Universidad Zaragoza

# Trabajo Fin de Grado

## Prevención de accidentes mediante un sistema colaborativo basado en una aplicación móvil


Autor

Jorge Martínez Lascorz

Director

F. Javier Zarazaga Soria

Escuela de Ingeniería y Arquitectura



2016

# PREVENCIÓN DE ACCIDENTES MEDIANTE UN SISTEMA COLABORATIVO BASADO EN UNA APLICACIÓN MÓVIL

## RESUMEN

El objetivo de este proyecto final de grado es dar una solución a un problema real existente en el ámbito de la seguridad vial. Concretamente, la finalidad principal de este TFG es ayudar a los usuarios en los desplazamientos por carretera. En ocasiones, la visibilidad juega un papel muy importante en los accidentes, así como las condiciones meteorológicas, los reflejos o la atención en la conducción.

El proyecto se basa en los principios de la economía colaborativa adaptados al entorno de la seguridad vial. Tiene como objetivo crear un sistema en el que todos los usuarios compartan información entre ellos, en este caso la ubicación y la velocidad actual, para poder conocer la posición actualizada en tiempo real de los usuarios que estén alrededor del propio cliente, y así evitar los problemas de visibilidad anteriormente mencionados. Para ello, la tareas que se han desarrollado han sido las siguientes:

- Creación de una base de datos no relacional indexada en base a los parámetros encargados de representar la localización de un usuario.
- Creación de un servidor intermediario entre el cliente y la base de datos capaz de realizar búsquedas geoespaciales y cálculo de distancias.
- Creación de un cliente para móviles que monitorice la ubicación del usuario en cortos intervalos temporales, así como obtener la información de los usuarios cerca de nuestra ubicación y poder representarlos en un mapa.
- Introducción de los principios de *'geofence [1]'* en el cliente con el fin de poder alertar al usuario en el momento en el que algún otro cliente acceda dentro del radio de monitorización.
- Permitir el uso de la aplicación en "segundo plano" para permitir una monitorización secundaria permanente.
- Crear un sistema de anonimato para mantener la privacidad de cada uno de los usuarios, independientemente de su posición, acceso, etc…
- Adaptar el rendimiento de la aplicación para facilitar el uso en cualquier tipo de smartphone, dentro del amplio abanico de procesadores existente.
- Reducir al máximo la tasa de datos necesaria para la conexión entre cliente/servidor debido a la gran cantidad de peticiones necesarias para su funcionamiento.

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

# DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. JORGE MARTINEZ LASCORZ,

con nº de DNI 17771032-J en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado, (Título del Trabajo) Prevención de accidentes mediante un sistema colaborativo basado en una aplicación móvil,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 21 de Marzo de 2016

Fdo: Jorge Martínez Lascorz

# ÍNDICE

# AGRADECIMIENTOS

A F. Javier Zarazaga, por llevar como director el desarrollo de este proyecto. Ha sido de gran ayuda su la labor de supervisión y los múltiples consejos sobre nuevas ideas a implantar.

A toda mi familia, que ha ayudado desde el comienzo a la realización pruebas de la aplicación en entornos reales y han sufrido la evolución hasta el final, indicando bugs, fallos o mejoras.

A todos los profesores del grado de ingeniería informática, sin ellos todo este proyecto no hubiera sido posible.

# 1 RESUMEN EJECUTIVO

GeoMe es una aplicación para dispositivos móviles Android cuya finalidad principal es prevenir accidentes, enviando alertas en tiempo real a los usuarios de la posición geográfica proporcionada por los propios usuarios. Su funcionalidad se basa en un sistema colaborativo basado en propiedades del geofencing, gracias a la ubicación GPS proporcionada por el dispositivo de todos los usuarios. Cada usuario de la aplicación tiene dos roles: servir su posición actual a los demás usuarios, y recibir la posición de los usuarios alrededor suyo. La aplicación es capaz de distinguir automáticamente tres tipos de usuarios: usuarios peatones, usuarios en bicicleta y usuarios conductores de vehículos motorizados, dependiendo de la velocidad media de desplazamiento tomada en varios puntos del trayecto. Todos los usuarios pueden configurar alertas en tiempo real de la localización proporcionada por los demás usuarios, es decir: alertas al acercarse un peatón, un ciclista o un vehículo, todo ello gracias al sistema colaborativo de posición de los usuarios.

Además de generar alertas, la aplicación muestra en tiempo real la ubicación de los usuarios alrededor de nuestra ubicación en un mapa ofrecido por el servicio de Google Maps. Gracias a él podremos ver qué tipos de usuarios hay a nuestro alrededor: peatones, ciclistas o vehículos. Un menú de Ajustes es el encargado de configurar todos los parámetros de la aplicación, es posible configurar: los tipos de alertas que queremos recibir, modificar el radio (en metros) alrededor del usuario en el que se activan las alertas, así como el tipo de alertas (sonido y/o vibración), configurar parámetros de rendimiento, etc…

La aplicación puede ejecutarse en segundo plano, dejando un proceso activo en el dispositivo encargado de realizar todas las funcionalidades mencionadas: monitorizar la ubicación actual y recibir alertas.

El sistema está preparado para ejecutarse en dispositivos Android con un sistema operativo 4.1 JellyBean o superior, por lo que abarca un 96.0% del *'mercado Android[2]'*, lo que es igual a un total de 1999,392 millones de usuarios disponibles. Por otro lado, la aplicación dispone de configuraciones de rendimiento, permitiendo su uso en móviles de todas las características, desde la gama alta hasta dispositivos de bajo rendimiento.

# 2 INTRODUCCIÓN

## 2.1 Contexto del TFG

Las bases de este proyecto se realizaron en el entorno de desarrollo del *'Grupo de Sistemas de Información Avanzados (IAAA) [3]'* en los laboratorios ubicados en el edificio Ada Byron de la EINA. Las principales ramas de investigación de este grupo se centran en tecnologías de software abierto, distribuido e interoperable, principalmente mediante servicios web y para sistemas de información geoespacial, abarcando áreas como Sistemas de Información Geográfica (SIG), Teledetección, Servicios Basados en la Localización (SBL) y, con una atención especial a las Infraestructuras de Datos Espaciales (IDEs).

Para este trabajo principalmente se han utilizado los conocimientos aprendidos sobre la creación y puesta en funcionamiento de nuevos servicios y aplicaciones, integrando utilidades de geoprocesamiento en sistemas de información geográfica.

## 2.2 Contexto Tecnológico

Para una mejor compresión del proyecto son necesarias unas breves explicaciones de su ámbito tecnológico que permita una visión global de su alcance y repercusión.

Hoy en día la *'economía colaborativa [4]'* está en auge gracias a la facilidad para comunicarnos que permiten los smartphones e internet. La idea se basa en poner en contacto a usuarios para compartir ciertos recursos y ahorrar dinero. Una gran cantidad de empresas como AirBnb , BlaBlaCar, Parkeo, Uber etc… se basan en este sistema para compartir todo tipo de productos: vehículo, alojamiento, plaza de parking… Intentar implementar estos principios en el ámbito de la seguridad ha sido uno de los objetivos llevados a cabo en este proyecto, compartiendo, en este caso, la propia ubicación entre los usuarios cercanos.

## 2.3 Motivación y problema que se aborda

La idea de prevenir accidentes con ayuda de la tecnología siempre ha sido uno de los objetivos principales de la seguridad vial. Desde los inicios de la automoción se usan los semáforos para prevenir accidentes en los cruces. ¿Porqué no ir un paso mas allá con la monitorización de vehículos?

Imaginemos que nos desplazamos en bicicleta por una carretera con poco arcén. En escenarios de poca visibilidad ocasionados por las condiciones meteorológicas (niebla, lluvia extrema…), así como en momentos en los que es imposible ver lo que nos

vamos a encontrar (rasantes, curvas cerradas…), estamos en potenciales situaciones de riesgo. El sistema desarrollado es una ayuda para prevenir este tipo de problemas. Aún más, permite monitorizar la ubicación de peatones y ciclistas, por lo que ayudará en condiciones de baja luminosidad. También será útil para usuarios que hayan sufrido un accidente o avería y tengan el vehículo en el arcén o tengan que dirigirse hacia un puesto de S.O.S. Las aglomeraciones también son un problema que se minimizará gracias al sistema colaborativo, ya que alertará al usuario con el fin de evitar una colisión múltiple.

Si se realiza una búsqueda exhaustiva en el mercado de Android (Play Store) mediante descriptores relacionados con la seguridad vial, la mayoría de aplicaciones disponibles tratan sobre juegos educativos vinculados al aprendizaje de normas de seguridad vial y aprendizaje de señales de tráfico.

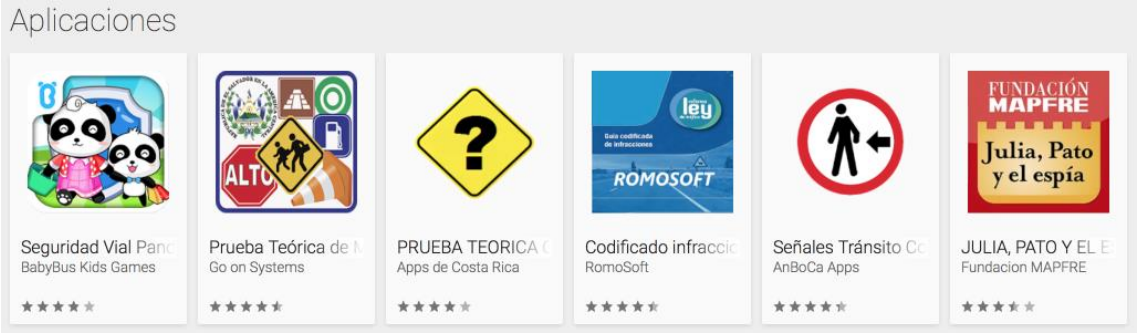


*Figura 1: Ejemplo de aplicaciones disponibles*

Todas estás aplicaciones son de temática similar. Destacando por ejemplo *'Guia de Educación Vial [5]'*, que dispone de un menú en el cual se pueden leer consejos para evitar accidentes de tráfico:


*Figura 2: Captura de pantalla de la aplicación "Guia de Educacion Vial"*

En caso de hacer búsquedas relacionadas con la monitorización, los resultados obtenidos son aplicaciones para localizar el dispositivo cuando éste se ha perdido, o aplicaciones que muestran el recorrido realizado por el dispositivo, generando una ruta visible a través de un mapa.

*'LocationTracker [6]'* es un ejemplo de ello, la aplicación permite monitorizar la ubicación del dispositivo durante un intervalo de tiempo, trazando la ruta realizada sobre un mapa:

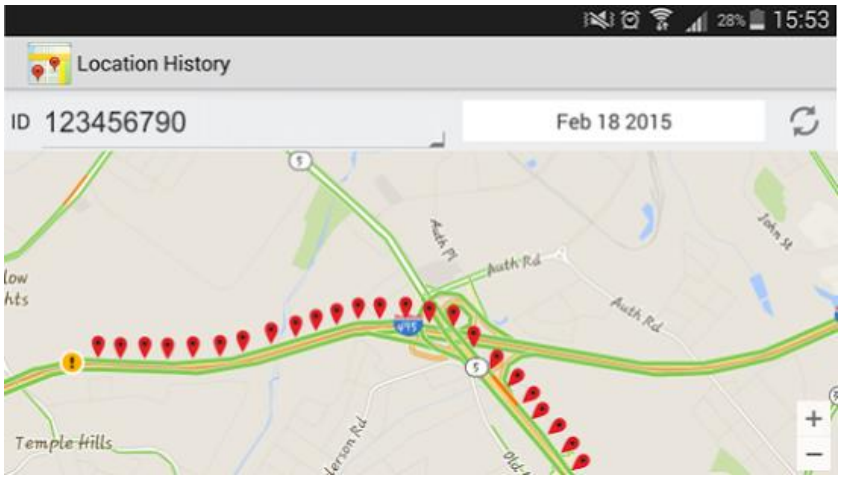


*Figura 3: Captura de pantalla de la aplicación "LocationTraker"*

Por último se han encontrado aplicaciones en las que se puede compartir la ubicación con los distintos amigos, aunque no con fines de seguridad vial. *'Map my Friends [7]'* realiza de forma similar al proyecto una monitorización de usuarios, pero no de forma global ya que solo permite ver la ubicación de usuarios "amigos".

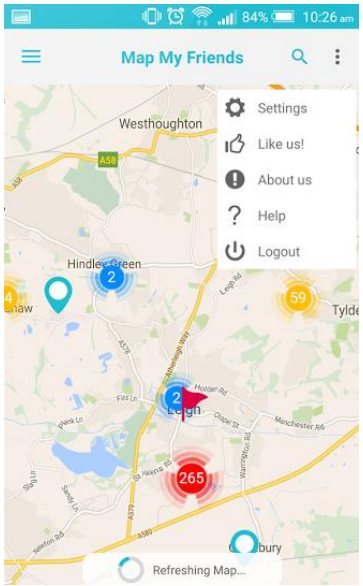


*Figura 3: Captura de pantalla de la aplicación "Map my friends"*

No se tiene conocimiento de la existencia de ningún sistema o aplicación parecido al desarrollado en este trabajo. En la actualizad sólo disponemos de ayudas físicas. En condiciones de poca visibilidad lo único disponible para usar son los distintos tipos de luces presentes en los vehículos (largas, cortas, niebla…), pero en caso de ser peatón o ciclista la cosa cambia, como mucho es posible usar un chaleco reflectante para alertar a los conductores de su posición. En curvas cerradas o rasantes, una técnica utilizada por los conductores en tocar el claxon cierto tiempo antes de tomar la curva, con la finalidad de avisar e indicar la posición en caso de que se encuentre un vehículo de frente. En ocasiones existen espejos en las curvas cerradas que permiten ver lo que hay al otro lado.

*Figura 5: Ejemplo de espejo en curva cerrada*

Como se puede apreciar, todas estas técnicas tienen una característica en común, alertar a los demás usuarios de la calzada sobre nuestra posición actual. Es por ello que en este proyecto se ha intentado dar un paso más allá, intentando aprovechar la popularidad de los smartphones y tecnologías como el GPS para llevar una monitorización de la ubicación de todos los usuarios que pueden realizar un desplazamiento por carretera.

Seguidamente se presenta una detallada lista de los requisitos que se abordan en todo el proyecto.

**Requisitos Funcionales**

- La aplicación mostrará un mapa basado en los servicios de *'Google Maps [8]'*.
- La aplicación mostrará en tiempo real la ubicación actual del usuario en el mapa, utilizando un pequeño icono superpuesto sobre el mapa.
- La aplicación mostrará en tiempo real los "*N*" usuarios en un radio de "*X*" metros alrededor de la ubicación del usuario, utilizando un pequeño icono superpuesto sobre el mapa.
- La aplicación será capaz de distinguir automáticamente entre usuarios en vehículos, usuarios en bicicleta y usuarios peatones. Dependiendo del tipo de usuario, el icono a mostrar en el mapa variará.
- La aplicación indicará las alertas al usuario a través de medios visuales, sonoros y físicos (vibración).
- Las alertas se generarán cuando otro usuario entre dentro del radio de muestreo del propio usuario.
- La aplicación permitirá configurar los tipos de alerta, el radio de muestreo, el número de usuarios a mostrar, y el intervalo de tiempo en el que se muestran los usuarios cercanos.
- La aplicación permitirá mantener la persistencia en los ajustes establecidos por el usuario.
- La aplicación deberá ser capaz de realizar todo su funcionamiento tanto en primer plano como en segundo plano.

**Requisitos No Funcionales**

- El icono de la ubicación del usuario será de color azul.
- El icono de un usuario vehículo será de color rojo.
- El icono de un usuario ciclista será de color amarillo.
- El icono de un usuario peatón será de color verde.
- Entre los avisos disponibles, se encontrarán: "Se acerca vehículo", "Se acerca ciclista", "Se acerca peatón" y "Aglomeración de vehículos"

## 2.4 Alcance, objetivos y limitaciones

El principal problema que repercute al alcance de este proyecto es la diversidad de smartphones con S.O Android. El rendimiento y la potencia de los procesadores es una diferencia que hay que tener en cuenta en este proyecto, ya que es necesario obtener información en tiempo real y trabajar con ella para poder visualizarla en un mapa. Estos componentes necesitan bastantes recursos si se desea conseguir la máxima fidelidad a la realidad (mostrar los usuarios alrededor en intervalos de tiempo muy cortos), es por ello que se ha gestionado el rendimiento de la aplicación a través de una opción en el menú “Ajustes” para conseguir abarcar todos los dispositivos posibles.

La privacidad es otra de las preocupaciones de todos los usuarios. Para evitar esto se ha reducido al mínimo la información almacenada sobre los usuarios activos de la aplicación, así como la identificación de éstos. En los siguientes apartados se explicará el funcionamiento de la gestión del anonimato.

Por parte del servidor, la constante monitorización de usuarios hace que el servidor tenga que atender a una gran cantidad de peticiones. A pesar de ser peticiones sencillas, el gran número de peticiones por minuto es importante tenerlo en cuenta, por lo que la escalabilidad es otro de los apartados críticos del sistema.

Finalmente es necesaria la ubicación por GPS para el funcionamiento de la aplicación, por lo que su uso se restringe a entornos con internet y cobertura GPS.

## 2.5 Herramientas de trabajo

El presente proyecto se ha desarrollado haciendo uso de las siguientes herramientas:

- Equipo de trabajo: Macbook Pro 13'
- Sistema Operativo: OS X El Capitan 10.11.4
- Versión de Java: Java Version 1.8.0_73-b02
- IDE: Android Studio 1.5.1
- IDE: IntelliJ IDEA 15 CE
- IDE: Atom v1.7.4
- Spring Framework 1.3.0 RELEASE: Framework de desarrollo de aplicaciones y contenedor de inversión de control utilizado para desarrollar el servidor. (Ver apartado *'3.3 Servidor'*)
- MongoDB 2.4: Sistema gestor de base de datos utilizado en el proyecto. La elección del mismo y sus ventajas se explican en el apartado *'3.2 Base de datos'*.
- Android SDK 16-23 (4.1 Jellybean – 6.0 Marshmallow): Versiones de Android soportadas. Se detalla su uso en el apartado *'3.7 Cliente Android'*.

## 2.6 Esquema general de la memoria del proyecto

El resto de la presente memoria se estructura de acuerdo a las siguientes secciones y anexos:

**Capítulo 3:** en este apartado se detallarán todas las tecnologías utilizadas en el proyecto, así como las distintas implementaciones que se han hecho de cada una de ellas. También se presentará una visión general de la arquitectura del sistema, destacando sus principales componentes. También se describe la metodología de trabajo seguida y las características que hacen de este proyecto un sistema escalable.

- Apartado 3.1: Diagramas de la arquitectura del sistema
- Apartado 3.2: Esquema de la base de datos, índices de indexación y parámetros geoespaciales.
- Apartado 3.3: Creación del servidor e interfaz de consultas.
- Apartado 3.4: Metodología de comunicación e interfaz ofrecida por el servidor.
- Apartado 3.5: Detalle del sistema de anonimato en base a la configuración de los mensajes intercambiados.
- Apartado 3.6: Selección de servicio para despliegue cloud y gestión.
- Apartado 3.7: Descripción del cliente Android, detalle del funcionamiento, ciclo de vida, ajustes disponibles y gráficas de rendimiento.
- Apartado 3.8: Metodología de trabajo seguida durante el desarrollo del proyecto.
- Apartado 3.9: Modularidad del servidor para realizar escalado.

**Capítulo 4:** resumen general del proyecto realizado, destacando lo más relevante, explicando las ideas para un posible desarrollo futuro y conclusión final.

- Apartado 4.1: Conocimientos adquiridos durante el trabajo.
- Apartado 4.2: Presentación de ideas futuras para mejorar el trabajo realizado.
- Apartado 4.3: Conclusiones obtenidas al finalizar el proyecto.

**Capítulo 5:** Bibliografía utilizada, artículos y enlaces al contenido utilizado.

**Anexo I:** Manual de usuario, indicando las opciones y información sobre la GUI disponible al usuario en cada una de las pantallas de la aplicación.

**Anexo II:** Documentación sobre las metodologías ágiles, tipos disponibles, utilidad, características en común, etc…

**Anexo III:** Detalle sobre los servicios REST utilizados en la comunicación entre cliente y servidor utilizados en este proyecto.

**Anexo IV:** Planificación de diseño empresarial para conseguir beneficios: análisis de clientes, diferenciación de producto, expansión de mercado…

# 3 TRABAJO DESARROLLADO

A continuación se muestra una visión de alto nivel de la arquitectura diseñada para el sistema, junto a dos diagramas: un diagrama de despliegue y uno de componentes. Posteriormente se detallarán individualmente las distintas tecnologías y los distintos usos que se han hecho de ellas para la implementación del proyecto, abarcando tanto la parte fron-end como la back-end.

## 3.1 Arquitectura software del sistema

La arquitectura del conjunto de componentes se basa en un modelo cliente/servidor sencillo. Para evitar problemas de congestión o cuellos de botella se puede llegar a crear en un sistema de colas en el que se distribuya la carga entre múltiples servidores que accedan a un clúster de base de datos. Esto se detallará más adelante, en el apartado *'4.2.2 Escalabilidad Back-end'*.

**Diagrama Despliegue**

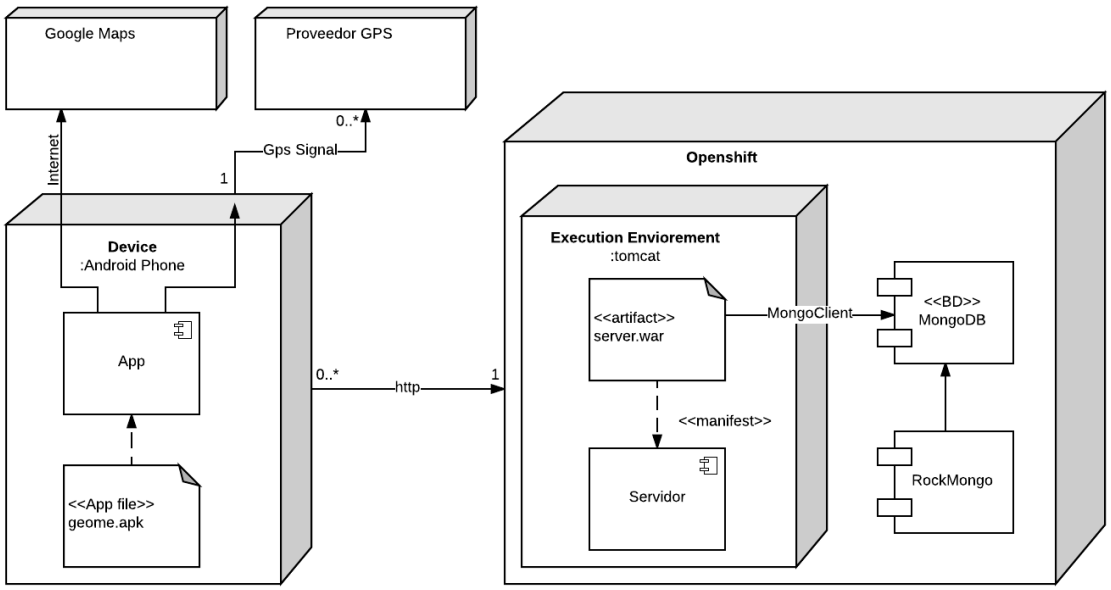


*Figura 6: Diagrama de despliegue*

El cliente está disponible para Android. El fichero .apk genera una aplicación que se conecta a través de Internet a la API de Google Maps para trabajar sobre el mapa mostrado en la interfaz. También recibe datos a través de la señal GPS que obtiene del dispositivo. Esta señal es recibida a través de los distintos proveedores GPS disponibles.

El cliente se conectará a través de http mediante una comunicación REST al servidor alojado en Openshift, que genera el servidor a través de un archivo .war desplegado en un entorno tomcat. La base de datos está desplegada directamente en el mismo servicio Openshift, además incluye un gestor de base de datos, RockMongo, para administrar la base de datos.

**Diagrama CyC**

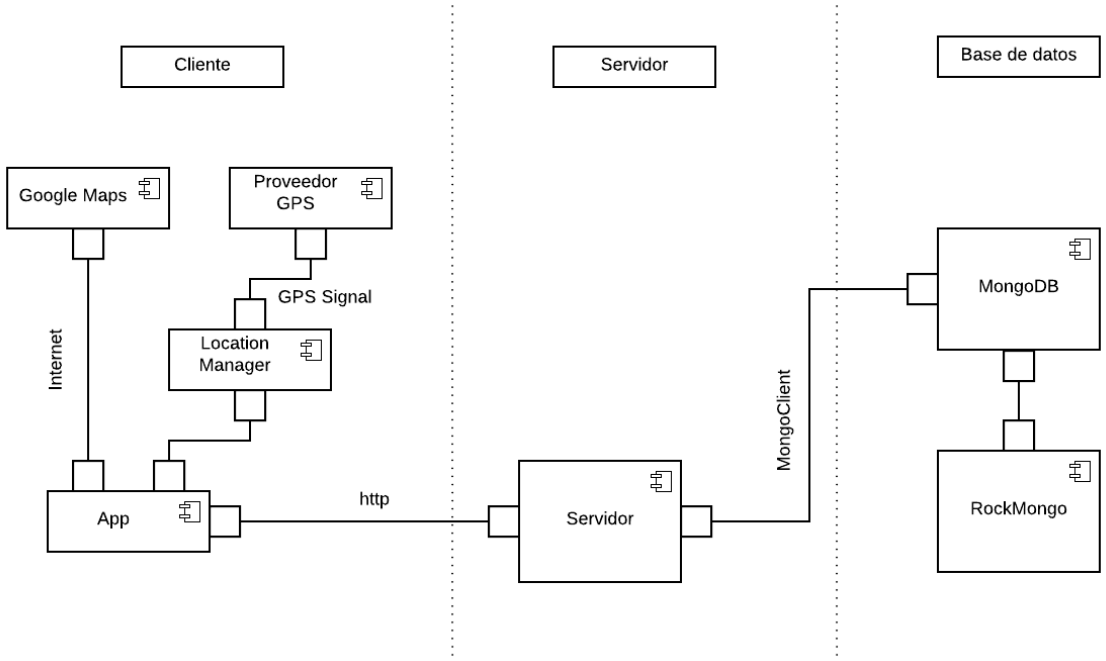


*Figura 7: Diagrama Componente y Conector*

La aplicación hace uso de un componente denominado LocationManager. Éste es el encargado de obtener la localización del usuario y trabajar con ella (Apartado *'3.7.2 Monitorización de ubicación y velocidad'*). A través de http enviará peticiones y recibirá datos del servidor en formato JSON-HATEOAS (Apartado *'3.5 Sistema de Anonimato'*). Por último, la base de datos es modificada a través del cliente MongoClient implementado en el servidor.

## 3.2 Base de datos

Con más de 40 años a sus espaldas, las bases de datos relacionales siguen vigentes en nuestros días. Este tipo de base de datos permite realizar operaciones de consulta en varias tablas y conseguir una estructura muy robusta de datos, sin embargo en este caso no es necesario tales características, es por ello que se ha utilizado una base de datos no relacional, como *'MongoDB [9]'*.

El modelo SQL permite crear desde bases de datos muy sencillas, hasta las más complejas. Parece la tecnología perfecta, pero el principal problema está en el rendimiento. Las bases de datos relacionales no son lentas, pero sí tienen un acceso costoso, por lo que peticiones de búsqueda e inserción constantes sobrecargarían el sistema. Resulta interesante usar bases de datos noSQL en casos en que la cantidad de información es muy grande y ágil, como es nuestro caso.

Una colección *"locations"* es la encargada de almacenar los Usuarios activos en la aplicación (localización y el tipo de usuario):

```
{
  "_class": "model.User",
  "type": NumberInt(0),
  "location": [
    41.66473363,
    -0.87482976
  ]
}
```

*Figura 8: Ejemplo de usuario almacenado en la base de datos*

El atributo *"Type"* identifica el tipo de usuario:

- 0: peatón
- 1: ciclista
- 2: vehículo

El atributo *"Location"* identifica la posición en el mapa del usuario. Para poder realizar búsquedas geoespaciales se ha utilizado una nueva función disponible en las versiones posteriores a 2.4 de MongoDB: *'indexación geoespacial [10]'*. Gracias a esta característica es posible realizar búsquedas por cercanía según la latitud y longitud usando el datum *'WGS84 [11]'*.

Google Maps usa coordenadas geográficas CGS en el sistema WGS84, por lo que la integración entre ambas tecnologías no requiere de ninguna transformación.

## 3.3 Servidor

Para realizar la comunicación es necesario minimizar la información transmitida para conseguir la máxima eficiencia. Para conseguir esto se ha utilizado un framework que permita la comunicación con MongoDB de manera sencilla y sea compatible con Android Java. La tecnología utilizada para el servidor es Spring Framework. Esta tecnología nos ofrece un marco de trabajo que nos facilita el ciclo de vida de los objetos y sus dependencias a la hora de crear un servidor. Su funcionamiento se basa en *"Módulos"* que pueden ser agregados a la aplicación. En nuestro caso existe un módulo denominado *'Spring Data MongoDB [12]'* el cual nos permite crear una capa de comunicación contra una base de datos MongoDB a través de objetos POJO, además está integrado con las funciones Geoespaciales, por lo que facilitará el trabajo de comunicación.

Todo el sistema implementado nos permite reducir las consultas contra la base de datos al siguiente código:

```
@RepositoryRestResource(collectionResourceRel = "people")
public interface UserRepository extends MongoRepository<User, String>
{
    List<User> findByLocationNear(Point p, Distance d);
}
```

*Figura 9: Ejemplo de consulta en Spring*

A través de MongoRepository se autoconfiguran peticiones de tipo RESTful y además permite realizar consultas geoespaciales a través del método *findByLocationNear*.

## 3.4 Comunicación

Como se ha mencionado, uno de los principales problemas con los que se tiene que trabajar es la gran cantidad de peticiones que se deben tratar en periodos cortos de tiempo. Es por ello que se debe minimizar la información transmitida para agilizar el proceso.

De todas las soluciones tecnológicas de servicios Web actualmente en boga, los servicios REST destacan por encima de los demás. Como se puede observar en la siguiente gráfica, soporta un número de peticiones por segundo superior frente a su competencia. Es por ello que para realizar la comunicación se ha seleccionado el sistema REST que además permite minimizar la información enviada en las peticiones.

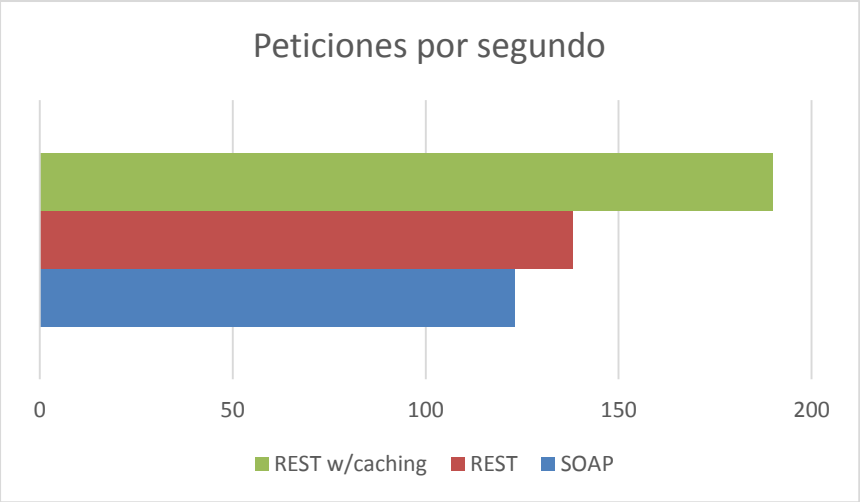


*Figura 10: Gráfica comparativa entre SOAP y REST*

Como vemos, una estructura con caché permite aumentar considerablemente el número de peticiones por minuto, pero en el dominio del problema al que nos enfrentamos no es posible usar este sistema, ya que los datos están en constante actualización. No podemos almacenar la posición de un usuario durante un periodo de tiempo ya que éste está en continuo movimiento, lo que provocaría que aparecieran localizaciones erróneas en el mapa de los demás usuarios.

La siguiente tabla muestra la interfaz API del servidor web:

| Acceso | Tipo | Atributos | Descripción |
|---|---|---|---|
| **/people** | POST | User.class | Encargado de recibir las peticiones de inserción de usuarios. |
| ***/href**** | DELETE | - | Encargado de recibir las peticiones de borrado de usuario a través de una URI identificativa. |
| ***/href**** | PUT | User.class | Encargado de recibir las peticiones de actualización de usuario a través de una URI identificativa. |
| **/location** | GET | Lat, lon, r ** | Encargado de recibir las peticiones de búsqueda geoespaciales. |

*Figura 11: Tabla informativa sobre la interfaz del servidor web*

***href:** *"href"* hace referencia al id que identifica el objeto almacenado en la base de datos gracias al sistema JSON-HATEOAS (Mirar apartado *'3.5 Sistema de anonimato'*).

****Lat, lon, r:** *"Lat y lon"* hacen referencia a las coordenadas geográficas en datum WGS84 y *"r"* representa la distancia en metros.

## 3.5 Sistema de anonimato

La privacidad es uno de los apartados más importantes para cualquier usuario de una aplicación como es la desarrollada en este proyecto. No debe quedar constancia en ningún momento de quien soy y de cual es mi ubicación actual. Si cada usuario es representado con un identificador, es posible obtener ese identificador que define ese usuario y por tanto podemos saber quién es.

La idea es usar el sistema *'HATEOAS [13]'* ofrecido por otro *'módulo Spring [14]'*. El sistema HATEOAS ofrece una representación de la arquitectura REST ofreciendo un link a cada objeto creado, dicho de otra manera, dado un punto de entrada genérico de nuestra API REST, podemos ser capaces de descubrir sus recursos basándonos únicamente en las respuestas del servidor, ya que éste ofrecerá información en forma de hipervínculos.

Un ejemplo de la información devuelta por el servidor ante una petición es el siguiente:

```
"people" : [ {
      "type" : 0,
      "location" : [ 41.68421, -0.88937666 ],
      "_links" : {
        "self" : {
          "href" : "http://locahost/people/5734674ce4b0d527ef759d7f"
        },
        "user" : {
          "href" : "http://locahost/people/5734674ce4b0d527ef759d7f"
        }
      }
    }, {
      "type" : 1,
      "location" : [ 41.683936, -0.888131666 ],
      "_links" : {
        "self" : {
          "href" : "http://localhost/people/57347d44e4b0d527ef759d83"
        },
        "user" : {
          "href" : "http://localhost/people/57347d44e4b0d527ef759d83"
        }
      }
    }
...
```

*Figura 12: Ejemplo de respuesto del servidor en formato HATEOAS*

De esta forma podemos observar que cada elemento de la lista *"People"* tiene una URI que identifica el propio elemento, pudiendo acceder a ella directamente a través de ese hipervínculo y realizar modificaciones.

Por parte del cliente se crea un nuevo usuario cada vez que se ejecuta la aplicación, lo que da lugar a una nueva URI que identifica a dicho usuario. Esta URI será el identificador del usuario y se realizarán modificaciones frente a ella. Una vez finalizado el uso de la aplicación, este usuario en la base de datos se borra, junto a su URI. En el momento de volver a abrir la aplicación, se creará un nuevo usuario con una nueva URI identificativa, por lo que será imposible conocer un usuario en base a un identificador concreto ya que éste variará con cada ejecución de la aplicación.

## 3.6 Despliegue Cloud

Una gran cantidad de servicios cloud están disponibles para el despliegue de servidores y bases de datos online. Heroku, Amazon Web Services, MongoLab o DigitalOcean son solo unos pocos ejemplos. Para este proyecto se ha utilizado una plataforma relativamente joven: *'OpenShift [15]'*.

OpenShift es una plataforma de código abierto que trabaja junto a Docker y Kubernetes para la gestión de containers. Desarrollada por RedHat ofrece un "*PaaS*" el cual permite integrar el servidor junto a la base de datos dentro de un mismo "gear". Esta ha sido la principal característica por la que se ha utilizado este servicio. A través del Blog que disponen online no ha sido gran esfuerzo desplegar un servidor Spring en dicha plataforma. Por otro lado se ha desplegado una base de datos MongoDB del mismo modo.

Para la gestión de la base de datos se ha utilizado RockMongo. Una interfaz gráfica implementada en PHP que facilita la gestión de los datos en una base de datos MongoDB, disponible también en OpenShift como un "Cartridge".

## 3.7 Cliente Android

Para el desarrollo del cliente se ha utilizado Android como plataforma de trabajo. Se ha utilizado como base el SDK 16, correspondiente a la versión 4.1 Jelly Bean de Android. Esto nos permite llegar a un 96.0% del total de mercado Android. También se ha implementado el nuevo sistema de permisos diseñado para Android 6.0 Marshmallow, pero el usuario deberá permitir el acceso a la ubicación de la aplicación manualemtne, en caso de denegar algún tipo de permiso, la aplicación se cerrará.

### *3.7.1 El mapa*

A través de la API ofrecida por Google se ha creado un mapa basado en los servicios de Google Maps. Este mapa es el encargado de superponer una capa con toda la información necesaria para el usuario. En el mapa se encuentran 4 tipos de iconos:

| Icono | Descripción |
|---|---|
|  | Este icono azul representa la ubicación del propio usuario. El mapa se encuentra centrado siempre sobre este icono. |
|  | Este icono representa el radio de acción del propio usuario en el que se mostrarán las ubicaciones de los usuarios cercanos. |
|  | Este icono rojo representa la ubicación de otro usuario de la aplicación de tipo vehículo (se encuentra en un desplazamiento por vehículo) |
|  | Este icono amarillo representa la ubicación de otro usuario de la aplicación de tipo ciclista (se encuentra en un desplazamiento con bicicleta) |
|  | Este icono verde representa la ubicación de otro usuario de la aplicación de tipo peatón (se encuentra en un desplazamiento a pie) |

*Figura 13: Tabla descriptiva de los iconos disponibles*

Estos iconos se actualizarán cada X metros recorridos por el usuario. La finalidad de esto es poder gestionar el intervalo en el que se actualizan los iconos, ya que es un proceso repetitivo que se realiza en intervalos cortos de tiempo y puede llegar a saturar el trabajo del procesador de ciertos smartphones. En el apartado *'3.7.4 Ajustes'* se detalla el funcionamiento de la gestión del rendimiento.

### 3.7.2 MONITORIZACIÓN DE UBICACIÓN Y VELOCIDAD

Para obtener la localización del usuario se utiliza los servicios de ubicación proporcionados por el GPS del dispositivo. Se ha optado por restringir el uso de la aplicación hasta que no haya una señal de GPS válida, ya que en caso contrario, podría ubicar al usuario en una posición incorrecta, lo que provocaría situaciones no deseables al resto de usuarios.

Al comienzo de la aplicación se muestra este cuadro de diálogo para informar al usuario del proceso:

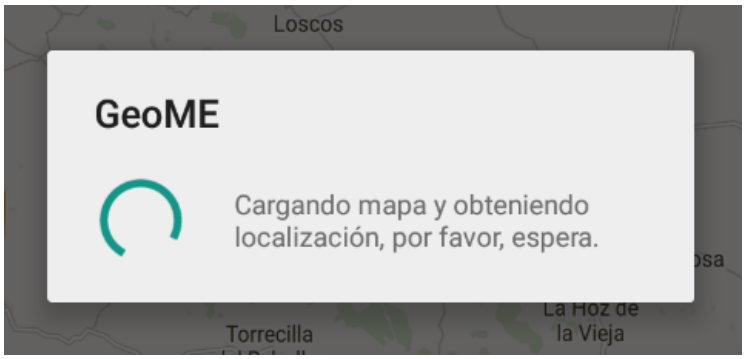


*Figura 14: Diálogo de carga de la aplicación*

Una vez obtenida la localización precisa, ésta se envía al servidor creando un nuevo usuario en la base de datos. Tal y como se ha explicado en el apartado *'3.5 Sistema de Anonimato'* se utiliza la URI para identificar a dicho usuario, por lo que esta URI es almacenada para poder modificar la localización de este usuario. A partir de este momento entra en juego el objeto *'LocationManager [16]'*.

LocationManager es una clase que permite acceder a los servicios de ubicación del sistema. Gracias a él podemos obtener actualizaciones periódicas de la ubicación geográfica del dispositivo. Una vez iniciado, en intervalos de 1000ms (1s) se enviará la nueva ubicación al servidor a través de la petición descrita en el apartado *'3.4 Comunicación'*.

Por otro lado, debemos conocer la velocidad actual del usuario para poder clasificarlo en los 3 niveles que dispone la aplicación: patón, ciclista o vehículo motorizado. La velocidad se obtiene a través del objeto LocationManager, pero no basta con conocer la velocidad actual, ya que un usuario conduciendo un vehículo debe parar ante un semáforo en rojo, lo que reduce su velocidad y por lo tanto variará su clasificación de tipo de usuario en la aplicación (Pasará de ser vehículo a peatón, siendo esto incorrecto). Para solucionarlo se ha utilizado la media de la velocidad medida en ciertos puntos concretos del trayecto durante intervalos de 30 segundos, obteniendo de manera más precisa la velocidad "real" de los usuarios. De esta manera cada 30 segundos se actualizará el tipo de usuario actual en la aplicación.

Para diferenciar cada tipo de usuario se han utilizado intervalos razonables de velocidad: la media de la velocidad de una persona caminando/trotando, la media de la velocidad de un ciclista estándar, y el resto de velocidades se consideran vehículos motorizados. Estos datos han sido extraídos de *'Wikipedia [17]'*. Tal y como se ha explicado antes, hay instantes en el que la velocidad puede ser superior o inferior a las cotas establecidas (peatón haciendo sprint, ciclista en tramo con pendiente, peatón subiéndose a un vehículo…), pero gracias a la caché de velocidades almacenada en la aplicación durante 30 segundos del recorrido, el fallo en el tipo de usuario será mínimo.

- Entre 0 y 9 Km/h: tipo peatón
- Entre 9 y 27 Km/h: tipo ciclista
- > 27Km/h: tipo vehículo


*Figura 15: Cotas de clasificación de usuarios*

### 3.7.3 CICLO DE VIDA

El *'ciclo de vida Android [18]'* es bastante complejo. Actualmente tiene un gran cantidad de estados: onCreate(), onStart(), onResume(), onPause(), onRestart(), onStop() y onDestroy(). Para este proyecto se ha implementado un sistema para eliminar ciertos estados innecesarios en la aplicación:

- **onPause():** este estado se ejecuta cuando el sistema abre una actividad previa a la actual, pausando la aplicación presente. En este caso no pausa la aplicación presente, si no que ejecuta el proceso en segundo plano en caso necesario, o termina la actividad principal. Esto se detallará mas adelante en la *'Figura [16]'*.
- **onStop():** este estado se ejecuta cuando la aplicación no está visible por el usuario. Por la misma razón que la anterior, este estado se omite para cerrar la aplicación inmediatamente.

A continuación se detallan los distintos estados disponibles del ciclo de vida actual de la aplicación:

- **Launch App:** estado inicial en el que se inicia la aplicación
- **Main Activity Running:** Es el estado por defecto de la aplicación. La actividad es ejecutada en primer plano, mostrando el mapa junto a la GUI. A partir de este este momento, el usuario puede salir de la aplicación, pasando al estado onPause().
- **onPause():** Al intentar salir de la aplicación se muestra un cuadro de diálogo en el que el usuario puede decidir si ejecutar la aplicación en segundo plano o no (*'Figura [17]'*).
    - **En caso de aceptar:** la aplicación pasará al estado "Call to Background service", estado en el cual inicia el *'Servicio [19]'* en segundo plano y termina la actividad principal (Activity Shutdown).
    - **En caso contrario:** directamente se cierra la aplicación y pasando al estado "Activity Shutdown"
- **Activity Shutdown:** estado en el que la aplicación está cerrada, sin ningún proceso trabajando en segundo plano.
- **Background Service Running:** en este estado la aplicación sigue monitorizando la posición del usuario en segundo plano y ejecuta las alertas tanto sonoras como físicas (vibración) en caso de que sea necesario. Para informar al usuario de la monitorización se muestra al usuario una notificación en la barra de estado (*'Figura [18]'*). Al interactuar con la notificación, pasamos al estado onDestroy().

- **onDestroy():** dependiendo de la acción ejecutada por el usuario podemos realizar dos acciones:
    - **Si descarta la notificación:** la monitorización acaba y termina el Service (Service Shutdown).
    - **Si pulsa sobre la notificación:** Si por el contrario se pulsa sobre la notificación, la aplicación vuelve a lanzarse y se termina con el Service (Service Shutdown).
- **Service Shutdown:** estado en el que el servicio en segundo plano está terminado.

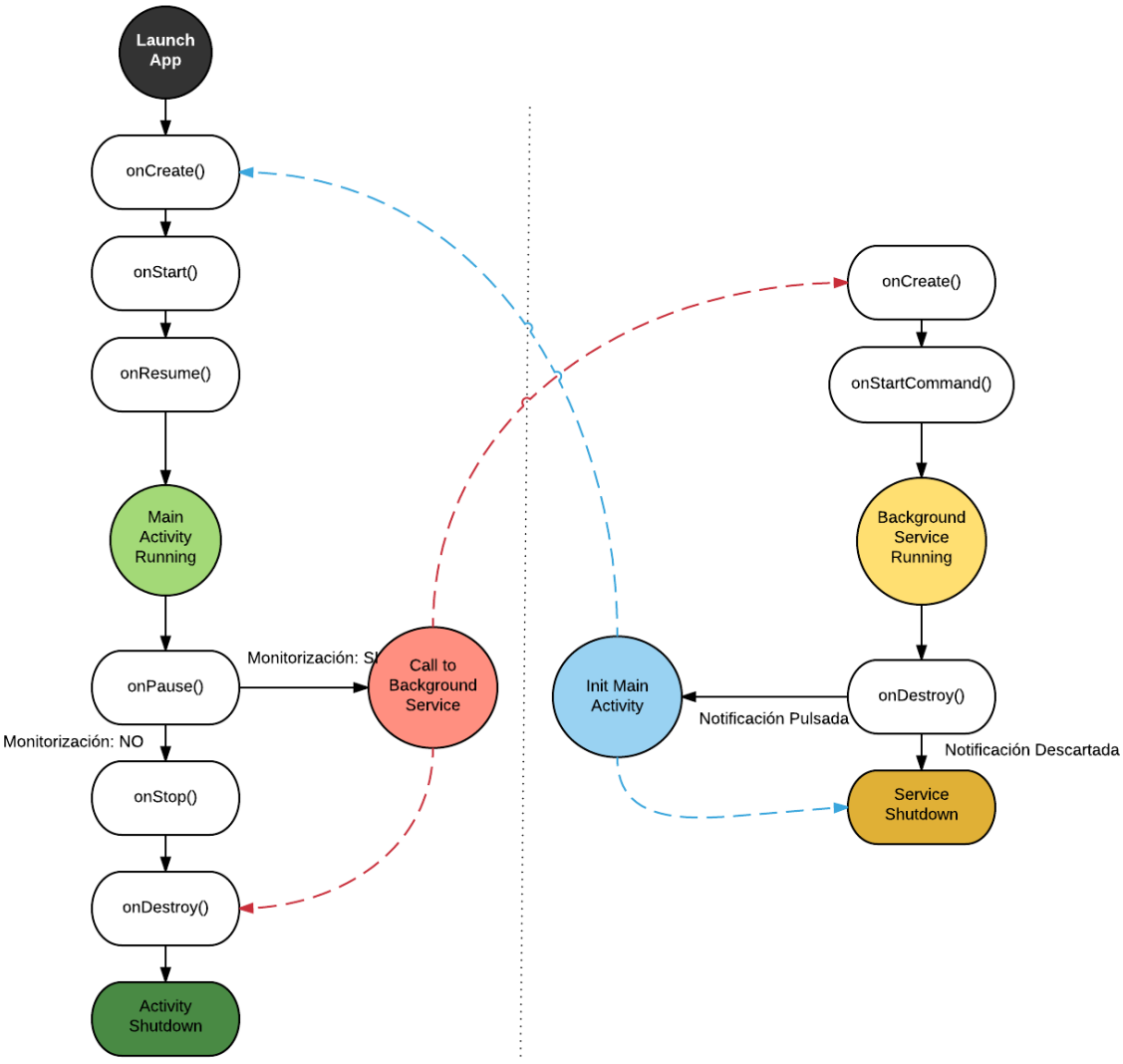


*Figura 16: Ciclo de vida de la Aplicación en primer y segundo plano*

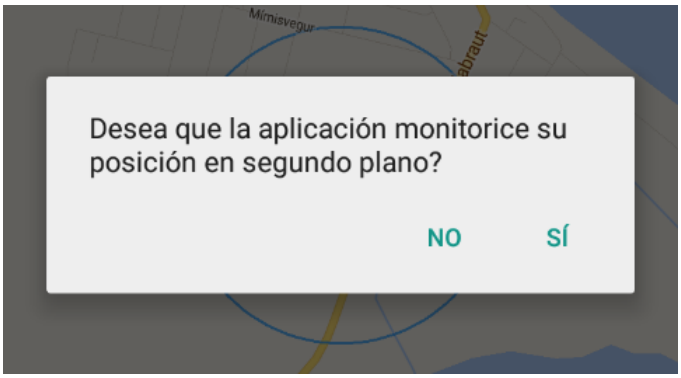


*Figura 17: Diálogo de salida de la aplicación*

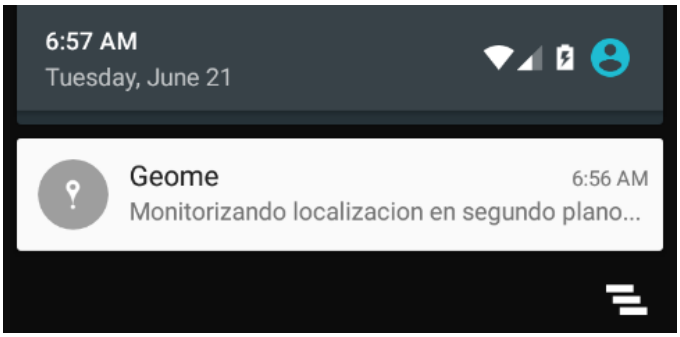


*Figura 18: Notificación de la ejecución en segundo plano*

### 3.7.4 AJUSTES

Hay disponible en la aplicación un botón en la parte superior derecha que dirige al usuario al menú de ajustes.

Es importante destacar que el menú ha sido creado extendiendo la clase *'PreferenceFragment [20]'* de Android. Esta clase guarda automáticamente los ajustes modificados por el usuario en la clase "SharedPreferences" de Android. Este método garantiza consistencia en los ajustes a pesar de que la aplicación sea cerrada, estos ajustes se almacenarán automáticamente al cerrar la aplicación y se cargarán al volver a abrirla. A través de este menú (*'Figura [19]'*) es posible gestionar una gran cantidad de parámetros de la aplicación:

- Permite seleccionar el tipo de alertas entre las cuatro disponibles que ejecutará la aplicación.
- Permite seleccionar un número máximo de iconos a mostrar en el mapa, así como el radio de muestreo (en metros) alrededor el propio usuario.
- El mapa podrá orientarse según las coordenadas polares del dispositivo
- El Servicio en segundo plano también es configurable a través de este menú. Permite activar, o no, las alertas tanto sonoras como de vibración en segundo plano.
- Por ultimo un ajuste de rendimiento, el cual configura la precisión con que se muestran los usuarios en el mapa. Este es el ajuste principal que afecta al rendimiento de la aplicación.

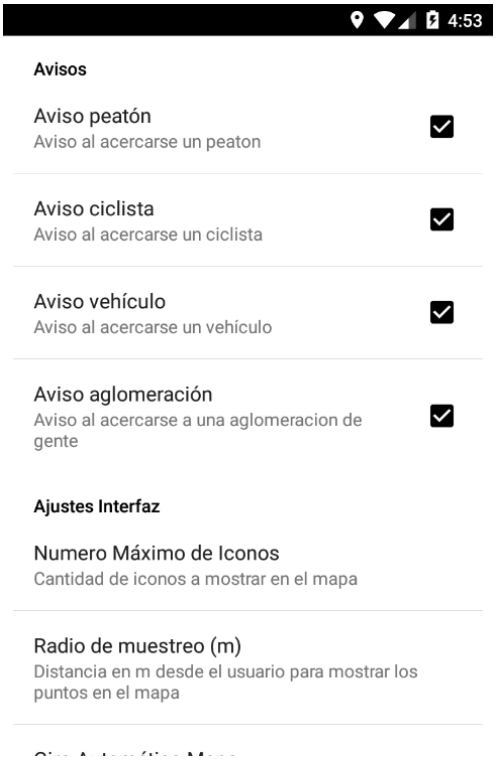


*Figura 19: Menú de ajustes de la aplicación*

### 3.7.5 RENDIMIENTO

Tal y como se ha descrito en la presentación del proyecto, se ha intentado acceder a la máxima cantidad de smartphones posibles, independientemente de la potencia de su procesador. El usuario será capaz de modificar el rendimiento de la aplicación a través de una opción en el menú Ajustes, tal y como se describe en el apartado *'3.7.4 Ajustes'*.

A continuación se muestra la diferencia entre el trabajo de CPU con el ajuste más básico, frente a otro que requiere de mayor velocidad de procesamiento:

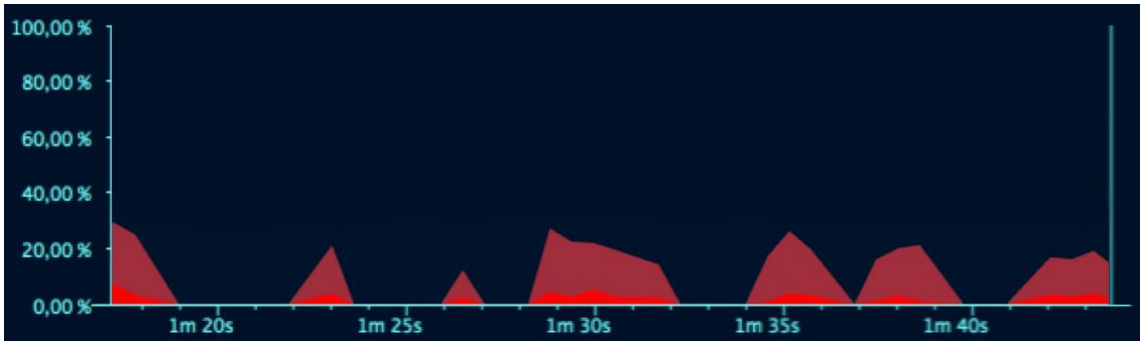


*Figura 20: Gráfica de trabajo de CPU en redimiento bajo*

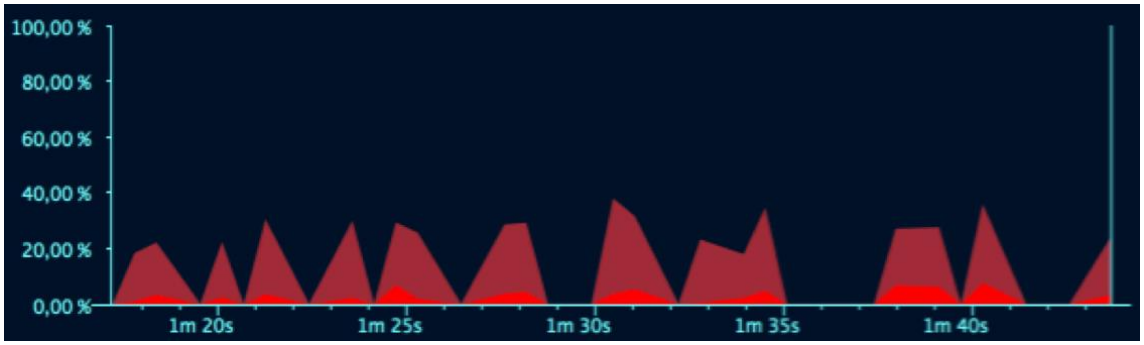


*Figura 21: Gráfica de trabajo de CPU en rendimiento alto*

Tal y como se aprecia en la gráfica, el trabajo de CPU se incrementa en los momentos que la aplicación tiene que representar los usuarios en el mapa. El intervalo de tiempo para pintar los usuarios es menor en el segundo ejemplo, por lo que el trabajo de CPU es mayor.

Por otro lado, en la siguiente gráfica (*'Figura [22]'*) se muestra el trabajo necesario para realizar la monitorización de la ubicación del usuario en segundo plano. Este proceso constante no consume ni un 5% del trabajo del procesador. De este modo se demuestra que el principal problema es la parte gráfica de la aplicación.

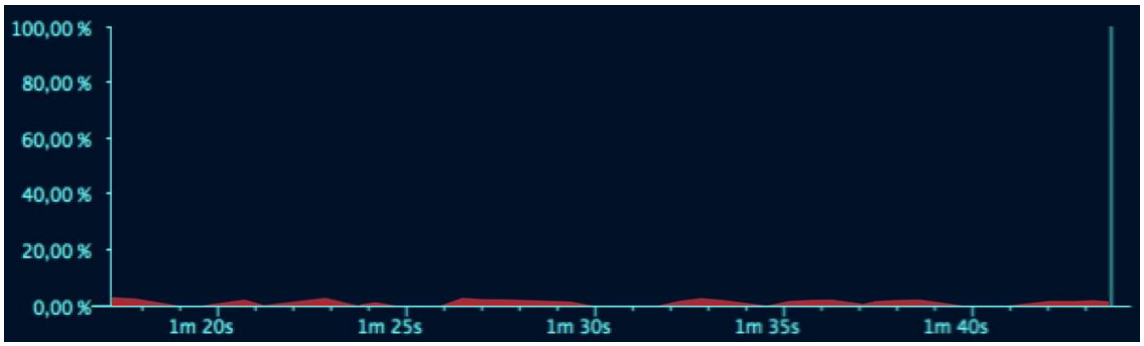


*Figura 22: Gráfica de trabajo de CPU de la aplicación en segundo plano*

### 3.7.6 ALERTAS

La gestión de las alertas puede parecer un proceso sencillo, pero al ser un proceso asíncrono trabajando en base a unos datos que se actualizan en tiempo real puede ser un problema.

Las alertas son sonidos que se reproducen cuando algún usuario entra en el radio de acción. Estos sonidos necesitan un tiempo para ser reproducidos, tiempo suficiente para que varíen los usuarios en el radio de acción y haya que reproducir otro tipo de alerta. Por ejemplo:

1. Hay dos usuarios en nuestro radio de acción: peatón y ciclista.
2. Se comienza a reproducir el aviso de peatón.
3. Los usuarios peatón y ciclista salen de nuestro radio de acción y aparece un nuevo usuario vehículo.
4. El sistema debe ser capaz de parar la reproducción de las alertas de peatón y ciclista para comenzar la reproducción de la alerta de vehículo.

Este sistema debe ser lo suficientemente robusto para saber en que momento los usuarios dentro del radio de acción varían y de qué modo: un peatón más, desaparece un vehículo y aparece un ciclista, desaparece el usuario del que estábamos alertando… Además hay que tener en cuenta el tiempo de reproducción del sonido, para evitar solapamientos entre alertas.

Para realizar esto se ha utilizado la clase *'MediaPlayer [21]'*. Esta clase permite ejecutar sonidos en los dispositivos Android y ejecuta un método (onCompletion()) al terminar de reproducir el sonido. Gracias a este método se ha creado un ciclo recursivo para reproducir las alertas almacenadas en un Array. En cada momento en el que haya que pintar los nuevos usuarios en el mapa, se calcula la diferencia entre el número de usuarios de cada tipo que había en el instante anterior, con el insatnte actual. Conociendo esta diferencia se puede conocer los cambios que ha habido en el radio de acción del usuario, lo que permite introducir las alertas pertinentes en un Array, el cual será usado por la clase MediaPlayer para reproducirlos.

## 3.8 Metodología de trabajo

La planificación del trabajo se ha realizado según los principios de las metodologías ágiles. Estas metodologías funcionan bien en equipos pequeños, por lo que se ha intentado implementar al trabajo individual de este proyecto. Se caracterizan principalmente por evitar el ciclo de vida en cascada tradicional a la hora de llevar a cabo proyectos. Esta característica junto al análisis y diseño "just in time" se han adaptado a la planificación y desarrollo del trabajo realizado. A continuación se describen las principales características obtenidas de las metodologías ágiles que se han seguido, *`Scrum [22]`* y *'Extreme Programming [23]'*, fusionando lo mejor de cada una de ellas con el fin de crear una nueva metodología que ayude al desarrollo del proyecto.

### 3.8.1 *Extreme Programming*

Una de las principales carencias de esta metodología es la falta de realizar un análisis inicial del proyecto. Esto se corrige con la planificación inicial realizada el principio del sprint por parte de la metodología Scrum. Al no haber análisis inicial, el equipo puede encontrar situaciones en las que no tienen los conocimientos necesarios para avanzar. En estos momentos el equipo debe experimentar y buscar una solución. Esto se le llama "Spike" ("púa", en español). Para este proyecto se ha tenido que hacer frente a tecnologías desconocidas como MongoDB o principios teóricos básicos sobre geolocalización, por lo que la utilización de estos "Spikes" para determinar momentos en los que se ha experimentado con distintas tecnologías hasta conseguir el objetivo deseado ha sido crucial.

Por otro lado, XP se focaliza en la revisión de código. Lo ideal es la programación por pares, sin embargo en proyectos individuales esto es imposible, por lo que se han realizado revisiones de código, realizando refactorizaciones cada cierto tiempo. En caso de encontrar inconsistencias o bugs, éstos se han corregido.

### 3.8.2 *Scrum*

Scrum centra su atención en llevar a cabo un conjunto de buenas prácticas que ayudan al trabajo en equipo para obtener el mejor resultado posible en el proyecto. El proceso de desarrollo se divide en Sprints/Iteraciones. Al comienzo de cada iteración se realiza un pequeño análisis y se planifica lo que se va a hacer en dicho sprint. Una vez terminado se debe tener un trabajo presentable ante un cliente, y se observan los resultados. Esto ha ayudado a la planificación del proyecto en 4 iteraciones:

- Primera Iteración: planteamiento de la idea y primer Spike ante tecnologías de geolocalización. Se planteó el uso de MongoDB, PostgreSQL, DynamoDB… Se realizó la implementación de base de datos que permitía estas características.

- Segunda Iteración: de nuevo un Spike para realizar el servidor. Se comenzó a experimentar con *'Play! Framework [24]'* como sistema para el servidor, pero se desechó la idea al no poder desplegarlo exitosamente en cloud. Finalmente se optó por usar Spring y se crearon los primeros puntos de acceso para interactuar con la base de datos.
- Tercera Iteración: Cliente Android. Ya se ha trabajado anteriormente con esta tecnología, por lo que no hubo problemas en su implementación.
- Cuarta Iteración: Finalización del proyecto y documentación final.

A pesar de la diferencia de ámbito entre los sprint, se estimó un tiempo de trabajo similar entre todos ellos. En la primera iteración la gran cantidad de tipos de bases de datos dificultó la tarea, en el segundo sprint a pesar de ser más sencillo, el despliegue cloud presentó bastantes problemas. Por último el tercer sprint, el cual, a pesar de haber trabajado con dicha tecnología, tiene muchísimas mas características, por lo que lleva mas tiempo realizarlo. La documentación y la finalización del proyecto final se ha decidido como un sprint a parte ya que el tiempo está prefijado en este tipo de proyectos, y siempre es preferible completar el trabajo a intentar añadir nuevas funcionalidades dejando el proyecto con fallos o características inacabadas.

Otra de las características de Scrum utilizadas ha sido el denominado "Scrum Board". De esta manera se puede ver de un solo vistazo las tareas del proyecto, clasificadas en: hechas, en proceso, o sin empezar. La pizarra está en constante cambio, moviendo post-it de un lado a otro, manteniendo actualizado el estado del proceso. En la siguiente figura (*'Figura [23]'*) se puede ver el Scrum Board utilizado para el desarrollo del proyecto. Las 4 columnas indican: tareas descartadas, tareas por hacer, tareas en proceso, y por último tareas ya realizadas:


*Figura 23: Scrum board utilizado en el proyecto*

## 3.9 Escalabilidad

La escalabilidad en una aplicación debe formarse en la fase de diseño, puesto que resulta prácticamente imposible adaptarla una vez realizada la implementación. Las decisiones que se toman en la fase de diseño son cruciales y determinarán en gran medida la escalabilidad de la aplicación.

### *3.9.1 ESCALABILIDAD SERVIDOR*

Disponemos de 2 métodos estándares para conseguir una aplicación escalable.

**Escalado vertical**: consiste en aumentar la escalabilidad utilizando un software más rápido, acompañado de un incremento de velocidad de procesamiento con más memoria, procesadores etc… Esta es la solución más cara, pero minimiza en número de errores en un único punto, por lo que su resolución será más rápida.

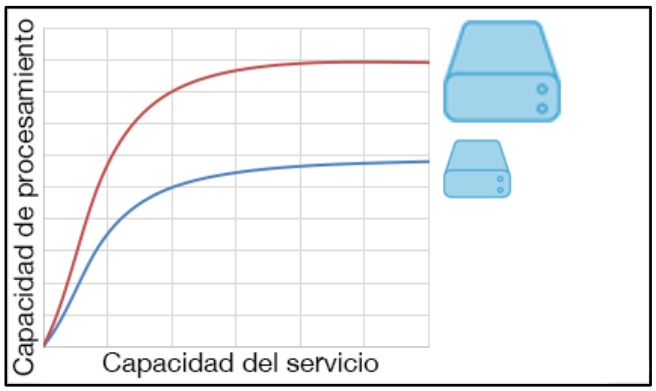


*Figura 24: Esquema de escalado vertical*

**Escalado horizontal**: esta alternativa consiste en distribuir la carga en varios servidores similares, en lugar de centrar todo el tráfico en un solo servidor potente. Esta opción suele ser más económica y sobre todo mucho más escalable, pero la aplicación debe soportar la modularidad de poder distribuir la carga en varios servidores, aunque la colección de servidores funcionará como un único equipo.

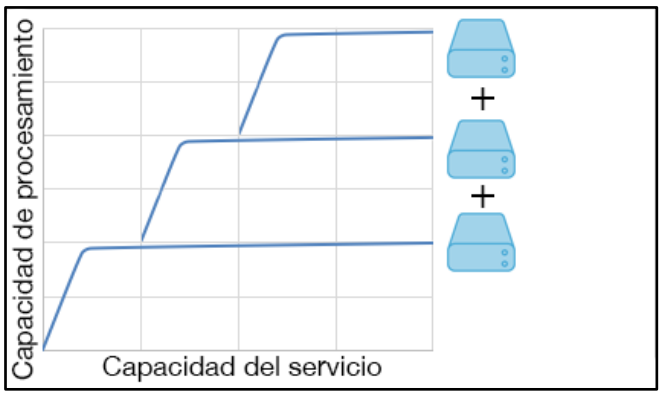


*Figura 25: Esquema de escalado horizontal*

En este caso, al tener un conjunto de operaciones reducida y modulares, es recomendable usar un escalado horizontal junto con un sistema de colas, el cual distribuya toda la carga del sistema entre todos los servidores. Para poder implementar este sistema en caso de que fuese necesario, el servidor cuenta con modularidad en los servicios ofrecidos a la aplicación.

### 3.9.2 ESCALABILIDAD BASE DE DATOS

La estructura de la base de datos es un factor muy importante a la hora de escalar una aplicación. Actualmente se dispone de una única tabla con todos los clientes activos de la aplicación. La indexación en base a las coordenadas geográficas hace que las búsquedas geoespaciales sean más eficientes, pero en el momento en el que la colección de datos sea bastante grande comenzará a dar problemas. Es por esto que se ha ideado un sistema de distribución de datos según la localización geográfica del cliente. Esto se detallará más adelante en el apartado *'4.2.2 Escalabilidad Back-end'*.

# 4 LECCIONES APRENDIDAS Y CONCLUSIONES

Para terminar, en los siguientes apartados se va a explicar los principales problemas a los que se ha enfrentado en la realización de este trabajo, destacando lo más relevante, así como las ideas para un futuro desarrollo en caso de que se fuese a realizar. Po último la conclusión final de todo el trabajo realizado.

## 4.1 Conocimientos adquiridos

Una vez finalizado el proyecto se ha comprobado todo el trabajo realizado. Comenzando por la base de datos se ha trabajado por primera vez con MongoDB, incluyendo indexación con objetos geográficos. Esta característica ha resultado complicada al principio, pero una vez aprendida la técnica va a ser mucho más fácil volver a implementarla en el futuro.

El servidor Spring ha sido una de las tecnologías usadas en una de las asignaturas de la carrera. Lo aprendido en esa asignatura ha servido como base para poder desarrollar este proyecto. Se han utilizado nuevos plugins como el de la conexión con la base de datos MongoDB, o el envío de datos en formato JSON-HATEOAS. Por otro lado se han obtenidos conocimientos de despliegue cloud, habiendo creado el servidor en una plataforma como Openshift.

Por último se han implementado características sobre sistemas de información geográficas sobre el cliente nunca antes vistas en la carrera, así como el funcionamiento asíncrono en dispositivos Android y el ciclo de vida de sus componentes.

## 4.2 Ideas Futuras

Con la finalización del proyecto se han quedado algunas ideas sin implementar. En los siguientes apartados se detallan algunas de ellas:

### 4.2.1 *Eliminación de usuarios "basura"*

Hay casos en los que el cliente cierra de manera inesperada la aplicación, cerrando la conexión con el servidor de manera repentina. Esto puede ocurrir al apagado repentino del dispositivo (caída al suelo y apagado instantáneo), o por problemas software en el que el dispositivo se "cuelga". Esto provoca que se generen en la base de datos "usuarios basura", es decir, clientes que ya no están activos en la aplicación pero que siguen apareciendo en la base de datos. Estos "usuarios basura" generan marcadores erróneos en los mapas de los demás clientes, por lo que deben ser borrados.

Los usuarios activos en la aplicación están en constante actualización en la base de datos, por lo que una manera de solucionar esto es crear un servicio monitorizado que elimine automáticamente los usuarios que no han actualizado su posición en un intervalo de tiempo determinado.

### 4.2.2 *Escalabilidad Back-End*

Por parte del cliente/servidor se ha mantenido la comunicación REST que disminuye la carga hacia el servidor, lo que ayuda a su escalabilidad. Tal y como se ha descrito en el apartado '3.9.1 Escalabilidad servidor', el sistema posee los componentes modularizados para implementar la escalabilidad horizontal.

En caso de la base de datos, actualmente solo se dispone de una colección. En caso de querer explotar la aplicación, la escalabilidad debería estar presente en este apartado. Como idea futura, se puede crear una distribución de carga entre distintas colecciones de datos, en base a los parámetros de las coordenadas geográficas del usuario a almacenar. Tomando como base una proyección sobre la Tierra, cada cliente accedería a una colección distinta dependiendo de la zona en la rejilla formada por dicha proyección.

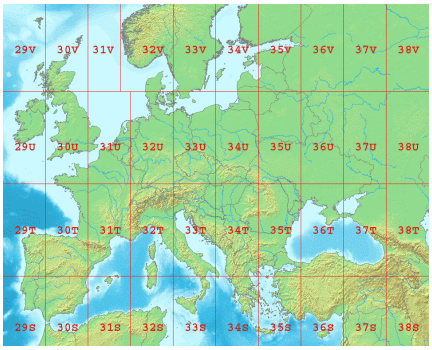


*Figura 26: Zonas UTM de europa*

### 4.2.3 ANÁLISIS DE DATOS

A parte de la principal finalidad del proyecto, el sistema ofrece una monitorización automática sobre la ubicación de los usuarios. Estos datos pueden ser recopilados y analizados con fines estadísticos.

Esta información pueden ser de gran utilidad para conocer qué carreteras son las más transitadas, en qué épocas hay más desplazamientos, zonas en las que haya más densidad de tráfico, atascos etc… Hay muchas posibilidades teniendo en cuenta que los datos se actualizan en tiempo real.

Estos datos pueden ser representados en gráficas o datos estadísticos, o directamente se puede presentar sobre una ubicación geográfica por medio de mapas de calor y un gradiente de colores. Al igual que en el ejemplo de la 'Figura 27', se pueden representar avenidas con más tránsito de peatones, ciclistas o vehículos, cada uno con unos fines de análisis específicos. Podemos observar qué carreteras tienen más tráfico para colocar una gasolinera, revisar las zonas con más ciclistas y analizar porqué es así (puede que en algún circuito haya ocurrido algún percance, o se haya cortado la vía y sea imposible se paso), o en caso de analizar la cantidad de peatones, el ayuntamiento puede revisar con más frecuencia la calzada en las zonas más transitadas.

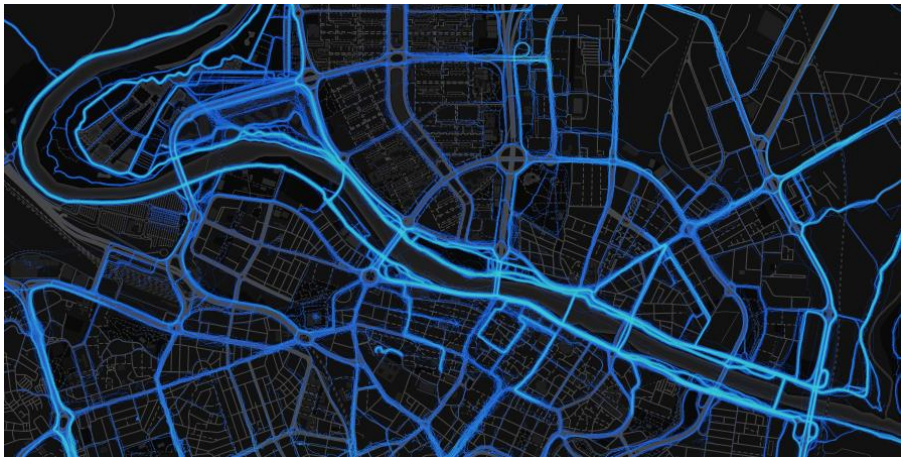

*Figura 27: Mapa de calor sobre trayectos más transitados realizando 'running en Zaragoza [25]'*

## 4.3 Conclusiones

Lo más difícil para llevar a cabo este proyecto de manera comercial será concienciar a los usuarios de las capacidades y de los beneficios que se obtiene con su uso. Un usuario "estándar" se niega a compartir datos propios, y mucho menos a compartir su ubicación en todo momento, por eso se debe concienciar del sistema de anonimato implementado en el sistema.

La situación ideal sería la instalación por defecto de esta aplicación en todos los dispositivos móviles, al igual que están algunas aplicaciones de Android o Google, con el fin de conseguir una gran cantidad de usuarios activos que puedan compartir su ubicación y poder llegar a cumplir los objetivos de este proyecto.

A pesar de las dificultades para su expansión, creo que se ha llevado a cabo un proyecto completo, con una idea que llevaba un par de años por mi cabeza y tenía ganas de desarrollar. Se han cumplido ampliamente todos los objetivos que se han planteado al inicio del proyecto, pese a trabajar con datos geográficos con los que nunca antes había tratado. Gracias a este proyecto he aprendido a trabajar de forma organizada, planificando el trabajo desde el principio. He adquirido conocimientos sobre un nuevo gestor de base de datos, y he ampliado mis conocimientos sobre la plataforma Android, así como los datos geoespaciales.

Ha sido una experiencia satisfactoria el poder saber que después de todos estos años de carrera, una idea que se me ocurrió ha podido llevarse a la realidad en un trabajo como este.

# 5 BIBLIOGRAFÍA

[1] Geofencing: https://es.wikipedia.org/wiki/Geovalla
[2] Distribución mercado Android: https://developer.android.com/about/dashboards/index.html
[3] Portal IAAA: http://iaaa.cps.unizar.es/showContent.do?cid=miembros_IAAA.ES
[4] Economía colaborativa: https://es.wikipedia.org/wiki/Consumo_colaborativo
[5] App Guia de Educación Vial: https://play.google.com/store/apps/details?id=appinventor.ai_diegotorop.GuiaEducacionVial
[6] App Location Traker: https://play.google.com/store/apps/details?id=com.WhizNets.locationtracker
[7] App Map My Friends: https://play.google.com/store/apps/details?id=com.vizapps.mapmyfriends&hl=es
[8] Google Maps API: https://developers.google.com/maps/documentation/android-api/?hl=es
[9] MongoDB: https://www.mongodb.com/es
[10] Indexación geoespacial MongoDB: https://docs.mongodb.com/manual/applications/geospatial-indexes/
[11] Datum WGS84: https://es.wikipedia.org/wiki/WGS84
[12] Spring Data MongoDB: http://docs.spring.io/spring-data/mongodb/docs/current/reference/html/
[13] HATEOAS: https://en.wikipedia.org/wiki/HATEOAS
[14] Spring HATEOAS: https://spring.io/understanding/HATEOAS
[15] Openshift: https://www.openshift.com
[16] Android LocationManager: https://developer.android.com/reference/android/location/LocationManager.html
[17] Wikipedia Velocidad Media: https://es.wikipedia.org/wiki/Kilómetro_por_hora
[18] Android Activity LiveCycle: https://developer.android.com/reference/android/app/Activity.html
[19] Android Service: https://developer.android.com/guide/components/services.html
[20] Android PreferenceFragment: https://developer.android.com/reference/android/preference/PreferenceFragment.html
[21] Android MediaPlayer: https://developer.android.com/reference/android/media/MediaPlayer.html
[22] Scrum: https://en.wikipedia.org/wiki/Scrum_(software_development)
[23] Extreme Programming: http://www.extremeprogramming.org
[24] Play! Framework: https://www.playframework.com
[25] Mapa calor runners Zaragoza: http://labs.strava.com/heatmap/#10/-0.87491/41.66468/blue/bike

# 6 ANEXO I. MANUAL DE USUARIO

En el siguiente anexo se detallarán las pantallas disponibles en la aplicación, así como el significado de cada uno de los componentes de la interfaz gráfica disponible para el usuario.

**Slashscreen:** pantalla inicial de la aplicación. Mientras está activa se irán cargando todos los componentes de la aplicación. Una vez que se hayan cargado todos los componentes necesarios, esta pantalla desaparecerá.

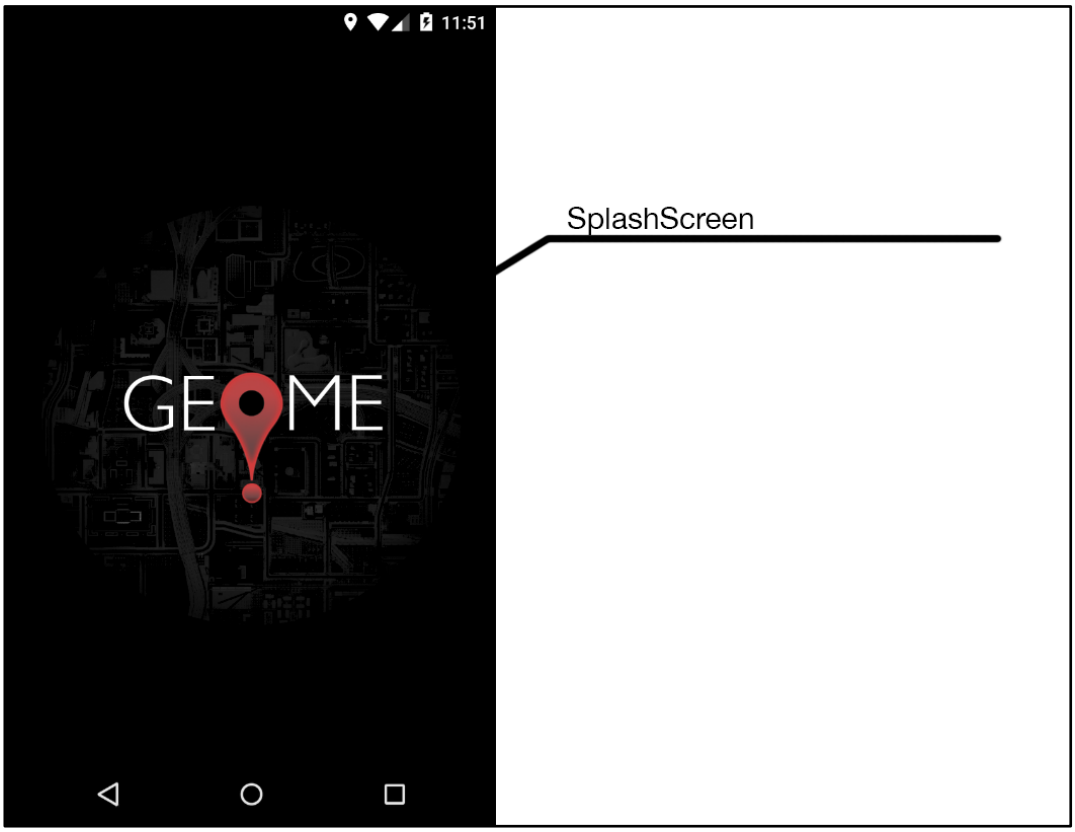


*Figura 28: Splashcreen*

**Diálogo de carga GPS:** en caso de que la ubicación GPS del dispositivo no sea lo suficientemente precisa, aparecerá el siguiente diálogo informando de ello. Una vez se haya obtenido la localización, el diálogo desaparecerá.

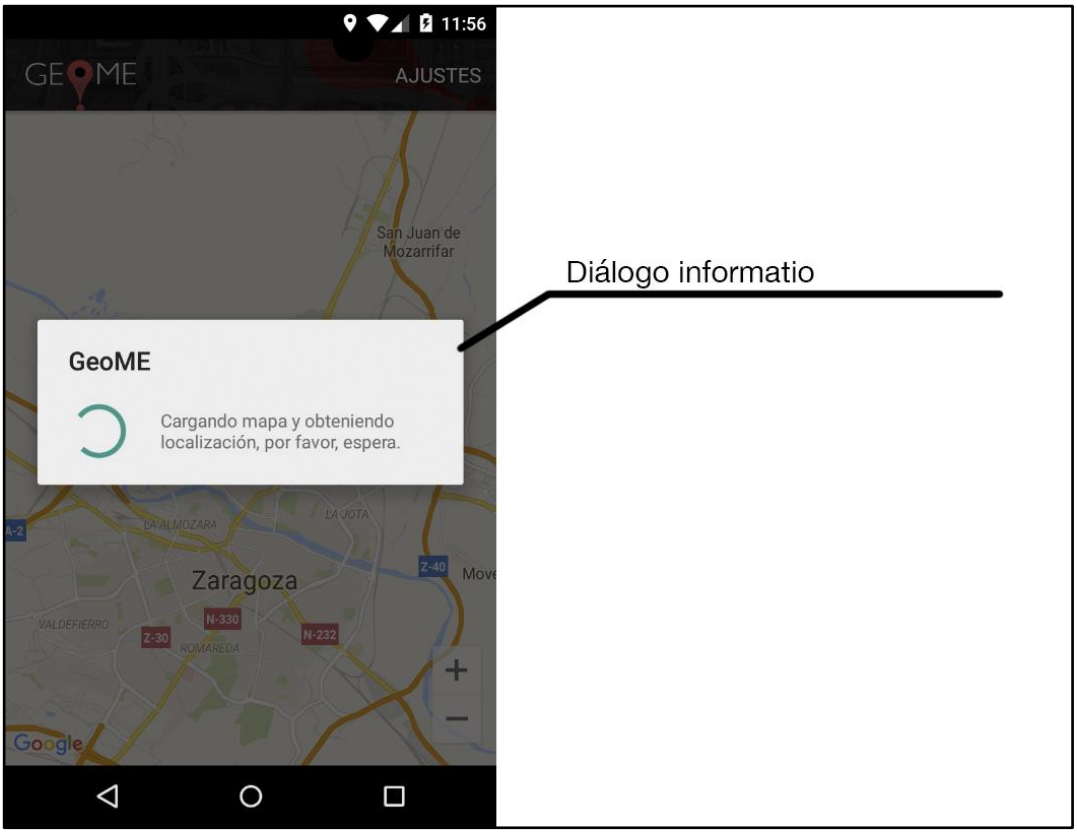


*Figura 29: Dialogo Informativo*

**Pantalla Principal:** menú principal de la aplicación. En la parte superior derecha hay disponible un botón para mostrar el menú de Ajustes. En la parte central estará ubicado un mapa que abarca la completitud de la pantalla. En dicho mapa aparecerá la ubicación actual del usuario a través de un icono azul. Los clientes alrededor aparecerán marcados con un icono verde, amarillo o azul, dependiendo del tipo que sean: peatón, ciclista o vehículo motorizado respectivamente. Un círculo de color azul muestra el radio de monitorización en la que aparecerán los usuarios cercanos.

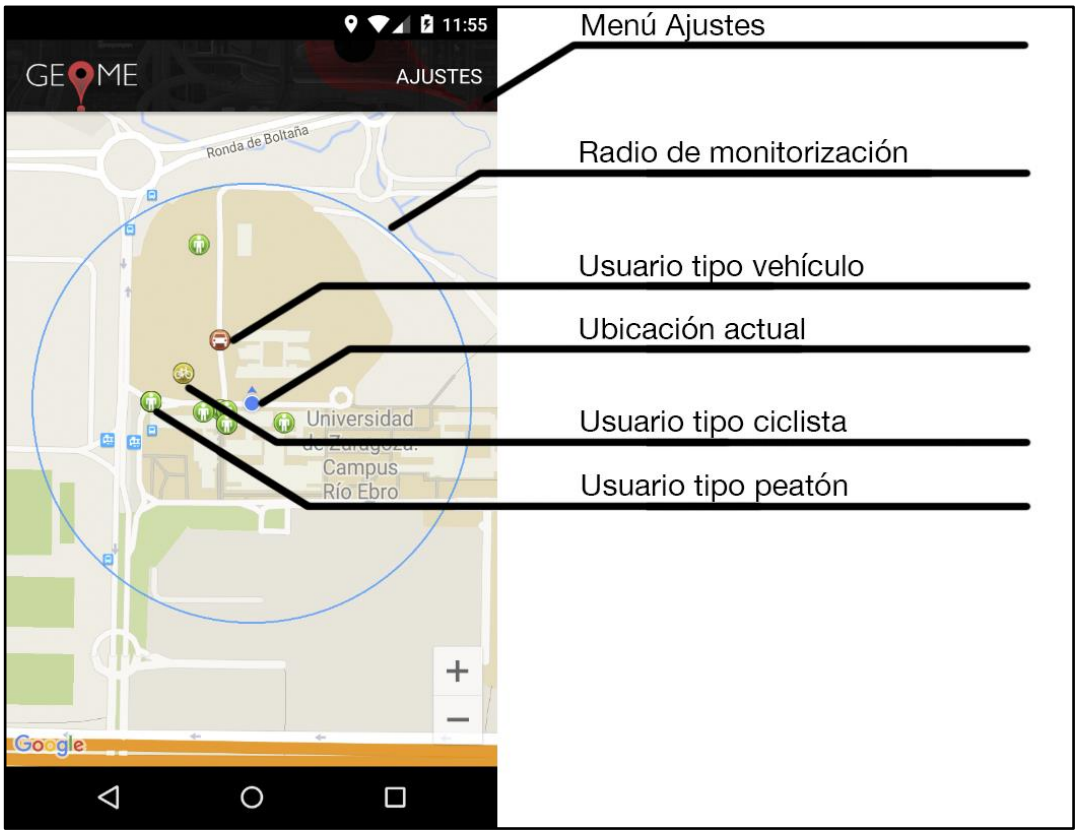


*Figura 30: Pantalla Principal*

**Pantalla de Ajustes:** a través de este menú el usuario puede configurar todos los parámetros disponibles de la aplicación. Es posible configurar los avisos de peatón, ciclista, vehículo y aglomeración; el número máximo de iconos a mostrar en el mapa, así como la distancia del radio de muestreo (en metros); la orientación automática del mapa; la precisión con la que se muestran los usuarios en el mapa; y por último se pueden configurar los avisos cuando la aplicación es ejecutada en segundo plano.


*Figura 31: Menu Ajustes*

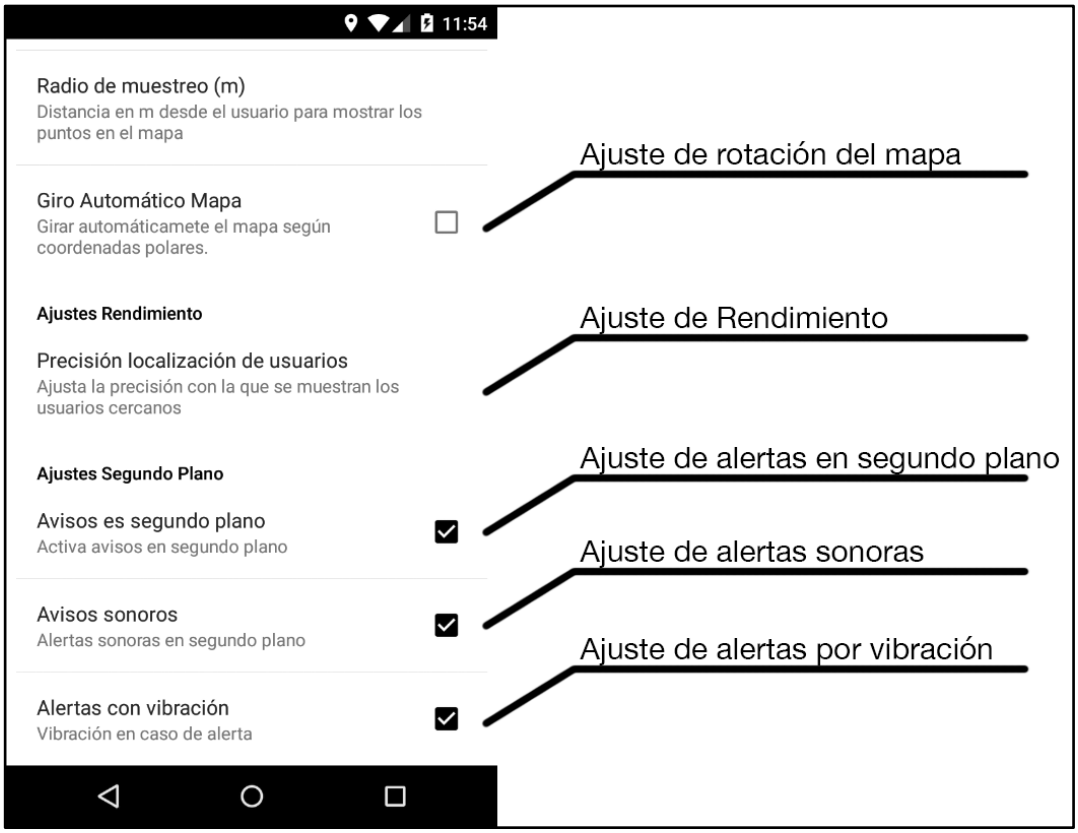


*Figura 32: Menu Ajustes*

# 7 ANEXO II. PLANIFICACIÓN ÁGIL

A finales de los 90 las metodologías ágiles comenzaron a ganar público. Se caracterizan por una colaboración cercana entre el quipo de programación y los expertos en negocio, entrega frecuente de valor al cliente, equipos auto-organizados… Estas características las hacen diferentes frente a la metodología tradicional del modelo en cascada.

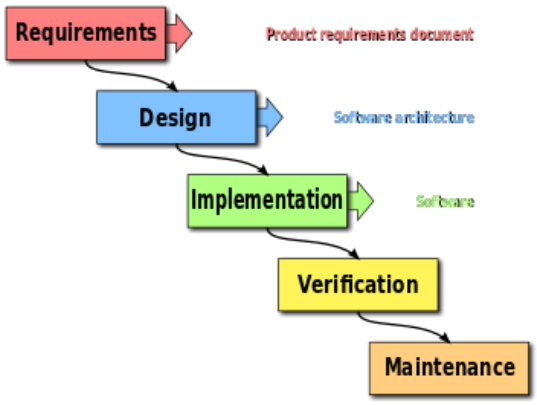


*Figura 33: Modelo ciclo de vida en cascada*

Al implementar estas metodologías en el desarrollo se obtienen ciertas ventajas que facilitan la gestión del proyecto. Los problemas que se presentan al inicio de cualquier proyecto suelen ser de gran envergadura, pero si aplicamos la política "divide y vencerás", dividiendo el problema en trozos más pequeños, el desarrollo es mucho más fácil de abordar. Las metodologías ágiles ayudan a centrarse en lo que realmente importa y olvidar el resto, además se adapta a cambios en mitad de la implementación, cosa que las metodologías tradicionales no lo permiten.

¿Por qué se ha elegido las metodologías ágiles para la realización de este proyecto? A pesar de estar diseñadas para pequeños equipos de trabajo, se han aprovechado los mejores principios de dos metodologías en concreto: Scrum y Extreme Programming. La estimación y la división del tiempo en sprints ayuda a la gestión, la revisión y validación permite corregir bugs, la entrega frecuente motiva a los desarrolladores… También ha influido el desconocimiento de las tecnologías a utilizar para su desarrollo, lo que provoca mucha dificultad a la hora de hacer la estimación, el análisis, y el diseño completo de todo el ciclo de vida del proyecto. Mediante las metodologías ágiles se ha podido distribuir todo este proceso en distintos sprints, lo que facilita mucho la tarea.

# 8 ANEXO III. SERVICIOS REST

## 8.1 ¿Qué son los servicios web?

Según el W3C (World Wide Web Consortium): "Un servicio Web (WS) es una aplicación software identificada por un URI (Uniform Resource Identifier), cuyas interfaces se puede definir, describir, y descubrir mediante documentos XML. Los Servicios Web hacen posible la interacción entre agentes de software (aplicaciones) utilizando mensajes XML intercambiados mediante protocolos de internet"

Desde siempre la comunicación entre las aplicaciones se realizaba a través de un único tipo de middleware, el cual posee los dos extremos de la comunicación. Por otro lado, los servicios web se basan en un conjunto de estándares que permiten la comunicación entre distintos sistemas sin la necesidad de un "moderador" que reenvíe estos mensajes, así como tampoco importa el lenguaje de programación o incluso el sistema operativo.

Existen muchas otras definiciones alternativas, lo que muestra la complejidad a la hora de concretar una adecuada descripción que englobe todo lo que son e implican los servicios web, es por ello que se describe a continuación el modelo REST utilizado durante el proyecto, describiendo su funcionamiento así como sus ventajas e inconvenientes de su implantación.

## 8.2 RESTful

Los principios de REST se basan estrictamente en cómo los recursos son definidos en las arquitecturas en red. La motivación de REST se basa en el principio de porqué la red ha sido tan exitosa. A groso modo, la Web es un espacio de URIs, las cuales identifican recursos. Esta representación se realiza por medio de mensajes a través de la Web. Este estilo de arquitectura subyacente a la Web es el modelo REST. ¿Cuáles son los principios de REST?

- Escalabilidad de la interacción con los componentes. La Web ha crecido exponencialmente sin degradar su rendimiento.
- Generalidad de interfaces. Gracias al protocolo HTTP, cualquier cliente puede interactuar con cualquier servidor HTTP sin ninguna configuración especial.
- Puesta en funcionamiento independiente. Los clientes y servidores pueden tener distintos ciclos de vida, un servidor antiguo debe ser capaz de comunicarse con clientes actuales y viceversa. HTTP permite la extensibilidad mediante el uso de las cabeceras, a través de las URIs.

- Compatibilidad con componentes intermedios. Los más populares intermediaros son varios tipos de proxys para Web. Algunos de ellos, las caches, se utilizan para mejorar el rendimiento, o también los firewalls, que permiten reforzar aspectos de seguridad.

¿Dónde es útil REST? Dependiendo del entorno, es necesario decidir cuál es el estilo adecuado para nuestra aplicación, algunos escenarios en los que sería útil usar REST serían los siguientes:

- El servicio Web no tiene estado. Una buena comprobación de esto consistiría en considerar si la interacción puede sobrevivir a un reinicio del servidor, lo que da la posibilidad de introducir o cambiar servidores averiados sin afectar a los usuarios.
- En una infraestructura de caching. Si los datos que el servicio Web devuelve pueden ser cacheados, entonces la infraestructura de caching que los servidores Web y los intermediarios proporcionan pueden incrementar el rendimiento.
- Cuando ambas partes deben estar de acuerdo en el modo de intercambiar de información. Tanto el productor como el consumidor del servicio conocen el contexto y contenido que va a ser comunicado.
- El ancho de banda es importante y necesita ser limitado. REST es particularmente útil en dispositivos con escasos recursos como teléfonos móviles, donde la sobrecarga de las cabeceras y capas adicionales de los elementos SOAP debe ser restringida.

# 9 ANEXO IV. MODELO DE NEGOCIO

El modelo de negocio tiene como principal objetivo generar ingresos y beneficios. Esto se consigue realizando un análisis y comprendiendo elementos tales como el cliente, el producto o el mercado. Tampoco es conveniente abusar de esto e intentar aplicar modelos de negocio donde no es conveniente, ya que puede generar malestar en el usuario (publicidad intrusiva, pagos por cualquier característica…) y puede llegar a reducir el número de usuarios. Debido a esto es esencial analizar el producto para conocer qué modelos de negocio aplicar.

Resulta interesante destacar que para cumplir el objetivo principal de este proyecto, el numero de usuarios tiene que ser muy elevado, por lo que no es conveniente obtener beneficios directamente el usuario (hacer la aplicación de pago). Hay que buscar otro plan de negocio. Tal y como se ha descrito en el apartado *'4.2.3 Análisis de Datos'*, la aplicación es capaz de llevar una monitorización de la ubicación de usuarios. Estos datos pueden ser de gran utilidad para ciertas empresas, ya que pueden conocer, por ejemplo, las áreas donde hay más tránsito de usuarios, lugar idóneo para abrir una nueva tienda. De esta manera se puede ofrecer a las empresas el análisis de esta información geográfica para obtener el máximo beneficio en sus tiendas, no sin antes recibir a cambio ingresos económicos o algún tipo de beneficio.

Otra metodología para generar ingresos sería introducir publicidad integrada con el funcionamiento la aplicación, de tal manera que no afecte a la usabilidad del usuario. La aplicación se basa en mostrar marcadores sobre un mapa, esta característica se puede explotar incluyendo 'Points Of Interests'. Estos puntos de interés pueden publicitar locales, restaurantes, cines, etc… De esta manera se puede incluir publicidad de tal forma que el usuario no sea consciente de ella, y la trate como una característica más de la aplicación.