verificar que se trata de un *framework* bastante conocido y con una comunidad activa.

CakePHP nos permite utilizar funciones de generación de código y *scaffolding*[11] para crear prototipos de una forma rápida. No necesita apenas configuración, simplemente necesitamos incluir y adaptar nuestra base de datos y ya estamos listos para empezar el desarrollo. CakePHP tiene muchas funcionalidades incluidas de serie: traducciones, acceso a la base de datos, almacenamiento en cache, validación, autenticación, etc. Y por último, en temas de seguridad, nos ofrece herramientas integradas para la validación de entrada de datos, protección CSRF[12], protección de manipulación en formularios, prevención de inyección SQL, etc.

6.2.2 Licencia

CakePHP define su licencia como una licencia "*friendly*", ya que éste está autorizado bajo licencia MIT[13] y por tanto nos permite la modificación, distribución y/o nueva publicación del código fuente, ya sea para uso comercial o para uso propio. Simplemente se debe cumplir la condición de dejar intactos los avisos de derechos de autor tal como define este tipo de licencia.

6.2.3 Descarga

En esta sección vamos a ilustrar todos los pasos a realizar para conseguir CakePHP y los requerimientos para su correcto funcionamiento.

Existen dos posibles métodos para adquirir CakePHP:

- Visitando la web oficial http://www.cakephp.org y siguiendo el enlace de descarga que se presenta en la página principal.
- Clonando el repositorio en *GitHub*[14] para obtener la versión estable más reciente. Para ello utilizaremos el siguiente comando:

```
git clone git://github.com/cakephp/cakephp.git
```

[10] RAD (Desarrollo rápido de aplicaciones): Del inglés Rapid application development, es un término general que se utiliza para referirse al desarrollo de software en el que se pone menor énfasis en las tareas de planificación y más énfasis en el desarrollo.

[11] Scaffolding: Técnica de CakePHP para crear aplicaciones web básicas que pueden crear, leer, editar y borrar entidades. Como se indica en el CookBook éste es un mecanismo inflexible y debería ser temporal.

[12] CSRF (Falsificación de petición en sitios cruzados): Del inglés Cross-site request forgery, es un tipo de ataque a una web en el que un usuario, en el que la web confía (p.ej. el administrador), es utilizado para transmitir comandos no autorizados desde otra web ajena a la primera.

[13] MIT: Licencia de software libre establecida por el 'Massachusetts Institute of Technology (MIT)'

[14] GitHub: Plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema Git.

Basándonos en la información que nos proporcionan en el tutorial de instalación de CakePHP para poder utilizarlo necesitamos:

- Un servidor HTTP (p.ej. Apache).
- PHP 5.3.0 o superior (aunque las versiones 2.6 y por debajo de CakePHP soportan PHP 5.2.8).
- Un motor de base de datos. No es un requerimiento pero en la actualidad cualquier aplicación web utilizará uno. CakePHP soporta una gran variedad de motores de base de datos: MySQL (4 o superior), PostgreSQL, Microsoft SQL Server y SQLite.
- Por último se nos informa de que todos los controladores incorporados requieren PDO[15], por tanto es nuestro deber asegurar que tenemos instalada la extensión PDO correcta.

6.2.4 ¿Por qué CakePHP?

Utilizar CakePHP fue elección de la empresa Tecnoprotel-Elian S.L. pero tras su uso y al entender mejor su funcionamiento podemos observar que se trata de un *framework* que permite crear aplicaciones web con bases de datos en poco tiempo. Otro aspecto a tener en cuenta es que se trata de un software gratuito y con una comunidad activa. Por último destacar que utilizar este *framework* frente al lenguaje PHP nativo facilita mucho la lectura y comprensión de las aplicaciones desarrolladas con él.

6.3 Node.js

6.3.1 Historia y Características

Node.js es una plataforma de código abierto para la capa del servidor basada en el motor de JavaScript V8 de Google. Fue creado por Ryan Lienhart Dahl en 2009 y actualmente está patrocinado por la empresa Joyent, de la que Dahl forma parte. Su lanzamiento inicial fue el 27 de mayo de 2009, en la actualidad esta plataforma se encuentra en la versión 0.12.2.

Node.js está catalogado como un marco orientado a eventos asíncronos y su principal cometido es construir aplicaciones de red altamente escalables (p.ej. servidores web). Node.js basa su funcionalidad en módulos, los cuales pueden ser compilados en el propio binario "módulos básicos" o pueden ser módulos de terceros que pueden extender node.js o añadir un nivel de abstracción.

6.3.2 Licencia

Al igual que describimos en el apartado de CakePHP, Node.js está bajo licencia MIT y reúne otros componentes libres bajo licencia *Open-source*.

6.3.3 Descarga

Nuevamente se nos ofrece la posibilidad de realizar la descarga de diversas formas:

- En el apartado 'Downloads' de la web oficial http://www.nodejs.org encontramos tanto el instalador para Windows como para MacOS y el código fuente comprimido para los demás sistemas operativos.

[15] PDO: Del inglés PHP Data Objects, se trata de una extensión para PHP 5 que permite hacer uso de las mismas funciones para hacer consultas y obtener datos de distintos manejadores de bases de datos.

- También podemos conseguir Node.js a través del uso del *package manager*. Las diferentes instrucciones varían según el sistema operativo, las podemos encontrar en https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager.

El único requerimiento que necesitamos para poder instalar correctamente Node.js es que si realizamos la descarga por otro método que no sea el instalador, es decir si descargamos los archivos comprimidos, es tener previamente instalado Python 2.6 o 2.7.

6.3.4 ¿Por qué Node.js?

Ya que se pretende realizar un servidor sencillo que escuche en una dirección IP y un puerto dado los mensajes de varios dispositivos GPS se optó por utilizar esta plataforma. Node.js ofrece eventos asíncronos y entrada y salida de datos no bloqueante por tanto es posible utilizarlo para escuchar múltiples conexiones sin necesidad de tener que utilizar recursos adicionales. Node.js es perfecto para aplicaciones de datos intensivos en tiempo real, como es nuestro caso.

6.4 Phonegap

6.4.1 Historia y Características

Phonegap es un *framework* que nos permite desarrollar aplicaciones móviles desde un marco de programación web, es decir utilizando herramientas genéricas como pueden ser HTML5 y JavaScript podemos obtener aplicaciones para cualquier dispositivo móvil. Hay que remarcar que las aplicaciones resultantes son "híbridas", es decir, no son realmente aplicaciones nativas.

Phonegap fue producido por Nitobi, el cual más tarde fue comprado por Adobe Systems el actual desarrollador de Phonegap. El lanzamiento inicial fue el 7 de agosto de 2008, en la actualidad este *framework* se encuentra en la versión 3.5.0.

La característica principal de Phonegap es la capacidad de manejar API[16] que permite tener acceso a diferentes componentes de cada dispositivo. En la web http://www.phonegap.com/about/feature se nos muestra la siguiente tabla con la información de a que componentes podemos acceder según su sistema operativo.

[16] API: Interfaz de programación de aplicaciones. Conjunto de subrutinas, funciones y procedimientos para conseguir abstracción en la programación, habitualmente entre los niveles o capas inferiores y los superiores del software.

	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 6.0+	Blackberry 10	Windows Phone 8	Ubuntu	Firefox OS
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓
Compass	X	✓	✓	X	✓	✓	✓	✓
Contacts	✓	✓	✓	✓	✓	✓	✓	✓
File	✓	✓	✓	✓	✓	✓	✓	X
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	X	✓	✓	✓	X
Network	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓

✓ - supported feature
X - unsupported feature due to hardware or software restrictions

Ilustración 19: Características accesibles en Phonegap

6.4.2 Licencia

Phonegap fue contribuido a la Apache Software Foundation (ASF) bajo el nombre Apache Córdoba. A través de la ASF el desarrollo futuro con Phonegap siempre será libre y de código abierto bajo la licencia Apache, Versión 2.0.

6.4.3 Descarga

Para instalar Phonegap necesitamos tener instalado previamente Node.js, ya que el proceso de instalación hace uso del node package manager (npm). Una vez que cumplamos este requisito podemos instalar la versión más actual de Phonegap ejecutando el siguiente comando:

```
C:\> npm install -g phonegap
```

Cabe destacar que también tenemos disponible la Phonegap Desktop App, tanto para Windows como para OS X, la cual se instala mediante un instalador automático pero la Phonegap Desktop App todavía se encuentra en fase Beta.

6.4.4 ¿Por qué Phonegap?

Se ha elegido Phonegap principalmente por la posibilidad que ofrece de desarrollar en diferentes plataformas móviles mediante un único proyecto desarrollado en lenguaje web. Esto es especialmente interesante si se desconoce alguno de los lenguajes necesarios para desarrollar en sistemas operativos móviles. Como se ha comentado no es posible exprimir tan en profundidad los dispositivos como lo haríamos programando en su lenguaje nativo pero es una muy buena opción a tener en cuenta siempre que el desarrollo de la aplicación no requiera tanto nivel de detalle.

6.5 Aplicación Web

En primer lugar y para tener más clara la arquitectura que maneja CakePHP vamos a definir claramente la jerarquía de carpetas que ha de seguir un proyecto en CakePHP.

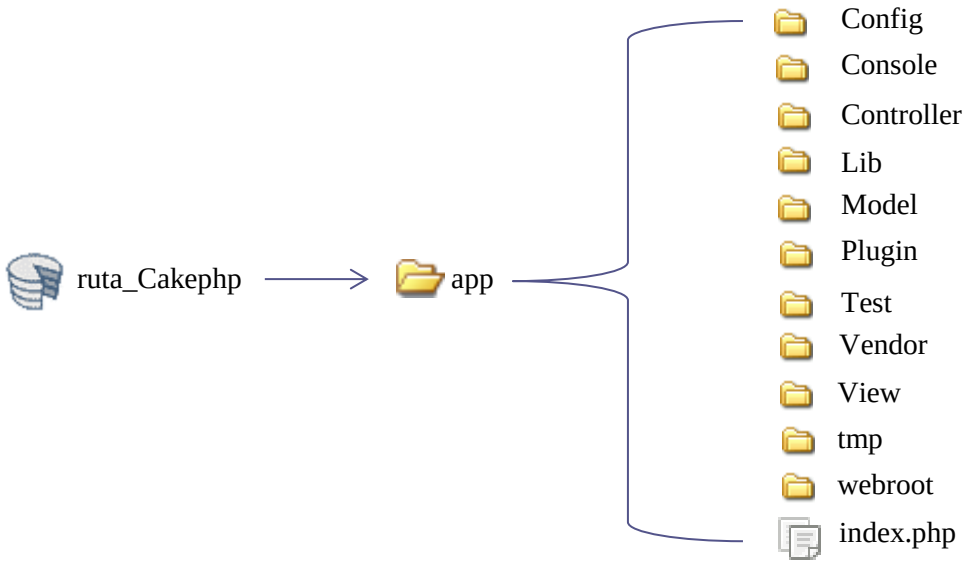


Ilustración 20: Jerarquía de carpetas de CakePHP

Como podemos ver la carpeta app es la contenedora de toda la funcionalidad y los recursos de nuestra aplicación web. Veamos más en detalle cada uno de sus componentes.

6.5.1 Configuración

Para conseguir que nuestra aplicación web funcione correctamente es necesario configurar correctamente el acceso a la base de datos. Esto se consigue de una forma muy sencilla en CakePHP, únicamente es necesario renombrar o copiar el contenido de database.php.default, que se encuentra dentro de la carpeta Config, al fichero database.php en esa misma carpeta y rellenar los campos de $result con nuestra configuración de base de datos.

```
<!-- Fichero: /ruta_CakePHP/app/Config/database.php -->
<?php
	class DATABASE_CONFIG {

		public $default = array(
			'datasource' => 'Database/Mysql',
			'persistent' => false,
			'host' => 'nombre host',
			'login' => 'nombre usuario',
			'password' => 'contraseña',
			'database' => 'nombre de base de datos',
			'prefix' => '',
			//'encoding' => 'utf8',
		);
	}
?>
```

6.5.2 Modelos

Como ya comentamos en el apartado (MVC) Modelo-Vista-Controlador los modelos son los encargados de interactuar con la base de datos. Es muy importante el nombre que contiene el campo en la base de datos sea el mismo a la hora de realizar su modelo, p.ej. si tenemos la tabla con nombre "*frames*" el archivo correspondiente a su modelo se llamará Frame.php. Deben estar localizados dentro de la carpeta Model.

En nuestra aplicación siguiendo el esquema de base de datos presente en el apartado Diagrama de Base de Datos tenemos las siguientes tablas: frames, geozones, gpsdevices, mails, mobiledevices, routes, sended_mails y users, por lo tanto debemos generar los siguientes archivos .php para crear sus modelos: **Frame.php**, **Geozone.php**, **Gpsdevice.php**, **Mail.php**, **Mobiledevice.php**, **Route.php**, **SendedMail.php** y **User.php**. El contenido inicial de estos archivos será el siguiente, siendo Modelo el nombre del archivo correspondiente:

```
<!-- Fichero: /ruta_CakePHP/app/Model/Modelo.php -->
<?php
	class Modelo extends AppModel{
		public $name = 'Modelo';
	}
?>
```

Una vez creados los archivos correspondientes todavía no podemos visualizar ningún tipo de contenido, si lo hacemos CakePHP nos avisará de que primero debemos crear los controladores correspondientes.

6.5.3 Controladores

Los controladores deben residir en la carpeta Controller y si seguimos las convenciones de CakePHP, lo cual se recomienda, deben nombrarse de la siguiente forma "*NombreModeloPlural+Controller.php*" por tanto nuestros archivos serán: **FramesController.php**, **GeozonesController.php**, **GpsdevicesController.php**, **MailsController.php**, **MobiledevicesController.php**, **RoutesController.php**, **SendedMailsController.php** y **UsersController.php.**

```php
<!-- Fichero: /ruta_CakePHP/app/Controller/GpsdevicesController.php
-->
<?php
	class GpsdevicesController extends AppController{

		public $helpers = array('Html', 'Form');

		public function beforeFilter() {
			parent::beforeFilter();
			$this->layout = 'layoutFinal';
		}

		public function index() {
			$this->paginate = array(
			'limit' => 20
			);
			$devices = $this->paginate('Gpsdevice');
			$this->set('devices', $devices);

			$this->set('lastCreated',
				$this->Gpsdevice->find('all', array(
				'order' => array(
					'Gpsdevice.modified' => 'desc')
			)));
		}
	}
  ?>
```

Como podemos observar se trata del controlador para el modelo Gpsdevice, vamos a describir las características que posee:

- `$helpers` → Un array en el cual vamos incluyendo objetos de tipo componente que nos ayudarán a la presentación de los datos en la vista.

- `function index` → Se trata de la función principal del controlador y podemos acceder a ella simplemente introduciendo en el navegador la url hacia el controlador (p.ej. http://www.traceus.es/admin/gpsdevices), no es necesario indicar al final /index.

- `$this->paginate` → Esta opción la utilizamos para paginar los resultados que posteriormente mostraremos en la vista, como puede observarse en primer lugar indicamos el límite de datos que deseamos por página, en este caso el límite es de 20, a continuación asignamos a la variable $devices el resultado de usar esa paginación con todos los datos recogidos por el modelo.

- `$this->set('devices', $devices)` → Set es la instrucción encargada de pasar parámetros desde el controlador a la vista, en este caso estamos pasándole a la vista la lista paginada que hemos definido anteriormente. En el segundo uso de set en el código [$this -> set('lastCreated', …)] vemos como incluso podemos asignar resultados de otras instrucciones, próximamente veremos cómo desde la vista accederemos a esta información.

- `$this->Gpsdevice->find('all',array("options"))` → Find es la instrucción encargada de recuperar información del modelo, en este caso estamos recuperando todos los datos de Gpsdevice, se mostrará incluso información de las relaciones que tenga Gpsdevice, y como peculiaridad estamos seleccionando

que se recuperen con el atributo modified en orden descendente, es decir, recuperaremos los Gpsdevices según el orden en el que se modificaron.

6.5.4 Vistas

Las vistas deben seguir unas instrucciones precisas para su correcta representación. En primer lugar debemos crear dentro de View una nueva carpeta que será la contenedora de todas las vistas relacionadas con ese controlador. Esta carpeta deberá llamarse exactamente como el controlador pero sin la palabra "Controller", es decir, "*NombreModeloPlural"*.

Una vez tengamos la carpeta creada simplemente debemos crear los archivos correspondientes a las funciones del controlador que queramos que posean vista, con extensión .ctp. En este caso crearemos el archivo **index.ctp** dentro de la carpeta **Gpsdevices**.

Llegados a este punto se debe tener en cuenta que para la realización de esta aplicación web la empresa a cargo de su realización (Tecnoprotel-Elian S.L.) adquirió un tema que combina módulos de Angular.JS[17] con estilos y funcionalidades de Bootstrap3[18].

Antes de ver en profundidad los archivos de vista vamos a explicar la estructura que se ha seguido para mostrar estas vistas en nuestra aplicación web. Hemos utilizado el esquema que se muestra en la siguiente imagen:

Ilustración 21: Aclaración funcionamiento de layouts

Dentro de cada controlador se define que layout va a utilizar, mediante `$this->layout = 'NombreLayout'`. De esta forma hemos podido definirnos diversos layouts para cada tipo de usuario. La peculiaridad de estos layouts es que permiten inyectar código de la vista de la función del controlador que se está ejecutando, mediante la instrucción `echo $this->fetch('content')`. En definitiva, el controlador llama al layout correspondiente, este inyecta el código de la vista que se está solicitando a su propio html y se muestra en el navegador. Estos sencillos pasos ahorran mucho código que, de otra forma, se repetiría en cada vista.

[17] Angular.JS: Framework de JavaScript desarrollado por Google.
[18] Bootstrap3: Conjunto de herramientas o Framework de software libre para diseño de sitios web.

```
<!-- Fichero: /ruta_CakePHP/app/View/Gpsdevices/index.ctp -->
<?php
    $paginator = $this->Paginator;
?>
<div class="bg-light lter b-b wrapper-md ng-scope">
    <h1 class="m-n font-thin h3">Dispositivos GPS</h1>
</div>
<div class="wrapper-md ng-scope">
      <div class="panel panel-default">
        <div class="panel-heading">
                  <?php echo $this->Html->link('<i class=
                        "fa fa-plus"></i>
                        Añadir Dispositivo GPS',
                        array('action' => 'add'),
                        array('type' => 'button',
                        'class' => 'btn m-b-xs btn-sm
                        btn-primary btn-addon',
                        'escape'=>false));
                  ?>
        </div>
        <table class="table table-bordered table-striped">
              <thead>
                  <tr>
                  <?php
                        echo "<th>" .
                        $paginator->sort('id', 'ID') . "</th>";
                        echo "<th>" .
                        $paginator->sort('name', 'Nombre') .
                        "</th>";
                        echo "<th>" .
                        $paginator->sort('imei', 'IMEI') .
                        "</th>";
                        echo "<th>" .
                        $paginator->sort('description',
                        'Descripción') . "</th>";
                        echo "<th>" .
                        $paginator->sort('created', 'Creado') .
                        "</th>";
                        echo "<th>" .
                        $paginator->sort('modified',
                        'Modificado') . "</th>";
                        echo "<th>" .
                        $paginator->sort('Acción') . "</th>";
                  ?>
                  </tr>
            </thead>
      <!-- Continúa -->
```

Como podemos observar se trata de un fichero que contiene tanto código en html como código en php, hemos seccionado esta parte del código para representar el funcionamiento del componente Paginator a la hora de crear cabeceras de tablas “sortables”, es decir, que al hacer click sobre ellas se ordenarán ascendente o descendentemente según el dato pulsado.

En primer lugar por comodidad asociamos el componente a la variable `$paginator` y a continuación usaremos la instrucción sort para conseguir la funcionalidad descrita. Sort requiere dos campos, el primero es el nombre de la clave que vamos a ordenar, es decir, el nombre de la columna en la base de datos; y el segundo será el nombre que se mostrará finalmente en la vista.

```
<!-- Fichero: /ruta_CakePHP/app/View/Gpsdevices/index.ctp -->
<tbody>
	<?php foreach ($devices as $device): ?>
		<tr>
			<td id="id">
				<?php echo $device['Gpsdevice']['id']; ?>
			</td>
			<td id="data">
				<?php  echo $this->Html->link(
					$device['Gpsdevice']['name'],
						array('action' => 'view',
					 $device['Gpsdevice']['id']));
				?>
			</td>
			<td>
				<?php echo $device['Gpsdevice']['imei'];
				?>
			</td>
			<td>
				<?php  echo $this->Text->truncate(
					device['Gpsdevice']['description'],
					22,
					array(
					'ellipsis' => '...',
					'exact' => false));
				?>
			</td>
			<!--. . .-->
			<td id="action">
				<button type="button"
					class="btn btn-xs btn-default">
					<i class="fa fa-times"></i>
					<?php
					echo $this->Html->link('Borrar',
					array('action' => 'delete',
					 $device['Gpsdevice']['id']),
					array('confirm' => 'Está seguro de
				que desea borrar el dispositivo?'));
					?>
				<!--. . .-->
			</td>
	<?php endforeach; ?>
</tbody>
```

En el siguiente fragmento de código observamos el modo principal de mostrar la información en una vista:

- Se hace uso de un foreach para recorrer la variable `$devices`, que fue definida en el controlador. Dentro de esta variable es posible acceder a cada uno de los elementos del modelo y por tanto mostrarlos a partir de la orden echo.
- Vale la pena señalar que se está haciendo uso del *helper* html (`$this->Html`) definido previamente en el controlador. En este caso se utiliza para crear enlaces a otras funciones del controlador.

Aquí podemos ver el resultado final de la combinación layout+vista, la cual se encarga de listar los dispositivos GPS almacenados y nos permite mediante enlaces añadir nuevas entidades, editar o eliminar.

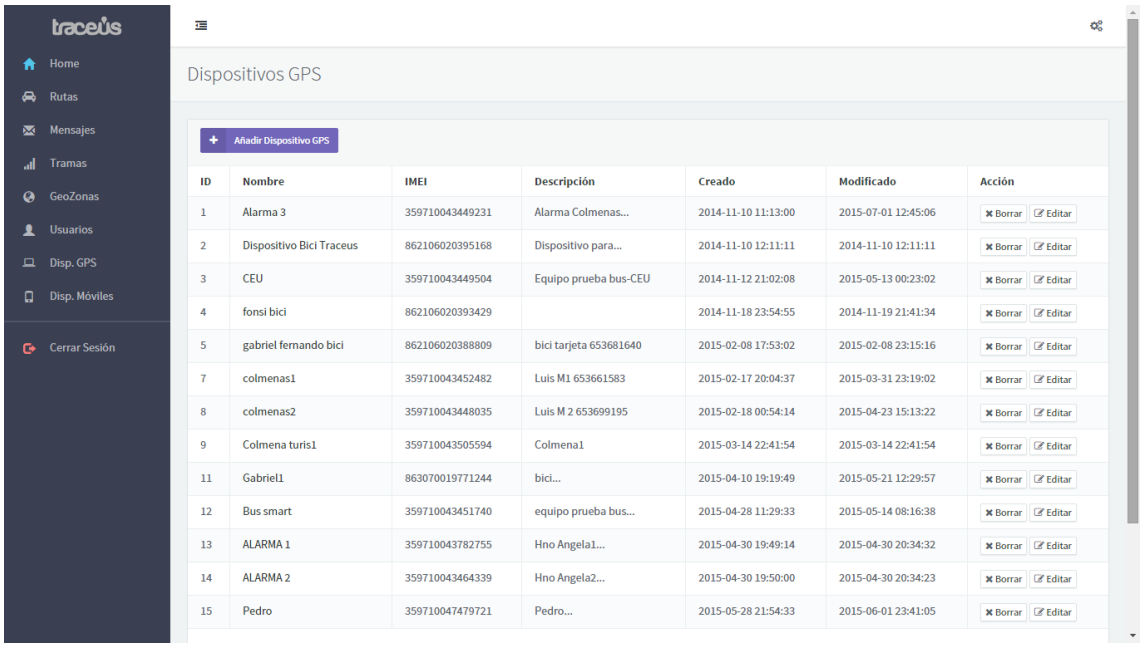


Ilustración 22: Ejemplo de relación layout + vista

6.5.5 Añadir Elemento

Una vez que hemos abordado el tema de mostrar los datos correctamente es hora de profundizar en ofrecer la posibilidad al usuario de añadir nuevos elementos.

Para conseguir esto debemos crear la función `add()` dentro del controlador.

```
<!-- Fichero: /ruta_CakePHP/app/Controller/GpsdevicesController.php
-->
public function add(){
      if($this->request->is('post')) {
            if($this->Gpsdevice->save($this->request->data)) {
                  $this->Session->setFlash(__
                        ('El dispositivo ha sido guardado.'),
                        'alert-box',
                        array('class'=>'alert-success'));
                  return $this->redirect(
                     array('action' => 'index'));
            }
            $this->Session->setFlash(__
                  ('Error al guardar el dispositivo.'),
                   'alert-box', array('class'=>'alert-danger'));
      }
}
```

Pasemos ahora a analizar el código, ya que se introducen aspectos no vistos hasta ahora:

- `$this->request->is('post')` → La función de esta instrucción es comprobar si la petición que se realiza se trata de un POST de HTTP, si no es así simplemente se mostrará la vista asignada.

- `$this->Gpsdevice->save($this->request->data)` → Como podemos ver esta instrucción se divide en dos partes, primero obtenemos mediante `$this->request->data` toda la información disponible del método POST y

posteriormente mediante un `save()` prentendemos almacenar los datos en nuestro modelo. Si conseguimos almacenar correctamente los datos, es decir sin errores de validación, devolvemos al usuario un mensaje de éxito, de no ser así se le devolverá un mensaje de error.

- Téngase en cuenta que utilizamos un elemento propio `alert-box` almacenado en la ruta app/View/Elements el cuál se encarga de mostrar correctamente, aplicándole un estilo Bootstrap, el mensaje que se pasa como argumento.

Una vez que tengamos definida nuestra función debemos crear el archivo que le dará vista, su ruta será /app/View/Gpsdevices/add.ctp

```
<!-- File: /app/View/Gpsdevices/add.ctp -->
<?php
      echo $this->Form->create('Gpsdevice' , array('class' =>
'form-horizontal'));
?>
<div ui-view="" class="fade-in ng-scope">
    <div class="bg-light lter b-b wrapper-md ng-scope">
      <h1 class="m-n font-thin h3">Añadir Dispositivo GPS</h1>
    </div>

    <div class="wrapper-md">
        <div class="panel panel-default">
            <div class="panel-body">
                <div class="form-group">
                    <label class="col-sm-2 control-label">
                          Nombre:
                </label>
                    <div class="col-sm-10">
                          <?php
                              echo $this->Form->input('name',
                          array('div' => null, 'type' => 'text',
                        'label' => false,
                        'class' => 'form-control ng-dirty ng-valid-
                        required ng-valid ng-touched', 'required'));
                          ?>
                    </div>
                </div>
                <!-- . . .-->
            </div>
            <footer class="panel-footer text-right bg-light lter">
                  <?php
                        echo $this->Form->end(array('div' =>
false, 'type' => 'submit', 'class' => 'btn btn-primary ', 'label'
=> 'Guardar Dispositivo GPS'));
                  ?>
             </footer>
        </div>
    </div>
</div>
```

Se puede observar que:

- Utilizamos (`$this->Form`) que es el segundo *helper* que habíamos definido en el controlador.

- `$this->Form->create` es el encargado de abrir el formulario en nuestra vista y por el contrario `$this->Form->end` es el encargado de cerrarlo mediante un botón de tipo *submit*.

6.5.6 Editar Elemento

Editar un elemento es muy similar a añadir un nuevo elemento, en primer lugar debemos crear la función correspondiente en el controlador. La única diferencia consiste en que al añadir un elemento no era necesario añadir ningún tipo de identificador ya que CakePHP se encarga de que al insertar un nuevo elemento éste reciba su id correspondiente.

```
<!-- Fichero: /ruta_CakePHP/app/Controller/GpsdevicesController.php
-->
public function edit($id = null) {
	if(!$id) {
		throw new NotFoundException(__(
			'Dispositivo inválido'));
	}

	$device = $this->Gpsdevice->findById($id);
	if (!$device) {
		throw new NotFoundException(__
			('Dispositivo inválido'));
	}

	if($this->request->is(array('device', 'put'))) {
		$this->Gpsdevice->id = $id;
		if ($this->Gpsdevice->save($this->request->data)) {
			$this->Session->setFlash(__
				('El dispositivo ha sido actualizado.'),
				'alert-box', array(
				'class'=>'alert-success'));
			return $this->redirect(array(
				'action' => 'index'));
		}
		$this->Session->setFlash(__
			('Error al actualizar el dispositivo'),
			'alert-box',
			array('class'=>'alert-danger'));
	}

	if(!$this->request->data){
		$this->request->data = $device;
	}
}
```

Podemos comprobar que en el caso de editar debemos proporcionar un `$id` y a partir de si este existe o no o si es un valor correcto o no continuaremos la ejecución o lanzaremos la excepción correspondiente. Posteriormente debemos crear el archivo de vista para la función edit, ya que esta vista será exactamente igual a la de añadir entidad omitiremos este paso, no sin antes indicar una sutil diferencia que no debe pasar por alto.

```
<!-- File: /app/View/Gpsdevices/edit.ctp -->
<?php
      echo $this->Form->input('id', array('type' => 'hidden'));
?>
```

Como mostramos en el código debemos introducir obligatoriamente un elemento oculto dentro del formulario de la vista, que se corresponderá con la id de la entidad que estemos editando. Sin esta id oculta al editar un elemento se nos creará uno nuevo y lo que pretendemos es actualizarlo.

6.5.7 Borrar Elemento

Por último podemos añadir la posibilidad de eliminar elementos. Nuevamente debemos crear la función que realizará la acción en el controlador, la peculiaridad de este caso es que al no ser necesaria la inserción de datos por parte del usuario no es necesario crear el archivo de vista.

```
<!-- Fichero: /ruta_CakePHP/app/Controller/GpsdevicesController.php
-->
public function delete($id = null) {
      $device = $this->Gpsdevice->findById($id);
      if (!$id) {
            throw new MethodNotAllowedException();
      }

      if($this->Gpsdevice->delete($id)) {
            $this->Session->setFlash(__
                  ('El dispositivo con nombre: %s ha sido
                  borrado.',
                   h($device['Gpsdevice']['name'])), 'alert-box',
                   array('class'=>'alert-warning'));
            return $this->redirect(array('action' => 'index'));
      }
}
```

Igual que en el caso de editar debemos indicar que `$device` vamos a eliminar, indicando para ello su `$id`. Si la operación ha tenido éxito lo mostramos al usuario y lo redirigimos a la vista index.

6.5.8 Login de Usuarios y Autorización

El *login* o inicio de sesión nos permite, mediante mecanismos que comentaremos a continuación, diferenciar entre usuarios y ofrecer a cada uno una vista personalizada dependiendo de sus privilegios. En nuestra aplicación únicamente los usuarios administradores serán capaces de realizar operaciones que actualice o modifique el estado de la base de datos, por tanto los usuarios que no sean administradores sólo serán capaces de visualizar la información contenida en la base de datos.

Basándonos en las convenciones de CakePHP debemos disponer de una tabla en nuestra base de datos llamada users. Esta tabla debe contener los campos *username* y *password* para hacer uso de la configuración automática de CakePHP, en nuestro caso hemos añadido de manera adicional el campo *type* para diferenciar tipos de usuario.

Por tanto creamos el controlador UsersController.php y lo configuramos como sigue:

```php
<!-- Fichero: /ruta_CakePHP/app/Controller/UsersController.php -->
<?php
App::uses('CakeSession', 'Model/Datasource');
class UsersController extends AppController {
    public $helpers = array('Html', 'Form', 'Tinymce');

    public function beforeFilter() {
        parent::beforeFilter();
        $this->Auth->allow('logout');
        $this->layout = 'layoutFinal';
    }

    public function login() {
             if ($this->Auth->login()) {
                if($this->Auth->user('role') == 'admin'){
                      return $this->redirect(array(
                           'controller' => 'index',
                            'action' => 'index'));
                }else{
                return $this->redirect(array(
                   'controller' => 'users',
                   'action' => 'viewUser',
                   $this->Auth->user('id')));
                }
              }else{
                return $this->Session->setFlash(__
                   ('Usuario o Contraseña inválidos.<br> Por favor,
                    inténtelo de nuevo.'), 'alert-box',
                    array('class'=>'alert-danger'));
             }
      }

      public function logout() {
        $this->Session->destroy();
           return $this->redirect($this->Auth->logout());
      }
}
?>
```

Se puede observar que se han creado dos funciones: **login**() y **logout**(). Como su nombre indica están son las funciones necesarias para identificarse y para dejar de estarlo respectivamente. Vamos a analizar las características destacables de este controlador:

- `beforeFilter`() → Ésta función nos permite especificar qué acciones del controlador queremos que sean públicas antes de la identificación del usuario. Tratándose de una aplicación web que requiere acceso para su visualización, únicamente permitimos la ejecución de **logout**().
- `$this->Auth->login()` → Gracias a esta función podemos comprobar si el usuario ha sido autenticado con éxito, y de ser así podemos identificar de qué tipo de usuario se trata.

`$this->Session->destroy()` → Mediante esta función nos encargamos de eliminar todos los datos del usuario que se hayan almacenado durante la sesión, esto es útil cuando utilizamos cookies para mantener activa la sesión del usuario.

Una vez que hemos realizado la creación del controlador anterior debemos añadir el *login* que será manejado por la clase `AuthComponent`. Para ello modificamos el controlador AppController.php añadiéndole los componentes de sesión y autenticación `Session` y `Auth`

```
<!-- Fichero: /ruta_CakePHP/app/Controller/AppController.php -->
<?php
App::uses('Controller', 'Controller');

class AppController extends Controller {

       public $components = array(
         'Session',
         'Auth' => array(
             'loginRedirect' => array(
                 'controller' => 'users',
                 'action' => 'login'
             ),
             'logoutRedirect' => array(
                 'controller' => 'pages',
                 'action' => 'display',
                 'home'
             )
         )
    );

    public function isAuthorized($user) {
           // Admin can access every action
           if (isset($user['role']) && $user['role'] === 'admin') {
               return true;
           }

           // Default deny
           return false;
       }
}
?>
```

Vale la pena destacar que es aquí donde en primer lugar enviamos al usuario al controlador UsersController, más concretamente a su función login(). También podemos observar que mediante `isAuthorized($user)` estamos otorgando a los usuarios con rol de administrador control total de la aplicación web, en nuestra aplicación cada controlador tiene su función `isAuthorized()` para así gestionar el acceso a funciones de cada usuario.

El siguiente paso es definirnos la vista encargada del *login* de los usuarios.

```
<!-- Fichero: /ruta_CakePHP/app/View/Users/login.ctp -->
<?php
      $this->layout = 'sigin';
      echo $this->Form->create('User');
?>

<div class="list-group list-group-sm">
      <div class="list-group-item">
        <?php echo $this->Form->input('username',
             array('label' => false, 'placeholder' => 'Usuario',
             'class' => 'form-control no-border',
             'required' => true)); ?>
      </div>
      <div class="list-group-item">
         <?php echo $this->Form->input('password',
            array('label' => false, 'type' => 'password',
            'placeholder' => 'Contraseña',
            'class' => 'form-control no-border',
             'ng-model' => 'user.password', 'required' => true));
            ?>
      </div>
</div>

<?php echo $this->Form->end(array('div' => false,
      'type' => 'submit',
      'class' => 'btn btn-lg btn-primary btn-block',
      'label' => 'Login'));
?>
```

Para terminar con el acceso de usuarios es necesario señalar que nunca debemos almacenar las contraseñas como texto plano, ya que esto supondría un problema de seguridad si alguien accediese a nuestra base de datos. Para evitar esto CakePHP nos ofrece la posibilidad de encriptar los *passwords* antes de guardar los datos de un usuario nuevo mediante la función `beforeSave()`.

```
<!-- Fichero: /ruta_CakePHP/app/Model/User.php -->
<?php
App::uses('SimplePasswordHasher', 'Controller/Component/Auth');

class User extends AppModel {

    public function beforeSave($options = array()) {
          if (isset($this->data[$this->alias]['password'])) {
              $passwordHasher = new SimplePasswordHasher();
              $this->data[$this->alias]['password'] =
                    $passwordHasher->hash(
                          $this->data[$this->alias]['password']
                    );
          }
          return true;
      }
}

?>
```

6.5.9 Seguridad

Una vez conocida la parte más técnica de nuestra aplicación debemos indicar que consideraciones, en cuanto a temas de seguridad, se han tomado a la hora de realizar el desarrollo de la aplicación web.

En primer lugar veamos que herramientas de seguridad nos ofrece el *framework* CakePHP. Todas las tareas comentadas a continuación se realizan dentro de un mismo componente de CakePHP, el componente *Security*:

- Restricción de métodos HTTP: Principalmente esta tarea nos permite indicar en nuestra aplicación que métodos HTTP (GET, POST, PUT, DELETE) requiere cada acción dentro de nuestro controlador. A continuación vamos a mostrar un ejemplo en el que se puede apreciar su uso con mayor nivel de detalle.

```
<!-- Fichero:
/ruta_CakePHP/app/Controller/GpsdevicesController.php -->
<?php
class GpsdevicesController extends AppController{

      public $helpers = array('Html', 'Form');
      public $components = array('Security');

      public function beforeFilter() {
        $this->Security->requirePost('add');
    }
```

  Podemos apreciar que generalmente el componente *Security* se utiliza dentro de la función **beforeFilter**() ya que se pretende que se active su funcionalidad desde el arranque.

- Conexión SSL: Gracias a *Security* también somos capaces de requerir que las acciones que indiquemos o que todas las acciones del controlador se realicen a través de una conexión segura SSL. En el siguiente ejemplo podemos constatar su simplicidad de uso.

```
<!-- Fichero:
/ruta_CakePHP/app/Controller/GpsdevicesController.php -->
<?php
class GpsdevicesController extends AppController{

      public $helpers = array('Html', 'Form');
      public $components = array('Security');

      public function beforeFilter() {
        if (isset($this->request->params['admin'])) {
            $this->Security->requireSecure();
        }
    }
```

  En este caso observamos que todas las acciones que se realicen con el parámetro admin en la petición, es decir las operaciones que hagan los usuarios administradores, se forzarán a que se realicen bajo conexión segura.

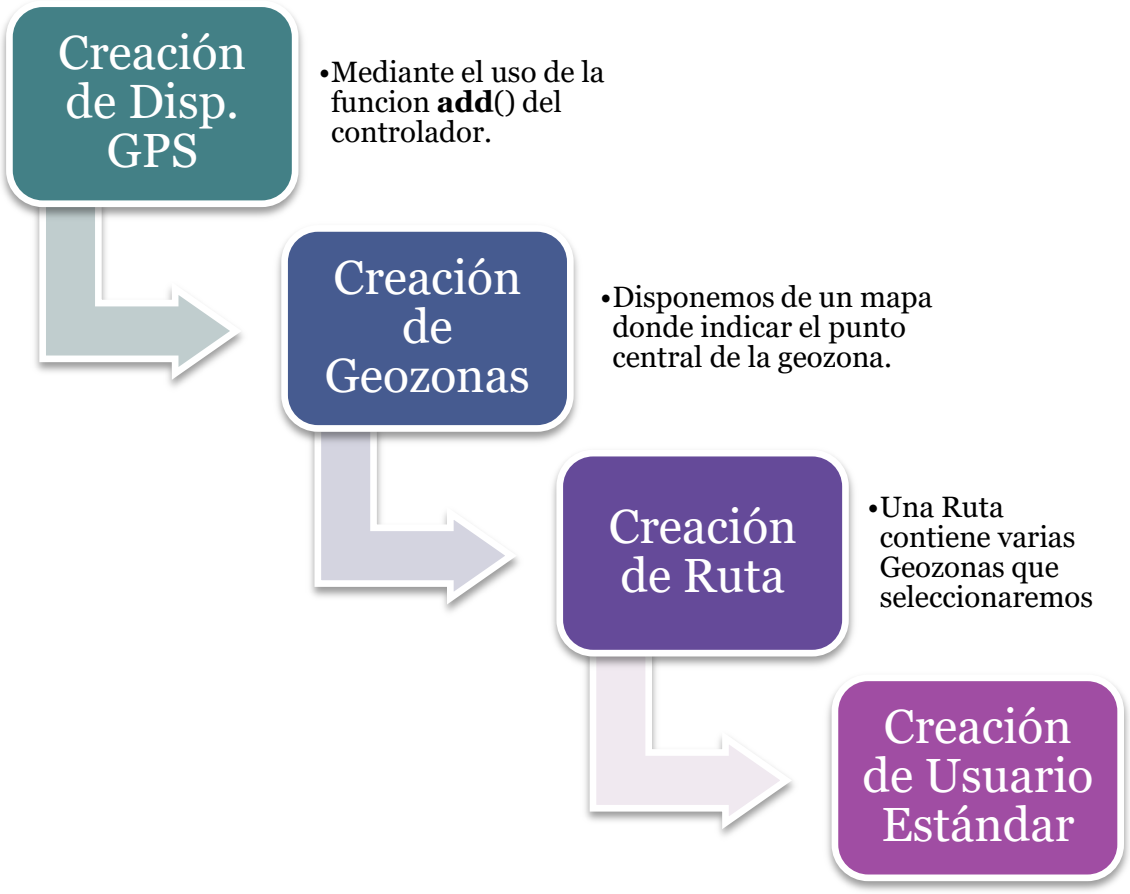
- Protección frente CSRF y manipulación de formularios: Simplemente con usar el componente *Security* ya obtenemos protección automática contra CSRF y contra la manipulación (*tampering*) de formularios, ya que el componente *Security* insertará automáticamente campos ocultos en nuestros formularios que posteriormente se revisarán para asegurar la correcta manipulación de éstos. Entre otras cosas no se permitirá enviar un formulario transcurrido cierto periodo de inactividad que se indicará con el campo `csrfExpires`.

Por otra parte en cuanto a temas de seguridad en el servidor, como puede ser la inclusión de un cortafuegos en la base de datos o el manejo frente ataques de denegación de servicios entre otros, al tratarse en nuestro caso de un servidor externo a la empresa la seguridad de éste corre a cargo del responsable de dicho servidor. La empresa Tecnoprotel-Elian S.L. fue consciente de este hecho en la contratación del servidor y por tanto realizó la elección de éste en consecuencia.

Para concluir este apartado solo resta comentar que, como puede observarse, el apartado de seguridad de la aplicación no estaba en el pliego de condiciones y por tanto se ha descuidado su inclusión, sin embargo, será un punto importante a tratar en futuras modificaciones en la aplicación.

6.5.10 Detalles de Funcionalidad

Llegados a este punto, en el que ya conocemos de qué manera se han realizado los controladores, las vistas, los modelos y cómo funciona el inicio de sesión en CakePHP, vamos a proceder a realizar una pequeña traza de las acciones que debe realizar el administrador para registrar con éxito un usuario estándar, con toda su configuración en el sistema.



Ahora ilustraremos esta traza con imágenes de todo el proceso:

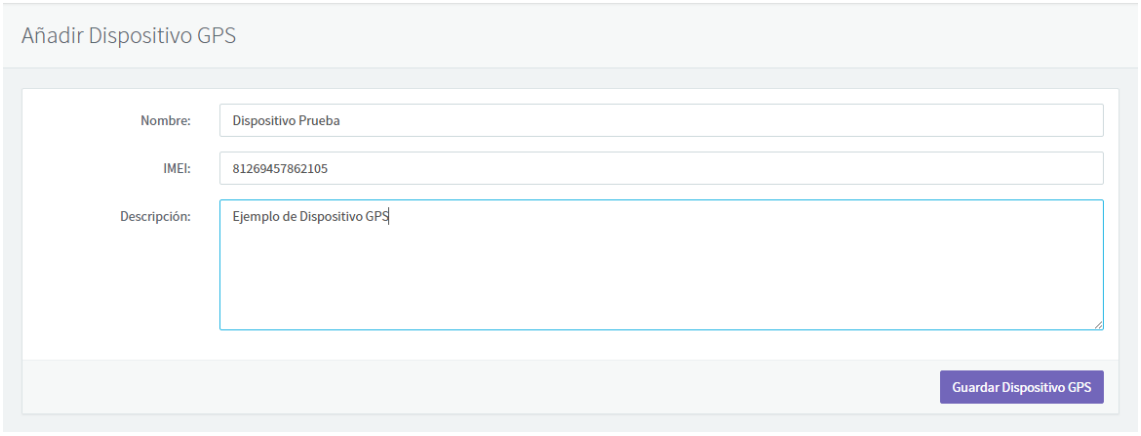


Ilustración 23: Vista añadir dispositivo GPS

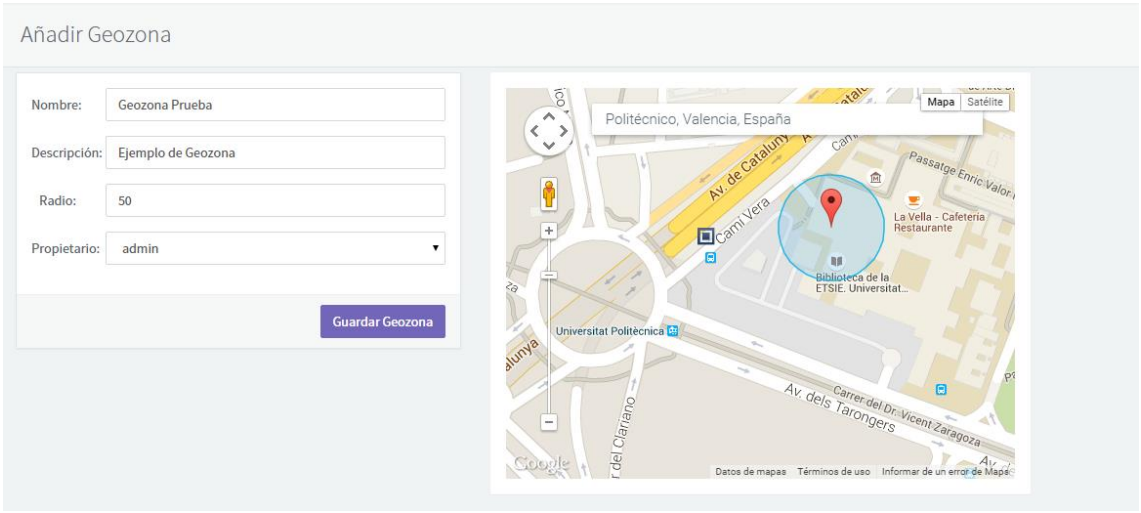


Ilustración 24: Vista añadir geozona

Ilustración 25: Vista añadir ruta

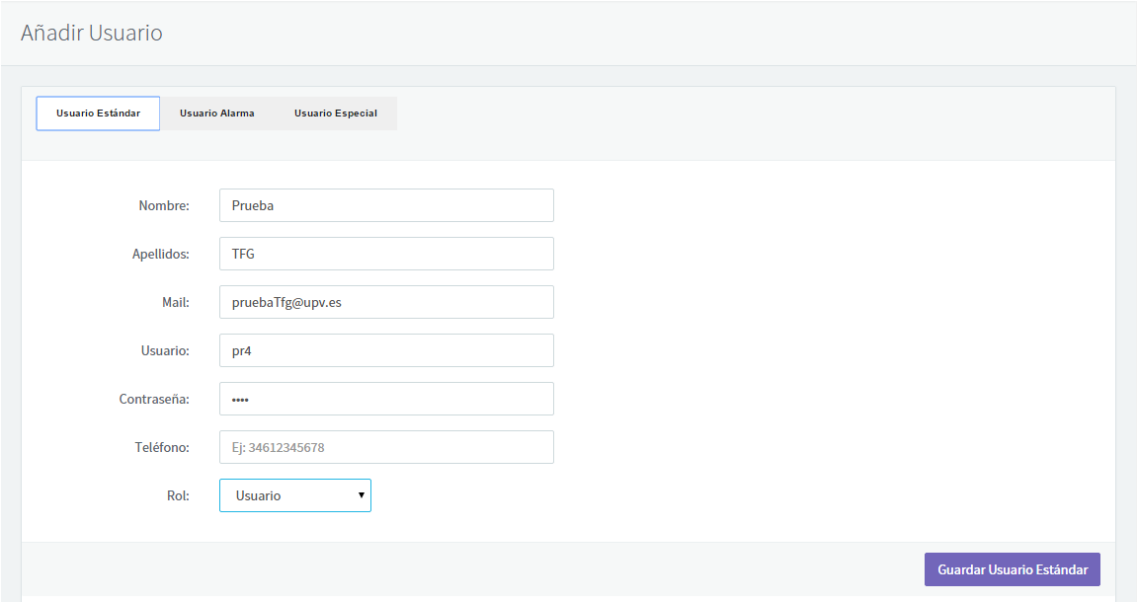


Ilustración 26: Vista añadir usuario estándar

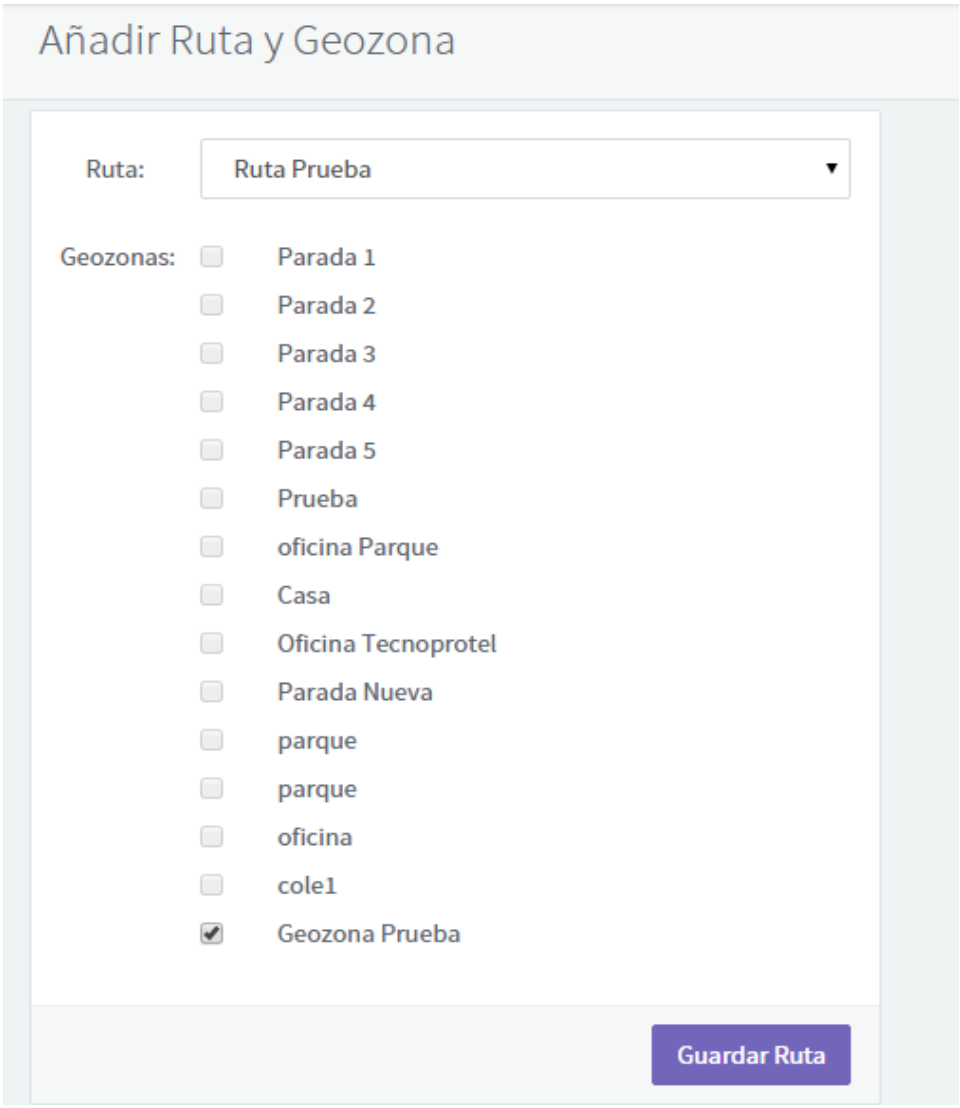


Ilustración 27: Vista selección ruta y geozona del usuario

Como aclaración al pulsar el botón de Guardar Usuario Estándar de la Ilustración 26 se nos redirige a la ventana de la Ilustración 27. En esta ventana solo es posible seleccionar las geozonas que se encuentren asignadas a la ruta seleccionada, las demás permanecerán inhabilitadas.

6.5.11 Recepción de Tramas

Como punto final de este apartado debemos profundizar en cómo se obtienen las tramas de los dispositivos GPS y se almacenan en nuestra base de datos.

Siguiendo las recomendaciones ofrecidas por la empresa Tecnoprotel-Elian para obtener los datos de geoposición de los dispositivos GPS se ha utilizado Node.JS para crear un servidor que escuche las diferentes tramas que envía cada dispositivo.

```
var mysql = require('mysql');


//***Conexion to DB***
//********************
var connection = mysql.createConnection({
  host     : '...',
  user     : '...',
  password : '...',
  database : '...'
});
```

En primer lugar debemos crear la conexión a nuestra base de datos para posteriormente ser capaces de almacenar correctamente los datos recibidos.

```
//***TCP Message Reception***
//***************************
var net = require('net');
var HOST = '0.0.0.0';
var PORT = 60001;

net.createServer(function(sock) {

    sock.on('data', function(data) {
      if(data.toString().match("##,imei:.*,A;")){
          var res = data.toString().split(",");
          var imei = res[1];
          sock.write('**,'+res[1]+',C,20s');
          sock.write('LOAD');
      }else if(data.toString().match("^\\d+")){
            sock.write('ON');
            console.log('ON Sended');
      }
          <-- Sigue a continuación -->
```

En este primer trozo del código de nuestro servidor TCP[19] podemos observar que creamos un socket que escuchará en la IP 0.0.0.0 (localhost) y en el puerto 60001 los mensajes TCP que se envíen. Añadimos un manejador para el evento “data”, el cual se activa siempre que haya datos en la conexión.

[19] TCP: Protocolo de control de transmisión, garantiza que los datos que transporta serán entregados sin error y en el orden en el que se produjo el envío.

Por último podemos observar que para el correcto funcionamiento de los dispositivos en determinadas situaciones es necesario enviar comandos a través de la conexión, vemos que se utilizan LOAD y ON. LOAD es necesario para indicar al dispositivo de que se le permite empezar a enviar los datos de las posiciones recibidas, normalmente este comando se manda cuando un dispositivo se conecta a nuestro servidor por primera vez, y ON es necesario para indicar al dispositivo que el servidor todavía permanece a la escucha, cada cierto tiempo el dispositivo manda una trama para comprobar si todavía se sigue recibiendo información en el servidor, de este modo si esto no fuese así el dispositivo GPS quedaría a la espera de un ON sin mandar posición para ahorrar batería.

```
else if(data.toString().match("##,imei:.*.tracker,A;")){
      var res = data.toString().split(",");
      var comdevid = res[0].toString().split(":");
      var devid = comdevid[1];
      var msg = res[1];
      var devdate = res[2];
      var admin_num = res[3];
      var info = res[4];
      var time = res[5];
      var code = res[6];
      var latitude = res[7];
      var posy = res[8];
      var longitude = res[9];
      var posx = res[10];
      var speed = res[11];
      var altitude = res[12];
      var date = new Date();
      //If we receive a SOS we cancel the repeat mode
      if(msg=="help me"){
         sock.write('**,'+res[0]+',E');
         console.log('SOS repeat disabled');
      }

      //***Transaction to DB***
      //***********************
      if(info == "F" ){
            var sql = "INSERT INTO frames (devid, msg,
            devdate, admin_num, info, time, code, latitude,
            posy, longitude, posx, speed, altitude, modified)
            VALUES ?";
            var values = [[devid, msg, devdate, admin_num,
            info, time, code, latitude, posy, longitude,
            posx, speed, altitude, date]];
            connection.query(sql, [values], function(err) {

            })
      }
}
```

Y por último nuestro servidor se encarga de dividir la trama recibida en variables que posteriormente, si el formato es el correcto, se almacenan en la base de datos.

Cabe mencionar que en esta sección del código también llevamos a cabo la cancelación de la repetición de la alarma, ya que si no lo hacemos recibiríamos la misma trama de alarma cada 3 min.

6.6 Aplicación Móvil

La aplicación móvil es una aplicación muy sencilla que muestra sobre un mapa de Google Maps[20] la posición actual del dispositivo así como la ruta que ha seguido hasta ese punto. Para empezar debemos observar la jerarquía de carpetas que presenta un proyecto en Phonegap

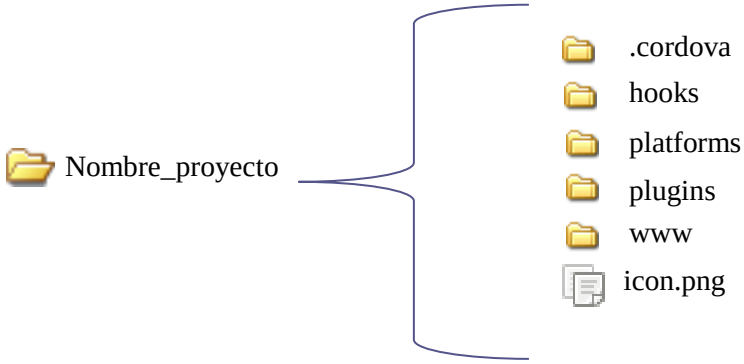


Ilustración 28: Jerarquía de carpetas de un proyecto Phonegap

Posteriormente describiremos como se realiza el intercambio de datos entre nuestra aplicación web y móvil y el funcionamiento de las notificaciones que se recibirán en la aplicación.

6.6.1 Desarrollo

Como podemos observar en la jerarquía presentada en la ilustración la carpeta www será la encargada de almacenar todo el código de la aplicación, siendo así muy similar a la estructura de una página web actual.

Dentro de la carpeta www se encuentra uno de los archivos más importantes en cuanto al correcto desarrollo de la aplicación, el archivo config.xml. Vamos a describir su contenido:

[20] Google Maps: Servicio de mapas web "web mapping", perteneciente a Google. Ofrece un API que permite manejar diferentes herramientas sobre el mapa.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!-- config.xml reference: https://build.phonegap.com/docs/config-xml -->
<widget xmlns     = "http://www.w3.org/ns/widgets"
        xmlns:gap = "http://phonegap.com/ns/1.0"
        id        = "com.phonegap.traceusGeo"
        version   = "1.0.5">

    <name>Traceus</name>

    <description>
        Aplicación para la Geolocalización de dispositivos.
    </description>

    <author href="http://traceus.es" email="info@traceus.es">
        Eduardo Yago Marco
        Javier Garrido Galdón
    </author>

    <!--
        Enable individual API permissions here.
        The "device" permission is required for the 'deviceready' event.
    -->
    <feature name="http://api.phonegap.com/1.0/device" />
<!—Continúa →
```

En esta primera parte del archivo podemos identificar varios atributos que compondrán posteriormente nuestra aplicación. Encontramos el paquete que la contendrá `"com.phonegap.traceusGeo"`, la versión de nuestra aplicación `"1.0.5"` y en la parte final vemos como se está cargando un *plugin* de la API de Phonegap, para ser más exactos, se trata del *plugin* `device`, que, como indica el texto comentado, es el encargado de recibir el evento *deviceready*, necesario para el correcto funcionamiento de muchos otros *plugins*.

```
<preference name="permissions"                value="none"/>

    <!-- Customize your app and platform with the preference element. -->
    <!-- <preference name="phonegap-version"      value="3.4.0" /> -->
<!-- all: current version of PhoneGap -->
    <preference name="orientation"                value="default" />
<!-- all: default means both landscape and portrait are enabled -->
    <preference name="target-device"              value="universal" />
<!-- … -->

    <!-- Define app icon for each platform. -->
    <icon src="icon.png" />
    <icon src="res/icon/android/icon-36-ldpi.png"   gap:platform="android"
gap:density="ldpi" />
    <!-- … -->

    <!-- Define app splash screen for each platform. -->
    <gap:splash src="res/screen/android/screen-ldpi-portrait.png"
gap:platform="android" gap:density="ldpi" />
    <!-- … -->
    <!--
        Define access to external domains.

        <access />
          - a blank access tag denies access to all external resources.
        <access origin="*" />
          - a wildcard access tag allows access to all external resource.

        Otherwise, you can specify specific domains:
    -->
    <access origin="http://127.0.0.1*"/> <!-- allow local pages -->
    <access origin="*"/> <!-- allow all external pages -->

</widget>
```

Por último en la segunda parte de este archivo encontramos las preferencias de nuestra aplicación, p.ej. la orientación, si se ejecutará en pantalla completa… A continuación vemos el apartado dedicado a los iconos de la aplicación y a los *splash screens*[21] separados por resolución y por plataforma. Para terminar llegamos a la definición de accesos a dominios externos, como se puede ver tanto el acceso a las páginas locales como a cualquier página externa está permitido. Este archivo xml será vital a la hora de crear nuestra apk.

6.6.2 Notificaciones

Para empezar vamos a explicar brevemente cómo funciona el envío de notificaciones a aplicaciones móviles desde nuestro servidor y posteriormente analizaremos como está implementada esta característica.

[21] Splash screens: Ventana de aplicación que se muestra al arranque de ésta y habitualmente muestra una imagen con el logo de la aplicación.

6.6.2.1 Notificaciones GCM (Android)

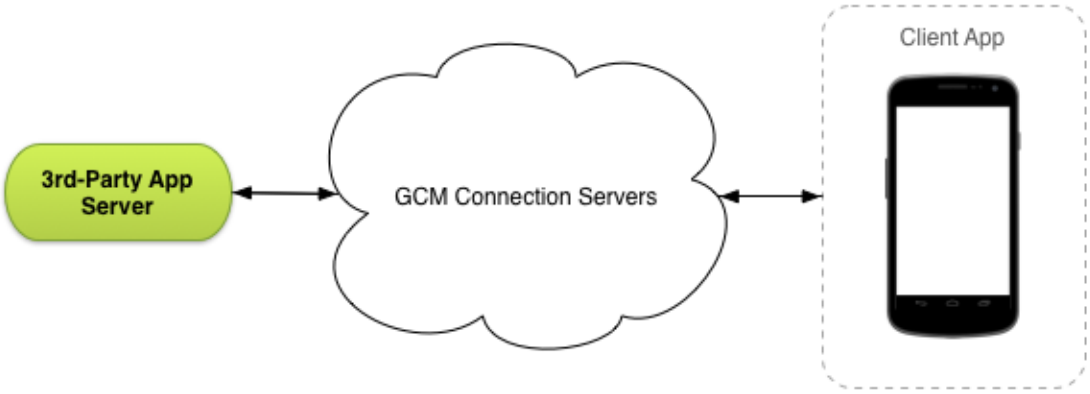


Ilustración 29: Arquitectura GCM [2]

A modo de resumen, lo que ilustra la imagen es el procedimiento básico para enviar una notificación a un dispositivo Android utilizando para ello Google Cloud Messaging (GCM).

El dispositivo móvil debe registrarse en el servidor GCM mediante el uso de la *Project key* perteneciente a nuestro proyecto, para conseguir esto debemos tener acceso a Google Developers Console y ahí crearnos un proyecto nuevo y añadimos el API *Google Cloud Messaging for Android*. Gracias a este identificador el dispositivo móvil será capaz de registrarse y obtendrá un nuevo identificador conocido como *registration token*, este nuevo identificador debe enviarse a nuestro servidor ya que será la forma de enviar notificaciones a ese dispositivo.

Una vez completados todos los pasos anteriores solo nos queda desde el servidor mandar el mensaje que deseamos en un formato especifico que nos indica Google en su documentación a su servidor GCM con el *registration token* del dispositivo y el servidor GCM será el encargado de enviarle esa notificación a ese dispositivo.

El registro se efectúa en la aplicación como se muestra a continuación:

```
function onDeviceReady() {
    deviceID = device.uuid;

    try
    {
        pushNotification = window.plugins.pushNotification;
        if (device.platform == 'android'
          || device.platform == 'Android' ||
                device.platform == 'amazon-fireos' ) {
                   pushNotification.register(successHandler, errorHandler,
                   {"senderID":"ProjectID”,
                           "ecb":"onNotification"});
        }
    }
    catch(err)
    {
        txt="Ocurrió un error en el dispositivo.\n\n";
        txt+="Error description: " + err.message + "\n\n";
    }
}

function onNotification(e) {

    switch( e.event )
    {
        case 'registered':
        if ( e.regid.length > 0 )
        {
            regID=e.regid;

        }
        break;
        ...

        default:
            alert('EVENT -> Unknown, an event was received and we do not
know what it is');
        break;
    }
}
```

Como podemos observar toda la tarea de registro recae sobre un *plugin* de Phonegap llamado *pushNotification* que con su función *register* realiza el registro en el servidor de GCM y nos lo devuelve en forma de objeto (`e.regid`), el cual nosotros almacenamos en una variable, al ejecutar el *callback*[22] `onNotification()`.

[22] Callback: Trozo de código que se pasa como argumento de una función. Esta función ejecutará ese código en un tiempo conveniente.

6.6.2.2 Notificaciones APNS (iOS)

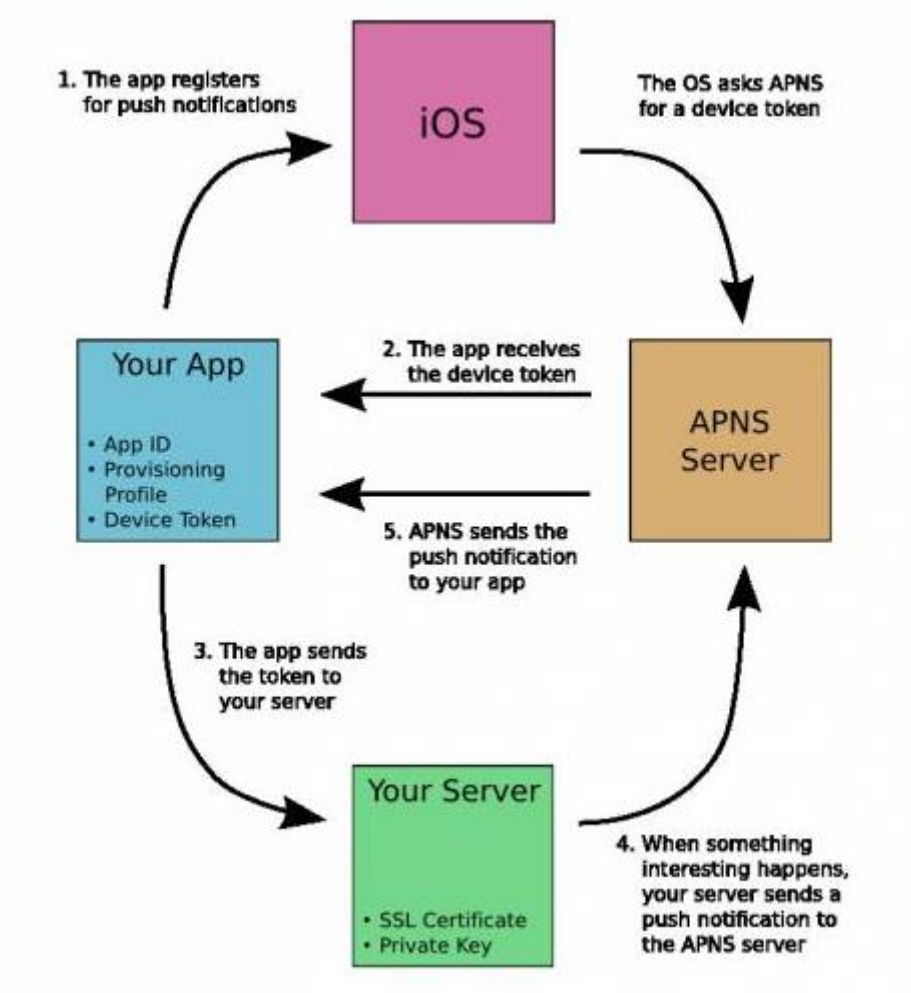


Ilustración 30: Arquitectura APNS [3]

Como podemos observar de un simple vistazo, la arquitectura que utiliza Apple para su Apple Push Notification Service (APNS) es similar a la vista en el caso de Google y su GCM.

La principal diferencia entre ambos sistemas de notificación es que en este caso no es necesario enviar ningún tipo de identificador para registrar nuestro dispositivo en los servidores de APNS. Como ocurría con las notificaciones GCM al producirse el registro los servidores de APNS nos suministran un identificador, llamado en la imagen *token*, que posteriormente debemos enviar a nuestro servidor para poder enviar notificaciones a ese dispositivo.

Una vez que nuestro servidor tenga el *token* en su poder simplemente resta crear un mensaje con las características que impone Apple, la más característica es la necesidad de genera un archivo de certificado .pem que se debe generar en una máquina con Mac OS, y enviarlo a su servidor APNS, el cual será el encargado de remitirlo al dispositivo.

El registro se efectúa de manera similar, gracias al plugin *pushNotification*:

```
function onDeviceReady() {
          deviceID = device.uuid;

          try
          {
          pushNotification = window.plugins.pushNotification;
          if (device.platform == 'android' || device.platform == 'Android'
                    ||device.platform == 'amazon-fireos' ) {
                              pushNotification.register(successHandler,
                              errorHandler, {"senderID":"ProjectID",
                                        "ecb":"onNotification"});
                    } else {
                    pushNotification.register(tokenHandler, errorHandler,
                     {"badge":"true","sound":"true",
                    "alert":"true","ecb":"onNotificationAPN"});
          }
     }
          catch(err)
          {
                    txt="Ocurrió un error en el dispositivo.\n\n";
                    //txt+="Error description: " + err.message + "\n\n";
                    navigator.notification.alert(txt);
          }
}

function tokenHandler (result) {
     regID=result;
}
```

Esta vez al tratarse de iOS el *plugin pushNotification* hace uso de su segundo tipo de *register* resultando este en un *callback* llamado *tokenHandler()* en el cual recibimos el identificador proporcionado por el servidor APNS y el cual nosotros almacenamos en una variable para su posterior envío a nuestro servidor.

6.6.2.3 Envío de notificaciones

El servidor es el encargado de comprobar si un dispositivo GPS se encuentra dentro de la zona definida en la geozonas y por tanto debe mandar una notificación a todos los dispositivos móviles de los usuarios que estén vinculados a ella. Esto lo realizamos mediante un fichero en PHP que se ejecuta cada minuto tal y como comentamos en el apartado Aplicación Web.

Se comprueban las tramas recibidas por los dispositivos GPS, si tienen su atributo *readed* a 0 quiere decir que están por leer, por tanto serán las que el fichero PHP recorra al ejecutarse.

Se calcula si la posición que recibe el gps está dentro de una geozona mediante la fórmula del Harvesine de la siguiente forma:

```php
function haversineGreatCircleDistance(
  $latitudeFrom, $longitudeFrom, $latitudeTo, $longitudeTo, $earthRadius =
6371000)
{
  // convert from degrees to radians
  $latFrom = deg2rad($latitudeFrom);
  $lonFrom = deg2rad($longitudeFrom);
  $latTo = deg2rad($latitudeTo);
  $lonTo = deg2rad($longitudeTo);

  $latDelta = $latTo - $latFrom;
  $lonDelta = $lonTo - $lonFrom;

  $angle = 2 * asin(sqrt(pow(sin($latDelta / 2), 2) +
    cos($latFrom) * cos($latTo) * pow(sin($lonDelta / 2), 2)));
  return $angle * $earthRadius;
}
```

Si el resultado de ejecutar la función con la longitud y latitud del GPS y las de la geozona definida es menor que el radio de la susodicha geozona podemos asegurar que el dispositivo GPS se encuentra en ese momento dentro de la geozona.

Por tanto de ser así debemos enviar la notificación pertinente a los usuarios autenticados en la aplicación móvil. Cómo explicaremos en el punto siguiente al realizarse la autenticación en la aplicación se nos envía el identificador necesario para mandar las notificaciones, por tanto en este punto disponemos de los identificadores de los usuarios que se hayan autenticado correctamente.

Realizamos el envío de las notificaciones usando dos funciones definidas en PHP. Se realiza esta separación ya que como se ha comentado anteriormente el mensaje que se envía a cada servidor de envío de notificaciones es totalmente diferente.

```
/* Send notifications to Android devices */

<?php
// API access key from Google API's Console
define( 'API_ACCESS_KEY', 'Our Google API Key' );

function pushnotify($id, $message, $title){
    $registrationIds = array($id);
    $msg = array
    (
        'message'    => $message,
        'title'      => $title,
        'vibrate'    => 1,
        'defaults'   => 0,
        'sound'      => "beep",
        'largeIcon' => 'large_icon',
        'smallIcon' => 'small_icon'
    );

    $fields = array
    (
        'registration_ids'   => $registrationIds,
        'data'           => $msg
    );

    $headers = array
    (
        'Authorization: key=' . API_ACCESS_KEY,
        'Content-Type: application/json'
    );

    $ch = curl_init();
    curl_setopt( $ch,CURLOPT_URL,
'https://android.googleapis.com/gcm/send' );
    curl_setopt( $ch,CURLOPT_POST, true );
    curl_setopt( $ch,CURLOPT_HTTPHEADER, $headers );
    curl_setopt( $ch,CURLOPT_RETURNTRANSFER, true );
    curl_setopt( $ch,CURLOPT_SSL_VERIFYPEER, false );
    curl_setopt( $ch,CURLOPT_POSTFIELDS, json_encode( $fields ) );
    $result = curl_exec($ch );
    curl_close( $ch );
}
```

Podemos ver como el primer requisito que se nos pide es tener un *API Key* de Google para poder enviar el mensaje a uno de sus servidores de GCM. Esta *API key* podemos obtenerla de Google Developers Console en el apartado APIs & auth en la pestaña Credentials. Una vez configurada la *key* simplemente le pasaremos el identificador único del dispositivo que nos remitió la aplicación móvil al registrarse en el servidor GCM, el mensaje a mostrar en la notificación y el título de ésta.

```
/* Send notifications to iOS devices */

<?php

function pushNotification($theMessage, $theDeviceToken)
{
    // Put your device token here (without spaces):
    $deviceToken = $theDeviceToken;

    // Put your private key's passphrase here:
    $passphrase = 'Pass for Private Key';

    // Put your alert message here:
    $message = $theMessage;

    $ctx = stream_context_create();
    stream_context_set_option($ctx, 'ssl', 'local_cert',
'../admin/app/webroot/files/cert.pem');
    stream_context_set_option($ctx, 'ssl', 'passphrase', $passphrase);

    // Open a connection to the APNS server
    $fp = stream_socket_client(
        'ssl://gateway.push.apple.com:2195', $err,
        $errstr, 60, STREAM_CLIENT_CONNECT|STREAM_CLIENT_PERSISTENT,
$ctx);

    if (!$fp)
        //exit("Failed to connect: $err $errstr" . PHP_EOL);
    exit("" . PHP_EOL);

    // Create the payload body
    $body['aps'] = array(
        'alert' => $message,
        'sound' => 'default'
        );

    // Encode the payload as JSON
    $payload = json_encode($body);

    // Build the binary notification
    $msg = chr(0) . pack('n', 32) . pack('H*', $deviceToken) . pack('n',
strlen($payload)) . $payload;

    // Send it to the server
    $result = fwrite($fp, $msg, strlen($msg));

    if (!$result)
        echo 'Message not delivered' . PHP_EOL;
    else
    fclose($fp);
}
?>
```

Aquí podemos apreciar las sutiles diferencias comentadas anteriormente, en este caso necesitamos disponer en nuestro servidor del archivo certificado, en nuestro caso el llamado `cert.pem` y por tanto debemos indicar en el campo `$passphrase` la contraseña que hemos utilizado al crearlo. Sólo nos queda indicar la ruta hasta dicho archivo y pasar a la función el mensaje que queremos enviar como notificación y el *token* correspondiente al dispositivo.

6.6.3 Autenticación y envío de parámetros.

Como hemos comentado en la sección anterior necesitamos enviar el identificador a nuestro servidor, para ello aprovecharemos la comprobación que efectuamos para autenticar a los usuarios y pasaremos este id como argumento.

```
function login(){
    inputuser = document.getElementById("user").value.toLowerCase();
    inputpass = document.getElementById("pass").value.toLowerCase();
    var req = new XMLHttpRequest();
    var url =
"http://www.traceus.es/admin/pass/save?user="+inputuser+"&pass="+inputpass
+"&devID="+deviceID+"&regID="+regID+"&plat="+isAndroid;
    req.open('GET', url, true);
    req.onreadystatechange = function (aEvt) {
        if(req.readyState == 4){
            if(req.status == 200){
                data_json = eval('(' + req.responseText + ')');
                for (i in data_json){
                    var _cond = data_json["cond"];
                    var _message = data_json["message"];
                }
                if(_cond == 3){
                    navigator.notification.alert(
                        _message,     // message
                        null,         // callback
                        'Número Máximo Alcanzado',    // title
                        'Aceptar'   // buttonName
                    );
                }else if(_cond == 4){
                    navigator.notification.alert(
                        _message,     // message
                        null,         // callback
                        'Error de identificación',    // title
                        'Aceptar'   // buttonName
                    );
                }else{
                    saveYshow();
                }
            }
        }
    };
    req.send(null);

}
```

Utilizamos un *XMLHttpRequest()* para realizar una petición GET a nuestro servidor pasando como parámetros: el nombre de usuario y la contraseña cifrada y en minúsculas, el Id del dispositivo que nos proporciona Phonegap mediante `device.uuid`, el identificador para las notificaciones y un valor booleano para determinar si se trata de un dispositivo Android o un dispositivo iOS.

6.6.4 Recepción de geolocalización

Ya que esta es la principal funcionalidad de nuestra aplicación debemos describir más en profundidad como se realiza la actualización de los datos en tiempo real.

```
function connection(user, pass){
    var data_json;
    var req = new XMLHttpRequest();
    var url =
"http://www.traceus.es/admin/pass/info?user="+user+"&pass="+pass;
    req.open('GET', url, true);
    if(!oFirstTimeLoaded){
        oFirstTimeLoaded = true;
        $("#main").mask("<img src='images/loading-128.gif' width='32'
height='32'><br>Cargando Geoposici&oacuten...");
    }else if(oMaskOpened == MASK_INFO || oMaskOpened == MASK_OPTIONS){
        $("#main").unmask();
    }
    oMaskOpened = MASK_RELOAD_MAP;
    $("#main").mask("<img src='images/connection-128.gif' width='32'
height='32'><br>Compruebe su conexi&oacuten a Internet...", 120000);
    req.onreadystatechange = function (aEvt) {
        if(req.readyState == 4){
            if(req.status == 200){
                data_json = eval('(' + req.responseText + ')');
                for (i in data_json){
                    var _codigo = data_json["code"];
                    var _message = data_json["message"];
                    _length = data_json["length"];
                    _latitude = data_json["latitude"];
                    _numPos = data_json["size"];

                }
                if(_codigo==2){
                    oMaskOpened = MASK_NONE;
                    $("#main").unmask();
                    oNavIsLocked = false;
                    _lengths = _length.toString().split(",");
                    _latitudes = _latitude.toString().split(",");
                    <. . .>

                }else if(_codigo==3){
                    oMaskOpened = MASK_NONE;
                    $("#main").unmask();
                    oNavIsLocked = false;
                    oMultipleRoutes = true;
                    _lengths = _length.toString().split(",");
                    _latitudes = _latitude.toString().split(",");
                    <. . .>

                }else{
                    deletelocal();
                }
            }else{
                console.log("Error read remote data");
            }
        }
    };
    req.send(null);
}
```

Al igual que ocurría con la función de autenticación, se realiza una petición GET mediante *XMLHttpRequest()* nuevamente indicando usuario y contraseña para conocer desde el lado del

servidor que usuario exacto realiza la petición. El servidor responde mediante objetos en JSON[23] con las posiciones ordenadas según el tipo de usuario que haya realizado la consulta.

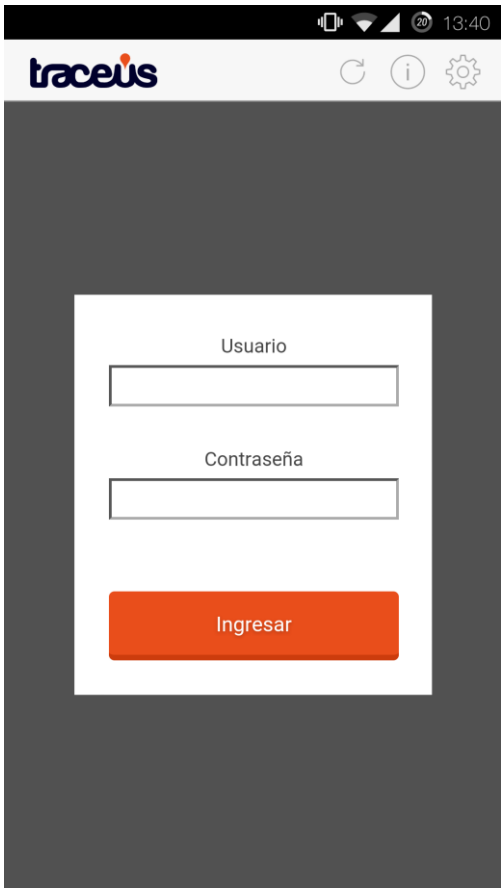
Téngase en cuenta que para que la posición vaya actualizándose con el tiempo es necesario realizar la función *connection()* dentro de un intervalo de tiempo. Esto se consigue mediante el uso de *setInterval()* como se muestra a continuación.

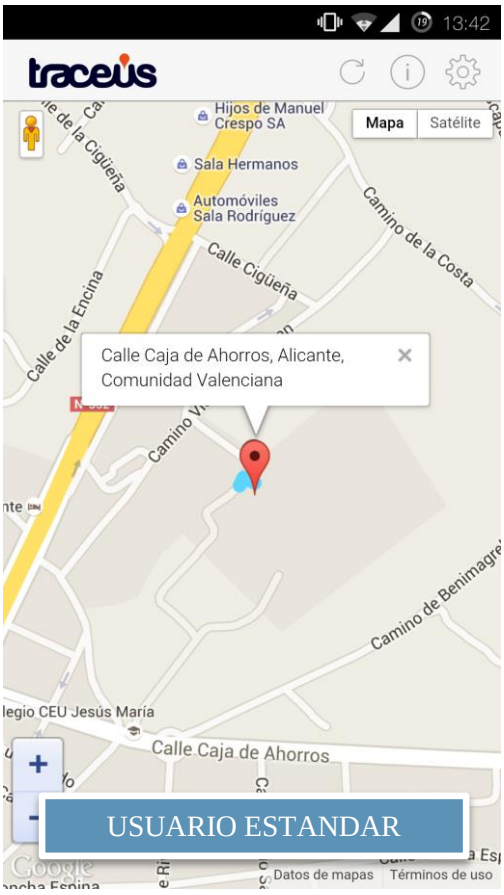
```
//Ejecutaremos connection con un intervalo de 30s
Interval = setInterval(function(){connection(usvalue, pavalue)}, 30000);

//Para dejar de ejecutar el intervalo
clearInterval(window.Interval);
```

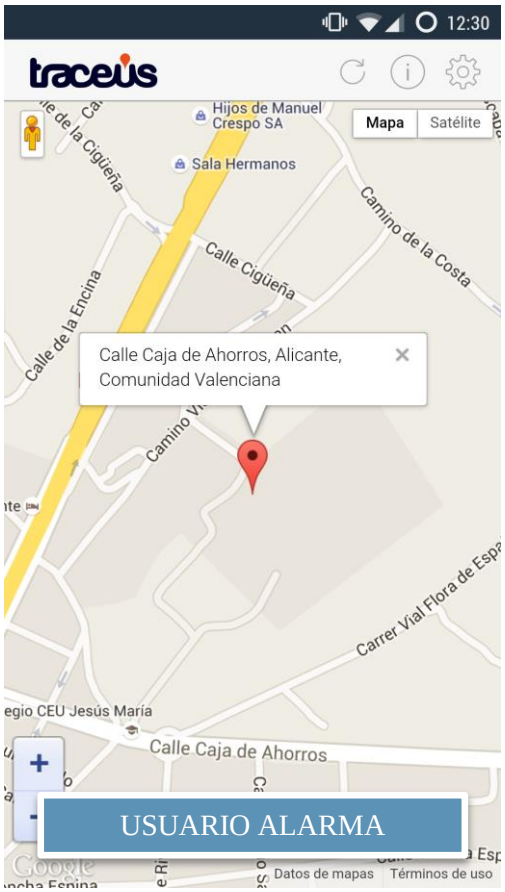
6.6.5 Aspecto Final

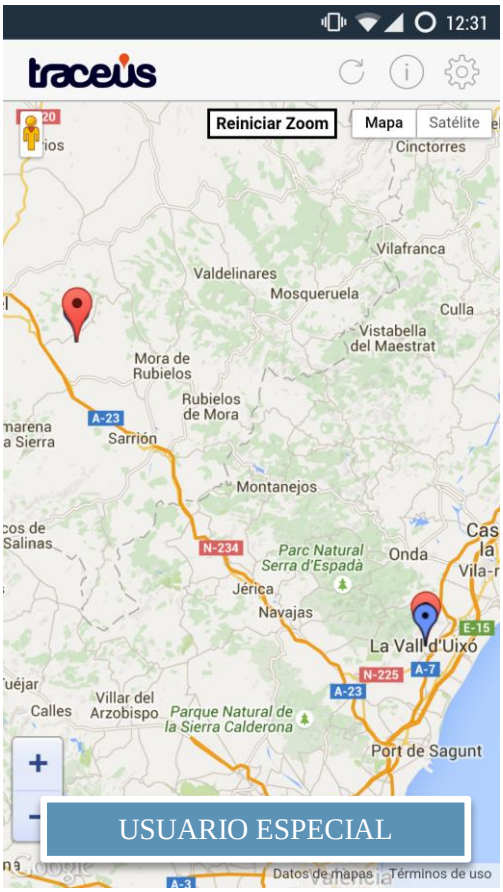
Vamos a mostrar el aspecto final de la aplicación móvil.

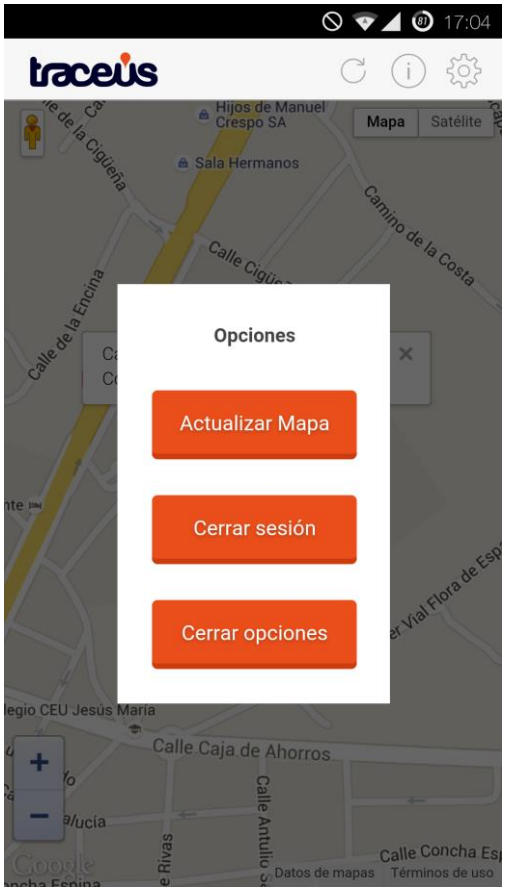


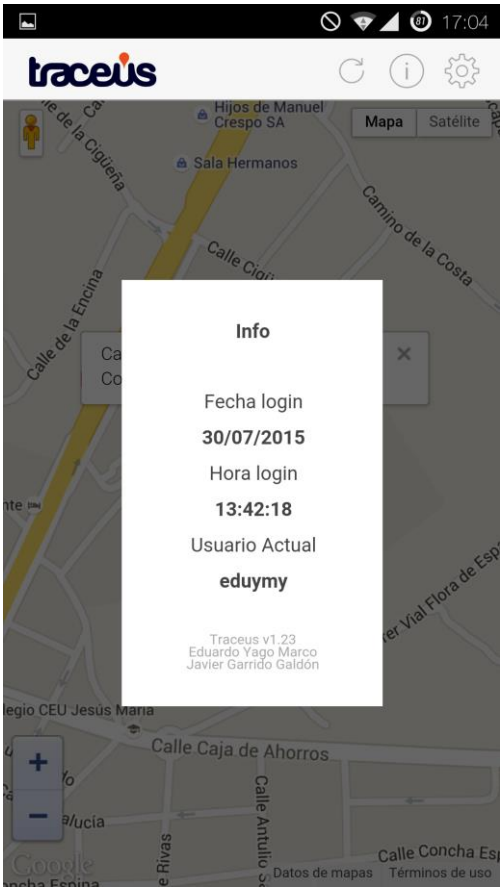


[23] JSON: *JavaScript Object Notation*, es un formato ligero de intercambio de datos cuya generación e interpretación es relativamente sencilla.


12:30
traceus
Mapa
Satélite
Hijos de Manuel Crespo SA
Sala Hermanos
Calle Cigüeña
Camino de la Costa
Calle de la Encina
Calle Caja de Ahorros, Alicante, Comunidad Valenciana
Carrer Vial Flora de Espa
Calle Caja de Ahorros
Datos de mapas
Términos de uso
USUARIO ALARMA


12:31
traceus
Reiniciar Zoom
Mapa
Satélite
Cinctorres
Vilafranca
Valdelinares
Mosqueruela
Culla
Vistabella del Maestrat
Mora de Rubielos
Rubielos de Mora
Sarrión
Montanejos
Parc Natural Serra d'Espadà
Onda
Jérica
Navajas
La Vall d'Uixó
Villar del Arzobispo
Parque Natural de la Sierra Calderona
Port de Sagunt
Datos de mapas
Términos de uso
USUARIO ESPECIAL


17:04
traceus
Mapa
Satélite
Hijos de Manuel Crespo SA
Sala Hermanos
Calle Caja de Ahorros
Opciones
Actualizar Mapa
Cerrar sesión
Cerrar opciones
Datos de mapas
Términos de uso


17:04
traceus
Mapa
Satélite
Hijos de Manuel Crespo SA
Sala Hermanos
Calle Caja de Ahorros
Info
Fecha login
30/07/2015
Hora login
13:42:18
Usuario Actual
eduymy
Traceus v1.23
Eduardo Yago Marco
Javier Garrido Galdón
Datos de mapas
Términos de uso

7. Test y Pruebas

En esta sección también debemos dividir las pruebas realizadas en ambas partes de nuestro proyecto (aplicación web y móvil) ya que se han realizado diferentes tipos de comprobaciones en cada caso.

7.1 Aplicación Web

Básicamente se pretende comprobar cuatro puntos fundamentales en nuestra aplicación: ¿Se reciben e interpretan correctamente los datos recibidos por los dispositivos GPS? ¿Se almacenan correctamente los cambios realizados en el sistema por los usuarios administradores? ¿Se muestran los datos correctamente por parte de los usuarios? Y el más importante de todos ¿el proceso de autenticación se realiza correctamente siendo imposible acceder a zonas no autorizadas para el usuario *logueado*?

Para resolver la problemática con los dispositivos GPS en primer lugar se realizaron pruebas para determinar que protocolo de red utilizaban para la comunicación, esto fue sencillo de resolver gracias al uso de sockets, de esta forma se confirmó que los dispositivos GPS envían la trama de datos mediante TCP. Para la correcta interpretación de los datos recibidos por parte del dispositivo nos remitimos a la información proporcionada por Tecnoprotel-Elian S.L.

El siguiente aspecto a tratar es ofrecer seguridad a los administradores de que cualquier cambio que realicen, ya sea añadir nuevos usuarios, crear rutas, etc., sea almacenado correctamente en el sistema con sus respectivos cambios en él. Para lograr esto nos apoyamos en las reglas de validación que nos ofrece CakePHP y por supuesto añadiendo código propio para comprobar que tanto la longitud como el tipo de datos sea el correcto.

Uno de los objetivos fundamentales de nuestra aplicación es que la interfaz visual de ésta sea altamente usable, al tratarse de una aplicación cuyo uso se realizará por una persona ajena a la misma es de especial interés que este usuario final no cometa fallos debido a un diseño pobre. Sabiendo esto, se ha puesto especial énfasis en este aspecto realizando pruebas de uso tanto con usuarios inexpertos como con usuarios con altos conocimientos en desarrollo web y por tanto aplicando las sugerencias de ambos.

Finalmente debemos realizar pruebas en la parte más delicada de nuestra aplicación que es la autenticación de usuarios. Para ello, como hemos comentado anteriormente, nos centramos en usar correctamente el componente Auth de CakePHP asegurándonos de distinguir entre usuarios, por medio de los roles, y creando vistas diferenciadas para cada uno de ellos. También hemos tenido en cuenta notificar al usuario de que el usuario o contraseña introducidos no sean correctos a la hora de autenticarse.

7.2 Aplicación Móvil

En este caso deseamos comprobar dos puntos críticos en nuestra aplicación: ¿Se realiza correctamente la comunicación entre la aplicación móvil y el servidor? ¿El sistema de notificaciones funciona correctamente?

Ya que la aplicación móvil basa su funcionalidad en la comunicación con el servidor consideramos que ésta es la parte más delicada y por tanto la que debemos comprobar minuciosamente. Como se comentó en el apartado 6.6 para la comunicación se utilizan *XMLHttpRequest()*, para asegurarnos de que la comunicación se realiza correctamente fue necesaria la inserción de códigos para cada posible fallo en la respuesta del servidor. De esta manera es relativamente sencillo comprobar que fallo obtenemos, en caso de error, simplemente mirando el código de respuesta recibido en la aplicación móvil.

Como punto final en la aplicación móvil debemos asegurarnos de que tanto el envío como la visualización de las notificaciones se realizan correctamente tanto en los dispositivos Android como en los dispositivos iOS. Ya mencionamos que todo el sistema de notificaciones del dispositivo móvil lo gestionaba un plugin de Phonegap, pues acompañando a este plugin el creador nos suministra dos archivos de test, desarrollados en Ruby[24], con los que mediante una configuración previa podemos comprobar que las notificaciones funcionan adecuadamente.

[24] Ruby: Lenguaje de programación dinámico de código abierto centrado en la simplicidad y productividad

8. Prospección de Futuro

Actualmente tanto la aplicación web como la aplicación para dispositivos móviles se encuentran disponibles y en funcionamiento. La empresa Tecnoprotel.Elian S.L ha conseguido que varios colegios de la comunidad valenciana prueben la plataforma Traceus, gracias a esto hemos obtenido información para seguir mejorando todo el sistema.

Al tratarse esta plataforma de un sistema protegido por patente, adquirida por la empresa, se pretende que en un futuro no muy lejano empiece a extenderse su utilización por el territorio nacional.

9. Conclusión

Una vez finalizado el proyecto llegamos a ciertas conclusiones tanto relacionadas con el uso del software necesario para su realización como respecto a la consecución de objetivos.

En primer lugar destacar que el aprendizaje durante el tiempo en el que se ha realizado el proyecto ha sido uno de los puntos fundamentales de éste. CakePHP ha demostrado ser un *framework* PHP muy potente y con una gran comunidad detrás, cosa que en mi opinión es muy importante hoy en día ya que se tiende a buscar soluciones mediante motores de búsqueda web y esto sin una comunidad grande no sería factible. Es cierto que puede resultar difícil en un primer contacto, pero una vez se adquiere algo de habilidad con él es muy sencillo de entender, muy completo y, siguiendo las convenciones que ofrece, muy ordenado y limpio en cuanto a código.

Análogamente el *framework* Phonegap ha sido toda una sorpresa en mi opinión, ya que previamente a la realización de este proyecto desconocía su existencia. Si únicamente necesitamos aplicaciones móviles que no requieran un control muy específico del dispositivo utilizar este *framework* puede ahorra muchísimo trabajo. Como sucedía con CakePHP, Phonegap es muy popular en la actualidad y gracias a su comunidad cada día se van incorporando nuevos *plugins* que dotan a este *framework* de nuevas funcionalidades. Personalmente lo que más me ha llamado la atención es la facilidad que ofrece al poder utilizar el mismo código para aplicaciones en dispositivos de diferentes sistemas operativos.

Por otra parte se han conseguido los objetivos que se propusieron antes del inicio de este proyecto. Ambas aplicaciones se encuentran disponibles y en funcionamiento en la actualidad, cabe destacar que a día de hoy se siguen realizando mejoras tanto funcionales como estéticas. Nuestra plataforma ofrece al usuario un servicio básico de localización y alertas sobre la posición del autobús escolar que se solicite.

Por ultimo a nivel personal la realización de este trabajo me ha supuesto un incremento en mi organización y estructura a la hora de escribir código, adquirir nuevas competencias y nuevos conocimientos centrados en el desarrollo web y por último conocer en primera persona como es el trabajo tanto a nivel personal como a nivel técnico en una empresa.

10. Bibliografía

[1]Glinz, M. (2007). On-Non-Functional Requeriments. *15th IEEE International Requirements Engineering Conference* (págs. 21-26). DOI 10.1109/RE.2007.45.: IEEE Computer Society.

[2]Google. (24 de Julio de 2015). Obtenido de Google Developers: https://developers.google.com/cloud-messaging/gcm

[3]Hafizji, A. (23 de Mayo de 2013). *Apple Push Notification Services in iOS 6 Tutorial: Part 1/2*. Obtenido de Raywenderlich: http://www.raywenderlich.com/32960/apple-push-notification-services-in-ios-6-tutorial-part-1

[4]Mamani, N. A. (Mayo de 1999). Integrando requisitos no funcionales a los requerimientos basados en acciones concretas. Rio de Janeiro, Brasil: Departamento de Informática (PUC).

[5]Moret Martínez, J. V. (Septiembre de 2014). Aplicación Web de bases de datos usando el Framework CakePHP. Valencia, Comunidad Valenciana, España.

[6]Rojo, S. d. (29 de Octubre de 2012). Requerimientos no funcionales para aplicaciones web. La Plata, Buenos Aires, Argentina.

[7]Shenzhen Xexun Technology Co. (Agosto de 2012). User Manual. *GSM/GPRS/GPS Tracker*. Guangdong, China.

[8]Skvorc, B. (28 de Marzo de 2015). *Best PHP Framework for 2015*. Obtenido de Sitepoint: http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/