GRADO EN INGENIERIA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE INFORMACIÓN

TRABAJO FIN DE GRADO

2015 / 2016

TÍTULO DEL TFG

WALKABILITY CAPTURER

nombre del documento

Memoria TFG

DATOS DE LA ALUMNA O DEL ALUMNO	**DATOS CODIRECCIÓN**	
NOMBRE David	NOMBRE Maider	NOMBRE Mikel
APELLIDOS Puerto Caldero	APELLIDOS Azanza Sesé	APELLIDOS Villamañe Gironés
	DEPARTAMENTO Lenguajes y sistemas informáticos	DEPARTAMENTO Lenguajes y sistemas informáticos
FDO.:	FDO.:	FDO.:
FECHA: 13-06-2016	FECHA: 13-06-2016	FECHA: 13-06-2016

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

1.1 RELEVANCIA CIENTÍFICA Y SOCIAL DEL PROBLEMA ABORDADO

La movilidad urbana es uno de los principales problemas (Cossío, 2013) que tienen las ciudades y más desde la generalización del uso del coche particular (Ecomotriz.com, 2013) para estos fines. Esto influye de manera negativa en la ciudad, ya que aumenta los niveles de polución, ruido, etc., así como problemas de aparcamiento. En grandes ciudades, como por ejemplo Madrid, ya se han dado situaciones de alerta por alta contaminación (García Gallo, 2015), lo que obligó a restringir la circulación en la ciudad hasta que los valores volviesen a niveles normales.

Lo ideal sería que los ciudadanos percibieran el moverse por la ciudad a pie como una actividad amigable, ya que esto repercutiría de manera positiva, tanto en el propio ciudadano, como en la ciudadanía en general. Por desgracia esto no es así (Ecomotriz.com, 2013). En muchos casos esta percepción es debida al diseño de las ciudades, como por ejemplo la presencia de numerosas calles con pendientes pronunciadas, o por falta de equipamiento, como puede ser la ausencia de farolas en una zona. Esto, unido a un estilo de vida cada vez más sedentario, hace que se opten por otros estilos de movilidad.

Desde el grupo de investigación CRIM-AP de la UPV/EHU[1] se ha venido trabajando para una mejor comprensión de la manera en la que el diseño urbano puede proveer de oportunidades para la integración y cohesión social, la conducta sostenible, y el uso libre y seguro de los espacios públicos. Actualmente sus líneas de trabajo buscan profundizar tanto en el modo en que los ciudadanos viven la seguridad en los espacios públicos; como en la manera en que toman sus decisiones de movilidad sostenible, especialmente a pie. Se interesan en este sentido, y entre otras variables, por la "seguridad percibida". Nos referimos al nivel de seguridad que de modo subjetivo perciben los ciudadanos, y que no siempre coincide con la seguridad objetiva; es decir, podemos percibir como inseguros lugares que no necesariamente presentan una alta tasa de delitos, y en consecuencia, evitarlos, limitando nuestra libertad de uso de los espacios públicos.

De hecho, investigaciones previas han mostrado la relevancia de esta variable para la toma de decisiones de movilidad en el espacio público urbano (San-Juan, Vozmediano, & Vergara, 2012), poniendo de manifiesto que la manera en que percibimos la ciudad, es, junto a la disponibilidad de diseños más o menos "walkables" (Cervero & Kockelman, 1997) -es decir pensados para promover la movilidad a pie- un aspecto relevante para una mejor comprensión de la movilidad sostenible y saludable. En definitiva, comprender la movilidad es entender que la ciudad no es solo como ha sido diseñada, también es como lo percibimos y la vivimos, y no resulta sencillo investigar, con las metodologías tradicionales, este último aspecto.

[1] http://www.ehu.eus/es/web/dms/equipo

1.2 NUEVAS ESTRATEGIAS PARA AVANZAR EN LA COMPRESIÓN DE LA RELACIÓN DISEÑO URBANO – MOVILIDAD SOSTENIBLE: EL PAPEL DE LOS DISPOSITIVOS MÓVILES

Es por ello que a la hora de avanzar en la compresión de los aspectos descritos, se ha juzgado como ineludible innovar en las metodologías empleadas. La metodología más comúnmente empleada en los estudios de la temática ha sido la de encuesta, con diseño transversal, que pide al participante que informe, en un único momento, de sus experiencias cotidianas de movilidad, percepción de la ciudad, rutas habitualmente empleadas… Esto presenta ciertas limitaciones como el recuerdo únicamente de experiencias salientes o falta de detalle en la información geográfica proporcionada.

Actualmente, y cada vez más (Morales, 2015), los dispositivos móviles están presentes en nuestras vidas y ofrecen la posibilidad de recabar tanto datos de movilidad así como información de los usuarios sobre sus percepciones y conductas recientes.

El desarrollo y difusión de la aplicación "Walkability capturer" permitirá alcanzar el objetivo de analizar la movilidad a pie en las ciudades y la toma de decisiones de movilidad con la información proporcionada por los usuarios de la aplicación móvil. A través de la aplicación se obtendrán datos agregados del uso de los escenarios urbanos al desplazarse a pie, de modo que conoceremos los lugares más y menos usados a la hora de caminar por la ciudad, recopilando además información sobre la toma de decisiones en este ámbito.

Más concretamente, la aplicación móvil tiene dos funcionalidades principales.

Por un lado, estará la monitorización de los movimientos del usuario. Lo que se hará será registrar todas las rutas a pie realizadas para su posterior estudio. Para que el desplazamiento del usuario sea tomado como una ruta a pie deberá tener una distancia mínima y estar realizado a una velocidad predeterminada. Estos valores son definidos por el equipo investigador y podrán variar a lo largo del periodo de recogida de datos. Además, depender de que el usuario recordase activar el seguimiento cada vez que se desplazase podría suponer una pérdida significante de datos, por lo que se ha decidido que se realice de manera automática. El dispositivo, gracias a los valores predeterminados ya mencionados, decidirá cuándo activar el registro de la ruta en cuestión.

Otra de las funcionalidades principales que tendrá la aplicación será la posibilidad de valorar las rutas realizadas. El sistema, de entre las rutas realizadas durante las últimas 24 horas, seleccionará una siguiendo el algoritmo especificado por el equipo de investigadores y realizará un breve cuestionario que variará dependiendo la finalidad de la ruta, el género del usuario y el horario en el que se ha realizado la ruta. Este cuestionario ha sido diseñado por el equipo investigador haciendo referencia a aspectos como la belleza de la ruta realizada o la cantidad de vegetación presente. También cabe destacar que se trata de un cuestionario dinámico, ya que podrá variar según las necesidades del estudio.

Entre las distintas opciones disponibles para el desarrollo de aplicaciones móviles, como puede ser el lenguaje Java para sistemas Android u Objective-C para iOS, se ha optado por el uso del framework de desarrollo móvil Apache Cordova[2]. Gracias a este framework, basado en las tecnologías estándar web como HTML5, CSS3 y JavaScript, se puede realizar un desarrollo multiplataforma. Esto permite que un único desarrollo funcione en las distintas plataformas, lo que evita tener que conocer el lenguaje nativo de cada una y tener que realizar varios desarrollos.

Debido a estas características mencionadas y los ligeros conocimientos previos sobre las tecnologías web, se ha elegido este modo de implementación. Esta elección puede aportar una ampliación de los conocimientos existentes, así como un mayor número de usuarios.

[2] https://cordova.apache.org/

2 PLANTEAMIENTO INICIAL

2.1 OBJETIVOS

La realización de este trabajo tiene como objetivo principal **crear una aplicación móvil**, la cual permitirá el registro de toda ruta a pie realizada por el usuario y la valoración de una ruta realizada durante el día, para la cual cada usuario/a informará de una serie de cuestiones relacionadas con la percepción de ese trayecto y la toma de decisiones de movilidad. Todo esto permitirá analizar cada ruta para determinar en qué medida las calles de la ciudad en la que se encuentre el usuario, son caminables, empleando el término en inglés, en qué medida presenta "walkability", es decir su diseño invita a caminar. Al ser un proyecto propuesto por el grupo de investigación CRIM-AP de la UPV/EHU[3], todos estos datos también serán usados para analizar cómo las personas perciben el espacio público de la ciudad y toman decisiones de movilidad, con lo que los resultados ayudarán a proponer medidas que hagan a las ciudades más caminables e incrementen la probabilidad de que los ciudadanos opten por alternativas de movilidad sostenibles y saludables.

Dentro de los objetivos secundarios del proyecto está el **aprendizaje de desarrollo de aplicaciones móviles**. Esta actividad está cada vez más solicitada, ya que las empresas lo ven como una ventaja competitiva (Peña, 2014) que les puede hacer destacar frente al resto de empresas del mercado. Hoy en día ya no basta con estar presentes en Internet, deben estar ahí donde estén sus clientes.

Otro de los objetivos secundarios es el **aprendizaje y familiarización con el framework *Apache Cordova*[4]**, que permite realizar aplicaciones nativas para los distintos sistemas operativos que tienen los smartphones (iOS, Android, etc.) sin la necesidad de programar en sus lenguajes nativos. Al tratarse de un aprendizaje autónomo, porque es una herramienta que no se ha visto durante la carrera, es posible alcanzar un alto nivel de conocimientos que beneficiarán en un futuro a la hora de buscar trabajo porque, como se ha mencionado en el párrafo anterior, es una actividad en auge.

Por último, uno de los objetivos secundarios que no se puede olvidar es **determinar la capacidad de realizar de manera autónoma un proyecto de principio a fin**. Durante la carrera se han realizado distintos proyectos, pero normalmente estos se realizaban en grupo y eran sencillos ya que se buscaba aplicar lo aprendido en clase. Aquí se deberán aplicar los distintos conocimientos adquiridos con el fin de demostrar que se está capacitado para la vida laboral.

[3] http://www.ehu.eus/es/web/dms/equipo
[4] https://cordova.apache.org/

2.2 ALCANCE

A la hora de llevar a cabo un proyecto, una de las tareas esenciales es definir su alcance. Esto significa definir las fases y funcionalidades que debe tener el proyecto para poder cumplir cada uno de los objetivos descritos anteriormente. También se definirá su ciclo de vida, o lo que es lo mismo, cómo se va a desarrollar.

2.2.1 CICLO DE VIDA

Para el desarrollo de este proyecto se ha elegido un ciclo de vida incremental. Este tipo de ciclo de vida está compuesto, a su vez, por ciclos de vida en cascada. Tras cada iteración, se consigue un prototipo con más funcionalidades que su predecesor, como podemos observar en la Ilustración 1 - Ciclo de vida incremental. Se podría comparar con las fases de construcción de una casa, ya que tras cada iteración conseguimos que el prototipo sea más completo.

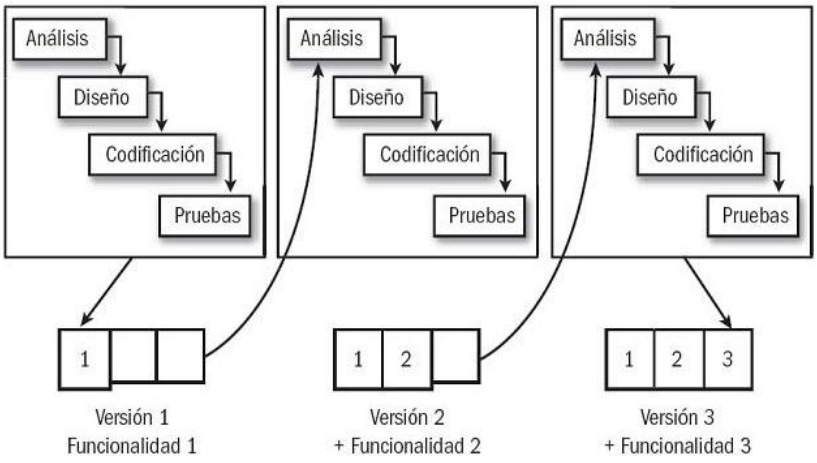

Ilustración 1 - Ciclo de vida incremental (Arana Roa, 2013)

Una de las razones principales por las que se ha elegido el ciclo de vida incremental es porque se basa en la generación de prototipos. Aunque debemos tener en cuenta que la gestión puede ser más compleja, este tipo de desarrollo se adecua a la forma en la que se pretende trabajar, ya que se puede apreciar progresivamente la consecución de los objetivos propuestos sin haber finalizado por completo el proyecto. Esto permite que la calidad del producto sea mayor y, en caso de error grave, este solo afectará al último prototipo.

2.2.2 FASES DEL PROYECTO

Como en todo proyecto, existe una serie de fases por las que este debe pasar y, a su vez, cada fase está dividida en una lista de tareas a desarrollar. En este caso las fases son:

- **Gestión**: es la primera fase por la que debe pasar el proyecto, aunque se alargará durante toda la duración del mismo. En ella se definirá el proyecto en sí, de lo que va a tratar el proyecto. Para lograr esto, habrá que buscar información y definir las tareas que lo van a componer. Además, durante su duración se controlará que se están cumpliendo los plazos establecidos, gestionar los riesgos, replanificar si fuera necesario, etc.

- **Análisis**: en esta fase se definen las funcionalidades que deberá tener el proyecto, así como el lenguaje y las herramientas necesarias para su desarrollo.

- **Diseño**: al llegar a esta fase se deberá definir cuál será la estructura que tendrá la aplicación, así como sus interfaces, cumpliendo con lo descrito en las fases anteriores de gestión y análisis. También se diseñarán los algoritmos necesarios para el funcionamiento de la aplicación para su posterior desarrollo.

- **Implementación**: se definirán los distintos prototipos en los que se dividirá el desarrollo de la aplicación y se implementarán. En este caso los prototipos a desarrollar serán 5.

    1. Almacenar y recuperar datos de una base de datos local.
    2. Almacenar y recuperar datos de una base de datos externa.
    3. Detectar una ruta y monitorear la ruta.
    4. Visualizar una ruta ya realizada.
    5. Valorar una ruta.

- **Plan de pruebas**: al tratarse de un ciclo de vida por prototipos, se realizará un plan de pruebas por cada prototipo implementado. Con esto se conseguirá un correcto funcionamiento del prototipo realizando las pruebas pertinentes cuando esté finalizado. Tras completar la implementación, se hará una serie de pruebas finales, para así asegurar el que el conjunto funciona correctamente.

- **Documentación**: esta fase será ejecutada en paralelo al proyecto. En ella se realizará un documento donde quedará plasmado el proyecto de manera escrita, así como lo acontecido durante su desarrollo. También incluye la preparación de la defensa.

2.2.3 ESTRUCTURA DE DESCOMPOSICIÓN DEL TRABAJO

En la Estructura de Descomposición del Trabajo (EDT) (Ilustración 2 - Gráfico EDT), se representan gráficamente las fases y tareas que componen el proyecto.

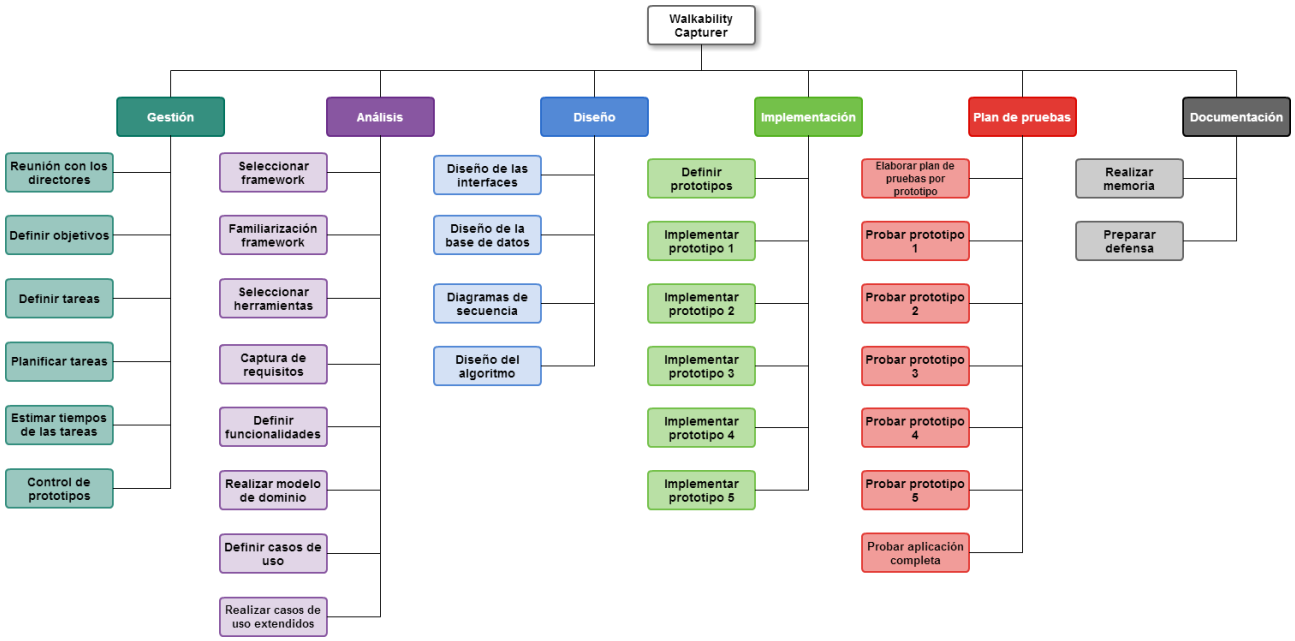

Ilustración 2 - Gráfico EDT

A continuación, se detallan el número de horas estimadas que se van a dedicar a cada tarea definida en el EDT

2.2.3.1 GESTIÓN

- **Reunión con los directores**: Reunión en la que se conocerá más en profundidad el proyecto y donde se determinarán los plazos de ejecución. Duración: 3 horas.
- **Definir objetivos**: Establecer los objetivos que se van a llevar a cabo en el proyecto. Duración: 2 horas.
- **Definir tareas**: Establecer las tareas que van a ser necesarias para llevar realizar el proyecto. Duración: 3 horas.
- **Planificar tareas**: Establecer un plan con el cual desarrollar todas las tareas definidas. Duración: 3 horas.
- **Estimar tiempos de las tareas**: Realizar una estimación aproximada del tiempo necesario a emplear en el desarrollo de cada tarea. Duración: 4 horas.
- **Control de prototipos:** Tras realizar cada prototipo, se comprobará si todo está yendo según lo planificado. Duración: 5 horas.

Total horas de la fase de gestión: 20 horas.

2.2.3.2 ANÁLISIS

- **Seleccionar framework**: Decidir qué framework se va a usar para el desarrollo de la aplicación buscando información. Duración: 30 horas.
- **Familiarización framework**: Tras seleccionar el framework, búsqueda y visualización de tutoriales que faciliten el aprendizaje. Duración: 45 horas.
- **Seleccionar herramientas**: Elegir las herramientas que se van a emplear para el desarrollo del proyecto. Duración: 20 horas.
- **Captura de requisitos**: Determinar los requisitos del usuario para, más adelante, definir las funcionalidades de la aplicación. Duración: 5 horas.
- **Definir funcionalidades**: Definir las funcionalidades que se van a implementar en la aplicación. Duración: 2 horas.
- **Realizar modelo de dominio**: Estudio de las necesidades de la base de datos y la relación entre las diferentes tablas. Duración: 3 horas.
- **Definir casos de uso**: Realización del diagrama de casos de uso de la aplicación, donde se especifica el papel que juegan los actores con los casos de uso relacionados. Duración: 2 horas.
- **Realizar casos de uso extendidos**: Realización de la documentación de todos los casos de uso de manera detallada, explicando en profundidad la funcionalidad de cada uno. Duración: 3 horas.

Total horas de la fase de análisis: 110 horas.

2.2.3.3 DISEÑO

- **Diseño de las interfaces**: Diseño en papel de todas las interfaces de la aplicación. Duración: 1 horas.
- **Diseño de la base de datos**: Diseño de la base de datos a utilizar a partir del modelo de dominio. Duración: 2 horas.
- **Diagrama de secuencia**: Realización de los diagramas de secuencia de la aplicación. Duración: 25 horas.
- **Diseño del algoritmo**: Definir el algoritmo de funcionamiento esencial de la aplicación. Duración: 15 horas.

Total horas de la fase de diseño: 43 horas.

2.2.3.4 IMPLEMENTACIÓN

- **Definir prototipos**: Especificar en cuántos prototipos se dividirá el desarrollo de la aplicación. Duración: 2 horas.
- **Implementar prototipo 1**. Duración: 20 horas.
- **Implementar prototipo 2**. Duración: 20 horas.
- **Implementar prototipo 3**. Duración: 50 horas.
- **Implementar prototipo 4**. Duración: 25 horas.
- **Implementar prototipo 5**. Duración: 15 horas.

Total horas de la fase de implementación: 132 horas.

2.2.3.5 *PLAN DE PRUEBAS*

- **Elaborar plan de pruebas por prototipo**: Desarrollar un plan de pruebas que se ajuste a lo implementado en cada prototipo y cuyo objetivo sea cubrir todas las posibilidades de error. Duración: 4 horas.
- **Probar prototipo 1**. Duración: 5 horas.
- **Probar prototipo 2**. Duración: 5 horas.
- **Probar prototipo 3**. Duración: 8 horas.
- **Probar prototipo 4**. Duración: 5 horas.
- **Probar prototipo 5**. Duración: 5 horas.
- **Probar aplicación completa**. Duración: 10 horas.

Total horas del plan de pruebas: 42 horas.

2.2.3.6 *DOCUMENTACIÓN*

- **Realizar memoria**: Realizar una memoria donde se recopile toda la labor realizada en el desarrollo de la aplicación, incluyendo toda la documentación elaborada para la realización del proyecto. Duración: 75 horas.
- **Preparar defensa**: Preparar la defensa del trabajo de fin de grado. Duración: 25 horas.

Total de horas de la fase de documentación: 100 horas.

2.3 *PLANIFICACIÓN TEMPORAL*

En este apartado se estimará cuántas horas durará el proyecto en función de la duración de cada tarea, así como determinar cuántas semanas serán necesarias para acabarlo (Tabla 1 - Planificación temporal). Además, se dispondrá de un gráfico Gantt para representar la planificación temporal a lo largo de las semanas.

Tareas	Dedicación estimada
Gestión	**20**
Reunión con los directores	3
Definir objetivos	2
Definir tareas	3
Planificar tareas	3
Estimar tiempos de las tareas	4
Control de prototipos	5
Análisis	**110**
Seleccionar framework	30
Familiarización framework	45
Seleccionar herramientas	20
Captura de requisitos	5
Definir funcionalidades	2
Realizar modelo de dominio	3
Definir casos de uso	2
Realizar casos de uso extendidos	3
Diseño	**43**
Diseño de las interfaces	1
Diseño de la base de datos	2
Diagrama de secuencia	25
Diseño del algoritmo	15
Implementación	**132**
Definir prototipos	2
Implementar prototipo 1	20
Implementar prototipo 2	20
Implementar prototipo 3	50
Implementar prototipo 4	25
Implementar prototipo 5	15
Plan de pruebas	**42**
Elaborar plan de pruebas por prototipo	4
Probar prototipo 1	5
Probar prototipo 2	5
Probar prototipo3	8
Probar prototipo4	5
Probar prototipo5	5
Probar aplicación completa	10
Documentación	**100**
Realizar memoria	75
Preparar defensa	25
Total	**447**

Tabla 1 - Planificación temporal

Estimando que se van a dedicar al proyecto un promedio de 35 horas semanales, este debería estar acabado en 13 semanas.

Teniendo en cuenta que el proyecto se realizará tanto sábados, domingos y otros días festivos, se ha realizado un diagrama de Gantt (Ilustración 3 - Diagrama Gantt (1/2) y Ilustración 4 - Diagrama Gantt (2/2)) en el que se muestra de manera detallada como quedaría la planificación organizada a través de las semanas. También se debe tener en cuenta que durante los meses de agosto y septiembre no se realizará ninguna tarea, por lo que se estima la finalización para principios del mes diciembre.

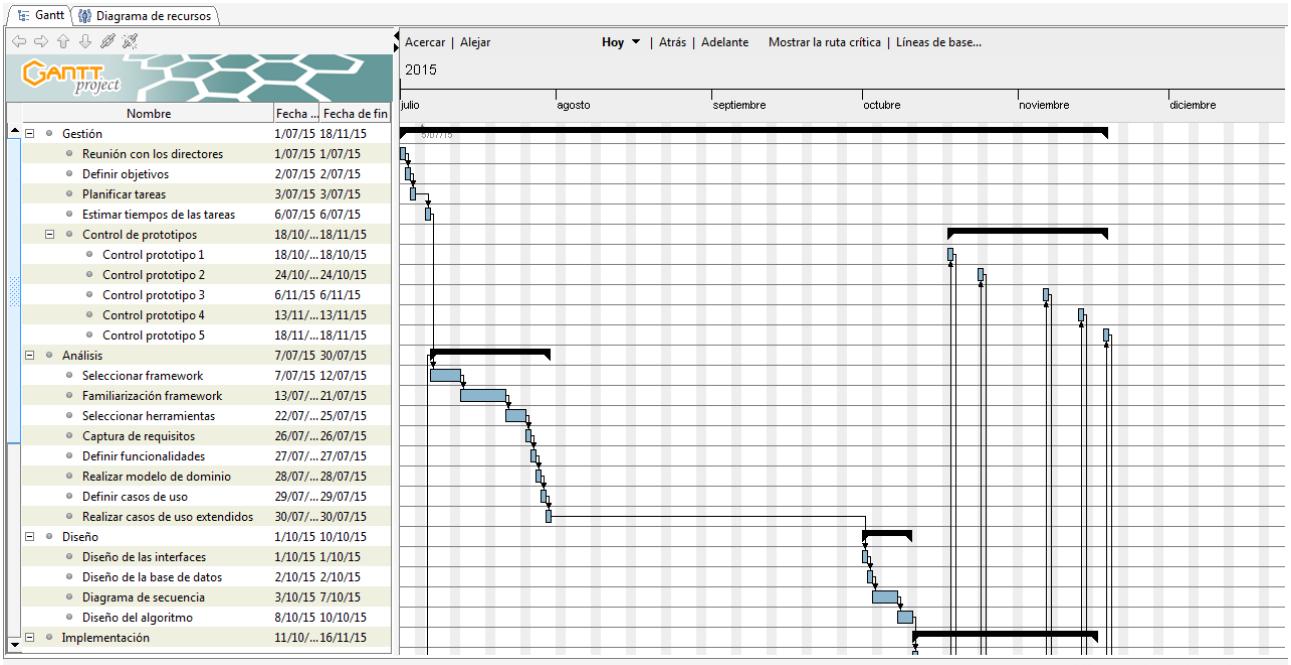

Ilustración 3 - Diagrama Gantt (1/2)

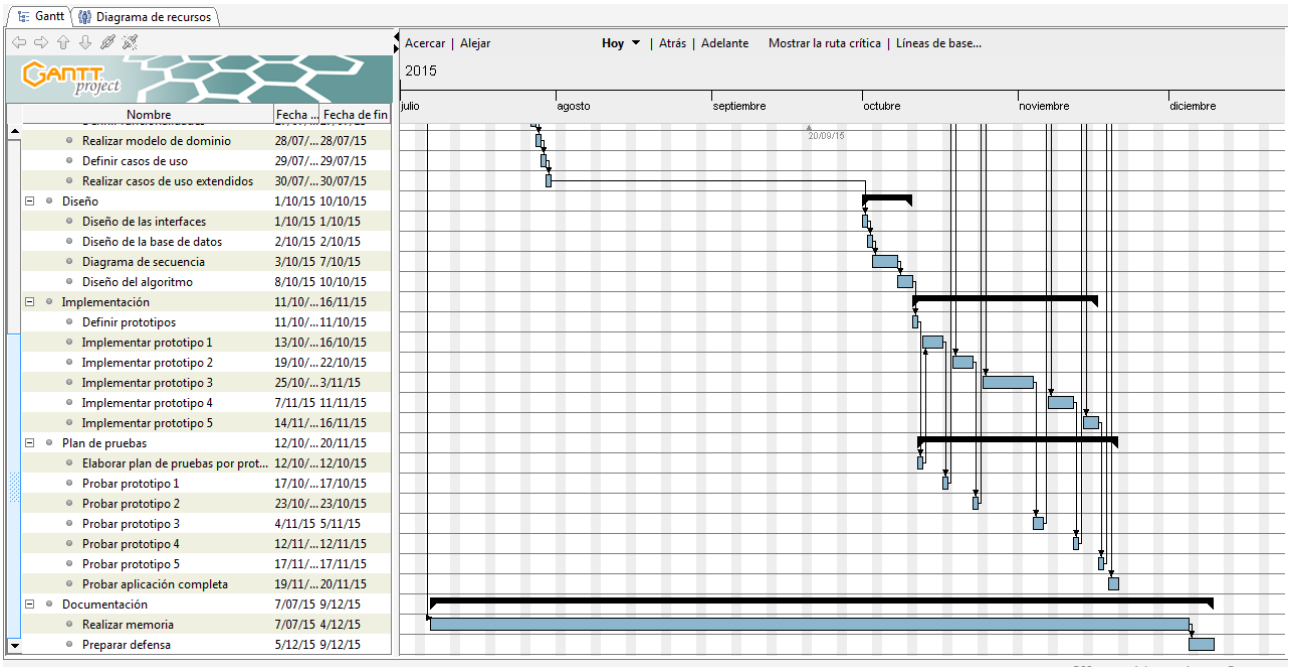

Ilustración 4 - Diagrama Gantt (2/2)

2.4 *ARQUITECTURA*

La arquitectura de la aplicación es de tipo cliente/servidor. La aplicación estará alojada en los dispositivos móviles donde se halla instalada y también hará uso del almacenamiento local de los terminales para guardar los datos recogidos. En el servidor se volcarán todos los datos recogidos por los distintos dispositivos que usen la aplicación para su posterior uso por parte del grupo de investigadores. Aquí también se almacenarán los datos de configuración proporcionados por el equipo de psicólogos para que la aplicación determine qué es una ruta a pie y qué no.

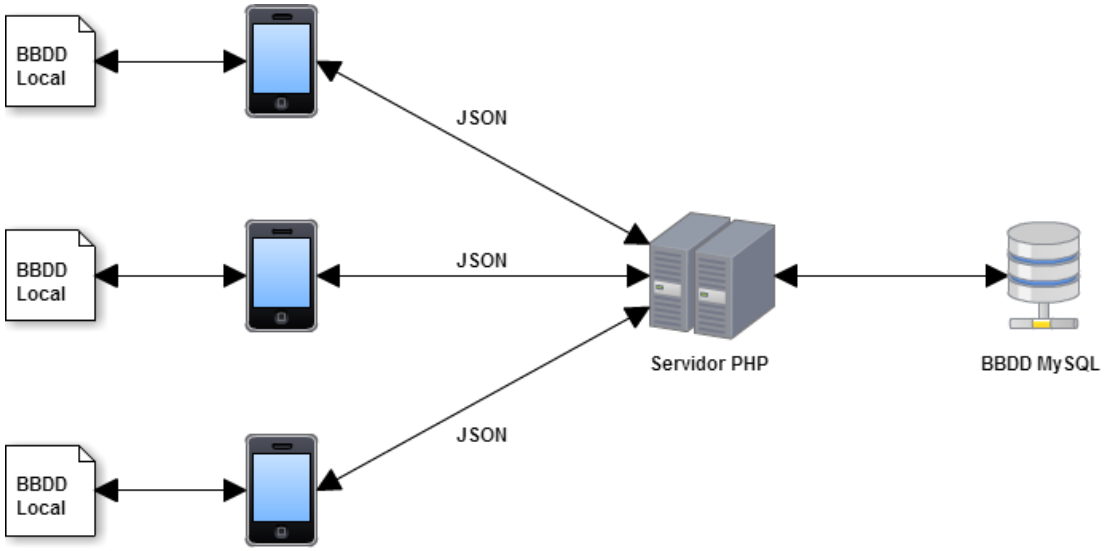

Ilustración 5 - Arquitectura en tres capas

La arquitectura resultante será la conocida como arquitectura o modelo de tres capas (Puente Cedillo, 2014). Al contenido de la base de datos se accederá mediante llamadas AJAX al archivo PHP correspondiente. Este será el encargado de realizar las tareas necesarias con la base de datos y, cuando se solicite, devolver datos a la aplicación. Tanto para enviar como para recibir datos se usará el objeto JSON[5].

2.5 HERRAMIENTAS

Para el desarrollo del proyecto se usarán las herramientas que a continuación se detallan.

2.5.1 SUBLIME TEXT 3

Sublime Text 3[6] es un editor de texto que será usado para realizar toda la lógica de la aplicación. Éste ha sido usado con anterioridad en otras asignaturas, por lo que es familiar a la hora de usarlo. También está considerado como uno de los mejores editores que existen actualmente (Román, 2014), por lo tanto será de gran ayuda. Permite la codificación en un gran número de lenguajes de programación, lo que brinda la oportunidad de no tener que usar varios programas para el desarrollo.

[5] http://www.json.org/
[6] https://www.sublimetext.com/

2.5.2 CACOO

Cacoo[7] es una interesante herramienta que permite crear diagramas de forma online con la ventaja de poder elaborarlos de forma colaborativa con otros usuarios. Tiene una interfaz sencilla y clara que facilita mucho su uso. También permite exportar los trabajos en formato PNG y compartirlos a través de enlaces y en las redes sociales. Con ella se desarrollarán desde los gráficos del EDT hasta los diagramas de secuencia.

2.5.3 API DE GOOGLE MAPS

La API de Google Maps[8] permite la inclusión de mapas tanto en aplicaciones móviles como en páginas web. Se trata de una herramienta que facilita el uso de los mapas además de proporcionar otro tipo de utilidades, como pueden ser geolocalización, street view, medida de distancias, etc…

2.5.4 APACHE CORDOVA

Cordova[9] es un framework de desarrollo móvil de código abierto. Éste permite utilizar las tecnologías estándar web como HTML5, CSS3 y JavaScript para el desarrollo multiplataforma, evitando el lenguaje de desarrollo nativo de cada plataforma móvil.

2.5.5 GANTT PROJECT

Gantt Project[10] es un programa que permite la representación gráfica de la planificación temporal del proyecto. En él, se van añadiendo las distintas tareas con la fecha de inicio y su duración y el propio programa las plasma a lo largo del tiempo gráficamente.

2.5.6 MICROSOFT OFFICE WORD 2007

Programa de ofimática que será utilizado para el desarrollo de la memoria y la documentación.

2.5.7 DROPBOX

Dropbox[11] es un servicio de alojamiento de archivos multiplataforma en la nube. El servicio permite a los usuarios almacenar y sincronizar archivos en línea y entre ordenadores y compartir archivos y carpetas con otros usuarios, además de con móviles y tablets.

2.5.8 GIT

Git[12] es un software de control de versiones, ya que es importante el mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Al tratarse de un almacenamiento en la nube, permite su recuperación en caso de pérdida de datos locales.

[7] https://cacoo.com
[8] https://developers.google.com/maps/?hl=es
[9] https://cordova.apache.org/docs/es/3.1.0/guide/overview/
[10] http://www.ganttproject.biz/
[11] https://www.dropbox.com
[12] https://git-scm.com/

2.6 GESTIÓN DE RIESGOS

El desarrollo de un proyecto conlleva una serie de riesgos y deben ser identificados con anterioridad para así evitar las causas que llevan a ellos o, en caso de que ocurran, poder actuar con rapidez.

2.6.1 PROBLEMAS DE ÍNDOLE PERSONAL

- Probabilidad: baja.
- Impacto: medio.
- Consecuencias: retraso en las fechas establecidas para cada tarea.
- Prevención: no se puede determinar, ya que esta depende del tipo de problema.
- Plan de contingencia: procurar dedicar más horas a realizar las tareas retrasadas.

2.6.2 PROBLEMAS CON EL HARDWARE DE DESARROLLO

- Probabilidad: baja
- Impacto: medio.
- Consecuencias: retraso en las fechas establecidas para cada tarea.
- Prevención: realizar un mantenimiento regular de los componentes del hardware usado. También verificar que las herramientas Dropbox y Git funcionan correctamente y se realiza el almacenamiento en la nube, ya que son nuestra copia de respaldo. Considerar, dentro de lo posible, disponer de un ordenador de sustitución para que, en caso de necesidad, volver al trabajo con rapidez.
- Plan de contingencia: intentar dedicar más horas a realizar las tareas retrasadas y recuperar las últimas copias del trabajo realizado para poder avanzar desde este punto.

2.6.3 PROBLEMAS DE SOFTWARE

- Probabilidad: media.
- Impacto: medio.
- Consecuencias: retraso en las fechas establecidas para cada tarea.
- Prevención: realizar un mantenimiento regular del sistema y procurar instalar el mínimo software nuevo posible.
- Plan de contingencia: intentar dedicar más horas a realizar las tareas retrasadas.

2.6.4 CAMBIOS EN LAS API USADAS

- Probabilidad: baja.
- Impacto: medio.
- Consecuencia: retraso en las tareas siguientes e incluso en la fecha de entrega del proyecto.
- Prevención: estar al día de las posibles actualizaciones que se vayan a realizar.
- Plan de contingencia: revisar a qué afectan las actualizaciones y, si es necesario, retocar el código para adaptarlo a las novedades introducidas.

2.6.5 PROBLEMAS A LA HORA DE INSTALAR EL SOFTWARE NECESARIO

- Probabilidad: media.
- Impacto: medio.
- Consecuencia: retraso en las tareas e incluso en la fecha de entrega del proyecto.
- Prevención: buscar y leer los tutoriales y manuales necesarios para la correcta instalación.
- Plan de contingencia: revisar la instalación para averiguar el posible fallo.

2.6.6 PLANIFICACIÓN INCORRECTA

- Probabilidad: media.
- Impacto: medio.
- Consecuencia: retraso en la fecha de entrega del proyecto.
- Prevención: planificar las tareas con amplios plazos, dejan margen de tiempo para cada tarea.
- Plan de contingencia: modificar la planificación original aumentando las horas de trabajo hasta el momento de la entrega.

2.7 EVALUACIÓN ECONÓMICA

A la hora de realizar un proyecto, es importante ver el impacto económico que este supondrá y estimar lo que va a costar su desarrollo, así como la amortización del mismo. Al tratarse de un proyecto desarrollado para el grupo de investigación CRIM-AP de la UPV/EHU, como se ha mencionado en el apartado de objetivos, se tratará de una aplicación gratuita por la cual no se percibirá ningún aporte económico.

En cuanto a personal, hay que tener en cuenta que un programador cobra **30€/hora** y sabiendo que el proyecto tiene una duración estimada de **447 horas**, el **coste total del personal** asciende a **13410€**.

En el apartado de hardware tendremos varios apartados:

- El ordenador portátil está **valorado** en **775€**, con una vida media de 5 años y un uso de un 33% a lo largo del proyecto. Por tanto:
  - Amortización anual: 775/5 = 155€
  - Gastos del portátil: ((160 * 13 semanas) * 0.33) / 52 semanas = **13,2€**
- El móvil está valorado en 150€ con una vida media de 2 años y un uso del 20% a lo largo del proyecto.
  - Amortización anual: 150/2 = 75€
  - Gastos del móvil: ((75 * 13 semanas) * 0.2) / 52 semanas = **3,75€**

Por tanto, los **gastos totales de hardware** ascienden a **16,95€**

En el apartado de software no hay gastos, ya que el software de pago como por ejemplo Microsoft Office Word, va incluido en el precio del portátil y el resto de software usado es gratuito o se ha hecho uso de su modalidad gratuita.

Llegados a este punto, se puede decir que el **coste total del proyecto** resulta ser de **13426,95€**

Como se ha mencionado al principio de este apartado, está aplicación ha sido solicitada por un equipo de investigadores del ámbito de las ciencias sociales de la UPV-EHU, por lo que no se busca un beneficio económico. El fin es profundizar en el conocimiento de la movilidad a pie en la ciudad así como en los aspectos que determinan la toma de decisiones de movilidad, particularmente en relación con el diseño *walkable* -o caminable, aquellos espacios que invitan a ser transitados a pie, ofreciendo a los ciudadanos la oportunidad para incrementar su actividad física, mejorando su salud (Pucher et al., 2010), reducir su impacto ambiental y relacionarse. Las ciudades pensadas para el peatón reducen el impacto en el medio ambiente, son más saludables y seguras y ofrecen más oportunidades para la interacción y la cohesión social; sin embargo, un diseño que favorezca el tránsito a pie sólo es exitoso si impacta en el proceso de toma de decisiones del ciudadano, que opta por esta estrategia de movilidad y no una alternativa. Es por ello que la aplicación recoge, tanto conducta efectiva (rutas por las que se ha transitado), como percepciones del espacio público utilizado e información sobre la toma de decisiones de los usuarios.

3 ANTECEDENTES

A la hora de llevar a cabo un proyecto, surge la duda de si ya se puede encontrar algo similar en el mercado actual o no. En este caso, y como se ha mencionado anteriormente, se trata de una idea propuesta por el grupo de investigación CRIM-AP de la UPV/EHU, por lo que se puede suponer que no hay nada disponible por el momento, ya que, en caso afirmativo, no hubiera surgido la necesidad de realizar este proyecto. Aún así, se han investigado las aplicaciones existentes por si se hubiera pasado algo por alto.

Existen numerosas aplicaciones que realizan funciones similares. Como ejemplos más significativos por ser de los más descargados, podemos encontrar Runtastic[13] y Endomondo[14]. A fecha de realización de este documento (Enero 2016), las podemos encontrar en el top 10 de aplicaciones más descargadas[15] en el apartado de salud y bienestar.

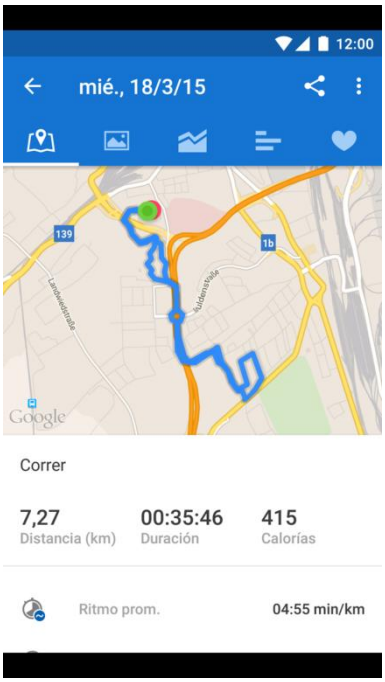

Ilustración 6 - Runtastic (Runtastic.com, 2012)

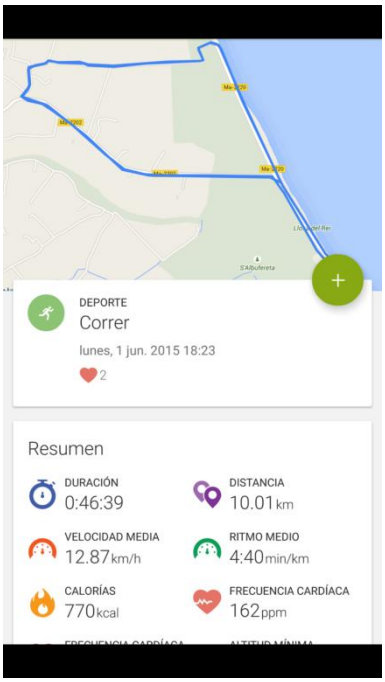

Ilustración 7 - Endomondo (Endomondo.com, 2012)

Estas dos aplicaciones se asemejan a un entrenador personal virtual. Mediante el GPS registran aquellas actividades realizadas, como pueden ser caminatas, running, distintas modalidades de ciclismo y otros tantos deportes. El usuario es el encargado de activar y desactivar el seguimiento por parte del programa. Tras acabar la actividad, este podrá revisarla y analizar distintos parámetros, como la velocidad media por ejemplo.

[13] https://www.runtastic.com/es
[14] https://www.endomondo.com/
[15] https://play.google.com/store/apps/category/HEALTH_AND_FITNESS/collection/topselling_free

Para el caso concreto del grupo de investigación CRIM-AP, estas aplicaciones no son funcionales debido a los siguientes inconvenientes. En primer lugar está la interacción por parte del usuario. Dejar en manos del usuario la responsabilidad de grabar toda ruta realizada podría suponer una pérdida importante de datos que podrían ser necesarios para el estudio. Por eso se decidió que esto se realizaría de manera automática. La aplicación a desarrollar en este proyecto estará continuamente recogiendo datos del usuario, como su localización GPS y su velocidad. Con esto y en función de los criterios establecidos por los investigadores para definir qué les interesa considerar como ruta (velocidad, distancia, etc.), el programa almacena los datos o los desecha.

Otro problema es el acceso a las rutas realizadas por todos los usuarios. El acceso sería exclusivo para el equipo de investigadores ya que necesitan acceder a ellos para realizar los análisis que consideren oportunos. Estas aplicaciones, al tratarse de aplicaciones comerciales, no permiten el acceso a las bases de datos donde se almacenan las rutas de todos los usuarios. Únicamente permiten acceder al usuario de la aplicación a las rutas realizadas por él mismo. Al crear este proyecto, se creará una base de datos donde almacenar todas las rutas que los usuarios hagan y posteriormente poder descargarlas para su análisis junto con la valoración realizada.

También se encontró el problema de las valoraciones de las rutas. No es posible modificar estas aplicaciones para incluir una encuesta en la cual el usuario valore la ruta que ha realizado. Como se ha comentado anteriormente, esta es una de las funcionalidades principales que tendría el proyecto, por lo que este es otro de los motivos que hace necesario su desarrollo.

Con todo esto, se puede decir que no se ha encontrado una aplicación que realice las funciones necesarias para el caso particular de este estudio.

4 CAPTURA DE REQUISITOS

La captura de requisitos es el primer paso para poder realizar una buena aplicación. Llevar a cabo este análisis se puede considerar como uno de los pasos más importantes en el proceso de desarrollo de un proyecto, ya que permitirá que este cubra las necesidades del cliente.

4.1 REQUISITOS PREVIOS

En las reuniones mantenidas con el grupo de investigadores, se determinó que la aplicación tuviera las siguientes funciones:

- Un cuestionario inicial que permitirá conocer datos del usuario, como por ejemplo su edad, género o cuándo querrá valorar las rutas realizadas.
- Que el programa, de manera autónoma, decidirá qué movimientos realizados por el usuario son una ruta a pie y cuáles no. Además, se registrarán aquellas rutas que cumplan los requisitos predefinidos por los miembros del grupo CRIM-AP.
- En el horario determinado por el usuario, se solicitará valorar una de las rutas realizadas durante las últimas 24 horas. Esta ruta será elegida según el algoritmo definido por los investigadores.
- Almacenamiento local y remoto de todos los datos recogidos.

4.2 CASOS DE USO

Tras realizar la captura de requisitos funcionales de la aplicación a desarrollar, el siguiente paso sería detectar los casos de uso que guardan relación directa con esos requerimientos.

En esta sección se van a exponer los casos de uso extraídos de las especificaciones dadas, así como la jerarquía de actores. En el Anexo I se encuentran los casos de uso extendidos. En ellos se detallan en profundidad los casos de uso, explicando la funcionalidad de cada uno de ellos.

4.2.1 JERARQUÍA DE ACTORES

La jerarquía de actores, para esta aplicación, constará de dos únicos actores. Estos serían:

- **Usuario**: es toda aquella persona que instale el programa es su dispositivo móvil.
- **Sistema**: es el sistema operativo del dispositivo móvil. Será el encargado de ejecutar acciones de manera automática, como por ejemplo solicitar la valoración de una ruta o subir los datos al servidor.

4.2.2 *MODELO DE CASOS DE USO*

En el gráfico siguiente (Ilustración 8 - Modelo de Casos de Uso) se encuentra representado el modelo de casos de uso asociado a la aplicación, en el que se pueden apreciar los diferentes casos de uso y las relaciones entre ellos.

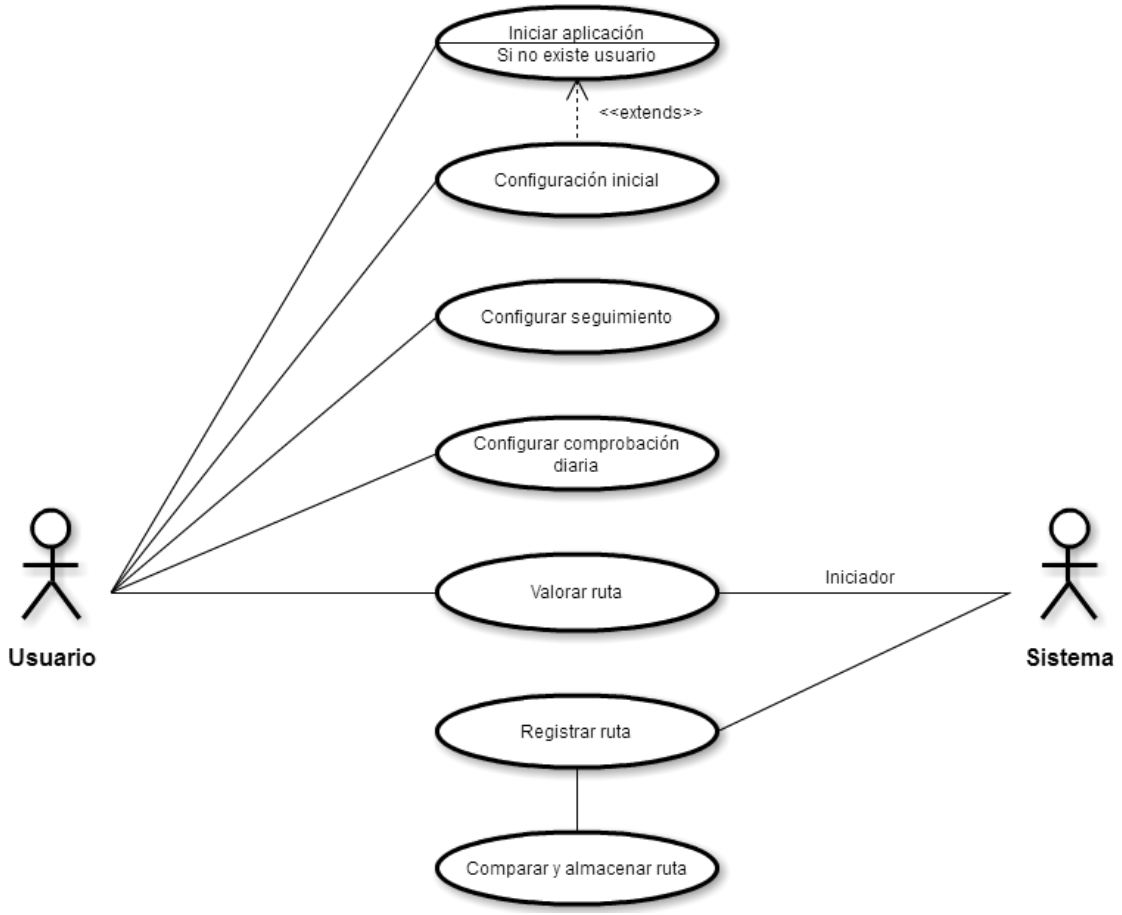

Ilustración 8 - Modelo de Casos de Uso

- **Iniciar aplicación:** cuando se inicia la aplicación se realizarán distintas comprobaciones, como la versión activa de la configuración de la aplicación o si existe un usuario registrado ya que habría que realizar la configuración inicial. También se cargarán las datos necesarios para el funcionamiento de la aplicación
- **Configuración inicial**: en la primera utilización de la aplicación, se le mostrará al usuario un breve cuestionario, donde conocer datos como su edad o su género. Estos datos se almacenarán para el posterior acceso por parte de los miembros del grupo CRIM-AP.
- **Configurar seguimiento**: el usuario podrá detener o poner en funcionamiento el seguimiento por parte del sistema para, por ejemplo, ahorrar batería.
- **Configurar comprobación diaria:** permitirá al usuario modificar la hora a la que el sistema comprobará si ha realizado o no alguna ruta

- **Valorar ruta:** cuando el usuario haya determinado en la configuración inicial, el sistema elegirá una ruta hecha por el usuario en las 24 horas anteriores basándose en el algoritmo dado por los investigadores y pedirá al usuario que la valore. Cuando no haya rutas nuevas, se preguntará por una ruta ya realizada siempre que haya pasado al menos un mes desde su última valoración.
- **Registrar ruta**: el sistema registrará las rutas a pie del usuario basándose en los parámetros definidos por los investigadores.
- **Comparar y almacenar ruta:** subcaso de uso donde el sistema comparará la ruta registrada con las existentes en la base de datos para determinar si es una ruta nueva o si ya ha sido realizada y la almacena en la base de datos.

4.3 MODELO DE DOMINIO

En el modelo de dominio aparecen reflejados todos los datos que van a ser almacenados, así como su relación entre ellos. En este caso, existe un modelo de dominio para los datos almacenados de manera local y otro para los datos almacenados remotamente.

En el modelo de dominio del dispositivo (Ilustración 9 - Modelo de Dominio del dispositivo) podemos encontrar las siguientes entidades con las correspondientes relaciones.

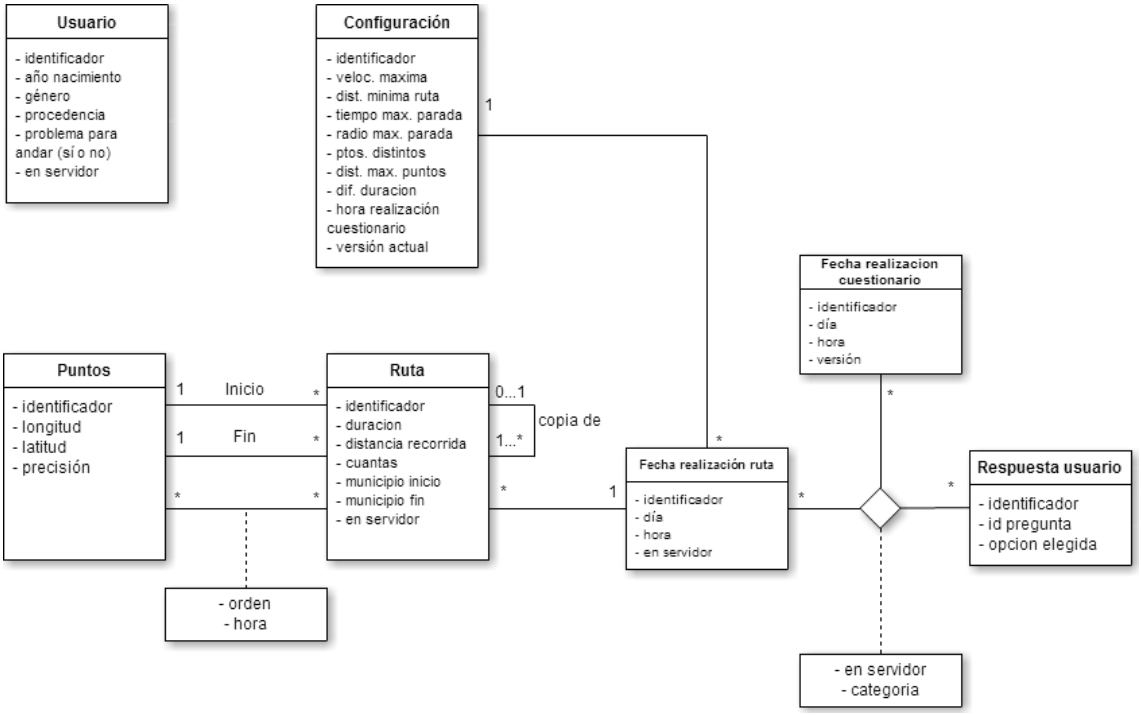

Ilustración 9 - Modelo de Dominio del dispositivo

- **Usuario**: en esta entidad se almacenan los datos que el usuario ha proporcionado en el inicio de la aplicación.
- **Configuración:** los valores que pueden variar a lo largo del estudio definidos por parte de los investigadores estarán almacenados en esta entidad.
- **Ruta**: aquí se almacenarán todos los datos pertenecientes a una ruta, como puede ser la duración de la misma y la distancia recorrida. Para facilitar el posterior análisis de los datos, el grupo CRIM-AP solicitó que en esta entidad se almacenasen también los municipios de inicio y fin, ya que así podría conocerse el origen y el fin de la ruta sin necesidad de pasar las coordenadas a un Sistema de Información Geográfico (SIG).
- **Puntos**: cada ruta está formada por una serie de puntos, así como de un punto de inicio y otro de fin. Es por esto que esta entidad está relacionada directamente con la entidad ruta. Aquí se guardarán los datos que definen un punto de la ruta, por ejemplo, la longitud o la latitud.
- **Fecha realización ruta**: aquí se guardará qué día y a qué hora se realiza una ruta en concreto.
- **Fecha realización cuestionario:** una ruta puede ser valorada más de una vez, por lo que se deberá almacenar cuándo ha sido valorada. Como los cuestionarios varían a lo largo del estudio, también se almacena con qué versión ha sido valorada.
- **Respuesta usuario**: aquí se guardarán las respuestas dadas por el usuario a una pregunta en concreto.

La relación entre **fecha realización ruta**, **fecha realización cuestionario** y **respuesta usuario** está presente debido a que es necesario saber la respuesta que se ha dado a una pregunta de un cuestionario para una ruta. Como una ruta se puede realizar varias veces, la percepción o la finalidad con la que ha sido realizada puede variar, por eso es necesario el atributo *categoría*.

Entre las entidades **punto** y **ruta**, además de las relaciones de inicio y fin necesarias para saber dónde comienza y finaliza una ruta, se encuentra una tercera relación con los atributos *orden* y *hora*. Éstos son necesarios para conocer la posición de cada punto dentro de cada ruta y para determinar la similitud entre dos rutas.

La entidad **ruta** cuenta con una relación hacia sí misma, y esto permite saber si una ruta es copia de otra y, en caso afirmativo, saber de quién es copia. Este dato será necesario a la hora de probar el algoritmo de similitud entre rutas, ya que permitirá saber cómo funciona y en qué medida hay que ajustar los parámetros como el porcentaje de puntos distintos para determinar que una ruta es distinta a otra o la distancia mínima que debe haber entre dos puntos para decir que son dos puntos distintos.

La relación entre **configuración** y **fecha realización ruta** se debe a que es necesario conocer con qué versión de configuración se ha realizado una ruta ya que puede variar a lo largo del tiempo.

En el modelo de dominio del servidor (Ilustración 10 - Modelo de Dominio del servidor) encontramos nuevas entidades, además de varias similitudes con el anterior modelo de dominio.

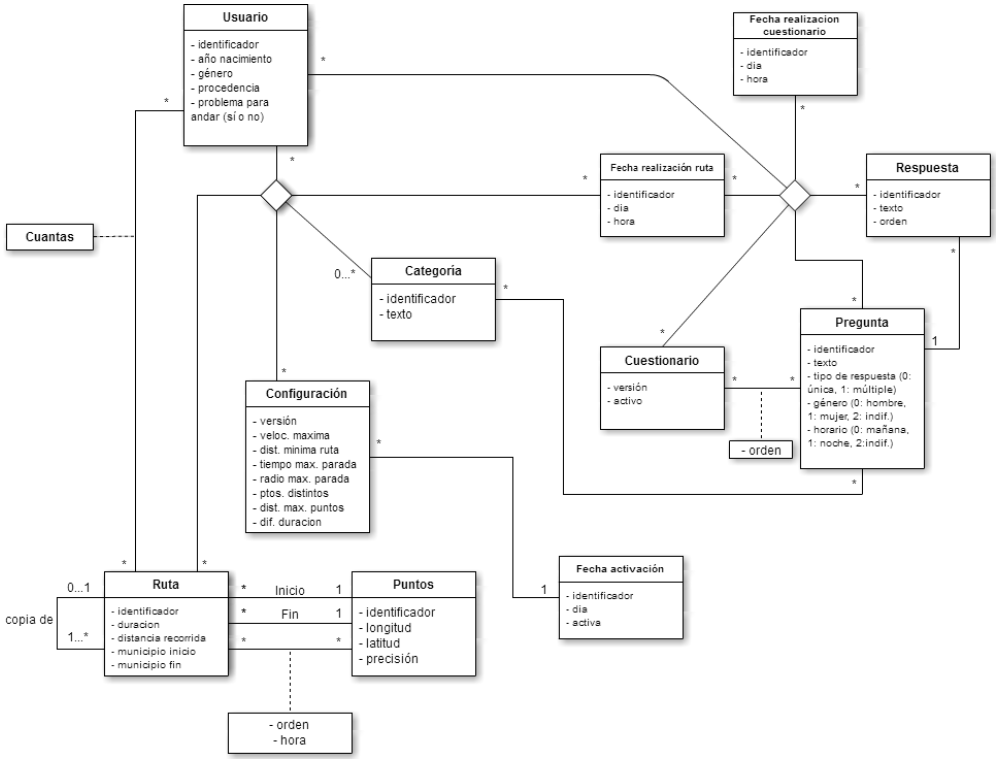

Ilustración 10 - Modelo de Dominio del servidor

- **Usuario**: en este caso, se almacenarán los datos de todos los usuarios de la aplicación.
- El atributo *Cuantas* de la relación entre **usuario** y **ruta** sirve saber cuántas veces ha realizado un usuario una ruta.
- **Fecha activación**: esta entidad guardará cuando se ha activado una configuración en concreto.
- **Cuestionario**: la entidad almacenará las versiones del cuestionario
- **Pregunta**: aquí se almacenarán los datos necesarios para formular las preguntas de los cuestionarios. Una misma pregunta puede pertenecer a varios cuestionarios a la vez.
- **Respuesta**: esta entidad contendrá el texto necesario para mostrar la respuesta a una pregunta en concreto.
- **Categoría**: esta entidad almacenará el tipo de ruta que realizará el usuario, la finalidad con la que ha sido hecha, como por ejemplo, hacer recados. La categoría solo será asignada si la ruta es elegida por el algoritmo de selección de rutas para realizar el cuestionario, por eso la relación entre **ruta** y **categoría** tiene cardinalidad 0…1. En la relación entre **categoría** y **pregunta** la cardinalidad es n…m, ya que una misma pregunta puede ser de una o más categorías.

La relación entre **usuario**, **ruta** y **fecha realización ruta** se debe a que es necesario saber qué usuario ha hecho qué ruta en qué momento. Un usuario puede hacer muchas rutas en distintos momentos. También están presentes las entidades **categoría** y **configuración**. Dependiendo el momento en el que una ruta sea realizada, la percepción de la misma puede cambiar, al igual que la configuración activa.

El atributo *orden* en la relación entre **cuestionario** y **pregunta** se debe a que las preguntas deben guardar un orden dentro del cuestionario porque es relevante para los investigadores.

La relación entre seis elementos aparece porque se necesita saber lo que ha contestado un usuario a una pregunta de un cuestionario en un momento concreto de una ruta hecha en un momento concreto.

5 ANÁLISIS Y DISEÑO

5.1 *ESTRUCTURA DE TRABAJO*

La elaboración del proyecto *Walkability Capturer* se podría dividir en dos partes. Por un lado está la parte local, que será la aplicación instalada en el dispositivo; y por otro está la parte del servidor que almacena la base de datos y los archivos PHP a los que accede remotamente la aplicación para poder hacer uso de la base de datos.

Como se ha comentado anteriormente (ver 2.5.4 Apache Cordova), se va a hacer uso del framework Apache Cordova. Por lo tanto, para su desarrollo se usarán los lenguajes HTML5, JavaScript y CSS3, así como del lenguaje PHP para los archivos alojados en el servidor con los que interaccionará la aplicación.

Por defecto, al crear un proyecto con Apache Cordova se obtiene el esquema de carpetas mostrado en Ilustración 11 - Estructura de la app.

Para dotar de funcionalidad a la aplicación, Cordova hace uso de los *plugins*. Éstos sirven de nexo entre la app y el dispositivo sobre el que corren, permitiendo acceder al hardware o a funciones más específicas del sistema. Si se quisiera que la aplicación, por ejemplo, tuviera geolocalización o permitiera la captura de fotos, sería necesario añadir al proyecto los plugins correspondientes.

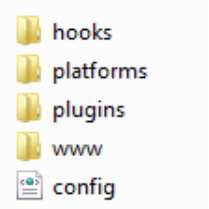

Ilustración 11 - Estructura de la app

Dentro de la estructura de la app se encuentra también la carpeta *hooks*. Los hooks son elementos que Cordova proporciona para extender su funcionalidad dependiendo de las necesidades del presente proyecto. Realmente se tratan de scripts que el desarrollador crea para automatizar tareas, como puede ser el caso de la carga de plugins a la hora de compilar un proyecto. Normalmente, cuando un desarrollador comparte su código, lo hace a través de un repositorio de código como Github[16], una plataforma de desarrollo colaborativo de software que utiliza el control de versiones Git (ver 2.5.8 Git) y donde el código se almacena de manera pública. Al tratarse de una aplicación multiplataforma, no es necesario compartir el código con los archivos necesarios para compilarla y se opta por compartir solo la carpeta *www*, que más adelante será explicada.

Por tanto, si la aplicación cuenta con un gran número de plugins, añadirlos de manera manual uno a uno puede resultar una tarea algo tediosa y pesada. Gracias a los hooks, esta tarea se puede simplificar. El desarrollador puede compartir su código con los hooks necesarios para que lo único que haya que realizar sea añadir la plataforma para la que queremos compilar la aplicación.

[16] https://github.com/

La siguiente carpeta es la de *platforms*. Como se ha mencionado anteriormente Apache Cordova es un framework multiplataforma, y para su desarrollo hace uso de HTML5, JavaScript y CSS3. Si los archivos desarrollados se ejecutasen directamente en un dispositivo, estos se mostrarían como una página web sin poder realizar ninguna función. Para que la aplicación pueda ser ejecutada como nativa, necesita ser "traducida" al lenguaje nativo del dispositivo. Esta carpeta almacena los archivos necesarios para realizar dicha traducción, cada uno en una carpeta nombrada con la plataforma a la que hace referencia.

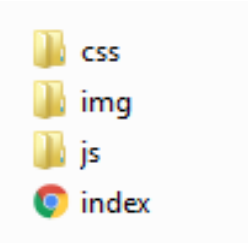

Ilustración 12 - Estructura de la carpeta www

Por último, en la carpeta *www* se alojarán todos los archivos desarrollados y los que darán forma a la aplicación una vez se ejecute. Como se puede ver en Ilustración 12 - Estructura de la carpeta www, en su interior se encuentra otro grupo de carpetas.

En *css* estarán todas las hojas de estilo utilizadas para dar forma y colocar al apartado visual del proyecto. La carpeta *img* contendrá aquellas imágenes que vayan a ser mostradas dentro de la aplicación. Otras imágenes, como pueden ser las utilizadas para el icono del proyecto o de las notificaciones, se almacenarán en el directorio correspondiente dentro de la carpeta *platform*. Por último, la carpeta *js* contendrá toda la funcionalidad de la aplicación, ya que todos los archivos JavaScript se encuentran en su interior. Con la intención de proporcionar una fácil comprensión, los archivos desarrollados estarán nombrados según su finalidad, es decir, el archivo que agrupe las funciones que interactúan con la base de datos se llamará *baseDatosLocal.js*, por ejemplo. En todo proyecto desarrollado en Apache Cordova, debe haber una página inicial. El archivo *index.html* sería esa página inicial, la que se mostrará en primer lugar al iniciar la aplicación.

Como se muestra en Ilustración 11 - Estructura de la app, también aparece un archivo *config.xml*. Aquí se definirán configuraciones globales del entorno. Entre las configuraciones, aquí se podrá decir qué archivo html se quiere definir como página inicial o, en el caso de que la aplicación lo necesite, bloquear la pantalla en una orientación concreta.

En el servidor los archivos PHP necesarios para que la aplicación acceda a la base de datos remota no se almacenarán bajo ninguna estructura de carpetas. Los archivos estarán desarrollados según las necesidades de la aplicación, es decir, al trabajar con objetos JSON cada archivo devolverá un objeto configurado según los datos solicitados a la base de datos.

5.2 DIAGRAMA RELACIONAL DE LA BASE DE DATOS

Como se ha mencionado en 4.3 Modelo de dominio, este proyecto cuenta con dos modelos de dominio distintos, uno para el dispositivo donde estará instalada la aplicación y otro para el servidor que almacenará los datos de cada uno de los usuarios para su posterior estudio por parte del equipo investigador.

En el caso del dispositivo, hay disponibles distintas herramientas ofrecidas por Apache Cordova para llevar a cabo el almacenamiento local[17]. Los sistemas de almacenamiento son: LocalStorage, WebSQL, IndexedDB y opciones basadas en plugins.

Excepto las opciones basadas en plugins, el resto de APIs son las mismas que los que usan los navegadores, como pueden ser Google Chrome u Opera. Esto se debe a que la aplicación será ejecutada como si de una página web se tratase y por tanto es mostrada a través de un pseudo-navegador, ya que el usuario lo percibe como una aplicación nativa.

Para este proyecto en particular, se ha optado por usar una base de datos relacional porque se almacenarán numerosos datos y deben guardar una relación entre ellos. Por tanto, la opción de LocalStorage será descartada, ya que se trata de un almacenamiento de tipo clave/valor, es decir cada valor añadido tendrá asociado una clave y para recuperarlo después se debe usar esa clave.

Lo mismo ocurre con IndexedDB. Se trata de un sistema de tablas indexado, con índices, por lo que no sería posible realizar una relación entre los distintos datos introducidos. También se puede dar el caso de que no se conozca el índice de los datos que se quieran recuperar y es por esto por lo que este sistema de tablas no es válido.

La última posibilidad de las APIs de almacenamiento ofrecidas por defecto por Apache Cordova es WebSQL. Se trata de una base de datos que usa un sistema de tablas relacional, necesario para este proyecto. Para su funcionamiento, hace uso del lenguaje Structured Query Language (SQL). Desde el 18 de octubre de 2010, esta especificación ha dejado de estar soportada por el W3C[18], aunque eso no quiere decir que no se pueda utilizar. Lo soportan los principales navegadores móviles y además, al usarlo con Apache Cordova, el propio framework se encarga de dar soporte en aquellos navegadores que no lo soportan de forma nativa. El inconveniente encontrado es que la cantidad total de almacenaje está limitada, por lo general, a 5MB[19]. Debido a esto, esta opción tampoco es válida para las necesidades actuales.

Tras revisar todas las opciones ofrecidas por defecto por Apache Cordova y desecharlas por las razones explicadas anteriormente, hay que recurrir a los plugins. Gracias a ellos, las funcionalidades disponibles se ven complementadas y ampliadas según las necesidades del desarrollador, como es en este caso. El plugin elegido es el **cordova-sqlite-storage**[20] recomendado por Apache Cordova. El funcionamiento general es muy similar al de WebSQL, ya que también usa el lenguaje SQL. La ventaja que tiene el uso de este plugin es que el espacio de almacenamiento no está limitado.

[17] https://cordova.apache.org/docs/en/latest/cordova/storage/storage.html
[18] https://www.w3.org/TR/webdatabase/
[19] https://cordova.apache.org/docs/en/latest/cordova/storage/storage.html#websql
[20] https://www.npmjs.com/package/cordova-sqlite-storage

Para la base de datos remota, se va a hacer uso de una base de datos alojada en el servidor de la universidad *galan.ehu.es*. Las conexiones a la base de datos se realizan a través de peticiones AJAX usando jQuery, que se explicará en el apartado de desarrollo. La base de datos es de tipo MySQL, ya que es un sistema de gestión de bases de datos relacional, multihilo y multiusuario.

MySQL permite elegir como motor de almacenamiento entre MyISAM o InnoDB. En este caso, se opta por elegir InnoDB ya que dota a MySQL de un motor de almacenamiento transaccional (conforma a ACID) con capacidades de commit (confirmación), rollback (cancelación) y recuperación de fallos. InnoDB realiza bloqueos a nivel de fila y también proporciona funciones de lectura consistente sin bloqueo en sentencias SELECT. Estas características incrementan el rendimiento y la capacidad de gestionar múltiples usuarios simultáneos. No se necesita un bloqueo escalado en InnoDB porque los bloqueos a nivel de fila ocupan muy poco espacio. InnoDB también soporta restricciones FOREIGN KEY. En consultas SQL, aún dentro de la misma consulta, pueden incluirse libremente tablas del tipo InnoDB con tablas de otros tipos (Arsys.es, 2012).

En base a los modelos de dominio expuestos anteriormente, para el caso del dispositivo se obtiene el siguiente diagrama relacional de la base de datos en producción:

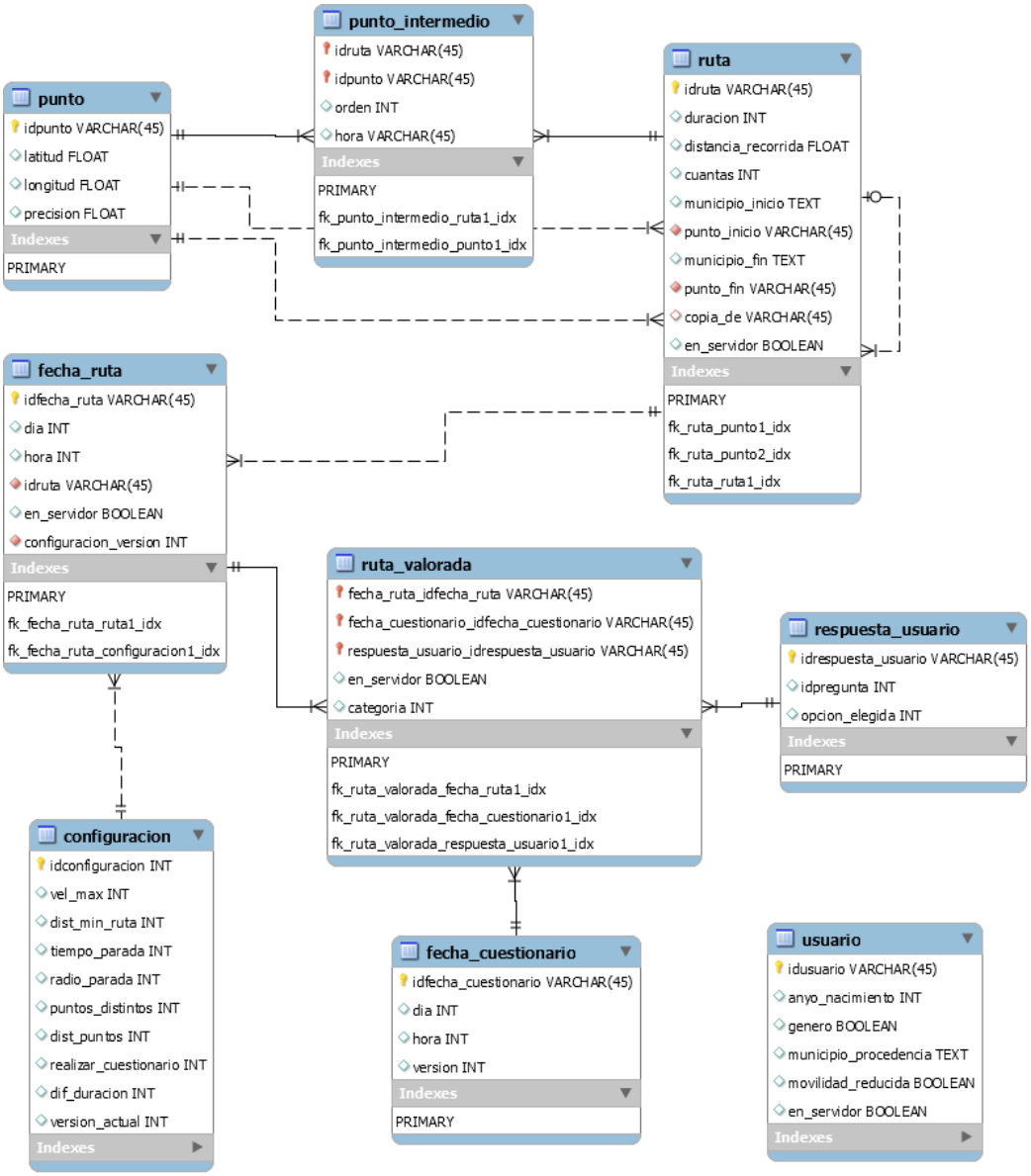

Ilustración 13 - Diagrama BD Relacional Dispositivo

En la Ilustración 13 - Diagrama BD Relacional Dispositivo, se pueden ver las diferentes tablas con sus atributos de los que está compuesta la base de datos, así como las relaciones que hay entre dichas tablas.

La tabla **usuario** almacena los datos que el usuario del dispositivo ha introducido al iniciar la aplicación. Estos datos serán también almacenados en el servidor externo para que el equipo investigador realice los estudios necesarios. Al usuario se le solicitará su año de nacimiento, su género, el municipio de procedencia y si sufre o no algún problema de movilidad. El sistema, al iniciarse, creará un identificador único para cada usuario que es utilizado como clave primaria de la tabla. Este identificador es necesario ya que en la base de datos del servidor habrá más de un usuario registrado y es necesario saber qué rutas y qué cuestionarios ha realizado cada uno. Se hará uso de otro atributo booleano para saber si los datos han sido ya mandados al servidor o no, ya que a la hora de subir los datos puede fallar la conexión.

La tabla **configuración** contendrá todos los valores que pueden ser modificados por parte del equipo investigador. Por un lado están los datos que determinan una ruta, como puede ser la velocidad máxima a la que puede moverse el usuario para considerar que se mueve a pie y la distancia mínima que tiene que recorrer para considerar que lo realizado es una ruta. Por otro lado están los que determinan cuándo se debe finalizar una ruta o una presunta ruta al pararse el usuario, que sería el tiempo máximo que puede estar parado y la distancia mínima que debe alejarse desde el punto de parada. Y por último están los datos que determinan cuándo dos rutas son distintas, que son el porcentaje de puntos distintos, la distancia máxima que debe existir entre dos puntos para determinar que son el mismo punto y el porcentaje de diferencia máximo en la duración entre dos rutas para determinar que son iguales. También se almacenará la hora que ha elegido el usuario para que la aplicación solicite realizar la valoración diaria y la versión de la configuración actual. Como clave primaria se crea un identificador, ya que el resto de datos será modificado según las necesidades del proyecto.

La tabla **ruta** contiene cada una de las rutas realizadas por el usuario. Por cada ruta realizada almacena el identificador, la duración, la distancia recorrida, cuántas veces se ha realizado, identificador del punto de inicio, municipio de inicio, identificador del punto de fin, municipio de fin y si está o no en el servidor. Esta tabla tiene como clave principal el identificador. Existe una relación consigo misma, ya que es necesario almacenar saber si una ruta es copia de otra almacenada anteriormente. Por lo tanto, en el atributo *copia_de* se almacenará el identificador de la ruta original en el caso de ser copia o null si se trata de la original. También está relacionada con la tabla **punto** y la tabla **fecha_ruta**. En el caso de la tabla **punto** existen varias relaciones. Por un lado están las relaciones de inicio y fin. Toda ruta tendrá un punto de inicio y un punto de fin, por lo que están relacionadas por los identificadores de los puntos en la tabla **ruta** como *punto_inicio* y *punto_fin* como claves foráneas. Para los puntos intermedios de la ruta, se crea una tabla intermedia llamada **punto_intermedio**, en la que se almacena el identificador del punto, el de la ruta, la posición del punto en la ruta y la hora a la que fue recogido. Los dos identificadores forman la clave primaria además de ser claves foráneas. En el caso de la **fecha_ruta**, ésta almacenará el identificador de la tabla **ruta** como clave foránea.

La tabla **punto** contiene cada uno de los puntos que forman una ruta. Por cada punto, se almacena el identificador, la longitud, la latitud y la precisión con la que ha sido registrado. Esta tabla tendrá como clave primaria el identificador. Como se ha mencionado anteriormente, esta tabla se relaciona con la tabla **ruta** mediante el identificador, siendo clave foránea en la tabla **ruta** como *punto_inicio* y *punto_fin*.

La tabla **fecha_ruta** contiene cuándo se ha realizado una ruta, ya que la misma ruta puede ser realizada más de una vez. Por cada ruta realizada, se almacena el identificador, la fecha y la hora a la que ha sido realizada. Esta tabla tiene como clave primaria el identificador y como clave foránea *idruta* que es el identificador de la ruta y la versión de la configuración actual porque se necesita conocer con que configuración se ha realizado cada ruta.

La tabla **fecha_cuestionario** contiene cuándo se ha realizado el cuestionario de una ruta, ya que una ruta puede ser valorada más de una vez. Por cada registro se almacenará el identificador, el día y la hora a la que ha sido realizado el cuestionario, además de la versión del cuestionario usada. El identificador es la clave primaria de la tabla

Por último, la tabla **respuesta_usuario** almacena la respuesta dada por el usuario a una pregunta. Por cada respuesta se almacena el identificador, el identificador de la pregunta a la que se ha contestado y el identificador de la respuesta elegida. Al igual que en otros casos, el identificador es la clave primaria,

Estas tres últimas tablas están relacionadas entre sí mediante la tabla **ruta_valorada**, ya que es necesario saber las respuestas que se han dado a un cuestionario de una ruta realizada en un momento concreto. Por lo tanto por cada registro se almacenará el identificador de cada una de las otras tres tablas, la categoría o finalidad de la ruta porque hasta que la ruta no es valorada no se conoce su categoría y un atributo para saber si los datos se encuentran o no ya almacenados en el servidor. Los tres identificadores forman la clave primaria de la tabla además de ser claves foráneas.

Como se ha dicho anteriormente, todo lo explicado ha sido para el caso del almacenamiento en el dispositivo. En la siguiente imagen (Ilustración 14 - Diagrama BD Relacional Servidor), podemos ver el diagrama relacional de la base de datos en el servidor, que es donde se recogen los datos de los distintos usuarios de la aplicación.

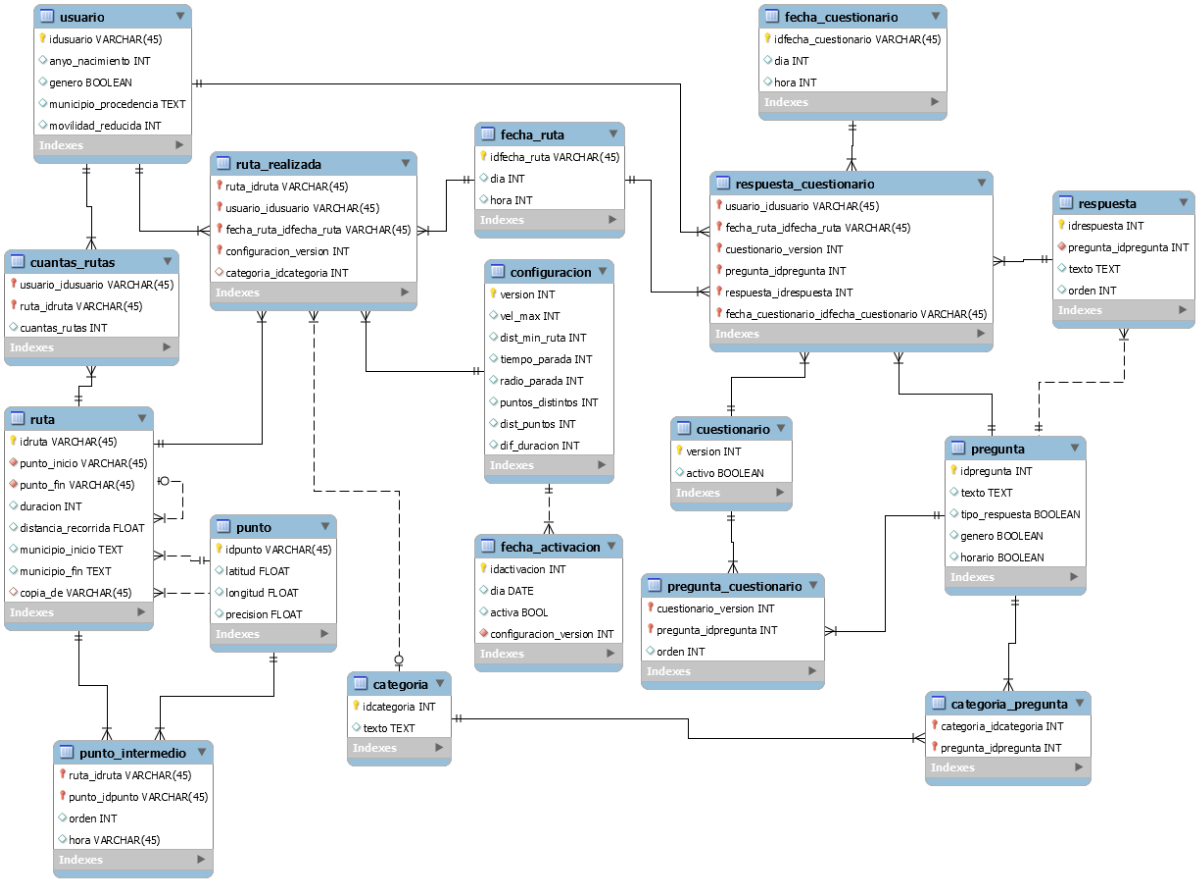

Ilustración 14 - Diagrama BD Relacional Servidor

La tabla **usuario** contiene los datos de los usuarios del sistema. Por cada usuario, almacena el identificador, el año de nacimiento, el género, el municipio de procedencia y si sufre o no problemas de movilidad. Al igual que en el dispositivo, la clave principal es el identificador. Esta tabla está relacionada por tres partes. Por una parte está directamente relacionada con la tabla **ruta**, aunque la relación tiene el atributo *cuantas* para saber cuántas veces ha realizado una misma ruta un usuario. Esto dará lugar a la tabla **cuantas_rutas** en la que se almacenará los identificadores de usuario y de ruta y el atributo *cuantas_rutas*. Los identificadores forman la clave principal, además son claves foráneas.

Por otra parte, está la relación quíntuple entre las tablas **usuario**, **ruta**, **fecha_ruta**, **categoría** y **configuración**. Gracias a esta relación se puede saber qué usuario ha hecho qué ruta en qué momento con qué configuración y con qué finalidad. Para ello, se ha creado la tabla **ruta_realizada**, que por cada registro almacena los identificadores de **usuario**, **ruta**, **fecha_ruta**, **categoría** y **configuración**. Estas forman la clave primaria y son claves foráneas, a excepción del identificador de la categoría que solo es clave foránea. Como la finalidad con la que se realiza una ruta se conoce únicamente a la hora de valorarla, en la tabla **ruta_realizada** el atributo *categoría_idcategoria* tendrá el valor null en el caso de que la ruta no haya sido valorada y el identificador correspondiente a la categoría seleccionada por el usuario en caso contrario. El atributo *categoría_idcategoria* también es necesario porque una ruta puede ser realizada con distintas finalidades.

En último lugar, podemos encontrar la relación a seis que se explicará al final de este capítulo.

La tabla **ruta**, al igual que en el dispositivo, contiene todas las rutas realizadas por los usuarios. Por cada registro almacena un identificador, la duración, la distancia recorrida, el identificador del punto de inicio, el municipio de inicio, el identificador del punto de fin y el municipio de fin. Como clave principal tiene el identificador. Al igual que en el dispositivo, la tabla **ruta** tiene una relación consigo misma para conocer la relación entre una ruta original y su copia, que almacenará el identificador de la original cuando la haya en el atributo *copia_de*. Toda ruta tiene un inicio y un fin, por eso la tabla está relacionada con la tabla **punto** y tiene como clave foránea *punto_inicio* y *punto_fin*, que serán los identificadores de los puntos de inicio y fin de la ruta. Además, la tabla **punto_intermedio** almacena el identificador del punto, el de la ruta, la posición del punto en la ruta y la hora a la que fue recogido.

La tabla **configuración**, al igual que en el caso del dispositivo, contendrá todos los valores que pueden ser modificados por el equipo investigador según las necesidades que vayan surgiendo en el desarrollo del estudio. La diferencia que existe con el caso del dispositivo es que aquí no se almacenará la hora a la que se debe realizar el cuestionario, ya que depende de cada usuario. Esta tabla está relacionada con la tabla **fecha_activación**, ya que es necesario registrar cuando se activa cada configuración. Por cada registro se almacena un identificador, el día y si está o no activa la versión. En toda la tabla solo deberá haber una tupla con el atributo *activo* con valor true, ya que solo puede haber una configuración activa. El identificador será la clave principal y como clave foránea, la versión de la configuración.

La tabla **cuestionario** contiene la versión actual del cuestionario que debe lanzar la aplicación al usuario para que valore una ruta. Por cada registro almacena la versión y si está activa o no. Al igual que en la tabla **fecha_activación**, solo debe haber una tupla activa. La versión es la clave principal.

La tabla **pregunta** contiene todos los datos necesarios para realizar una pregunta en el cuestionario. Por cada registro, almacena un identificador, el texto de la pregunta, el tipo de respuesta que tiene la pregunta que puede ser de respuesta única o múltiple. También se almacenará el género del usuario o la franja horaria en la que se ha realizado la ruta. Por defecto, se ha determinado que horario de mañana es desde las 07:00h hasta las 22:59h y el horario de noche desde las 23:00h hasta las 06:59h. Estos dos atributos, junto a la categoría, se tendrán en cuenta a la hora de elegir que preguntas se mostrarán al usuario.

La tabla **pregunta** tiene varias relaciones. Por una parte, está relacionada con la tabla cuestionario con una cardinalidad n:m, por lo que se genera una tabla intermedia llamada **pregunta_cuestionario** en la que, por cada registro, se almacena los identificadores de las dos tablas, que forman la clave primaria y son claves foráneas, y el orden en el que deben ser mostradas. Por otro lado, se relaciona con la tabla **respuesta**. Esta tabla, por cada registro, almacena un identificador, el texto de la respuesta y el orden en el de deberá ser mostrada. El identificador será la clave principal y el identificador de la pregunta será clave foránea.

También estará relacionada con la tabla **categoría** ya que, como se ha mencionado, se tendrá en cuenta a la hora de mostrar el cuestionario. Como una pregunta puede pertenecer a más de una categoría, se crea la tabla **categoría_pregunta** que almacena los identificadores de las dos tablas relacionadas y además son claves foráneas.

En la tabla **categoría** se almacenarán las categorías o finalidades que tiene una ruta para el usuario. Cada registro contiene un identificador y el texto que define la categoría. Inicialmente, las categorías serán:

1. Desplazamiento a y desde el trabajo o centro de estudios.
2. Desplazamientos durante la jornada laboral, como parte del trabajo.
3. Tareas familiares (por ejemplo acompañar a familiares al colegio, al centro de salud…).
4. Ocio, tiempo libre.
5. Compras y recados.

Cuando una pregunta sea válida para cualquier categoría, se relacionará con todas las categorías. De esta manera a la hora de seleccionar las preguntas será suficiente indicar la categoría elegida.

Como ocurre en el dispositivo, es necesario saber cuándo un usuario realiza una ruta y un cuestionario. Es por ello que se crean las tablas **fecha_ruta** y **fecha_cuestionario** y almacenan los mismos datos que en el dispositivo.

Como se explica en el modelo de dominio del servidor, para almacenar las respuestas del cuestionario es necesario saber:

- Quién ha respondido (**usuario**)
- Qué ha respondido (**respuesta**)
- A qué lo ha respondido (**pregunta**)
- En qué versión del cuestionario (**cuestionario**)
- Cuándo lo ha respondido (**fecha_cuestionario**)
- Sobre qué lo ha respondido (**fecha_ruta**, que junto con **usuario** permite saber sobre qué ruta estaba respondiendo)

Es por esto que se crea la tabla **respuesta_cuestionario** que, por cada registro, almacena los identificadores de las distintas tablas relacionadas que forman la clave principal y además son claves foráneas.

6 DESARROLLO

Este apartado se centra en describir de forma accesible, cómo se ha hecho el trabajo de fin de grado y qué herramientas se han utilizado para su desarrollo, justificando las decisiones que se han tomado.

6.1 ACLARACIONES PREVIAS

Antes de empezar con la explicación técnica del desarrollo realizado, es necesario explicar unos detalles.

En primer lugar, ¿qué es una ruta? O mejor dicho, ¿qué entienden por ruta los investigadores? Será considerado una ruta el desplazamiento a pie durante, por lo menos, una distancia mínima X por parte del usuario sin sobrepasar una velocidad máxima Y, ya que de ser así indicaría que el desplazamiento puede estar siendo realizado corriendo, en bici, en coche, etc. Esta distancia, al igual que la velocidad máxima, serán configurables a lo largo del estudio. También hay que tener en cuenta cuándo una ruta deja de cumplir las condiciones para ser ruta. Cuando el usuario se detenga por un tiempo superior al máximo estipulado o que durante ese tiempo no se mueva más de un cierto número de metros medidos desde el punto de parada hará que la ruta se dé por finalizada. Estos valores también serán variables a lo largo del estudio.

Para que la aplicación, usando los valores proporcionados por el grupo investigador, sepa determinar cuándo un desplazamiento a pie debe ser considerado una ruta, se diseñó un diagrama de transición de estados.

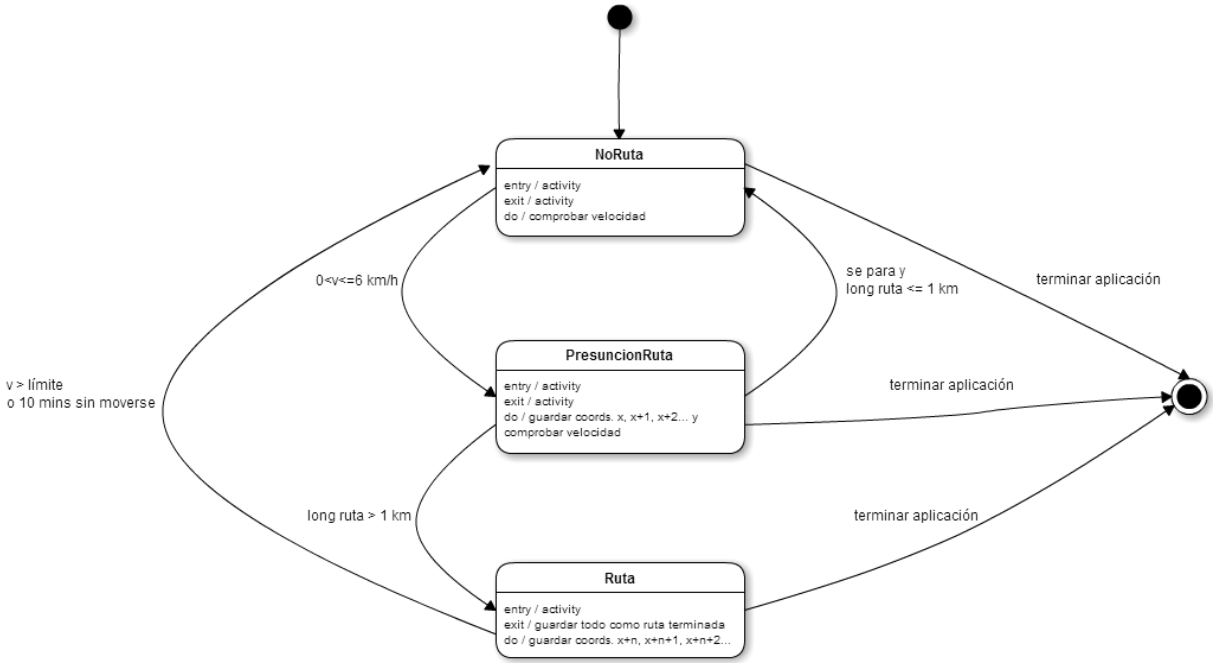

Ilustración 15 – Diagrama de transición de estados

En el estado **NoRuta** se vigila el cambio de velocidad. Si la velocidad obtenida se encuentra entre los valores permitidos, el punto se almacenará como el origen y se cambia al estado **PresuntaRuta**.

En este nuevo estado, además de vigilar la velocidad, se tiene en cuenta la distancia recorrida. En el caso de que la velocidad máxima permitida sea superada en dos ocasiones consecutivas y no se haya desplazado la distancia mínima necesaria para que el desplazamiento sea definido como una ruta, se volverá al estado anterior desechando los puntos almacenados hasta el momento. La condición de que la velocidad sea superada en dos ocasiones consecutivas para retroceder al primer estado se debe a que el usuario la puede sobrepasar al cruzar un semáforo, por ejemplo. De este modo, si esa velocidad excesiva solo se da una vez, la ruta continuará.

En el caso de que el usuario se detenga, existen dos condiciones. La primera es que esté detenido durante un tiempo máximo. Si durante ese tiempo, el usuario se mueve, también se tendría en cuenta que esos movimientos sean en un radio superior al permitido por los investigadores porque, si no, no serían relevantes para el estudio. Por ejemplo, el usuario puede detenerse en un semáforo para esperar a poder pasar y luego continuar su ruta, por lo que el seguimiento debe continuar. Por el contrario, si se detiene y no se aleja del punto de parada durante el tiempo máximo determinado, la ruta debe considerarse como finalizada.

En el estado **Ruta** se debe vigilar la velocidad como en el estado de **PresuntaRuta**, con la diferencia de que al cumplirse las condiciones para volver al estado de **NoRuta** la ruta realizada hasta el momento debe ser almacenada.

También cabe destacar que existe una condición añadida por el desarrollador a la hora de dar por finalizada una ruta. A la hora de probar la geolocalización se percató de que cuando se accedía a un edificio, por ejemplo, el plugin realizaba una petición de una localización nueva sin éxito. Si esto no se tratase, cuando el usuario saliera al exterior la ruta seguiría como si esa parada no hubiera existido. Para evitarlo, se añadió la condición de que, si tras 15 minutos no se recibe un punto nuevo, la ruta debe considerarse finalizada.

A la hora de almacenar una ruta, hay que tener en cuenta que ésta puede haber sido realizada anteriormente, por lo que se debe comprobar esa posible similitud. De esta tarea también se encarga la aplicación y por ello se ha diseñado el siguiente algoritmo de comparación:

```
Se seleccionan todas las rutas realizadas originales
      Por cada ruta
            Se comparan los puntos de inicio y fin con los de la ruta recién
            recogida
            Si coinciden
                  Se comparan las duraciones de las rutas
                  Si la diferencia es aceptable
                        Se comparan las rutas punto a punto
                        Si la cantidad de puntos diferentes es aceptable
                              Posible ruta igual
                        Si no
                              Ruta distinta
                  Si no
                        Ruta distinta
            Si no coinciden
                  Ruta distinta
Si hay posibles rutas
      Si hay una
            Se añade la nueva ruta indicando cuál es su original
      Si hay más
            Se elige la que más coincidencias tenga
Si no
      Se almacena como original
```

Como se puede ver en el algoritmo, se habla de comparación de puntos y de duraciones. En el caso de los puntos, para que dos puntos sean considerados iguales deben estar separados una distancia máxima de 15 metros. En el caso de la duración, la diferencia entre ellas no debe ser superior al 40%. Y, por último, para que dos rutas sean consideradas iguales no pueden tener más del 25% de los puntos distintos. Estos datos son una estimación inicial, ya que podrán variar en función de las decisiones que tomen los investigadores a la vista del comportamiento del algoritmo.

6.2 EL LENGUAJE DE LA APLICACIÓN

Cuando se propuso el proyecto, se ofrecieron varias opciones de desarrollo. En primer lugar estaba la opción de desarrollar la aplicación para la plataforma Android[21] exclusivamente. En segundo lugar estaba el entorno Xamarin[22]. Utilizar este entorno permite que, programando bajo lenguaje C#, se consiga una aplicación nativa multiplataforma. Esto significa que el mismo proyecto funcionaría bajo el sistema Android y el sistema iOS, por ejemplo, ahorrando tiempo de desarrollo ya que no sería necesario realizar dos proyectos distintos que realizasen las mismas funciones. Y en último lugar estaba el entorno Apache Cordova. Éste, al igual que Xamarin, permite el desarrollo multiplataforma, pero en este caso haciendo uso de la tecnología web, como se menciona en 2.5.4 Apache Cordova.

[21] https://www.android.com/intl/es_es/
[22] https://www.xamarin.com/

Elegir crear una aplicación exclusiva para Android provocaría que el número de usuarios estuviera limitado a los propietarios de dispositivos que usasen esta plataforma. Esto sería un inconveniente para el estudio, ya que cuantos más datos se recojan, mejores análisis podrá realizar el equipo investigador. Por este motivo se descartó el desarrollo en exclusiva para Android.

En el caso de Xamarin, este entorno hubiera sido una buena elección por las posibilidades de crear una app multiplataforma, pero se descartó debido al desconocimiento del lenguaje C# por parte del desarrollador ya que aún quedaban opciones de desarrollo. En el caso de que no hubiera más opciones, habría que haber realizado un período de adaptación, tanto al lenguaje como al entorno Xamarin, buscando cursos y manuales en la red.

Finalmente se eligió el entorno Apache Cordova. La elección tuvo dos motivos básicos. Uno fue que permite un desarrollo multiplataforma y otro que se usa la tecnología web HTML5, CSS3 y JavaScript. Al haber cursado la asignatura optativa Desarrollo de Aplicaciones Web Enriquecidas (DAWE), se tenían unos conocimientos previos acerca de estas tecnologías y el uso de este entorno permitiría aumentarlos.

Aunque, como se ha comentado, Apache Cordova permite realizar la aplicación multiplataforma, debido a la falta de recursos para desarrollar y probar la compatibilidad con iOS, el desarrollo se centró en Android. Aún así, se tuvo cuidado de que todos los plugins que se utilizasen fuesen compatibles con iOS para permitir una ampliación futura.

6.3 EL ENTORNO APACHE CORDOVA

Para comenzar a hacer uso de Apache Cordova, hubo que instalar la Interfaz de Línea de Comandos (o CLI por sus siglas en inglés). Esta herramienta se distribuye como un paquete npm (node package manager)[23]. Por lo tanto, lo primero que hubo que instalar en el ordenador fue Node.js[24]. Tras la instalación, existe la posibilidad de descargar e instalar un cliente de la plataforma Git, ya que CLI lo usa para descargar recursos cuando se utiliza una URL que hace referencia a un repositorio Git. Por último, para instalar Apache Cordova se hizo uso de la utilidad npm de Node.js. El módulo Cordova fue descargado automáticamente por npm. Desde el símbolo del sistema de Windows se escribió la siguiente línea:

```
C:\>npm install –g cordova
```

[23] https://www.npmjs.com/
[24] https://nodejs.org/en/download/

Tras la descarga y la instalación, ya estaba disponible Cordova desde la línea de comandos. El modificador *–g* le dice a npm que instale Cordova de modo global. Además de CLI, se deben instalar otros recursos dependiendo de la plataforma para la que se quiera desarrollar. En el caso de Android en Windows, hubo que instalar Java Development Kit (JDK)[25] y Android Studio[26]. Para trabajar con estos recursos la herramienta CLI de Cordova necesita que se declaren variables de entorno. Por lo general, CLI las declara automáticamente, pero hay casos en los que es necesario hacerlo de manera manual.

- Se creó la variable de entorno **JAVA_HOME** indicando la carpeta de instalación del JDK para su localización.
- Se añadió la variable de entorno **ANDROID_HOME** indicando el directorio de instalación del Android SDK para su localización.
- También se recomienda añadir a la variable de entorno **PATH** los directorios **tools** y **platform-tools** del Android SDK.

Con todos estos pasos realizados, ya se pudo empezar a crear la estructura del proyecto a desarrollar. Lo primero fue crear el proyecto Cordova y para ello se usó CLI pasándole como parámetro el directorio donde se alojó el proyecto, el nombre del paquete y el nombre del proyecto respectivamente. Para este proyecto, el comando quedó de la siguiente manera:

```
C:\>cordova create walkability com.proyecto.walkability Walkability
```

Con el directorio creado se accedió a él mediante la instrucción:

```
C:\>cd walkability
```

Ya dentro, se añadieron las distintas plataformas objetivo para las que iba a estar disponible el proyecto:

```
C:\walkability\>cordova platform add android
```

Una vez realizados estos pasos, el último paso que hubo que realizar fue añadir los plugins que se iban a necesitar para el funcionamiento del proyecto:

```
C:\walkability\>cordova plugin add nombre-del-plugin
```

De todos modos, no es estrictamente necesario añadir en ese punto los plugins. Estos pueden ser añadidos o eliminados en cualquier momento.

[25] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[26] http://developer.android.com/sdk/installing/index.html?pkg=tools

Tras esto, se puede decir que se tenían los mínimos para empezar a desarrollar el proyecto. Los pasos dados dieron lugar a la estructura vista en el apartado 5.1 Estructura de trabajo.

A la hora de compilar y generar un instalador de la aplicación desarrollada, la instrucción *cordova build* crea ese instalador para todas las plataformas instaladas en el proyecto. Si se necesita para una plataforma en particular bastará con añadir la plataforma a la instrucción:

```
C:\walkability\>cordova build

C:\walkability\>cordova build android
```

Para realizar tests de la aplicación, hay disponibles dos posibilidades. Por un lado, Cordova hace uso de los emuladores que normalmente incorporan los SDKs de desarrollo. Mediante la siguiente instrucción Apache Cordova abre un emulador con la aplicación desarrollada para realizar las pruebas necesarias.

```
C:\walkability\>cordova emulate android
```

Por otro lado, si se quiere realizar una prueba más real, se puede hacer uso de un dispositivo físico. La siguiente instrucción transferiría e instalaría la aplicación en el dispositivo objetivo y ya estaría disponible para su uso.

```
C:\walkability\>cordova run android
```

6.4 PLUGINS, DOTANDO DE FUNCIONALIDAD A LA APLICACIÓN

Como se explica en 5.1 Estructura de trabajo, los plugins son una parte importante del framework Apache Cordova (ApacheCordova.org, 2014), ya que internamente las aplicaciones desarrolladas funcionan como una página web sin ninguna funcionalidad más allá de las proporcionadas por JavaScript. Es por esto que los plugins se podrían dividir en dos categorías.

En la primera están los llamados plugins del núcleo. Este grupo está formado por aquellos plugins que permiten el acceso al hardware del dispositivo, como pueden ser la cámara o el acelerómetro. Estos están bajo mantenimiento por parte del proyecto Apache Cordova y es esta comunidad la encargada de realizar las actualizaciones pertinentes en caso de ser necesarias.

En la segunda categoría están los plugins de terceros. Éstos otorgan características y funcionalidades extras que no se encuentran en los plugins proporcionados por Apache Cordova. Al estar realizados por desarrolladores independientes, un plugin en concreto puede no estar disponible para todas las plataformas. Existen directorios donde, gracias a un buscador, se puede acceder a los distintos tipos de plugins disponibles. Desde la propia página de Apache Cordova se puede acceder a su buscador[27]. Aquí se pueden encontrar todos los plugins que han sido publicados en npm, aunque el mayor directorio es el de Github, ya que aquí los desarrolladores comparten todas sus creaciones.

Al iniciar una aplicación basada en Apache Cordova, se dispara un evento llamado *deviceready*. Este se lanza cuando todos los componentes de la aplicación han sido cargados y están disponibles para su uso. Por lo general, se recomienda que la ejecución del código no comience hasta que este evento sea lanzado, ya que así se garantiza el correcto funcionamiento de todo el código.

6.4.1 LA GEOLOCALIZACIÓN

Para este proyecto, la geolocalización es una parte esencial, ya que es necesario registrar las rutas realizadas por el usuario. Por ello, se hizo uso del plugin de geolocalización proporcionado por Apache Cordova[28].

Tras la instalación del plugin y siguiendo las indicaciones encontradas en la documentación, se realizaron algunas pruebas de su uso y funcionamiento. Al realizarlas, surgió el primer problema. Cuando la aplicación se encontraba en segundo plano, aunque el GPS continuaba solicitando y recibiendo información, el código JavaScript no se ejecutaba. Esto suponía un inconveniente, ya que la idea principal de este proyecto es que la aplicación se ejecute en segundo plano sin necesitar la interacción del usuario, salvo para ciertas funciones como la valoración de la ruta.

Por ello, era necesario encontrar un plugin que permitiera la ejecución del código JavaScript en segundo plano. Haciendo uso del buscador de la página oficial de Apache Cordova se encontró un sustituto.

Se trata del plugin ***cordova-background-geolocation-lt***[29]. Este plugin, además de funcionar en segundo plano, incorpora una característica especial. El propio código decide cuándo hacer uso del GPS en función de la velocidad a la que se mueva el dispositivo.

Para ello, redondea la velocidad a la que se mueve el dispositivo al 5 más cercano que, por lo que explica el desarrollador en la documentación, se refiere al múltiplo de 5 más cercano. Después, el valor resultante lo eleva al cuadrado y le suma un valor configurable llamado *distanceFilter* medido en metros.

[27] https://cordova.apache.org/plugins/
[28] https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-geolocation/index.html
[29] https://www.npmjs.com/package/cordova-background-geolocation-lt

Por tanto:

- Si una bicicleta se mueve a 7.7m/s con un distanceFilter configurado en 30m:

  Round(7.7, 5)^2 + 30 -> (10)^2 + 30 -> 130

  El sistema hará uso del GPS cuando el dispositivo se mueva 130 metros.

- En cambio si un coche se mueve a 30 m/s con el mismo distanceFilter:

  Round(30, 5)^2 + 30 -> (30)^2 + 30 -> 930

  El sistema hará uso del GPS cuando el dispositivo se mueva 930 metros.

En resumen, para un mismo valor del parámetro *distanceFilter*, cuanto mayor sea la velocidad, mayor será la distancia que habrá entre cada medición.

El inconveniente que se encontró en este plugin es que era de pago, por lo que se desechó esta opción. En su lugar se eligió otro basado en ese mismo plugin, pero ampliado y gratuito.

El plugin es ***cordova-plugin-mauron85-background-geolocation***[30] y ha sido el utilizado finalmente para el desarrollo del presente proyecto. Aunque se trata de un plugin diseñado para funcionar en segundo plano, también funciona en primer plano. Esto permite su uso de manera exclusiva sin necesidad de tener dos plugins de geolocalización distintos dependiendo de si la aplicación se encuentra en primer o segundo plano.

Entre las características ampliadas del nuevo plugin se encuentra la posibilidad de elegir entre la opción de dejar decidir al sistema cuándo hacer uso del GPS (modo *distanceFilter*) o la opción de ser el propio desarrollador el que elija cada cuánto tiempo se debe solicitar la posición.

Este modo es el llamado *fusedLocation*. Gracias a las APIs de Android FusedLocationProvider[31] y ActivityRecognition[32], permiten reducir el consumo de batería ya que detecta si el dispositivo se encuentra en movimiento o no. El intervalo definido por el desarrollador para que se solicite un punto nuevo solo es tenido en cuenta cuando se detecta movimiento.

Durante el período de pruebas se probaron ambos modos y en el apartado 8.2 Problemas durante la realización del TFG se detallan cuál fue el finalmente usado y los motivos que llevaron a esa elección.

En la aplicación desarrollada se hace uso principalmente de tres funciones del plugin.

[30] https://www.npmjs.com/package/cordova-plugin-mauron85-background-geolocation
[31] http://bit.ly/1UjvkCB
[32] http://bit.ly/1TZZ2kf

La primera es `backgroundGeoLocation.start()` y, como el propio nombre indica, sirve para iniciar la geolocalización. Al iniciar la aplicación, y tras realizar unas verificaciones previas, como por ejemplo comprobar que el usuario ya se encuentra registrado en la aplicación, se ejecuta esta función para comenzar el seguimiento del usuario en sus rutas a pie. La siguiente función usada es `backgroundGeoLocation.configure(success, fail, option)`. Al realizar una petición de posición al GPS se ejecuta el callback *success* cuando todo haya ido correctamente. Dentro del callback se encuentra el código mediante el cual se decide lo que es y no una ruta a pie, así como cuándo se inicia y cuándo se termina la ruta. El callback devuelve el punto proporcionado por el GPS y, mediante el código y los valores proporcionados por el equipo investigador, se trata para determinar si el usuario está en movimiento o no y decidir el estado de la ruta. En el callback *fail* se recogen los posibles fallos que se puedan dar. El apartado *option* permite configurar el plugin en distintos parámetros, como pueden ser la precisión deseada o el distanceFilter.

La tercera función necesaria se ejecuta dentro del callback *success* que sería la función `backgroundGeoLocation.finish()`. Esta función hay que ejecutarla para avisar al sistema de que la tarea en segundo plano ha acabado, ya que de no ser así el sistema acabaría con la tarea al estar mucho tiempo en ejecución y así evitar el colapso del procesador.

Además de estas funciones principales, hay disponibles otras complementarias. Es el caso de `backgroundGeoLocation.watchLocationMode(success, fail)` que permite al sistema detectar cambios en la configuración de los servicios de localización del dispositivo. Esta opción solo está disponible para la plataforma Android, pero al tratarse de una función complementaria, no interfiere en el funcionamiento de la aplicación. Al activar o desactivar los servicios de localización, se ejecutaría el callback *success* que devolverá true o false dependiendo de si están o no activos. En el caso de estar desactivados, se ejecutaría la función `backgroundGeoLocation.showLocationSettings()` que mostraría la opción de configurar los servicios de localización que incorpora el sistema operativo en su menú de ajustes.

6.4.2 ALMACENAMIENTO LOCAL

Como se comenta en el apartado 5.2 Diagrama relacional de la base de datos, a la hora de elegir el modo de almacenamiento interno hay disponibles varias herramientas y se optó por la basada en plugins. Apache Cordova recomienda el uso del plugin **cordova-sqlite-storage**[33], por lo que se eligió éste ya que se adaptaba a las necesidades del proyecto.

Tras instalar el plugin, se puede comprobar si la instalación ha sido correcta. Para ello, después de que el evento *deviceready* sea lanzado, se hace uso de `window.sqlitePlugin.echoTest(successCallback, errorCallback)`. En caso de que la instalación haya sido correcta, se ejecuta *successCallback*. Si por el contrario ocurre algún error, se ejecuta *errorCallback*. Esta comprobación no es obligatoria, pero si recomendada para verificar que el plugin funciona correctamente.

[33] https://www.npmjs.com/package/cordova-sqlite-storage

Después de esta comprobación, el plugin estaba listo para su uso. Lo siguiente que hay que hacer es abrir la base de datos.

Para ello, como puede verse en el siguiente extracto del código, se usan el nombre de la base de datos y el path donde estará ubicada. Aunque en la documentación pone que el path solo es necesario si la aplicación va a funcionar bajo iOS, hay que decir que en Android muestra un error a la hora de usar la aplicación si no se especifica, por lo que también es necesario su uso. La opción *androidLockWorkaround* se debe a un problema detectado por un usuario del plugin. El problema era que, a veces cuando la aplicación se paraba o cancelaba sin cerrar la base de datos, la base de datos se bloqueaba o la información insertada se perdía. Debido a esto se creó una solución temporal que cierra y vuelve a abrir la base de datos tras cada transacción, de esta manera se aseguran los datos y no se bloquea.

```
db = window.sqlitePlugin.openDatabase({name: 'walkability.db',
          androidLockWorkaround: 1, location: 1});
```

En la variable *db* se crea un objeto database que permite la interacción con la base de datos. Para hacer cualquier transacción, se hace referencia a la base de datos de la siguiente forma:

```
db.transaction(function(tx){
           tx.executeSql('SELECT * FROM configuracion', [], function(tx, rs){
                 if (rs.rows.length == 0) {
                       //SOLICITAR ULTIMA VERSION
                             solicitarConfig(0);
                 } else{
                       //COMPROBAR SI ES LA VERSION ACTIVA
                       comprobarVersionConfig();
                 }
           });
     }, function(err){console.log('ERROR existeConfig: ' + err.message);
error = err;}, function(){console.log('TODO OK existeConfig');});
```

El código anterior pertenece a la función que comprueba si existe ya una configuración previa. La transacción está compuesta por tres funciones. La primera es la que contiene la sentencia sql que se va a hacer contra la base de datos. Genera un objeto *tx* que es el encargado de ejecutar la sentencia. El siguiente componente de *executeSql* es un array que contiene los datos que, opcionalmente, se le pueden pasar a la sentencia SQL. Por ejemplo, si se quiere seleccionar la tupla que tenga un identificador en concreto, se realizaría de la siguiente manera.

```
tx.executeSql('SELECT * FROM configuracion WHERE identificador = ?',
[identificador], function(tx, rs){});
```

La variable *identificador* contendría el valor referente al identificador sobre el que se quiere realizar la consulta.

En último lugar se encuentra la función resultante de la consulta. Mediante *rs.rows.length* se puede saber si la consulta arroja alguna tupla o no. Aquí, como se puede ver en el ejemplo, se ejecutaba el código cuando la consulta devolvía algún resultado.

Las dos siguientes funciones de las tres que componen *transaction* se ejecutan cuando ocurra algún error o cuando la transacción finalice correctamente, respectivamente. Debido a la ejecución asíncrona, estas funciones eran necesarias para el funcionamiento secuencial del código. Siempre que la secuencia necesitase el acceso a la base de datos, lo conveniente era que el código posterior se ejecutase al ser lanzadas una de estas funciones.

Por ejemplo, la sección de código a continuación pertenece a la función que crea el esquema de la base de datos. Tras crearlo, se hace la llamada a las funciones que comprueban si existe una configuración anterior y si el usuario está creado. Si no se hiciera de esta manera, mientras se está creando la base de datos esas funciones serían llamadas y lanzarían un error ya que las tablas *configuración* y *usuario* no existirían.

```
db.transaction(function(tx) {
//código
}, function(err){console.log('ERROR inicializarBD: ' + err.message);},
function(){console.log('BD CREADA'); existeConfig(); existeUsuario();});
```

6.4.3 NOTIFICACIONES LOCALES

En el apartado 3 Antecedentes se comenta que una de las características necesarias que debe tener este proyecto y que no se han encontrado en otras aplicaciones existentes es la de poder valorar las rutas con una encuesta personalizable por los investigadores. Diariamente, cuando el usuario así lo haya definido, la aplicación de manera autónoma debe comprobar si se ha realizado alguna ruta. En caso afirmativo, si la ruta realizada es la primera vez que se realiza o, aunque haya sido realizada previamente, nunca ha sido valorada, se debe avisar al usuario de que tiene una valoración pendiente.

Para poder llevar a cabo la función de comprobación diaria, Apache Cordova no dispone de ningún tipo de alarma o programador de forma nativa. En JavaScript existen las funciones *setInterval()* que permite realizar una acción en intervalos regulares de tiempo y *setTimeout()* que ejecuta una acción cuando el tiempo configurado llega a cero. Por lo general, estas funciones trabajan con valores pequeños, medidos en segundos o milisegundos, por lo que se decidió buscar una alternativa.

Usando el buscador de plugins de Apache Cordova, se encontró el plugin **de.appplant.cordova.plugin.local-notification**[34]. Entre las características destacables del plugin está la de poder programar notificaciones en intervalos que pueden ir desde segundos o minutos hasta meses o años. También permite definir cuándo debe ser la primera vez que se lance y, a partir de ahí, cumplir los intervalos definidos.

A la hora de instalar el plugin, éste tenía dependencias de otros plugins, por lo que fueron instalados automáticamente. Los plugins instalados fueron **cordova-plugin-device**[35], **cordova-plugin-app-event**[36] y **cordova-android-support-v4**[37].

Tras su instalación, el plugin ya estaba disponible para su uso. A la hora de programar una notificación, ésta era bastante configurable. Tanto en Android como en iOS se permite configurar el identificador de la notificación (necesario para cancelar la notificación, seleccionarla o actualizar alguna de sus opciones), el texto a mostrar, el sonido de la notificación, etc.

Para el proyecto se necesitaba una notificación que se lanzase una vez al día, pero teniendo en cuenta cuándo quería el usuario que se le mostrase. Como el usuario podía lanzar la aplicación en cualquier momento, había que determinar cuándo sería la primera vez que debía ser lanzada y, a partir de ahí, seguir los intervalos diarios. Por ejemplo, si el usuario determinó que las valoraciones las hará a las 22:00 y lanza la aplicación a las 20:00, la notificación debería ser lanzada a las 22:00 del día actual. Por el contrario si determinó que las valoraciones las hará a las 20:00 y lanzó la aplicación a las 22:00 se debería esperar a las 20:00 del día siguiente para lanzar la notificación.

La notificación se definiría de la siguiente manera:

```
cordova.plugins.notification.local.schedule({
                        text: "Verificacion diaria",
                        id: 1,
                        every: "day",
                        at: momento
                        });
```

La opción *every* determina el intervalo que cumplirá la notificación. En la opción *at* se fija cuándo debe ser lanzada por primera vez la notificación. En este caso, la variable *momento* contendrá un objeto de tipo Date configurado con los datos de cuándo la notificación debe ser lanzada.

[34] https://www.npmjs.com/package/de.appplant.cordova.plugin.local-notification
[35] https://github.com/apache/cordova-plugin-device
[36] https://github.com/katzer/cordova-plugin-app-event
[37] https://github.com/floatinghotpot/cordova-plugin-android-support-v4

Las notificaciones también generan unos eventos, útiles para el funcionamiento de la aplicación. Se generan varios tipos de eventos, pero los que se utilizaron en el proyecto fueron *trigger* y *click*. El evento *trigger* se ejecuta cuando una notificación se lanza.

```
cordova.plugins.notification.local.on("trigger", onTrigger);
```

Cuando el evento se lanza, es capturado por el código anterior que ejecuta la función *onTrigger*. La función devuelve el objeto de la notificación que ha lanzado el evento. En el proyecto, este evento solo fue tenido en cuenta cuando lo lanzaba la notificación diaria.

```
function onTrigger(notification){
      //Se comprueba que sea la diaria (por defecto tiene el ID 1)
      if(notification.id == 1){
            //Se cancela para que no se muestre y no molestar al usuario en
caso de que no tenga nada pendiente de valorar
            cordova.plugins.notification.local.cancel(1,
            function(){console.log("Notificacion id " + notification.id + "
            quitada");});
            //Se vuelve a programar para el día siguiente (data_base.js)
            activarNotificacionDiaria();
            //Se comprueba si hay alguna ruta que valorar (cuestionario.js)
            obtenerHoraCuestionario();
      }
}
```

La notificación diaria no se debe mostrar al usuario, ya que no siempre tendrá algo pendiente de valoración. El plugin dispone de la función *clear* que elimina la notificación del centro de notificaciones, pero no la elimina de la barra de notificaciones. Por ello, se decidió hacer uso de la función *cancel* que elimina por completo la notificación, para después volver a programarla.

El otro evento usado en el proyecto es el de *click*, que solo podía ser lanzado por la notificación programada cuando hay disponible una ruta para valorar. Ésta, por defecto, tendrá el identificador 10 y lanzará la siguiente función al ser capturado el evento *click*.

```
//Cuando se hace click en una notificación

function onNotificationClick(notification){

      //Se comprueba que sea la lanzada tras detectar que hay rutas
      pendientes de valoración (por defecto tiene el ID 10)

      if (notification.id == 10) {

            //La notificación almacena en formato JSON el ID de la ruta y
            el de la fecha_ruta. Se transforma a texto

            parseado = JSON.parse(notification.data);

            //Tras la transformación a texto, se lanza el cuestionario
            (cuestionario.js)

            realizarCuestionario(parseado.idruta, parseado.idfecha_ruta);

      }

}
```

Las notificaciones también pueden contener datos, almacenados en formato JSON. En este caso, la notificación contendrá los datos necesarios para lanzar el cuestionario, como por ejemplo, el identificador de la ruta y el identificador de la fecha de la ruta. Esta notificación se lanzará cada hora hasta que el usuario realice el cuestionario, de esta manera, si el usuario eliminase la notificación o dejase el cuestionario a medias, se le recordaría.

Aprovechando que el plugin de notificaciones tenía dependencias con el plugin **cordova-plugin-device**, se utilizó para dar solución a un problema. El problema era obtener un identificador único para almacenar los datos en la base de datos, tanto local como remota. En un principio se pensó usar el objeto Date que al crearse genera un objeto con la hora y fecha del día actual según la hora local. Gracias a esto y accediendo a la función *getTime()* del propio objeto, se obtenían los milisegundos transcurridos desde la media noche exacta del 01 de enero de 1970 en formato UTC.

Si solo hubiera sido necesario almacenar los datos de manera local, hubiera sido una solución aceptable, ya que los valores obtenidos no se repetirían en el tiempo. Pero en el caso del presente proyecto no era una solución válida, ya que, aunque era algo complicado, se podía dar el caso que dos usuarios justo en el mismo momento realizaran una acción que diera lugar a un mismo identificador.

El plugin **cordova-plugin-device**, entre otras cosas, permite acceder al Identificador Único Universal (UUID por sus siglas en inglés). Este identificador es como la dirección MAC de las tarjetas de red e identifica de manera única al dispositivo. Se genera en el primer encendido del dispositivo, por lo que persiste durante toda la vida del dispositivo. Gracias a este identificador, y unido a los milisegundos obtenidos a través de la función *getTime()* del objeto Date(), se obtiene un identificador único. En el código siguiente se puede ver cómo se crea el identificador del usuario. Se obtienen el UUID del dispositivo y los milisegundos y se concatenan para formar el identificador.

```
uuid = device.uuid;
date = new Date();
idusuario = uuid.concat(date.getTime());
```

6.5 SERVIDOR PHP

Como se comenta en el apartado 2.4 Arquitectura, el encargado de acceder a la base de datos MySQL y realizar acciones con ella es el servidor PHP. En función de las acciones que se quieran realizar (insertar, seleccionar o actualizar datos), existirá un archivo en el servidor que contenga estas acciones y los parámetros necesarios para cada consulta.

Todas las consultas iban a ser realizadas sobre la misma base de datos con los mismos datos de conexión, por lo que se decidió encapsular los datos en el siguiente archivo.

```
<?php
$host = "galan.ehu.eus";//host
$usuario = "Xdpuerto001";//usuarioBD
$pass = **********;//Contraseña
$bd = "Xdpuerto001_servidor_db";//Base deDatos

$servidor = mysqli_connect($host, $usuario, $pass, $bd);

if (mysqli_connect_errno()) {
      printf("Connect failed: %s\n", mysqli_connect_error());
      exit();
}

//Se elige el formato de datos para la conexion UTF8
mysqli_set_charset($servidor, "utf8");
mysqli_autocommit($servidor, FALSE);
?>
```

Este archivo fue llamado *connect.php* y para poder hacer uso de él, en los archivos se añadió la instrucción *require 'connect.php'* al principio del archivo de tal modo que en primer lugar se ejecutaría el código que realiza la conexión y después el código de cada archivo.

En la última línea del código se encuentra la función *mysqli_autocommit()*. Ésta permite desactivar el autocommit de la conexión, ya que por defecto está activo. Al final de cada conexión habrá que confirmar todos los cambios hechos en la base de datos. De esta manera, si durante la conexión algo fuese mal, no quedarían acciones inacabadas.

En el código siguiente se puede ver como se realiza la solicitud del texto necesario para mostrar las categorías en los cuestionarios. En primer lugar, se realizaría la conexión a la base de datos a través del archivo *connect.php* y después se selecciona el texto de las categorías para, finalmente, insertarlos en un array y devolverlos a través del objeto JSON.

```
<?php
require 'connect.php';

$consulta = "SELECT `texto` FROM `categoria` ORDER BY `idcategoria`
ASC";

$sql = mysqli_query($servidor, $consulta);
if ( ! $sql) {
      echo "La conexion no se logró ".mysqli_error($sql);
      //Se declara que esta es una aplicacion que genera un JSON
      header('Content-type: application/json');
      //Se abre el acceso a las conexiones que requieran de esta
aplicacion
      header("Access-Control-Allow-Origin: *");
      die;
}
$datos = array();
while ($obj = mysqli_fetch_object($sql)) {
      $datos[] = array('categoria' => $obj->texto);
}
mysqli_free_result($sql);
echo '' . json_encode($datos) . '';
//Se declara que esta es una aplicacion que genera un JSON
header('Content-type: application/json');
//Se abre el acceso a las conexiones que requieran de esta aplicacion
header("Access-Control-Allow-Origin: *");

mysqli_close($servidor);//Se cierra la conexion
?>
```

Con el fin de tratar los posibles errores que se pueden dar al conectar con la base de datos o al realizar cualquier consulta, se añadió una condición. Al ejecutarse la consulta a través de la función *msqli_query()*, ésta devolverá *FALSE* en caso de que haya ocurrido un error. Si se produce el error, se genera un JSON que avisa a la aplicación. Más adelante se explicará cómo se tratan estos errores.

6.6 APROVECHANDO JQUERY

jQuery es uno de los complementos más esenciales para el desarrollo web, usado en millones de sitios en Internet, ya que facilita mucho el desarrollo de aplicaciones enriquecidas del lado del cliente, en JavaScript, compatibles con todos los navegadores (Desarrolloweb.com, 2012).

6.6.1 ENVÍO Y RECEPCIÓN DE DATOS

Una de las tecnologías usadas en el desarrollo web para enviar y recibir datos es AJAX (Asynchronous JavaScript And XML en inglés). Se trata de una tecnología asíncrona, ya que los datos se solicitan y se cargan en segundo plano sin interferir en el funcionamiento de la aplicación.

Este método de envío y recepción de datos fue elegido para el desarrollo del presente proyecto por su sencillo funcionamiento a través de jQuery y también porque ya se tenían unos conocimientos previos de su uso.

El funcionamiento de AJAX a través de jQuery es de la siguiente manera. En primer lugar, en la variable *type* se define el tipo de conexión que se realizará. Por preferencia, en todas las conexiones se eligió el método POST. En la variable *url* se añade la dirección al archivo php al que se debe acceder. Dependiendo de la conexión, la dirección puede ser distinta. La variable *data* contiene los valores que se mandarán al servidor al realizar la llamada. Estos irán en formato JSON. Por último *success* y *error* contienen el código que se ejecutará en caso de conexión exitosa o de error, respectivamente.

El siguiente código es el usado a la hora de comprobar la versión de configuración de la aplicación que actualmente se encuentra activa en el servidor. Para ello, a través de POST y usando JSON, se envía a la dirección definida en *url* la versión de configuración que tiene el dispositivo. En el archivo php se recoge la versión y se comprueba si coinciden. En caso de coincidencia se crea un array con la variable *cambiar* con valor 0. Si no hay coincidencia, se selecciona la nueva configuración y se crea un array con la variable *cambiar* con valor 1 y los datos de la nueva configuración. Finalmente, el array creado se devuelve usando también JSON.

Si no ha ocurrido ningún error, a través de *success* se recibiría el objeto JSON devuelto por el servidor. Aquí se comprobaría el valor de la variable *cambiar* para determinar si hay que actualizar la configuración o no.

Si hubiera ocurrido algún error, se ejecutaría el código contenido en *error*. En este caso, no se realizaría ninguna acción.

```
$.ajax({type: "POST",
      url:
      "http://galan.ehu.eus/dpuerto001/WEB/comprobarVersionConfig.php",
      data: {dato:version},
      success: function(data){
                  cambiar = parseInt(data[0].cambiar);

                  if (cambiar == 1) {
                        actualizarConfig(data[0]);
                  }
      },
      error: function(e){console.log("ERROR");},
});
```

6.6.2 *INTERACCIÓN CON DOM*

Otra de las posibilidades que ofrece jQuery es poder interactuar con los elementos de DOM[38]. El Modelo de Objetos del Documento (DOM por sus siglas en inglés) es una API para documentos HTML y XML. Con el DOM, los programadores pueden construir documentos, navegar por su estructura, y añadir, modificar o eliminar elementos y contenido (Miembros del W3C, 2001). Por ejemplo, se puede utilizar para ocultar de la visión del usuario un bloque de contenido o cambiar el texto a una etiqueta desde una simple función.

En ocasiones es necesario recoger los datos que contienen estos objetos, como por ejemplo a la hora de recopilar información introducida por el usuario. Esta acción también es soportada por jQuery permitiendo un acceso rápido y sencillo.

Aplicando estas funcionalidades al presente proyecto, se pueden ver varios ejemplos de uso.

El código siguiente es un ejemplo de cómo mostrar y ocultar bloques de contenido al usuario. En primer lugar se recoge el evento disparado al hacer click sobre el botón con identificador *btnInicio* del panel lateral de la aplicación. Cuando el evento es recogido, se ejecuta la función que muestra el bloque con identificador *principal* y oculta los bloques *configuración* y *acerca*.

[38] https://www.w3.org/DOM/

```
//Muestra el div principal y oculta el resto
$("#btnInicio").on( "click", function() {
      $('#principal').show();
      $('#configuracion').hide();
      $('#acerca').hide();
});
```

Como se ve en el ejemplo, con jQuery también se pueden controlar los eventos. Esta función permite capturar los eventos disparados y ejecutar código dependiendo del evento.

A continuación se puede ver el código que se diseñó para crear el cuestionario y mostrárselo al usuario. Antes de comenzar a crear nada, se comprueba si hubiera alguna pregunta en el bloque de las preguntas y en caso afirmativo, se elimina. Tras esto, y usando los datos devueltos por el servidor, se comienza a crear el cuestionario.

Por cada dato contenido en el array devuelto se crea y añade un objeto tipo *legend* utilizado para mostrar el texto de la pregunta. El siguiente paso es crear y añadir el bloque que contendrá las posibles respuestas a la pregunta. A este bloque, en el atributo *name*, se le añade el identificador de la pregunta para así poder identificarlo a la hora de recoger las respuestas. Por cada pregunta se crea y añade un objeto *label* que contiene el texto de la respuesta y, dependiendo del valor del variable *tipoRespuesta*, se crea un objeto tipo *radioButton* o tipo *checkBox* ya que las preguntas pueden ser de respuesta única o múltiple. Cuando todas las respuestas son creadas y añadidas se define el bloque de respuestas como *controlgroup*, lo que agrupará los objetos contenidos para mostrarlos como un solo grupo.

Por último, se almacena la versión del cuestionario para su posterior uso a la hora de recoger los datos y se muestra el cuestionario.

```
function crearCuestionario(data){
      if ($("#preguntas div[id^='pregunta']").length != 0) {
            $("#preguntas").empty();
      }

      preguntasRespuestas = data.preguntasRespuestas;
      for (var i = 0; i < preguntasRespuestas.length; i++) {
            var enunciado = $("<legend>", {id:"texto"+ (i+1),
text:preguntasRespuestas[i].textoPregunta});
            $("#preguntas").append(enunciado);
            var pregunta = $("<div>", {id:"pregunta"+ (i+1),
name:preguntasRespuestas[i].idPregunta});
            $("#preguntas").append(pregunta);
            var tipoRespuesta = preguntasRespuestas[i].tipoRespuesta;
            for (var j = 0; j <
preguntasRespuestas[i].respuestas.length; j++) {
                  var respuesta = preguntasRespuestas[i].respuestas[j];
                  var label = $("<label>", {for: respuesta.idRespuesta,
text: respuesta.textoRespuesta});
                  $("#pregunta"+ (i+1)).append(label);
                  if (tipoRespuesta == 0) {
                        var opcion = $("<input>", {type:"radio",
name:"pregunta"+ (i+1), id:respuesta.idRespuesta});
                  } else {
                        var opcion = $("<input>", {type:"checkbox",
name:"pregunta"+ (i+1), id:respuesta.idRespuesta});
                  }
                  $("#pregunta"+ (i+1)).append(opcion);
            }
            $("#pregunta"+ (i+1)).controlgroup();
      }
      $("#cuestionario").attr("name", data.version);
      $("#cuestionario").show();
      $("body").removeClass("loading");
```

Como se ha comentado en el apartado anterior, la comunicación a través de AJAX se realiza de manera asíncrona. Además, por temas de ejecución interna de la aplicación, puede darse el caso de que los datos tarden en mostrarse, lo que puede hacer pensar que la aplicación se ha bloqueado. Para evitar esto y aprovechando jQuery se creó una pantalla de espera. Se trata de un bloque de clase modal, que al mostrarse bloquea la interacción del usuario con la aplicación mientras ésta realiza las ejecuciones necesarias. En la última línea del bloque de código anterior se puede ver como se elimina la pantalla de espera tras la creación del cuestionario.

6.6.3 JQUERY PAGES

Al comienzo del desarrollo, se pensó dividir las distintas pantallas de la aplicación en archivos html separados. El problema fue que esto iba a generar código duplicado porque, por ejemplo, para que el panel lateral fuera visible en todas las páginas, el código que lo forma debería estar en cada uno de los archivos.

Otra opción fue crear un bloque por cada página. Esto permitía mostrar solo lo necesario en cada caso, pero surgían problemas de visualización a la hora de gestionar la visualización de ciertos apartados, como puede ser el texto y test iniciales, que solo se deben mostrar cuando se inicia por primera vez la aplicación.

Tras investigar en internet se encontró una solución. Ésta fue jQuery Mobile[39], un framework optimizado para dispositivos táctiles dependiente de jQuery usado para crear diseños web adaptables, es decir, que se adaptan a los distintos dispositivos en los que se muestran. Esto permitió, además de darle un toque más personal a la interfaz, hacer uso de las distintas funciones que aporta.

Entre ellas se encontró jQuery Mobile Page[40] que permitía que, en un mismo archivo, se crease la estructura de distintas páginas, cada una con su *header*, *body* y *footer*. Gracias a esto, no sería necesario cargar un archivo html distinto cada vez que hubiera que cambiar de página, solo ejecutar el código para mostrar la página solicitada.

jQuery Mobile Page también cuenta con una serie de evento que fueron útiles a la hora de controlar ciertos funcionamientos de la aplicación. Por ejemplo, había que controlar la carga del mapa mostrado en la pantalla principal. Gracias a los eventos de jQuery Mobile Page y sus controladores, fue posible gestionar esta carga.

El código siguiente corresponde al controlador que captura el evento *pagehide*. Este se genera tras la transición de una página a otra. En este caso, el evento se generará cuando la pagina con identificador *testInicial* se oculta y ocurrirá únicamente cuando se pasa del test inicial a la pantalla principal de la aplicación.

[39] https://jquerymobile.com/
[40] http://demos.jquerymobile.com/1.4.0/pages/

```
//Al ocultarse la página testInicial, cargar la configuración,
inicializar el mapa principal e iniciar el seguimiento
$(document).on("pagehide", "#testInicial", function() {
      cargarConfig();
      initMap();
      $("#btnSeguimiento").trigger("click");
});
```

La última línea de código generará un evento click en *btnSeguimiento* lo que provocará que se inicie el seguimiento.

6.7 TEMPORIZADORES

Al principio de este capítulo, en el apartado 6.1 Aclaraciones previas, se comenta que al estar tanto en el estado de **PresuntaRuta** como en el de **Ruta** hay que controlar el tiempo que el usuario se detiene, así como el tiempo que transcurre en la recepción entre puntos. En ambos casos debe darse por finalizada la ruta.

Como se explica en el apartado 6.4.3 Notificaciones locales, Apache Cordova no cuenta con una alarma de forma nativa, por lo que fue necesario hacer uso de la función de JavaScript *setTimeout()*. El funcionamiento de esta función es sencillo ya que lo que permite es ejecutar una función definida transcurrido un intervalo de tiempo una única vez.

A continuación, se puede ver el código diseñado para el temporizador de puntos. En él se definió que el tiempo que debería transcurrir son 900000 milisegundos, lo que corresponde a 15 minutos. La función establecida para ejecutarse transcurridos los 15 minutos lo que hace es, en primer lugar, parar si estuviera activo el temporizador de parada. Después cambiar el estado a 1, que corresponde al estado **NoRuta**. Por último, si se encontraba en el estado 3 (**Ruta**), almacenará los puntos recogidos hasta ahora.

```
temporizadorPunto = setTimeout(function(){

      if (temporizadorParada != 0) {
            clearTimeout(temporizadorParada);
            temporizadorParada = 0;
      }

      estadoAnterior = estado;
      estado = 1;

      if (estadoAnterior == 3) {
      recogerRutas(ruta, distanciaTotal);
      }
}, 900000);
```

El temporizador de parada siguió una configuración similar, la diferencia fue que el intervalo de tiempo de parada estaba almacenado en la variable *tiempo_parada*, ya que varía según las necesidades del estudio.

6.8 API GOOGLE MAPS

Al tratarse de un proyecto cuya función principal es la geolocalización, se hizo necesaria la presencia de la API Google Maps[41] ya que ofrece más funciones que la de representar mapas.

Hasta la realización del proyecto, el desarrollador pensaba que la API Google Maps se limitaba a la representación de mapas, pero tras investigar al respecto, se dio cuenta del potencial que tenía gracias a las librerías[42] con las que cuenta. Estas son **drawing**, **geometry**, **places** y **visualization**.

La biblioteca **drawing** proporciona a los usuarios una interfaz para dibujar polígonos, rectángulos, polilíneas, círculos y marcadores en el mapa. La biblioteca **geometry** incluye funciones de utilidades para calcular valores geométricos escalares (como la distancia y el área) sobre la superficie terrestre. La biblioteca **places** permite a la aplicación buscar sitios, como establecimientos, ubicaciones geográficas o puntos de interés destacados, dentro de un área definida. Por último, la librería **visualization** proporciona representaciones visuales de datos, como los de mapas de calor y los del motor de Google Maps.

Para el proyecto, la única librería que hizo falta fue la de **geometry**, ya que era necesario realizar mediciones de distancia, como por ejemplo la distancia total recorrida o la distancia entre dos puntos.

Con el siguiente código se carga la API en la aplicación. Este tenía que ir dentro de la cabecera del archivo html, es decir, entre las etiquetas <header>.

```
<script async defer
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&libraries=geometry"><
/script>
```

En el código anterior, se puede ver que el valor del parámetro *key* es *YOUR_API_KEY*. Esto se debe a que todas las aplicaciones API para JavaScript requieren autenticación con una clave de API[43]. Ésta permite, entre otras cosas, controlar el uso de la API por parte de la aplicación, habilitar límites de uso, etc.

Una de las utilidades de la API es la creación del objeto *LatLng*. Este objeto es solicitado por las distintas funciones de la API que se usaron en el proyecto cuando se necesitaba pasar parámetros referentes a las coordenadas.

[41] https://developers.google.com/maps/?hl=es
[42] https://developers.google.com/maps/documentation/javascript/libraries?hl=es
[43] https://developers.google.com/maps/documentation/javascript/get-api-key?hl=es

En el código a continuación, se puede ver cómo se crea el objeto *LatLng*. El primer parámetro correspondería a la latitud y el segundo a la longitud.

```
latlon = new google.maps.LatLng(location.latitude, location.longitude);
```

Para crear el mapa era necesario crear un objeto del tipo *Map*. Como parámetros había que pasar al constructor el contenedor donde iba a estar ubicado y las opciones con las que debía ser mostrado el mapa. Entre las opciones posibles, estaba dónde debía estar centrado el mapa, el nivel de zoom o el tipo de mapa.

Las secciones de código que se encontrarán a continuación corresponden al contenido en la función *pintarMapa*(), usado para mostrar al usuario la ruta que ha realizado antes de responder al cuestionario. El siguiente código genera el mapa a mostrar.

```
var mapa = new google.maps.Map(mapholder,{
      center:puntos[0],
      zoom: 13,
      mapTypeId: google.maps.MapTypeId.HYBRID,
      mapTypeControl:false,
      navigationControlOptions:{style:google.maps.NavigationControlStyle
      .SMALL},
      streetViewControl: false,
      minZoom: 13,
      maxZoom: 17,
      rotateControl: false
});
```

Para que la ruta mostrada tuviera un detalle mayor, se hizo uso del objeto *Polyline*. Éste, recibiendo como parámetro una serie de puntos ordenados, genera una línea que une todos los puntos. De esa manera, es posible visualizar enseguida la ruta que se ha seguido.

El siguiente código es el encargado de generar la línea. El parámetro *path* contiene el grupo de puntos necesario para dibujar la línea, *strokeColor* define el color de la línea, *strokeOpacity* la opacidad que debe tener la línea, *strokeWeight* define el grosor de la línea y *map* dónde se dibujará la línea.

```
var rutaPath = new google.maps.Polyline({
      path: puntos,
      strokeColor: '#FF0000',
      strokeOpacity: 1.0,
      strokeWeight: 2
      map: mapa
});
```

Otro objeto ofrecido por la API es el de tipo *Marker*. Con éste se puede colocar en las coordenadas pasadas por parámetro un marcador indicando el punto. El siguiente código es el utilizado para indicar el inicio de la ruta. El parámetro *position* contiene las coordenadas del punto en formato *LatLng*. El parámetro *label* permite añadir al marcador texto y en este caso se añadió la letra "I". Finalmente, el parámetro *map* indica dónde irá colocado el marcador

```
var marcaInicio = new google.maps.Marker({
      position: puntos[0],
      label: "I",
      map: mapa
});
```

Con la intención de dar más detalle a la representación de la ruta sobre el mapa, se añadieron marcadores a cada uno de los puntos que conforman la ruta. En este caso, el marcador mostrado no tendría el símbolo por defecto, si no que se colocaría un círculo de color verde. El código a continuación corresponde a esos marcadores.

```
marcador = new google.maps.Marker({
      map: mapa, //Objeto Map sobre el que se muestra el marcador
      icon: {
            path: google.maps.SymbolPath.CIRCLE, //Tipo de
            símbolo del marcador
            scale: 3, //Tamaño
            fillColor: 'green', //Color de relleno
            strokeColor: 'green', //Color de línea del símbolo
            strokeWeight: 5 //Anchura de la línea del símbolo
      },
      position: puntos[i] //Posición en el mapa
      });
```

Una de las necesidades desde el inicio del proyecto era conocer la distancia recorrida por el usuario para determinar cuándo se debía pasar del estado **PresuntaRuta** al estado **Ruta**. La función *computeLength()* facilitaba esta tarea, ya que pasando a la función como parámetro un array con los objetos *LatLng* referentes a los puntos recogidos hasta el momento, esta devolvía la distancia total recorrida.

```
distanciaTotal = google.maps.geometry.spherical.computeLength(pathRuta);
```

En otras ocasiones también era necesario conocer la distancia entre dos puntos en concreto, por ejemplo a la hora de comparar dos rutas punto a punto. Por ello, haciendo uso de la función *computeDistanceBetween()* y pasando como parámetros el objeto *LatLng* de los dos puntos a medir, ésta devolvía la distancia entre los dos puntos.

```
                        var distancia =
google.maps.geometry.spherical.computeDistanceBetween(punto1, punto2);
```

Con el fin de facilitar el posterior análisis, el grupo investigador solicitó que, además de las coordenadas de inicio y fin de cada ruta, se obtuviera el municipio de inicio y fin. Gracias al servicio de geocodificación[44] de la API Google Maps, esta tarea fue sencilla. En realidad, lo que se realizaba era una geocodificación inversa.

La diferencia es que la geocodificación convierte direcciones en coordenadas geográficas que se pueden usar para colocar marcadores o posicionar el mapa. En cambio, en la geocodificación inversa convierte una ubicación del mapa en una dirección en lenguaje natural.

En el siguiente código se puede ver cómo se definió dentro de la función *anadirRuta()*. Se pasó como parámetro la latitud y la longitud del punto. Tras esto, se realiza la solicitud al servicio de geocodificación. Al completarse la solicitud, en el método *callback* se tratan los resultados devueltos. El geocodificador inverso a menudo devuelve más de un resultado. Las "direcciones" no representan solo direcciones postales, sino cualquier forma de asignar nombres geográficos a una ubicación.

```
var geocoder = new google.maps.Geocoder;

geocoder.geocode({'location': {lat:ruta[0].latitud,
lng:ruta[0].longitud}}, function(results, status) {
      if (status == google.maps.GeocoderStatus.OK) {
            municipio_inicio =
            results[1].address_components[0].short_name;
      } else {
            municipio_inicio = "No determinado";
      }
});
```

6.9 *LIBRERÍAS DE TERCEROS*

A la hora de crear el test inicial, una de las cuestiones que se realiza al usuario es su municipio de procedencia. En un principio se optó por colocar un *TextArea* a través del cual ingresar el municipio. Esto suponía tener que controlar no solo que no se dejase vacío y que solo se utilizasen letras, sino también que los datos introducidos correspondieran a municipios españoles.

[44] https://developers.google.com/maps/documentation/javascript/geocoding?hl=es#Geocoding

Viendo esta problemática, se decidió indagar en internet a ver si era posible buscar una solución sencilla. Resultado de esta investigación fue el hallazgo de la librería PSelect.js[45] del usuario IagoLast en Github. Esta librería rellenaba de manera automática selects con las provincias y municipios españoles.

Para su correcto funcionamiento, el desarrollador en las instrucciones de uso solicitaba la presencia de la librería jQuery en el proyecto, lo que no era un problema ya que ya estaba siendo usada. Tras esto, únicamente se añadieron dos *selects*: uno con la clase "ps-prov" y otro con la clase "ps-mun" para las provincias y los municipios, respectivamente.

```
<legend>¿De dónde eres?</legend>
<select class="ps-prov"></select>
<select class="ps-mun"></select>
```

[45] https://github.com/IagoLast/pselect

7 VERIFICACIÓN Y EVALUACIÓN

Es este capítulo se detallan algunas de las pruebas realizadas a lo largo de todo el proceso de desarrollo. Gracias a la constancia y a la atención puesta a la hora de realizar las pruebas, se han ido identificando y rectificando los errores encontrados en cada uno de los prototipos realizados.

Además de las pruebas expuestas aquí, se han realizado infinidad de pruebas con la finalidad de solventar los errores existentes. Tras realizar el prototipo final, éste también fue sometido a pruebas en un entorno real por los dos directores del proyecto.

Como se comentó en el apartado 2.2.1 Ciclo de vida, se eligió un ciclo de vida incremental en el que se irían creando prototipos cada vez más completos, realizando pruebas tras la consecución de cada prototipo. Las pruebas que se presentan a continuación son una muestra de todas las realizadas.

7.1 PRUEBAS PROTOTIPO 1

Se comprueba que la interacción con la base de datos local se realiza correctamente.

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1	Insertar un dato que no está en la BD	Mensaje indicando que ha sido insertado correctamente	Mensaje indicando que ha sido insertado correctamente	Correcto
2	Insertar un dato que ya está en la BD	Mensaje indicando que no admiten valores duplicados	Mensaje indicando que no admiten valores duplicados	Correcto
3	Extraer dato de la BD presente en ella	Mensaje mostrando el dato extraído	Mensaje mostrando el dato extraído	Correcto
4	Extraer dato de la BD no presente en ella	Mensaje vacío	Mensaje vacío	Correcto

Tabla 2 - Pruebas prototipo 1

7.2 PRUEBAS PROTOTIPO 2

Se comprueba que la interacción con la base de datos remota se realiza correctamente.

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1.1	Enviar e insertar dato no presente en la BD externa	Mensaje indicando que ha sido insertado correctamente	Mensaje de error	Incorrecto
1.2	Después de corregir el código, enviar e insertar dato no presente en la BD externa	Mensaje indicando que ha sido insertado correctamente	Mensaje indicando que ha sido insertado correctamente	Correcto
2	Enviar e insertar dato presente en la BD externa	Mensaje de error	Mensaje de error	Correcto
3.1	Recuperar dato de la BD externa presente en ella	Mensaje mostrando el valor devuelto	Mensaje vacío	Incorrecto
3.2	Después de corregir el código, recuperar dato de la BD externa presente en ella	Mensaje mostrando el valor devuelto	Mensaje mostrando el valor devuelto	Correcto
4	Recuperar dato de la BD externa no presente en ella	Mensaje vacío	Mensaje vacío	Correcto

Tabla 3 - Pruebas prototipo 2

PRUEBA 1.1

Al crear la llamada AJAX, el parámetro data contiene los datos que van a ser enviados al servidor. Los datos eran añadidos directamente, sin formato JSON. Identificado el problema, se formatearon los datos y fueron recibidos e insertados correctamente en la base de datos.

```
//Código erróneo
data: datosJSON,
//Código válido
data: {dato:datosJSON},
```

PRUEBA 3.1

Tras recibir los datos del servidor, estos se parseaban con la función *JSON.parse()*. Esta operación devolvía un error por consola porque los datos devueltos ya estaban en formato JSON, así que se podía acceder directamente a ellos.

7.3 PRUEBAS PROTOTIPO 3

Las pruebas para el prototipo 3, donde se desarrolló la detección y seguimiento de la ruta, se dividieron según el estado en el que se encontrase.

7.3.1 ESTADO 1

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1	El usuario no se mueve	Se permanece en el estado 1	Se permanece en el estado 1	Correcto
2	El usuario se mueve a velocidad permitida.	Se guarda el punto y se cambia al estado 2	Se guarda el punto y se cambia al estado 2	Correcto
3	El usuario se mueve a velocidad superior a la permitida	Se permanece en el estado 1	Se permanece en el estado 1	Correcto

Tabla 4 - Pruebas prototipo 3, estado 1

7.3.2 ESTADO 2

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1	El usuario supera la velocidad permitida una vez sin superar distancia mínima	Se almacena el punto y se permanece en el estado 2	Se almacena el punto y se permanece en el estado 2	Correcto
2	El usuario supera la velocidad permitida dos veces seguidas	Se pasa al estado 1	Se pasa al estado 1	Correcto
3	El usuario supera la velocidad permitida una vez superando distancia mínima	Se almacena el punto y se pasa al estado 3	Se almacena el punto y se pasa al estado 3	Correcto

Tabla 5 - Pruebas prototipo 3, estado 2 (1/2)

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
4	El usuario se mueve a la velocidad permitida sin superar la distancia mínima	Se almacena el punto y se permanece en el estado 2	Se almacena el punto y se permanece en el estado 2	Correcto
5	El usuario se mueve a la velocidad permitida superando la distancia mínima	Se almacena el punto y se pasa al estado 3	Se almacena el punto y se pasa al estado 3	Correcto
6.1	El usuario se para y no se mueve en el tiempo máximo permitido	Se pasa al estado 1	No se detecta la parada	Incorrecto
6.2	Después de corregir el código, el usuario se para y no se mueve en el tiempo máximo permitido	Se pasa al estado 1	Se pasa al estado 1	Correcto
7	El usuario se para y se mueve antes de cumplir el tiempo, sin alejarse los metros mínimos del punto de parada	Se pasa al estado 1	Se pasa al estado 1	Correcto
8	El usuario se para y se mueve antes de cumplir el tiempo, alejándose los metros mínimos del punto de parada	Se almacena el punto y se permanece en el estado 2	Se almacena el punto y se permanece en el estado 2	Correcto
9	El usuario se para y se mueve antes de cumplir el tiempo, alejándose los metros mínimos del punto de parada y superando la distancia mínima de ruta	Se almacena el punto y se pasa al estado 3	Se almacena el punto y se pasa al estado 3	Correcto
10	No se recibe punto nuevo antes de 15 minutos	Se pasa al estado 1	Se pasa al estado 1	Correcto

Tabla 6 - Pruebas prototipo 3, estado 2 (2/2)

PRUEBA 6.1

En las pruebas realizadas al prototipo, se descubrió que el plugin no devolvía una velocidad igual a 0 exactamente, sino que devolvía un valor entre 0 y 1 km/h. La condición para que se detectara la parada era que la velocidad fuese igual a 0, por lo que esta parada nunca era detecta. La solución fue cambiar la condición a que la velocidad fuese inferior a 1. De este modo se consiguió detectar las paradas del usuario.

7.3.3 ESTADO 3

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1.1	Tras superar velocidad máxima una vez en estado 2, se pasa a estado 3 y se vuelve a superar	Se pasa al estado 1	Se almacena la ruta y se pasa al estado 1	Incorrecto
1.2	Después de corregir el código, tras superar velocidad máxima una vez en estado 2, se pasa a estado 3 y se vuelve a superar	Se pasa al estado 1	Se pasa al estado 1	Correcto
2	El usuario supera la velocidad permitida una vez	Se permanece en el estado 3	Se permanece en el estado 3	Correcto
3	El usuario supera la velocidad permitida dos veces seguidas	Se almacena la ruta y se pasa al estado 1	Se almacena la ruta y se pasa al estado 1	Correcto
4	El usuario se mueve a la velocidad permitida	Se almacena el punto y se permanece en el estado 3	Se almacena el punto y se permanece en el estado 3	Correcto
5	El usuario se para y no se mueve en el tiempo máximo permitido	Se almacena la ruta y se pasa al estado 1	Se almacena la ruta y se pasa al estado 1	Correcto
6	El usuario se para y se mueve antes de cumplir el tiempo, sin alejarse los metros mínimos del punto de parada	Se permanece en el estado 3	Se permanece en el estado 3	Correcto
7	El usuario se para y se mueve antes de cumplir el tiempo, alejándose los metros mínimos del punto de parada	Se almacena el punto y se permanece en el estado 3	Se almacena el punto y se permanece en el estado 3	Correcto
8	No se recibe punto nuevo antes de 15 minutos	Se almacena la ruta y se pasa al estado 1	Se almacena la ruta y se pasa al estado 1	Correcto

Tabla 7 - Pruebas prototipo 3, estado 3

PRUEBA 1.1

Al pasar al estado 3, hay que controlar si la velocidad ya había sido superada una vez en el estado anterior y, en caso afirmativo, la ruta debería darse por finalizada. En las pruebas esto ocurría así, pero además se almacenaba la ruta cuando no se debía almacenar porque solo se controlaba que se estuviera en el estado 3. Para resolver este error, se añadió una variable en la que se almacena el valor del estado anterior. En caso de superar la velocidad por segunda vez, para que la ruta se almacenase tanto la variable *estado* como la variable *estadoAnterior* deben ser igual a 3.

7.4 PRUEBAS PROTOTIPO 4

Se prueba que se recupere una ruta de la base de datos y se muestre correctamente en un mapa.

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1	Se recupera una ruta de la BD	Se muestra correctamente	Se muestra correctamente	Correcto

Tabla 8 - Pruebas prototipo 4

7.5 PRUEBAS PROTOTIPO 5

Se prueba que se realice correctamente la selección de una ruta para valorar, así como la creación del cuestionario.

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1	Comprobar la configuración de la comprobación diaria	Se lanza a la hora configurada por el usuario	Se lanza a la hora configurada por el usuario	Correcto
2	Seleccionar primero rutas nuevas cuando se han realizado rutas nuevas, rutas sin valorar y rutas ya valoradas	Se selecciona una ruta realizada por primera vez	Se selecciona una ruta realizada por primera vez	Correcto
3	Seleccionar primero rutas sin valorar cuando se ha realizado rutas sin valorar y ruta ya valoradas	Se selecciona una ruta sin valorar	Se selecciona una ruta sin valorar	Correcto

Tabla 9 - Pruebas prototipo 5 (1/3)

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
4.1	Seleccionar ruta realizada en las últimas 24 horas y valorada hace un mes o más	Se selecciona una ruta valorada hace un mes	No se muestra la ruta correctamente	Incorrecto
4.2	Después de corregir el código, seleccionar ruta realizada en las últimas 24 horas y valorada hace un mes o más	Se selecciona una ruta valorada hace un mes	Se selecciona una ruta valorada hace un mes	Correcto
5	Mostrar notificación de valoración pendiente al seleccionar una ruta	Se muestra la notificación	Se muestra la notificación	Correcto
6	Cargar el cuestionario al hacer click en la notificación	Se carga la pantalla del cuestionario	Se carga la pantalla del cuestionario	Correcto
7	Cargar las categorías al hacer click en Sí	Se cargan las categorías	Se cargan las categorías	Correcto
8.1	Seleccionar otra ruta al hacer click en No y hay más rutas para valorar	Se carga de nuevo la pantalla de cuestionario con la nueva ruta	No se cargan los datos de la nueva ruta	Incorrecto
8.2	Después de corregir el código, seleccionar otra ruta al hacer click en No y hay más rutas para valorar	Se carga de nuevo la pantalla de cuestionario con la nueva ruta	Se carga de nuevo la pantalla de cuestionario con la nueva ruta	Correcto
9	Salir de la pantalla cuestionario al hacer click en No habiendo más rutas realizadas, pero no para valorar porque no ha pasado un mes desde su última valoración	Se muestra aviso y se pasa a la pantalla principal	Se muestra aviso y se pasa a la pantalla principal	Correcto

Tabla 10 - Pruebas prototipo 5 (2/3)

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
10.1	Salir de la pantalla cuestionario al hacer click en No y no hay más rutas realizadas	Se muestra aviso y se pasa a la pantalla principal	No sucede nada	Incorrecto
10.2	Después de corregir el código, salir de la pantalla cuestionario al hacer click en No y no hay más rutas para valorar	Se muestra aviso y se pasa a la pantalla principal	Se muestra aviso y se pasa a la pantalla principal	Correcto
11	Cargar las preguntas al seleccionar una categoría	Se cargan las preguntas	Se cargan las preguntas	Correcto
12	Guardar y enviar las respuestas	Se guardan y envían correctamente	Se guardan y envían correctamente	Correcto*
13	Mostrar cada hora la notificación de valoración pendiente si no se finaliza el cuestionario	Se muestra la notificación cada hora	Se muestra la notificación cada hora	Correcto

Tabla 11 - Pruebas prototipo 5 (3/3)

PRUEBA 4.1

A pesar de que en la base de datos solo se almacena una ruta la primera vez que se realiza, hasta comprobar que el algoritmo de comparación funciona correctamente al 100%, también se almacenarán las rutas copia. Por eso, a la hora de mostrar una ruta en el cuestionario, se debe mostrar la ruta copia. Según estaba realizado el código, éste mostraba la ruta original, por lo que si el algoritmo de comparación había seleccionado como igual una ruta erróneamente, no se detectaría y no podría ser eliminada de la base de datos.

Tras modificar el código, se muestra correctamente la ruta.

PRUEBA 8.1

La carga del mapa que muestra la ruta realizada estaba configurada para que se realizara solo al mostrarse la pantalla del cuestionario. Para subsanar el error, hubo que añadir una instrucción que cargase el mapa cuando se ejecutase la función *realizarCuestionario()*.

PRUEBA 10.1

A la hora de mostrar un mensaje avisando de que no había más rutas para valorar, este solo se lanzaba cuando había más rutas realizadas, pero que no se podían valorar porque no había pasado un mes desde su última valoración. Por lo tanto, cuando se eliminaban todas las rutas realizadas y no quedaban más, la aplicación no realizaba ninguna acción. Por ello hubo que añadir una condición en la que, si habiendo eliminado previamente una ruta no hay más rutas realizadas, se muestre un aviso y se pase a la pantalla principal. Se debe controlar que haya habido una eliminación previa porque, de no ser así, los días que no se realizase ninguna ruta también se mostraría un mensaje y se molestaría al usuario sin necesidad.

PRUEBA 12

Aparentemente esta acción se realiza correctamente, pero a la hora de hacer una revisión de los errores devuelto por la aplicación a través de la consola, se comprueba que se recibe un error del servidor. Aunque el identificador es enviado como un string, a la hora de usarlo para insertar datos en la base de datos remota, este es tratado como un dato de tipo double. El problema se solucionó colocando la variable entre comillas simples.

```
//Código erróneo
$consulta = "INSERT INTO `fecha_cuestionario`
(`idfecha_cuestionario`, `dia`, `hora`) VALUES
($idfecha_cuestionario, $dia, $hora)";
//Código corregido
$consulta = "INSERT INTO `fecha_cuestionario`
(`idfecha_cuestionario`, `dia`, `hora`) VALUES
('$idfecha_cuestionario', $dia, $hora)";
```

Además de esto, también se comprobó que solo se enviaba la primera respuesta almacenada debido a un error de índices. Aunque se recorría todo el array de datos, solo se leía el primero, por lo que se enviaba el mismo dato tantas veces como longitud tuviera el array. Modificando el índice se solucionó el problema.

7.6 PRUEBAS APLICACIÓN COMPLETA

Se realizan pruebas para comprobar el buen funcionamiento tras desarrollar por completo la aplicación.

Código	Descripción	Resultado esperado	Resultado obtenido	Observaciones
1	Registrar un usuario en la BD local	Se añade correctamente	Se añade correctamente	Correcto
2	Registrar un usuario en la BD remota	Se añade correctamente	Se añade correctamente	Correcto
3	Comparar dos rutas con puntos de inicio separados menos de 15 metros	Se clasifican como iguales	Se clasifican como iguales	Correcto*
4	Comparar dos rutas con puntos de inicio separados más de 15 metros	Se clasifican como distintas	Se clasifican como distintas	Correcto
5	Comparar dos rutas con diferencia de duración inferior al 40%	Se clasifican como iguales	Se clasifican como iguales	Correcto
6	Comparar dos rutas con diferencia de duración superior al 40%	Se clasifican como distintas	Se clasifican como distintas	Correcto
7.1	Comparar dos rutas con menos del 25% de los puntos distintos	Se clasifican como iguales	No ocurre nada	Incorrecto
7.2	Después de corregir el código, comparar dos rutas con menos del 25% de los puntos distintos	Se clasifican como iguales	Se clasifican como iguales	Correcto
8	Comparar dos rutas con más del 25% de los puntos distintos	Se clasifican como distintas	Se clasifican como distintas	Correcto

Tabla 12 - Pruebas aplicación completa

PRUEBA 3

Aunque aparentemente se realizaba la comparación correctamente, al comprobar los valores que se devolvían al recuperar todas las rutas originales realizadas de la base de datos, se comprueba que no se realiza correctamente. Dado que las consultas a la base de datos se realizan de modo asíncrono, todos los puntos eran almacenados dentro del array para las rutas originales como si pertenecieran a la ruta consultada en primer lugar.

Para solucionar este problema hubo que rehacer el código y utilizar la recursividad. A la función recursiva se le pasaba como parámetros el índice que es 0, el límite que es la cantidad de rutas dentro del array de rutas y el array con las rutas originales. Además se incluía la ruta que se acaba de registrar y la distancia de la ruta para su uso posterior.

Esto consiguió resolver el problema.

PRUEBA 7.1

A la hora de llamar a la función que compara una ruta con otra punto a punto, esta no se ejecutaba. Tras comprobar el código, se detectó que la llamada a la función estaba mal escrita y en realidad se volvía a llamar a la función donde se comparan el inicio y fin y la duración de las rutas.

Tras modificar la llamada a la función se resolvió el problema.

8 CONCLUSIONES Y TRABAJO FUTURO

En este apartado se tratan las conclusiones del proyecto, tanto a nivel de gestión y desarrollo, como las conclusiones finales desde un punto de vista crítico y personal.

8.1 REVISIÓN DE OBJETIVOS

Una de las principales tareas que tiene todo proyecto es definir los objetivos que se pretenden conseguir con su realización. Tras acabarlo, llega el momento de revisar si esos objetivos fijados con anterioridad han sido completados o no y, en caso afirmativo, determinar en qué medida.

CREACIÓN DE UNA APLICACIÓN MÓVIL

Este primer objetivo se puede dar claramente por conseguido ya que uno de los requisitos indispensables para la finalización de este proyecto era la creación de una aplicación móvil. Esto permitirá conocer cómo están diseñadas las ciudades y de qué manera deben cambiar para incentivar el desplazamiento a pie frente a otros medios de transporte disponibles.

APRENDIZAJE DE DESARROLLO DE APLICACIONES MÓVILES

Gracias al proyecto se ha podido ampliar la capacidad de desarrollo, ya que hasta el momento se limitaba al desarrollo para ordenadores. Esto puede marcar un punto de partida en el desarrollo de aplicaciones móviles que permita aumentar los conocimientos y las plataformas para las cuales desarrollar.

APRENDIZAJE Y FAMILIARIZACIÓN CON EL FRAMEWORK APACHE CORDOVA

Sin duda alguna el camino seguido hasta la consecución de este objetivo ha permitido al desarrollador crecer profesionalmente. Se trata de un framework que, en opinión del desarrollador, de ser usado sin la intermediación de otro framework, resulta algo complejo de usar. El punto débil es la compleja identificación de errores durante la compilación. En Internet existen numerosas comunidades, tanto de habla inglesa como de habla española, en la cual buscar una solución y una traducción a esos errores.

Quizás, el uso de un framework intermedio hubiera simplificado el desarrollo y hubiera minimizado los problemas surgidos.

DETERMINAR LA CAPACIDAD DE REALIZAR, DE MANERA AUTÓNOMA, UN PROYECTO DE PRINCIPIO A FIN

Aún con dificultades y guiado en todo momento por los directores del proyecto, se ha conseguido que el proyecto se complete. A lo largo de la carrera se ha aprendido a realizar planificaciones, pero nunca de un modo tan real como este. También se han realizado proyectos, pero no de una envergadura tan grande. Conseguir este objetivo ha supuesto un crecimiento tanto personal como profesional en el desarrollador ya que ha permitido conocer en primera persona lo que es un desarrollo real de una aplicación y algo que realizará durante su vida laboral.

8.2 PROBLEMAS DURANTE LA REALIZACIÓN DEL TFG

La realización de un proyecto como puede ser el TFG nunca está exento de problemas debido a la magnitud y extensión. Entran en juego demasiadas variables que hay que tener en cuenta y controlarlas todas es muy complicado.´

Un problema o más bien un hándicap para este proyecto ha sido la antigüedad del equipo usado durante el desarrollo. Al tratarse de un equipo portátil que ronda los 8 años no era capaz de hacer funcionar el emulador que acompaña a Android Studio. Esto supuso, que cada vez que se quería hacer cualquier comprobación, existía la necesidad de compilar el código y ejecutarlo en el móvil. A pesar de la pérdida de tiempo que esto conllevaba, no fue impedimento para realizar el proyecto.

Uno de los problemas importantes encontrados, sobre todo durante el desarrollo, es la aparente inestabilidad de los proyectos desarrollados con Apache Cordova. El motivo concreto no se sabe cuál es, quizás una mala configuración inicial, aunque lo más probable puede ser problemas de dependencias entre plugins. En varias ocasiones, tras añadir un plugin necesario, a la hora de compilar el código para crear un instalador y poderlo probar en el dispositivo, la compilación fallaba.

Aprovechando la ocasión, otro problema encontrado en Apache Cordova es la dificultad encontrada para leer los mensajes de error derivados de la compilación. No eran mensajes claros, lo que provocó que varias veces hubiera que crear un proyecto Apache Cordova desde cero, añadir los plugins necesarios y copiar el código creado hasta el momento. Aunque se disponía de un repositorio de código y se hacía uso de él para retroceder hasta una versión anterior, esto no servía ya que al instalar de nuevo el plugin el error continuaba y hubo que tomar la decisión radical porque el plugin era necesario para continuar con el desarrollo y la única opción era crear un proyecto nuevo.

En Internet existen frameworks que prometen un desarrollo sencillo basado en Apache Cordova. Entre los disponibles, se encuentra con licencia gratuita el framework Ionic[46], pero también los hay de licencia de pago como puede ser Telerik[47]. Debido al poco conocimiento relacionado con la plataforma Apache Cordova, se tomó la decisión de desarrollar sin utilizar ningún framework, lo que ha podido complicar el desarrollo, pero no impedirlo.

Otro problema importante ocurrido durante el desarrollo y pruebas del proyecto fue provocado por el plugin de geolocalización. Como se comenta en 6.4.1 La geolocalización, este plugin cuenta con dos tipos de seguimiento. Los problemas encontrados fueron que, por ejemplo, cuando se usaba el modo *distanceFilter* en muchas ocasiones el tiempo de recepción de punto era demasiado elevado, lo que provocaba que la ruta se diera por finalizada. A menudo esto ocurría cuando no se había superado la distancia mínima de recorrido.

El modo *fusedLocation* tampoco estuvo exento de problemas. En este caso, los puntos eran devueltos según estaban configurados, pero sin devolver datos referentes a la velocidad. Esto provocaba que en ningún momento se pasase del estado de **NoRuta**. La decisión que se tomó fue calcular la velocidad usando su fórmula, donde la velocidad es igual a la distancia recorrida entre el tiempo empleado. Tras esto se descubrió que, debido a fallos ajenos al código desarrollado, en ocasiones la precisión fuese elevada lo que provocaba que los puntos recibidos fuesen distantes de los anteriores y la velocidad aumentase considerablemente. Basándose en las rutas anteriormente creadas, se estableció un filtro con el que, si la precisión con la que el punto había sido recogido era superior, el punto no era tenido en cuenta. Esto permitió también crear rutas lo más precisas y reales posibles.

Tras realizar pruebas con ambos modos, se determinó usar el modo *fusedLocation* ya que, además de realizar un seguimiento más detallado de las rutas, permitía un ahorro superior de batería.

Tanto este plugin, como los otros usados, son plugins que están en continuo desarrollo por lo que no están exentos de problemas. Gracias a su presencia en Github y al desarrollo colaborativo típico en esa plataforma, los fallos suelen ser solventados con relativa rapidez.

Otro de los problemas surgidos, y que se comentarán más adelante, fue la falta de dedicación al proyecto, generando un retraso general de la planificación realizada al inicio. Al no haber realizado un proyecto de esta magnitud hasta el momento, se pensaba que era algo sencillo que se acabaría en poco tiempo. Claramente esto no fue así, y antes de que el problema fuese a mayores, el desarrollador decidió implicarse y sacar a adelante el proyecto.

[46] http://ionicframework.com/
[47] http://www.telerik.com/platform/appbuilder

8.3 CONCLUSIONES EN LA GESTIÓN: PLANIFICACIÓN, RIESGOS Y COSTES

Como se comenta en el apartado anterior, la planificación inicial no se ha cumplido. Además de por la falta de dedicación, los problemas surgidos y una planificación temporal escasa han provocado que el proyecto se extienda en el tiempo.

En un principio, la fecha de entrega se estimó para la convocatoria de febrero de 2016. Al ver que era se trataba de algo imposible, se aplazó hasta la convocatoria de abril/ mayo 2016, algo que no se cumplió ya que aún faltaban tareas por realizar. Finalmente, fue posible la entrega para la convocatoria de junio/julio 2016.

Por lo tanto, unos de los riesgos tenidos en cuenta y temidos, la planificación incorrecta, se ha cumplido. Las consecuencias únicamente han sido la demora en la entrega del proyecto.

A continuación se muestra el diagrama de Gantt real. En él se han intentado plasmar del modo más fiel posible la nueva planificación temporal, así como la demora producida por los motivos citados al principio.

Si se analiza el diagrama, se puede ver que desde el inicio se empieza a retrasar todo el proyecto. En la planificación inicial se tuvo en cuenta que iba a existir un retraso debido a la época estival, pero finalmente el retraso comenzó antes. Esto, sumado a los continuos retrasos entre las tareas y a una mala planificación temporal, dio como resultado a los aplazamientos en las entregas comentados anteriormente.

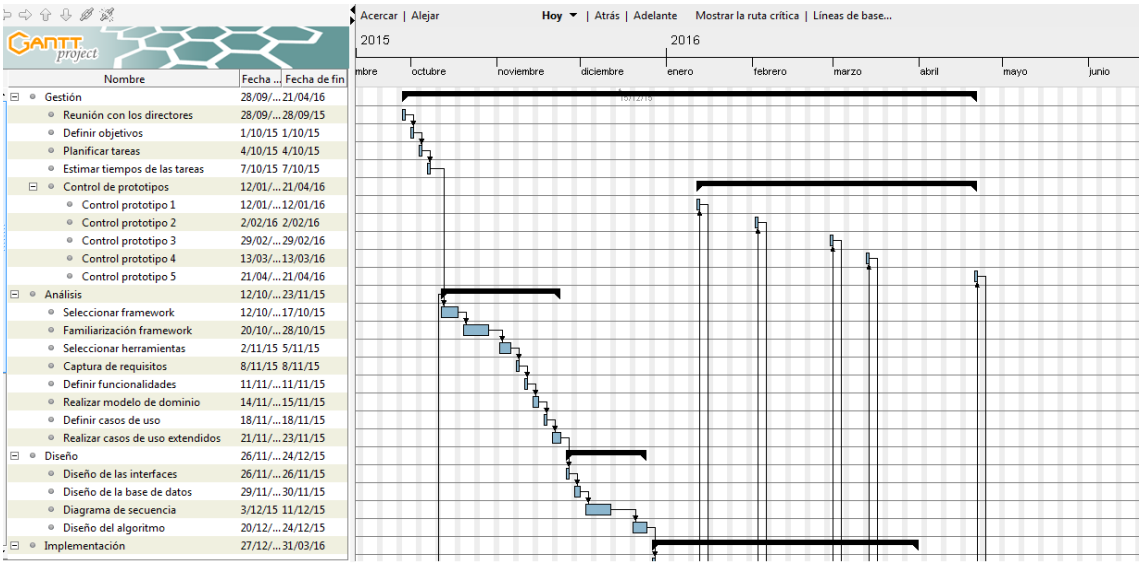

Ilustración 16 - Diagrama Gantt Real (1/2)

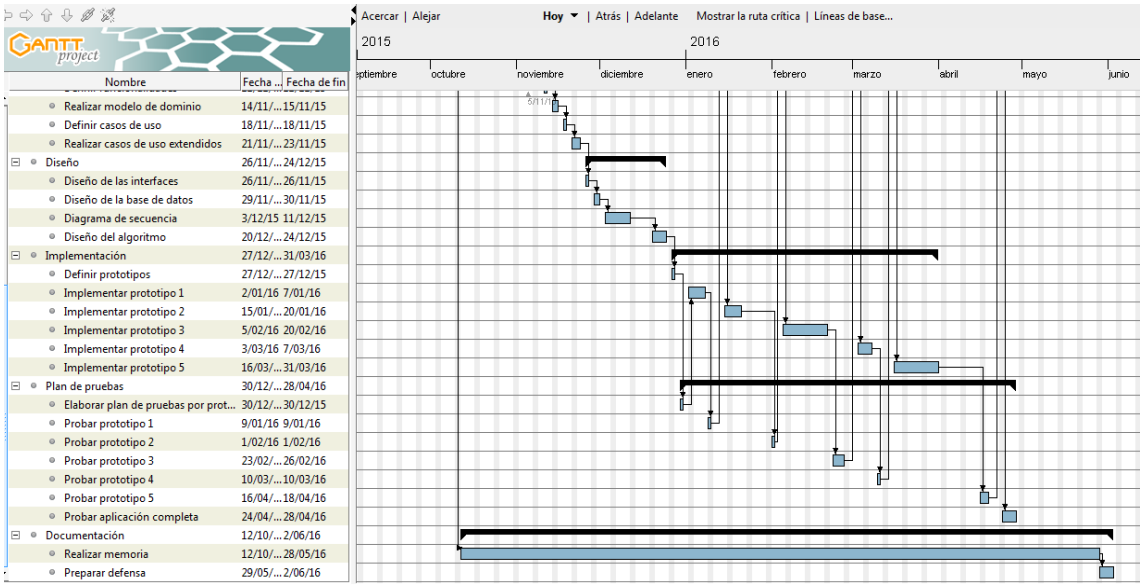

Ilustración 17 - Diagrama Gantt Real (2/2)

Si se compara la dedicación estimada en 2.3 Planificación temporal con la dedicación que ha sido necesaria, se puede ver que la estimación ha sido escasa. Aproximadamente ha habido que dedicar algo menos del doble de tiempo estimado, como se muestra en Ilustración 18 - Gráfico comparativa dedicación estimada vs real y en Tabla 13 - Comparativa dedicación estimada vs. real.

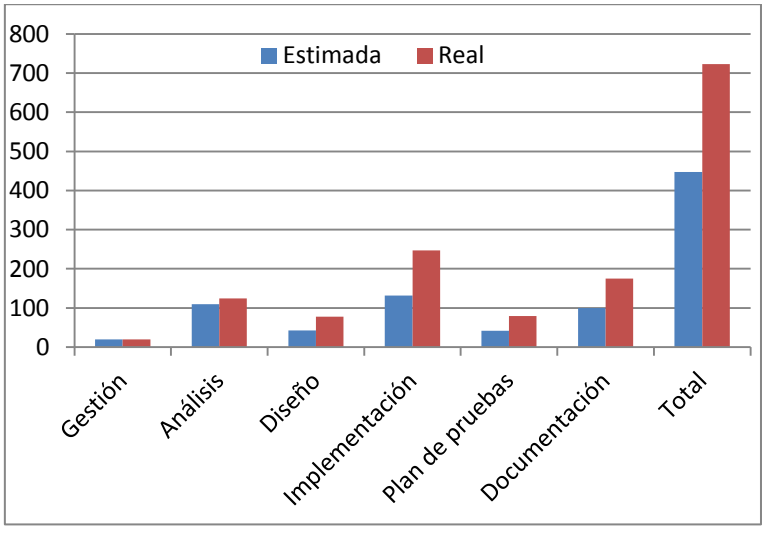

Ilustración 18 - Gráfico comparativa dedicación estimada vs real

Tareas	Estimada	Real
Gestión	**20**	**20**
Análisis	**110**	**124**
Diseño	**43**	**78**
Implementación	**132**	**247**
Plan de pruebas	**42**	**79**
Documentación	**100**	**175**
Total	**447**	**723**

Tabla 13 - Comparativa dedicación estimada vs. real

Como se puede comprobar en el gráfico y la tabla anterior, la necesidad de aumentar el tiempo de dedicación ha sido general, exceptuando en las dos primeras tareas. Para la tarea de diseño se necesito más tiempo para realizar los diagramas de secuencia y el algoritmo, debido a la extensión de la secuencia y al rediseño tras realizar las pruebas y encontrar errores.

En la tarea de implementación, la dedicación aumentó en todas las subtareas porque el tiempo estimado era insuficiente, además de por los problemas comentados a la hora de añadir plugins nuevos.

Para realizar las pruebas también se necesitó más tiempo al haber estimado menos horas de las realmente necesarias para realizar todas las planificadas.

En último lugar, para realizar la documentación también se duplicaron las horas dedicadas porque la estimación resultó ser escasa.

Teniendo en cuenta la evaluación económica realizada en 2.7 Evaluación económica, hay que recalcular todos los costes con los datos de las horas reales aproximadas. De las 447 horas estimadas se pasa a 723 horas reales, 276 horas más. Esto supone un aumento de 8280€ más de los 13410€ previstos en el coste del personal, arrojando un total de 21690€. Al aumentar las horas, hay que calcular las nuevas amortizaciones del ordenador portátil (20,66€) y del dispositivo móvil (6,06€), aumentando el coste total.

Gastos		
	Estimados	Reales
Personal	13410€	21690€
Hardware	16,54€	26,72€
Software	0€	0€
Total	13426,54	21716,72€

Tabla 14 - Comparativa gastos estimados vs. Reales

Estos datos simplemente son orientativos porque, como se comenta en el apartado 2.7 Evaluación económica, no se busca un beneficio económico al desarrollar la aplicación, sino aportar algo que beneficie a toda la sociedad.

8.4 TRABAJO FUTURO

A continuación se presenta un listado con tareas de cara al futuro que permitan conseguir una aplicación mejor y que no se quede obsoleta.

APLICACIÓN COMO SERVICIO

A la entrega del proyecto, la aplicación depende del usuario para su funcionamiento. Si por ejemplo se reinicia el móvil o elimina la aplicación del apartado de aplicaciones recientes, esta se detiene. Un gran aporte sería crear un servicio que permitiera a la aplicación seguir funcionando sin depender del usuario. Esta dependencia implica que haya momentos en los que el usuario no sea consciente de que no tiene activa la aplicación, por lo que se pierde información. Ya se trata de un proyecto donde la interacción del usuario es limitada para su funcionamiento y de esta manera se limitaría aún más. Realizar esta mejora resultaría un beneficio tanto para el usuario porque no se tiene que preocupar de activar la aplicación como para el grupo investigador porque existiría una menor pérdida de datos.

GEOLOCALIZACIÓN OPTIMIZADA

Como se comenta en 6.4.1 La geolocalización, para realizar evitar puntos imprecisos se implementó un filtro que solo permitía tratar aquellos puntos con una precisión concreta. Esta criba de puntos, implica la pérdida de datos sobre la ruta que en ocasiones quizás sean necesarios. Es por esto que una mejora de cara al futuro sería la optimización de la geolocalización para así capturar la mayor información posible sobre la ruta.

AMPLIACIÓN A iOS

Como se comenta en 6.2 El lenguaje de la aplicación, aunque Apache Cordova permite un desarrollo multiplataforma, solo se ha desarrollado y probado para Android debido a la falta de recursos necesarios para desarrollar también para iOS. Por tanto, un trabajo futuro sería la verificación y corrección de la aplicación para que también esté disponible para dispositivos iOS ya que, aunque se ha tenido cuidado de elegir plugins y tecnologías válidas para ambas plataformas, es posible que exista algún error o incompatibilidad.

MEJORAS EN EL DISEÑO DE LA INTERFAZ

Otra mejora que se podría añadir sería la mejora de la interfaz gráfica de la aplicación así como la personalización de los iconos. Aunque se ha intentado que todo apareciese correctamente, no ha sido posible pulir al 100% los aspectos gráficos ya que es un aspecto que se consideró secundario.

REVISIÓN PERIÓDICA DE LOS PLUGINS

Al tratarse de plugins desarrollados por terceros, estos se encuentran en continuo desarrollo. Sería recomendable una revisión periódica para estar al tanto de las mejoras y corrección de errores que se vayan realizando a los mismos. También permitiría el cambio del modo de seguimiento en caso de que se obtengan mejoras.

8.5 CONCLUSIONES DESDE UN PUNTO DE VISTA CRÍTICO Y PERSONAL

Llegó la hora de echar la vista a atrás y ver todo el camino recorrido. Ha sido un camino difícil, lleno de dificultades y de alegrías personales. Elegir Apache Cordova frente a las otras opciones me ha permitido realizar un auto-aprendizaje en el que he tenido que recurrir en numerosas ocasiones a buscar ayudas y manuales en Internet para poder solucionar los problemas que van surgiendo. La verdad que es una satisfacción enorme llegar a este punto donde ves que todo el trabajo realizado ha merecido la pena.

Al no haber realizado antes un proyecto como este, no sabía lo que implicaba. Pensaba que esto en dos ratos que le dedicase iba a estar hecho por lo que lo iba dejando y dejando hasta darme cuenta que, o me ponía a ello o esto nunca tendría fin. También había momentos en los que veía todo lo que me quedaba por delante, lo que hacía que me desmotivase y perdiese el interés. Poco a poco y viendo que todo iba funcionando, conseguí dar el empujón necesario para ser capaz de acabar lo que empecé

Me ha permitido crecer como persona porque he sido consciente de mis límites, de ver la capacidad que tengo para solucionar problemas y de aprender por mi mismo. También profesionalmente porque el auto-aprendizaje se basa en ir encontrando errores y solucionarlos sin ayudas y pienso que es de la manera que más se aprende.

El investigar y profundizar en las tecnologías web me ha hecho que surja en mí un interés que hasta el momento no existía y que puede abrirme puertas en mi futuro laboral.

9 BIBLIOGRAFÍA

ApacheCordova.org. (12 de Junio de 2014). *Apache Cordova.* Recuperado el 12 de Abril de 2016, de https://cordova.apache.org/docs/en/latest/guide/overview/index.html#plugins

Arana Roa, B. (22 de Agosto de 2013). *Ingeniería del Software.* Recuperado el 11 de Noviembre de 2015, de http://isescom.blogspot.com.es/2013/08/desarrollo-en-cascada-vs-desarrollo.html: http://3.bp.blogspot.com/-fwY-x5e5GTg/Uhad3GFsC0I/AAAAAAAAAB0/-t1nr3Eu45s/s1600/FasesIterativoIncremental.jpg

Arsys.es. (4 de Abril de 2012). *Blog de Arsys.* Recuperado el 21 de Marzo de 2016, de http://blog.arsys.es/myisam-o-innodb-elige-tu-motor-de-almacenamiento-mysql/

Cervero, R., & Kockelman, K. (1997). Travel demand and the 3Ds: Density, diversity, and design. En R. Cervero, & K. Kockelman, *Transportation Research Part D: Transport and Environment, 2 (3)* (págs. 199-219). Elsevier.

Cossío, M. T. (16 de Enero de 2013). *La ciudad viva.* Recuperado el 3 de Febrero de 2016, de http://www.laciudadviva.org/blogs/?p=15737

Desarrolloweb.com. (19 de Septiembre de 2012). *Desarrolloweb.com.* Recuperado el 23 de Mayo de 2016, de http://www.desarrolloweb.com/manuales/manual-jquery.html

Ecomotriz.com. (22 de Junio de 2013). *Ecomotriz.com*. Recuperado el 8 de Febrero de 2016, de Noticias sobre movilidad sostenible: http://www.ecomotriz.com/2011/10/movilidad-urbana-1/

Endomondo.com. (26 de Julio de 2012). *Endomondo - Correr & Ciclismo.* Recuperado el 9 de Febrero de 2016, de https://play.google.com/store/apps/details?id=com.endomondo.android&hl=es: https://lh3.googleusercontent.com/0I31xk78EV_jftzPp7Wcki-5EEQ2CNUFRpfdbrPQxxoknX93Zki1hrOLpJQ1leqMLP5d=h900-rw

García Gallo, B. (25 de Diciembre de 2015). *El País*. Recuperado el 3 de Febrero de 2016, de http://ccaa.elpais.com/ccaa/2015/12/25/madrid/1451041531_535829.html

Miembros del W3C. (30 de Junio de 2001). *HTML con clase.* Recuperado el 1 de Junio de 2016, de http://html.conclase.net/w3c/dom1-es/introduction.html

Morales, M. (18 de Mayo de 2015). *ComputerHoy.com*. Recuperado el 3 de Febrero de 2016, de http://computerhoy.com/noticias/life/espana-cabeza-del-uso-del-movil-europa-creciendo-28595

Peña, A. (20 de Noviembre de 2014). *iSocialWeb*. Recuperado el 11 de Noviembre de 2015, de http://www.isocialweb.es/por-que-tu-empresa-necesita-una-app-o-aplicacion-movil/

Puente Cedillo, O. M. (6 de Agosto de 2014). *Programación Web.* Recuperado el 26 de Abril de 2016, de Instituto Tecnólogico de Matehuala: https://programacionwebisc.wordpress.com/2-1-arquitectura-de-las-aplicaciones-web/

Román, J. (21 de Mayo de 2014). *Emezeta.com*. Recuperado el 3 de Enero de 2016, de http://www.emezeta.com/articulos/los-mejores-editores-de-texto-para-programar

Runtastic.com. (23 de Noviembre de 2012). *Runtastic Running y Fitness.* Recuperado el 9 de Febrero de 2016, de https://play.google.com/store/apps/details?id=com.runtastic.android&hl=es: https://lh3.googleusercontent.com/5nr8tnPvgK4maIafYOztJD40bM9lrQrFPjFA-S0GpCDWDUY20jjRMmmP9-KbCNd0F0Vm=h900-rw

San-Juan, C., Vozmediano, L., & Vergara, A. (2012). Self-protective behaviours against crime in urban settings: an empirical approach to vulnerability and victimization models. En C. San-Juan, L. Vozmediano, & A. Vergara, *European Journal of Criminology, 9 (6)* (págs. 652-657).

ANEXO I – CASOS DE USO EXTENDIDOS

En el anexo I se van a detallar los casos de uso anteriormente nombrados, explicando de forma más detallada las diferentes funcionalidades que se quieren implementar.

A continuación, se representa la interfaz inicial (Ilustración 19 - Interfaz inicial) de la aplicación, cuando se inicia por primera vez la app en el terminal. Se muestra el texto introductorio a la app y al estudio.

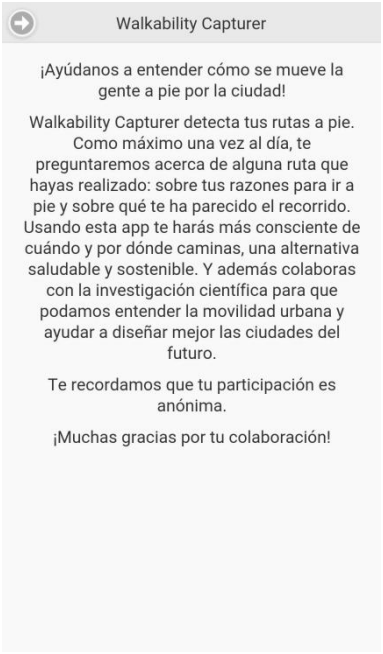

Ilustración 19 - Interfaz inicial

INICIAR APLICACIÓN

Nombre: Iniciar aplicación.

Descripción: La aplicación es lanzada por el usuario y es inicializada.

Actores: Usuario

Precondiciones: Ninguna.

Requisitos no funcionales: Conexión a Internet activada.

Flujo de eventos:

1. El usuario lanza la aplicación.
2. Se abre la base de datos.
3. Se comprueba que exista un esquema de tablas.
   3.1. Si no existe, se crea.
4. Se comprueba que exista configuración.
   4.1. Si existe, se comprueba que sea la versión activa.
      4.1.1. Si no es la activa, se descarga y almacena.
   4.2. Si no existe, se solicita la última versión.
      4.2.1. Si surge un error, se muestra un aviso (Ilustración 20 - Error solicitud configuración) y se reintenta.
         4.2.1.1. Si es el segundo intento se muestra un aviso (Ilustración 21 - Error conexión) y se impide el uso de la app.
      4.2.2. Si no, se descarga y almacena.
5. Se comprueba que exista el usuario.
   5.1. Si existe, se cargan los datos de configuración, se programa la comprobación diaria y se envían los datos pendientes al servidor que no han podido ser enviados en la última comprobación diaria.
      5.1.1. Se muestra la pantalla principal (Ilustración 22 - Interfaz principal) y se inicia el seguimiento.
   5.2. Si no, se inicia el caso de uso Configuración inicial.

Postcondiciones: Si el usuario existe, se ha iniciado el seguimiento. Si no, se ha iniciado la configuración inicial.

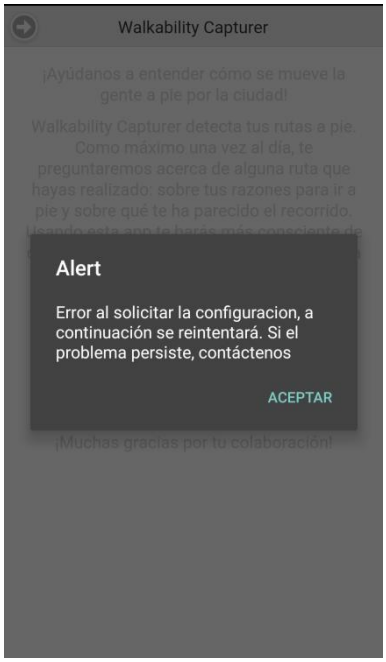

Ilustración 20 - Error solicitud configuración

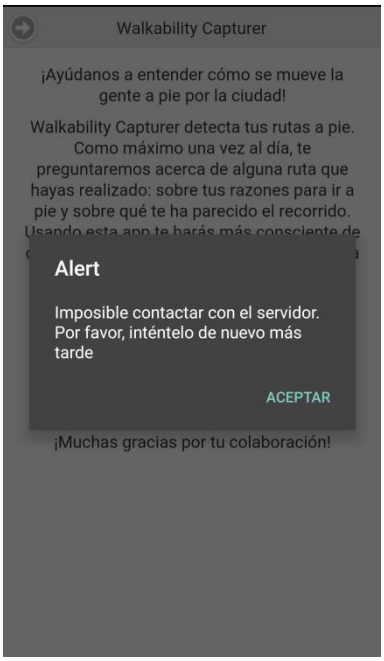

Ilustración 21 - Error conexión

Ilustración 22 - Interfaz principal

CONFIGURACIÓN INICIAL

Nombre: Configuración inicial.

Descripción: Permite al usuario registrarse en el sistema para poder comenzar a usar la aplicación.

Actores: Usuario.

Precondiciones: No existe un usuario registrado.

Requisitos no funcionales: Conexión a Internet activada.

Flujo de eventos:

1. El usuario pulsa el botón de la esquina izquierda para pasar al test inicial (Ilustración 23 - Botón interfaz inicial).
2. El usuario seleccionará los datos referentes a su persona de los apartados disponibles. Algunos, como puede ser el apartado de género, vienen marcados con una opción por defecto (Ilustración 24 - Test Inicial).
   2.1. Si pulsa el botón de la esquina izquierda habiendo dejado algún campo sin seleccionar, se muestra un aviso (Ilustración 25 - Aviso con opciones sin seleccionar) indicando el primero no seleccionado.
   2.2. En el apartado donde se selecciona la procedencia, si selecciona el checkbox se muestra un área de texto donde poder insertar el municipio (Ilustración 26 - Checkbox activo).
   2.3. Rellena todos los campos y pulsa el botón de la esquina izquierda.
      2.3.1. Si no se ha seleccionado el año, se muestra un aviso (Ilustración 27 - Aviso año no seleccionado).
      2.3.2. A la hora de comprobar el municipio de procedencia.
         2.3.2.1. Si está activo el checkbox.
            2.3.2.1.1. Si está vacío, se muestra un aviso (Ilustración 28 - Aviso TextArea vacío).
            2.3.2.1.2. Si no tiene solo letras, se muestra un aviso (Ilustración 29 - Aviso solo letras).
         2.3.2.2. Si no está activo el checkbox.
            2.3.2.2.1. Si no se ha seleccionado una provincia, se muestra un error (Ilustración 30 - Aviso seleccionar provincia).
            2.3.2.2.2. Si no se ha seleccionado un municipio, se muestra un error (Ilustración 31 - Aviso seleccionar municipio).
      2.3.3. Si no hay errores, se muestran los datos recogidos al usuario para su confirmación (Ilustración 32 - Confirmación datos).
      2.3.4. Si los confirma, se almacena en la base de datos local y se envía al servidor. Se muestra una ventana dando la bienvenida (Ilustración 33 - Usuario creado).
         2.3.4.1. Se cargan los datos de configuración y se programa la comprobación diaria.

2.3.4.2. Se muestra la pantalla principal y se inicia el seguimiento.

2.3.5. Si no, se permanece en la pantalla actual para modificar los datos necesarios.

Postcondiciones: El usuario estará insertado tanto en la base de datos local como en la base de datos remota. La aplicación redirige automáticamente a la interfaz principal (Ilustración 34 - Interfaz principal). A partir de ahora, cada vez que se inicie la app, se mostrará esta pantalla. Se inicia automáticamente el seguimiento.

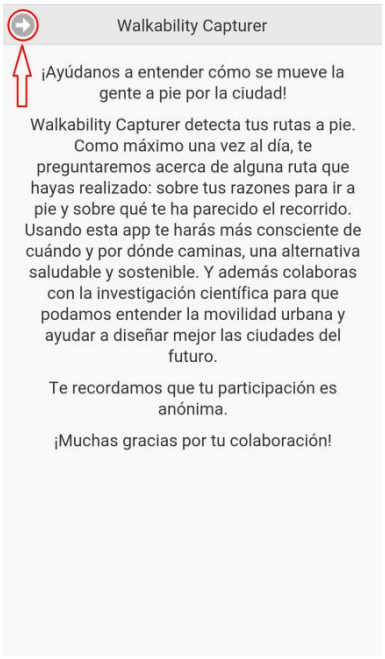

Ilustración 23 - Botón interfaz inicial

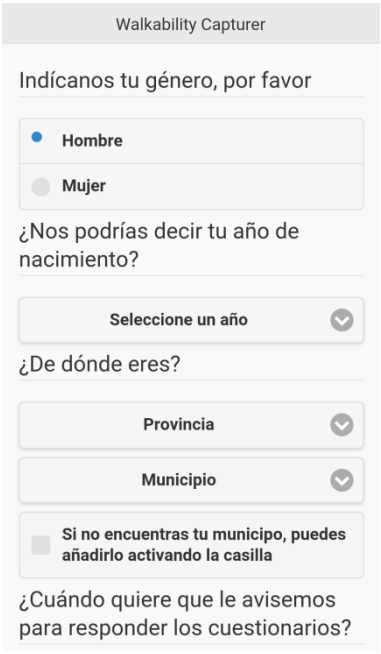

Ilustración 24 - Test Inicial

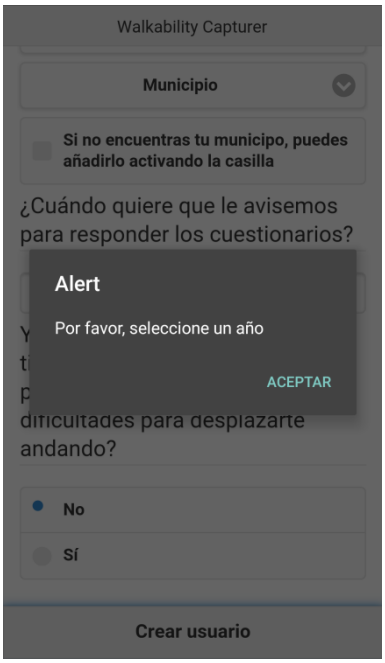

Ilustración 25 - Aviso con opciones sin seleccionar

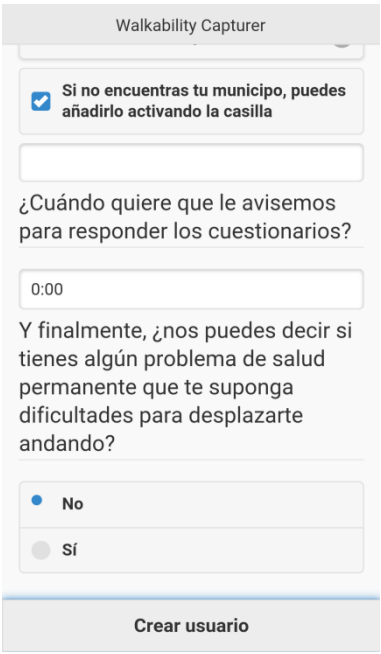

Ilustración 26 - Checkbox activo

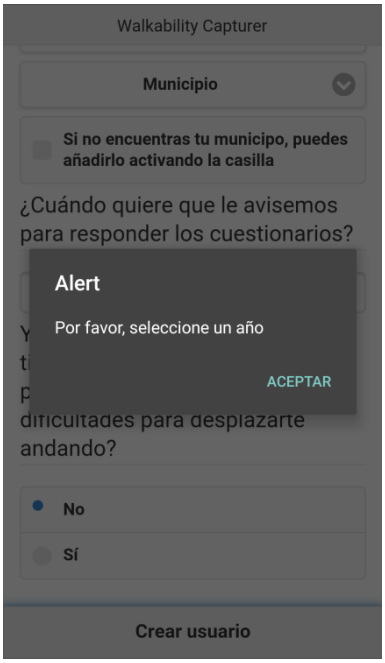

Ilustración 27 - Aviso año no seleccionado

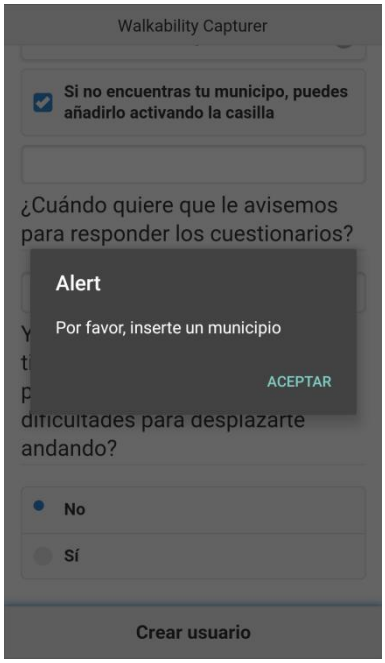

Ilustración 28 - Aviso TextArea vacío

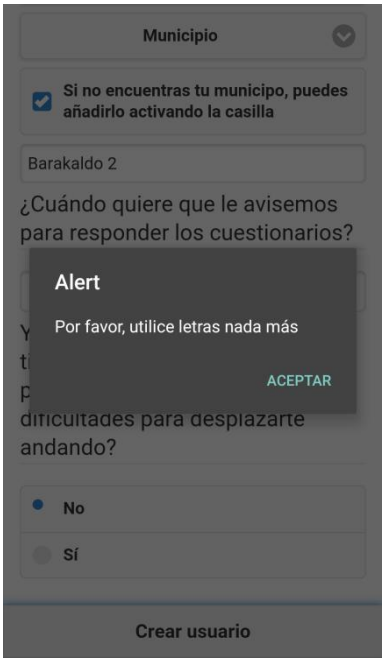

Ilustración 29 - Aviso solo letras

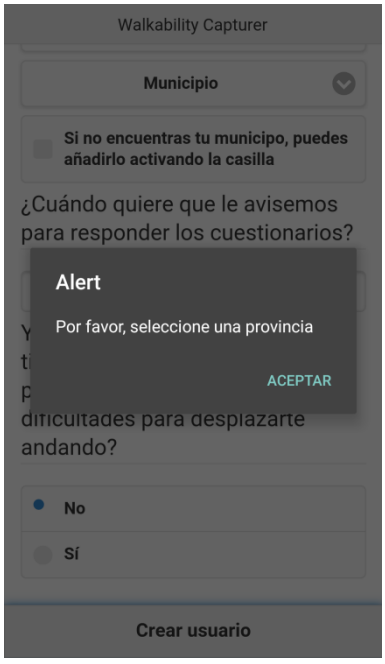

Ilustración 30 - Aviso seleccionar provincia

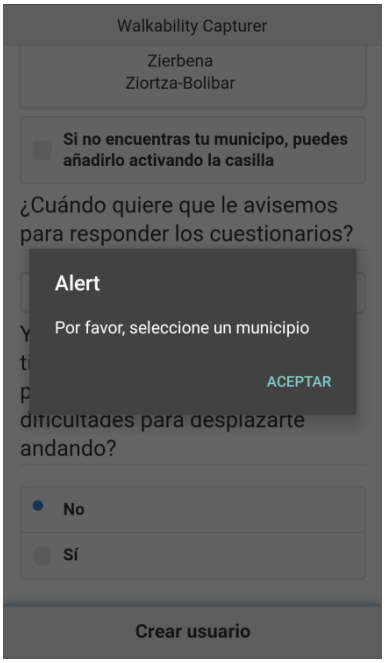

Ilustración 31 - Aviso seleccionar municipio

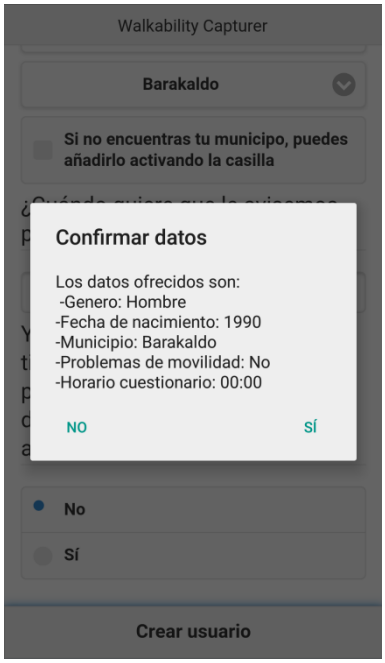

Ilustración 32 - Confirmación datos

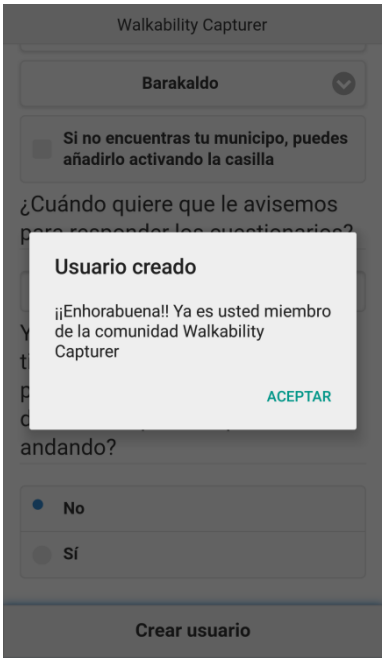

Ilustración 33 - Usuario creado

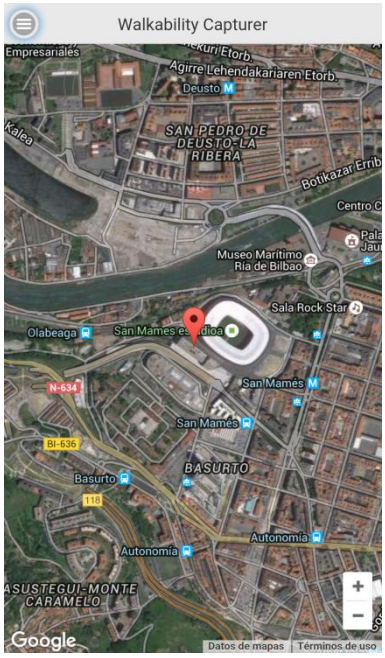

Ilustración 34 - Interfaz principal

CONFIGURAR SEGUIMIENTO

Nombre: Configurar seguimiento

Descripción: Permite al usuario detener el seguimiento de la aplicación para, por ejemplo, ahorrar batería. Si el seguimiento está detenido, permite ponerlo en funcionamiento.

Actores: Usuario.

Precondiciones: Ya hay un usuario registrado en el sistema

Requisitos no funcionales: Conexión a Internet y GPS activados.

Flujo de eventos:

1. El usuario hace click en el botón superior izquierdo (Ilustración 35 - Principal botón superior).
2. Se muestra el panel donde se encuentran los distintos apartados de la app.
   2.1. Si el seguimiento está activo, al hacer click en "Parar Geo." (Ilustración 36 - Parar Geo.) la app detendrá el seguimiento.
      2.1.1. Si se encuentra en el estado 3, se almacena la ruta (Comparar y almacenar ruta).
   2.2. Si el seguimiento está parado, al hacer click en "Iniciar Geo." (Ilustración 37 - Iniciar Geo.) la app comenzará el seguimiento.

Postcondiciones: Ninguna.

Ilustración 35 - Principal botón superior

Ilustración 36 - Parar Geo.

Ilustración 37 - Iniciar Geo.

CONFIGURAR COMPROBACIÓN DIARIA

Nombre: Configurar comprobación diaria.

Descripción: Permite al usuario modificar la hora en la que el sistema comprobará si ha realizado alguna ruta en las últimas 24 horas. Esta hora ha sido establecida al iniciar la aplicación.

Actores: Usuario.

Precondiciones: Existe un usuario registrado y una hora previa almacenada en la base de datos local.

Requisitos funcionales: Ninguno.

Flujo de eventos:

1. El usuario hace click en el botón superior izquierdo (Ilustración 38 - Principal botón superior).
2. Se muestra el panel donde se encuentran los distintos apartados de la app.
3. El usuario hace click en "Configuración" (Ilustración 39 - Configuración).
4. Se muestra la pantalla de configuración con la hora previamente definida (Ilustración 40 - Pantalla configuración).
   4.1. El usuario hace click en "Cambiar horario" (Ilustración 41 - Botón cambiar horario).
   4.2. Se muestra un campo Time donde modificar la hora y un botón "Guardar horario" con el que confirmar el nuevo horario (Ilustración 42 - Campo modificar hora).
   4.3. Se selecciona un nuevo valor con el campo Time.
      4.3.1. Si el usuario pulsa "Guardar horario" (Ilustración 43 - Botón guardar horario).
         4.3.1.1. Si el campo está vacío, se muestra un aviso (Ilustración 44 - Error hora vacía).
         4.3.1.2. Si el campo tiene un valor correcto, se muestra un aviso de que ha sido correctamente modificado (Ilustración 45 - Hora actualizada).

Postcondiciones: La hora a la que se debe realizar la comprobación diaria ha sido modificada.

Ilustración 38 - Principal botón superior

Ilustración 39 - Configuración

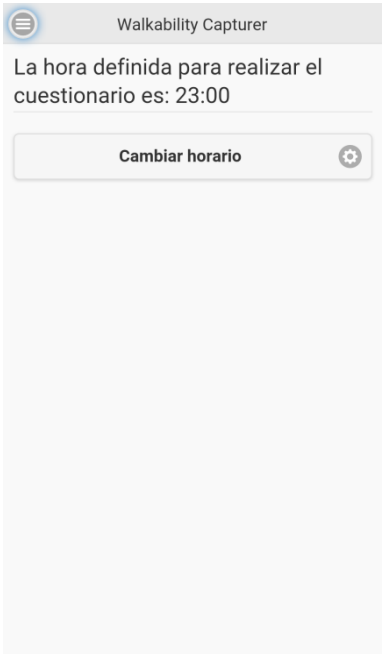

Ilustración 40 - Pantalla configuración

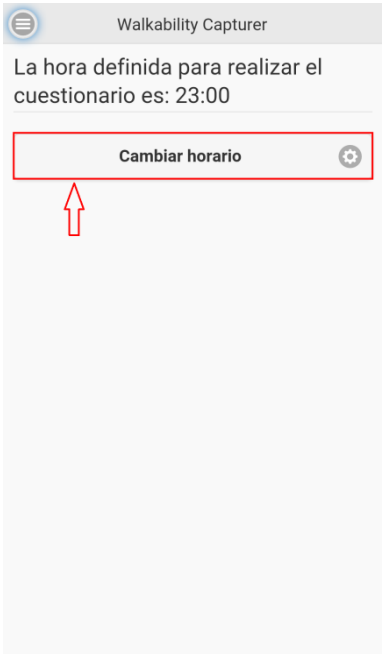

Ilustración 41 - Botón cambiar horario

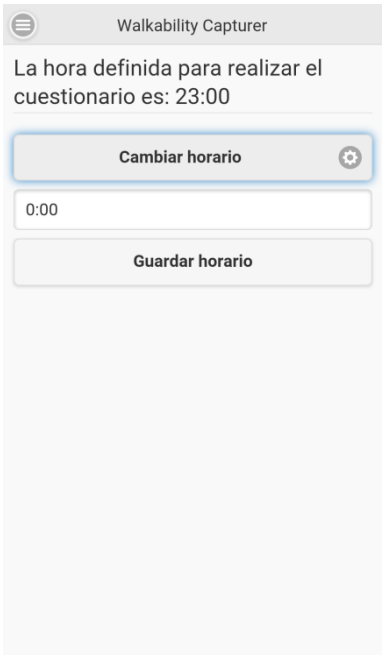

Ilustración 42 - Campo modificar hora

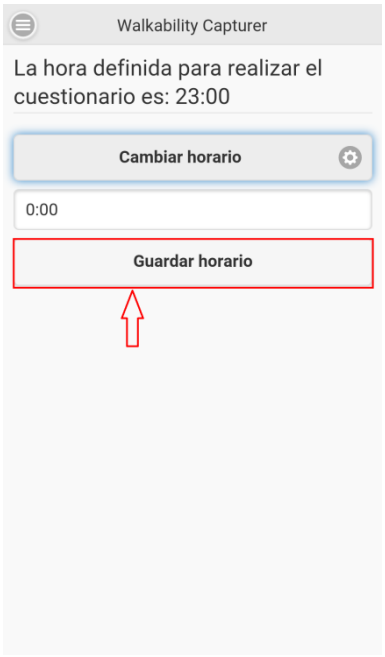

Ilustración 43 - Botón guardar horario

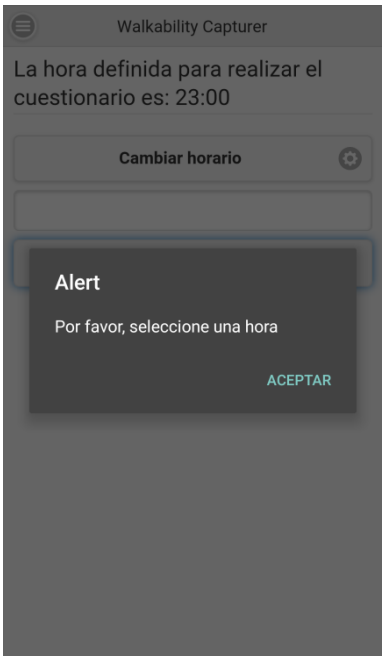

Ilustración 44 - Error hora vacía

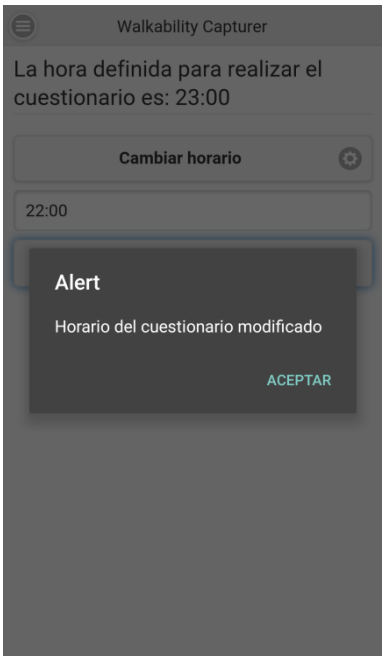

Ilustración 45 - Hora actualizada

REGISTRAR RUTA

Nombre: Registrar ruta.

Descripción: Mediante la geolocalización, se controla la posición del usuario y gracias al algoritmo diseñado, el sistema decide si la ruta realizada debe considerarse como ruta o no.

Actores: Sistema.

Precondiciones: El seguimiento está iniciado.

Requisitos no funcionales: Servicio de localización activado.

Flujo de eventos:

1. El usuario se mueve.
2. Se recoge un punto nuevo.
   2.1. Si la precisión es inferior a 55, el punto se tiene en cuenta.
      2.1.1. Se detiene el temporizador de espera de punto.
      2.1.2. Se calcula la velocidad teniendo en cuenta el punto recogido y el anterior.
      2.1.3. Si no hay punto anterior, la velocidad es cero.
      2.1.4. Si el estado es 1, se detiene el temporizador de parada si está activo y se comprueba la velocidad.
         2.1.4.1. Si la velocidad es superior a cero e inferior a la máxima permitida, se guarda el punto como inicio y se pasa al estado 2. Se activa el temporizador de espera de punto.
      2.1.5. Si no, se comprueba la velocidad.
         2.1.5.1. Si la velocidad es superior a la máxima permitida, se comprueba si está activo el temporizador de parada.
            2.1.5.1.1. Si está activo, se comprueba si se ha alejado la distancia mínima del punto de parada.
               2.1.5.1.1.1. Si se ha superado, se para el temporizador de parada. También, se contabiliza y se comprueba cuántas veces se ha superado la velocidad máxima.
                  2.1.5.1.1.1.1. Si se ha superado una vez, se almacena el punto y se activa el temporizador de espera de punto..
                     2.1.5.1.1.1.1.1. Si el estado es 2 y se superado la distancia mínima de ruta, se pasa al estado 3.
                  2.1.5.1.1.1.2. Si se ha superado dos veces, se vuelve al estado 1. Además, si el estado era 3 al recoger el punto actual y cuando se ha recogido el punto anterior también era 3, se elimina el último punto recogido y se almacena la ruta (Comparar y almacenar ruta).
               2.1.5.1.1.2. Si no, se activa el temporizador de espera de punto.
            2.1.5.1.2. Si no está activo, se contabiliza y se comprueba cuántas veces se ha superado la velocidad máxima.

2.1.5.1.2.1. Seguir 2.1.5.1.1.1.1

2.1.5.2. Si la velocidad es inferior a 1 y el temporizador de parada no está activo, se activa. Además se almacena el punto y se toma como punto de referencia de parada. Se activa el temporizador de espera de punto.

2.1.5.3. Si la velocidad está entre los valores permitidos, se comprueba si está activo el temporizador de parada.

2.1.5.3.1. Si está activo, se comprueba si se ha alejado la distancia mínima del punto de parada.

2.1.5.3.1.1. Si se ha superado, se para el temporizador de parada y se almacena el punto. Se activa el temporizador de espera de punto.

2.1.5.3.1.2. Si no, se activa el temporizador de espera de punto.

2.1.5.3.2. Si no, se almacena el punto.

3. Si se cumple el tiempo del temporizador de espera de punto, se detiene el temporizador de parada si está activo y se pasa al estado 1.
   3.1. Si el estado era 3, se almacena la ruta (Comparar y almacenar ruta).
4. Si se cumple el tiempo del temporizador de parada, se detiene el temporizador de espera de punto y se pasa al estado 1.
   4.1. Si el estado era 3, se almacena la ruta (Comparar y almacenar ruta).

Postcondiciones: En caso de haber llegado al estado 3, se obtiene una ruta que será almacenada en la base de datos después de ser trata para saber si ha sido o no realizada anteriormente en el subcaso de uso Comparar y almacenar ruta.

5.1. COMPARAR Y ALMACENAR RUTA

Nombre: Comparar y almacenar ruta.

Descripción: Subcaso de uso donde se compara la ruta registrada con las almacenadas para determinar si es la primera vez que se hace o no. Tras la comparación, la ruta es almacenada en la base de datos.

Actores: Sistema.

Precondiciones: Se ha registrado una ruta en el caso de uso Registrar ruta.

Requisitos no funcionales: Conexión a Internet activada.

Flujo de eventos:

1. Se comprueba si hay rutas originales almacenadas.
   1.1. Si hay, se comparan con la registrada.
      1.1.1. Por cada ruta almacenada, se compara el inicio y el fin.
         1.1.1.1. Si ambas tienen mismo inicio y mismo fin (la distancia entre puntos de inicio y entre puntos de fin es menor a la máxima permitida), se compara la duración.
            1.1.1.1.1. Si la diferencia en la duración entre ambas es menor al máximo permitido, se compara la cantidad de puntos.
               1.1.1.1.1.3. Si la ruta registrada es mayor que la almacena, se toma como referencia la registrada.
               1.1.1.1.1.4. Si no, se toma como referencia la almacenada.
               1.1.1.1.1.5. Se comparan las rutas punto a punto.
                  1.1.1.1.1.5.1. Comparar mientras haya puntos por comparar en la ruta tomada como referencia y el número de puntos distintos sea inferior al máximo permitido.
                     1.1.1.1.1.5.1.1. Se compara un punto.
                        1.1.1.1.1.5.1.1.1. Si la distancia entre ellos es superior a la permitida, se compara el mismo punto de la ruta referencia con el siguiente punto de la otra ruta.
                        1.1.1.1.1.5.1.1.2. Si no, el punto es considerado igual por lo que se comparan los siguientes en la ruta. Se contabiliza el número de puntos iguales y se deja constancia del último punto igual.
                        1.1.1.1.1.5.1.1.3. Si se llega al final de la ruta referencia sin encontrar similitud, se contabiliza como punto distinto y se empieza a comparar a partir del siguiente punto al último punto igual.
                     1.1.1.1.1.5.1.2. Si el número de puntos distintos es superior al permitido, las rutas son distintas.
                     1.1.1.1.1.5.1.3. Si no, son iguales.
               1.1.1.1.1.6. Si se determina que las dos rutas son iguales, se guarda como candidata de ser la ruta original de la ruta registrada. También se guarda el número de puntos iguales.
               1.1.1.1.1.7. Si existe ya una ruta candidata, se comprueba si el grado de similitud de la recién comparada es mayor.
                  1.1.1.1.1.7.1. Si es mayor, se almacena.
               1.1.1.1.1.8. Si no existe, se guarda.
      1.1.2. Si hay alguna candidata a ruta original, se asigna a la ruta registrada y se almacena como ruta copia de esa ruta original.
      1.1.3. Si no, la ruta registrada se almacena como ruta original.
   1.2. Si no, se almacena la ruta registrada en la base de datos como ruta original.

Postcondiciones: Se ha registrado una nueva ruta.

VALORAR RUTA

Nombre: Valorar ruta.

Descripción: Una vez al día, el sistema comprueba si se han realizado rutas durante las últimas 24 horas. Si es así, selecciona una entre las realizadas para que sea valorada por el usuario. Para seleccionar una ruta, ésta debe ser: nueva (ha sido la primera vez que se realizaba), realizada anteriormente pero sin valoración o realizada anteriormente y que haya pasado un mes desde la última valoración. Al finalizar, los datos son enviados al servidor.

Actores: Sistema (iniciador) y usuario.

Precondiciones: Han pasado 24 horas desde la última comprobación.

Requisitos no funcionales: Conexión a Internet activada.

Flujo de eventos:

1. A la hora indicada por el usuario, el sistema inicia la comprobación.
2. Se comprueba si se ha realizado alguna ruta en las últimas 24 horas.
    2.1. Si se ha realizado alguna, se comprueba si alguna es nueva.
        2.1.1. Si hay alguna nueva, se comprueba si es solo una.
            2.1.1.1. Si es solo una, se comprueba que la notificación de valoración esté activa.
                2.1.1.1.1. Si está activa, se actualiza con los datos de la ruta elegida y se crea el cuestionario.
                2.1.1.1.2. Si no, se activa la notificación de cuestionario.
            2.1.1.2. Si son más, se selecciona una aleatoriamente y se comprueba que la notificación de valoración esté activa.
                2.1.1.2.1. Si está activa, se actualiza con los datos de la ruta elegida y se crea el cuestionario.
                2.1.1.2.2. Si no, se activa la notificación de cuestionario.
        2.1.2. Si no, se comprueba si alguna ha sido realizada anteriormente y no ha sido valorada hasta el momento.
            2.1.2.1. Si hay alguna, se comprueba si es solo una.
                2.1.2.1.1. Si es solo una, se comprueba que la notificación de valoración esté activa.
                    2.1.2.1.1.1. Si está activa, se actualiza con los datos de la ruta elegida y se crea el cuestionario.
                    2.1.2.1.1.2. Si no, se activa la notificación de cuestionario.
                2.1.2.1.2. Si son más, se selecciona una aleatoriamente y se comprueba que la notificación de valoración esté activa.
                    2.1.2.1.2.1. Si está activa, se actualiza con los datos de la ruta elegida y se crea el cuestionario.
                    2.1.2.1.2.2. Si no, se activa la notificación de cuestionario.
            2.1.2.2. Si no, se comprueba si hay alguna realizada y valorada.

2.1.2.2.1. Si hay alguna, se comprueba cuántas han sido valoradas hace un mes o más.
2.1.2.2.1.1. Si hay solo una, se comprueba que la notificación de valoración esté activa.
2.1.2.2.1.2. Si hay más, se selecciona una aleatoriamente y se comprueba que la notificación de valoración esté activa.
2.1.2.2.1.2.1. Si está activa, se actualiza con los datos de la ruta elegida y se crea el cuestionario.
2.1.2.2.1.2.2. Si no, se activa la notificación de cuestionario.
2.1.2.2.2. Si no, se avisa al usuario de que no quedan más rutas pendientes de valoración (Ilustración 46 - Aviso no más rutas).
2.1.2.2.2.1. Se cancela la notificación de valoración pendiente, se realiza el envío de datos pendientes si los hubiera y se pasa a la pantalla principal (Ilustración 47 - Pantalla principal).

2.2. Si no, se realiza el envío de datos pendientes si los hubiera. Si se ha descartado una ruta anteriormente, se avisa al usuario de que no quedan más rutas pendientes de valoración (Ilustración 46 - Aviso no más rutas).
2.2.1. Se cancela la notificación de valoración pendiente y se pasa a la pantalla principal (Ilustración 47 - Pantalla principal).

3. Cuando el sistema activa la notificación de valoración pendiente, se muestra al usuario (Ilustración 48 - Notificación valoración pendiente).
4. Al hacer click en ella, se lanza el cuestionario.
4.1. Se muestra el mapa y se pregunta al usuario si ha realizado esa ruta o no (Ilustración 49 - Inicio cuestionario (1/2) e Ilustración 50 - Inicio cuestionario (2/2)).
4.1.1. Si responde que sí (Ilustración 51 - Botón Sí), se solicitan las categorías al servidor.
4.1.1.1. Si ocurre algún error, se muestra un aviso (Ilustración 52 - Fallo conexión categorías).
4.1.1.2. Si no, se muestran las categorías al usuario (Ilustración 53 - Categorías).
4.1.1.3. Al seleccionar una categoría y pulsar el botón Siguiente (Ilustración 54 - Botón siguiente), se solicitan las preguntas al servidor.
4.1.1.3.1. Si ocurre algún error, se muestra un aviso (Ilustración 55 - Fallo conexión preguntas).
4.1.1.3.2. Si no, se muestran las preguntas al usuario (Ilustración 56 - Preguntas).
4.1.1.3.3. Cuando el usuario completa el cuestionario, hace click en Enviar cuestionario (Ilustración 57 - Botón Enviar cuestionario).
4.1.1.3.3.1. Se agradece al usuario su colaboración (Ilustración 58 - Agradecimiento).
4.1.1.3.3.2. Se almacenan las respuestas y se envían al servidor.
4.1.1.3.3.3. Se envían el resto de datos recogidos durante las últimas 24 horas y los pendientes de envío y se cancela la notificación de valoración pendiente.

4.1.1.3.3.4. Se muestra la pantalla principal (Ilustración 47 - Pantalla principal).

4.1.2. Si responde que no (Ilustración 59 - Botón No), el sistema realiza la comprobación de nuevo.

5. Si la notificación se elimina de la barra de notificaciones o si el usuario no finaliza el cuestionario, se mostrará una notificación cada hora hasta que se finalice.

Postcondiciones: Se ha almacena correctamente el cuestionario y se han enviado todos los datos al servidor. Si ha ocurrido algún error, en la comprobación se enviarán.

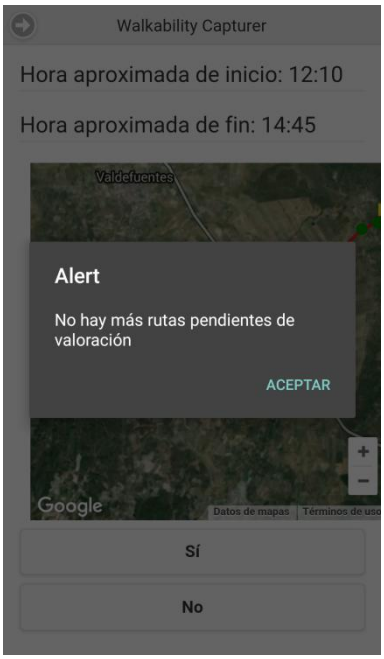

Ilustración 46 - Aviso no más rutas

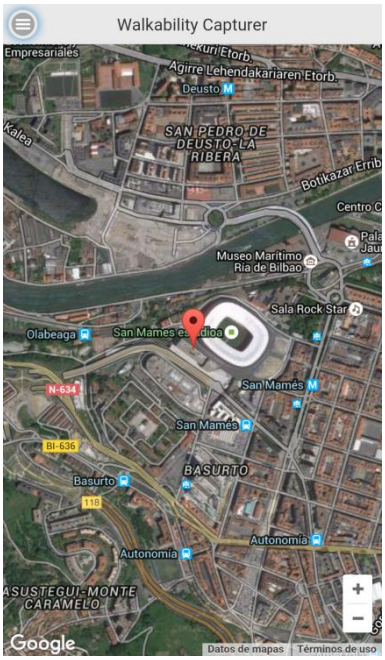

Ilustración 47 - Pantalla principal

Ilustración 48 - Notificación valoración pendiente

Ilustración 49 - Inicio cuestionario (1/2)

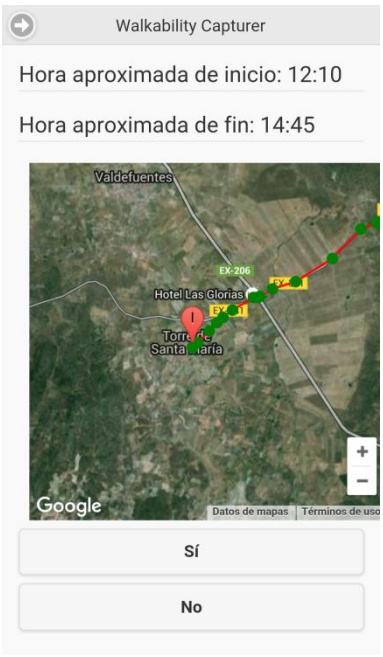

Ilustración 50 - Inicio cuestionario (2/2)

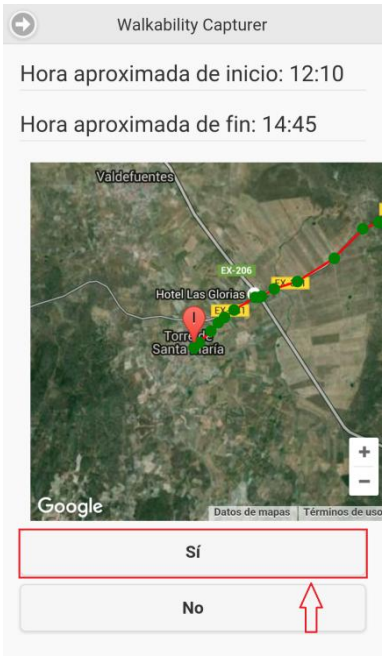

Ilustración 51 - Botón Sí

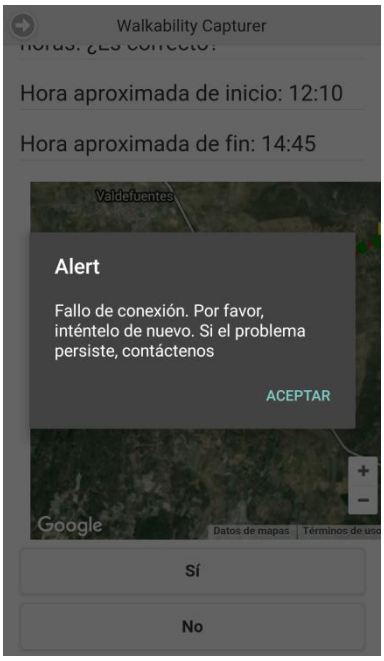

Ilustración 52 - Fallo conexión categorías

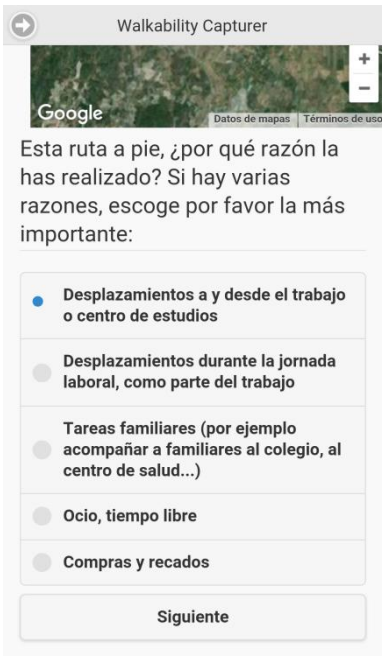

Ilustración 53 - Categorías

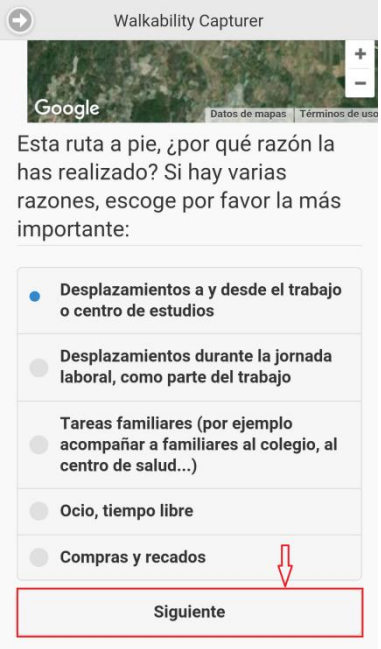

Ilustración 54 - Botón siguiente

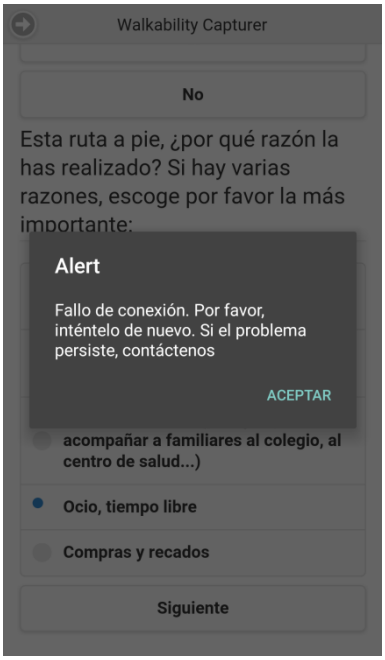

Ilustración 55 - Fallo conexión preguntas

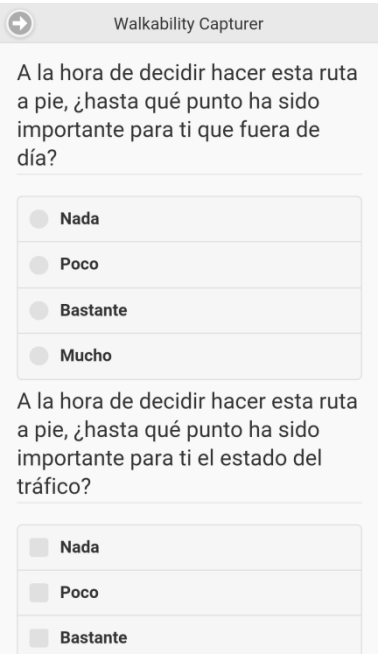

Ilustración 56 - Preguntas

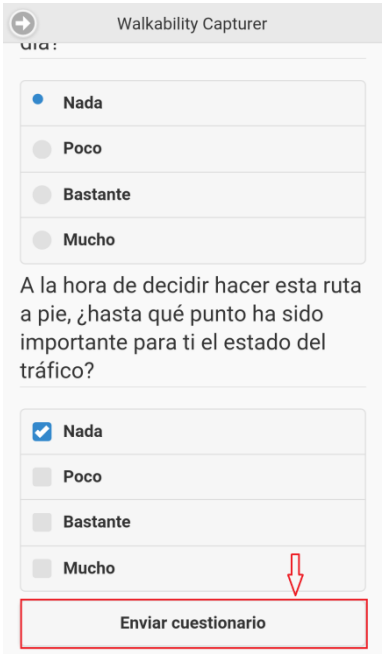

Ilustración 57 - Botón Enviar cuestionario

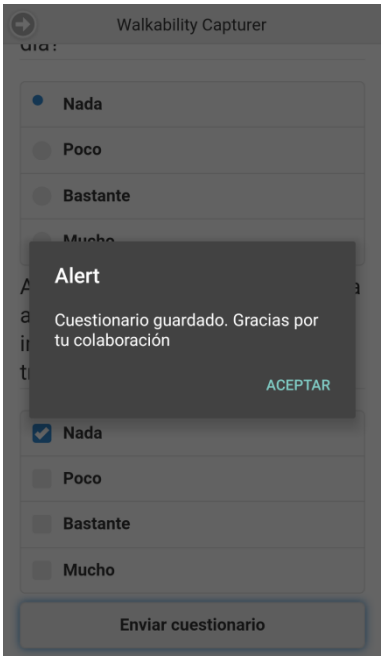

Ilustración 58 - Agradecimiento

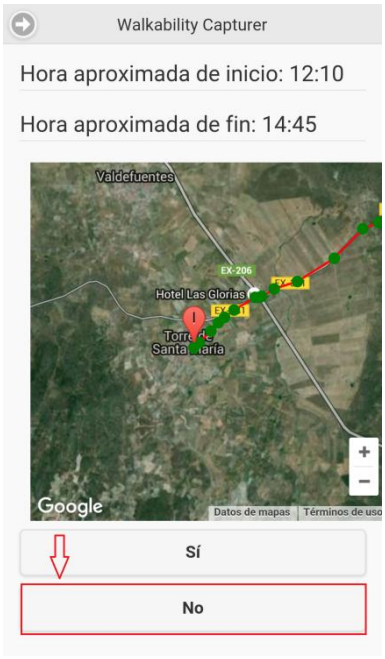

Ilustración 59 - Botón No

ANEXO II – DIAGRAMAS DE SECUENCIA

En el anexo II se van a desarrollar los diagramas de secuencia. Para una mejor lectura de los diagramas, algunos se presentarán en modo apaisado.

INICIAR APLICACIÓN

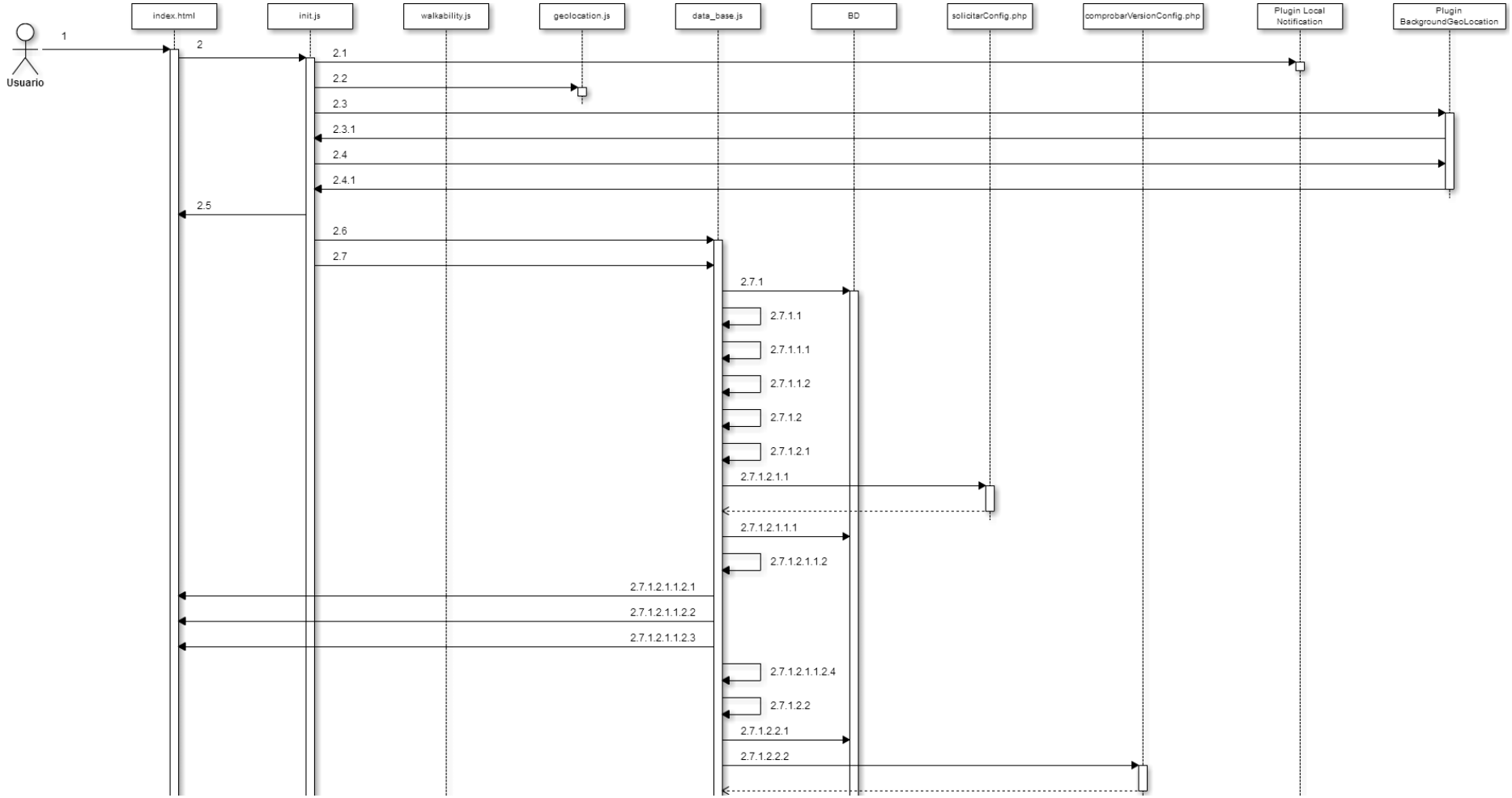

Ilustración 60 - Diagrama de secuencia: Iniciar aplicación (1/2)

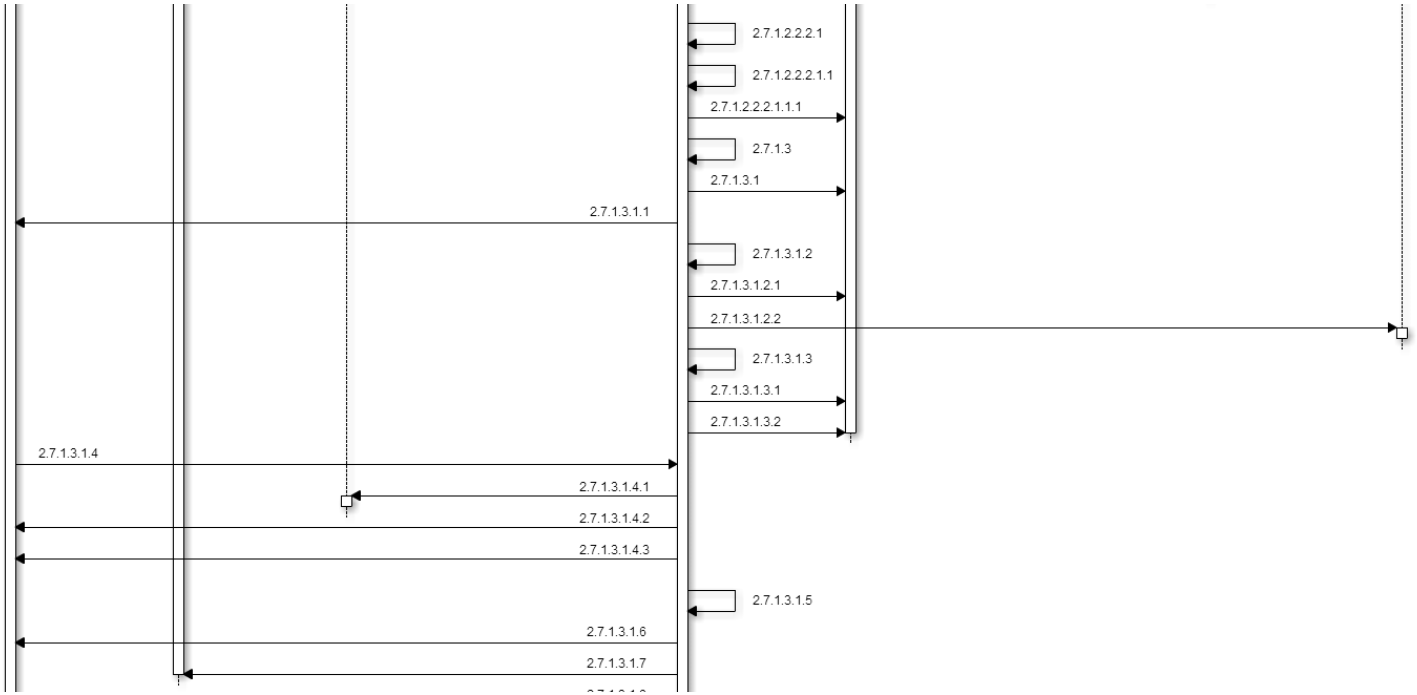

Ilustración 61 - Diagrama de secuencia: Iniciar aplicación (2/2)

1. El usuario lanza la aplicación.
2. Se ejecuta el evento *deviceready → document.addEventListener("deviceready", onDeviceReady, false)*
   2.1. Por si hubiera alguna notificación activa, se cancelan todas → *cordova.plugins.notification.local.cancelAll(function(){},this)*
   2.2. Se configura la geolocalización → *configurarBackgroundGeoLocation()*
   2.3. Se controla que el servicio de localización del dispositivo está activo → *backgroundGeoLocation.isLocationEnabled(locationCheck, fail)*
      2.3.1. Si el servicio de localización está desactivado y el usuario lo permite, se abren los ajustes del servicio de localización → *backgroundGeoLocation.showLocationSettings()*
   2.4. Se controla que durante el funcionamiento de la aplicación no se desactive el servicio de localización → *backgroundGeoLocation.watchLocationMode(locationCheck, fail)*
      2.4.1. Si el servicio se desactiva y el usuario lo permite, se abren los ajustes del servicio de localización → *backgroundGeoLocation.showLocationSettings()*
   2.5. Se bloquea el uso de la aplicación hasta completar la inicialización → *$("body").addClass("loading")*
   2.6. *abrirBD()*
   2.7. *existeBD()*
      2.7.1. Se comprueba que exista la tabla "usuario" → *'SELECT * FROM sqlite_master WHERE name = "usuario"'*
      [Si no hay esquema de BD]
         2.7.1.1. *inicializarBD()*
            2.7.1.1.1. *existeConfig() → 2.7.1.2*
            2.7.1.1.2. *existeUsuario() → 2.7.1.3*
      [Si hay esquema]
         2.7.1.2. *existeConfig()*
         [Si no hay configuración previa]
            2.7.1.2.1. *solicitarConfig(intento)*
               2.7.1.2.1.1. *$.getJSON('http://galan.ehu.eus/dpuerto001/WEB/solicitarConfig.php', function(data){...}).fail(function() {...})*
               [Si la conexión se realiza correctamente]
                  2.7.1.2.1.1.1. *'INSERT INTO configuracion (vel_max, dist_min_ruta, tiempo_parada, radio_parada, puntos_distintos, dist_puntos, version_actual, dif_duracion) VALUES (?,?,?,?,?,?,?,?,?)'*
               [Si ocurre algún error]
                  2.7.1.2.1.1.2. Se comprueba el parámetro intento
                  [Si intento == 3]
                     2.7.1.2.1.1.2.1. *alert("Imposible contactar con el servidor. Por favor, inténtelo de nuevo más tarde")*
                     2.7.1.2.1.1.2.2. Se bloquea el uso de la aplicación porque no hay configuración → *$("body").addClass("loading");*
                  [Si no]

2.7.1.2.1.1.2.3. *alert("Error al solicitar la configuracion, a continuación se reintentará. Si el problema persiste, contáctenos")*
2.7.1.2.1.1.2.4. *solicitarConfig(intento)*
[Fin si]
[Fin si]
[Si hay configuración previa]
2.7.1.2.2. *comprobarVersionConfig()*
2.7.1.2.2.1. *'SELECT version_actual FROM configuracion'*
2.7.1.2.2.2. Se consulta al servidor la versión activa → *$.ajax({...})*
[Si se realiza correctamente]
2.7.1.2.2.2.1. Se comprueba la variable cambiar
[Si cambiar == 1]
2.7.1.2.2.2.1.1. *actualizarConfig(data[0])*
2.7.1.2.2.2.1.1.1. *'UPDATE configuracion SET vel_max = ?, dist_min_ruta = ?, tiempo_parada = ?, radio_parada = ?, puntos_distintos = ?, dist_puntos = ?, version_actual = ?, dif_duracion = ? WHERE idconfiguracion = 1'*
[Fin si]
[Fin si]
[Fin si]
2.7.1.3. *existeUsuario()*
2.7.1.3.1. *'SELECT * FROM usuario'*
[Si el usuario existe]
2.7.1.3.1.1. *window.location.href="app"*
2.7.1.3.1.2. *activarNotificacionDiaria()*
2.7.1.3.1.2.1. *'SELECT realizar_cuestionario FROM configuracion WHERE idconfiguracion = 1'*
2.7.1.3.1.2.2. *cordova.plugins.notification.local.schedule({...})*
2.7.1.3.1.3. *cargarConfig()*
2.7.1.3.1.3.1. *'SELECT vel_max, dist_min_ruta, tiempo_parada, radio_parada, puntos_distintos, dist_puntos, dif_duracion FROM configuracion'*
2.7.1.3.1.3.2. *'SELECT idusuario FROM usuario'*
2.7.1.3.1.4. *$(document).on("pagehide", "#textoInicial", function() {...})*
2.7.1.3.1.4.1. *initMap()*
2.7.1.3.1.4.2. *$("body").removeClass("loading")*
2.7.1.3.1.4.3. *$("#btnSeguimiento").trigger("click")*
2.7.1.3.1.5. *enviosPendientes()*
[Si no existe]
2.7.1.3.1.6. *window.location.href="#textoInicial"*
2.7.1.3.1.7. Se rellena el select de los años → *rellenarSelect()*
2.7.1.3.1.8. *$("body").removeClass("loading")*
[Fin si]

CONFIGURACIÓN INICIAL

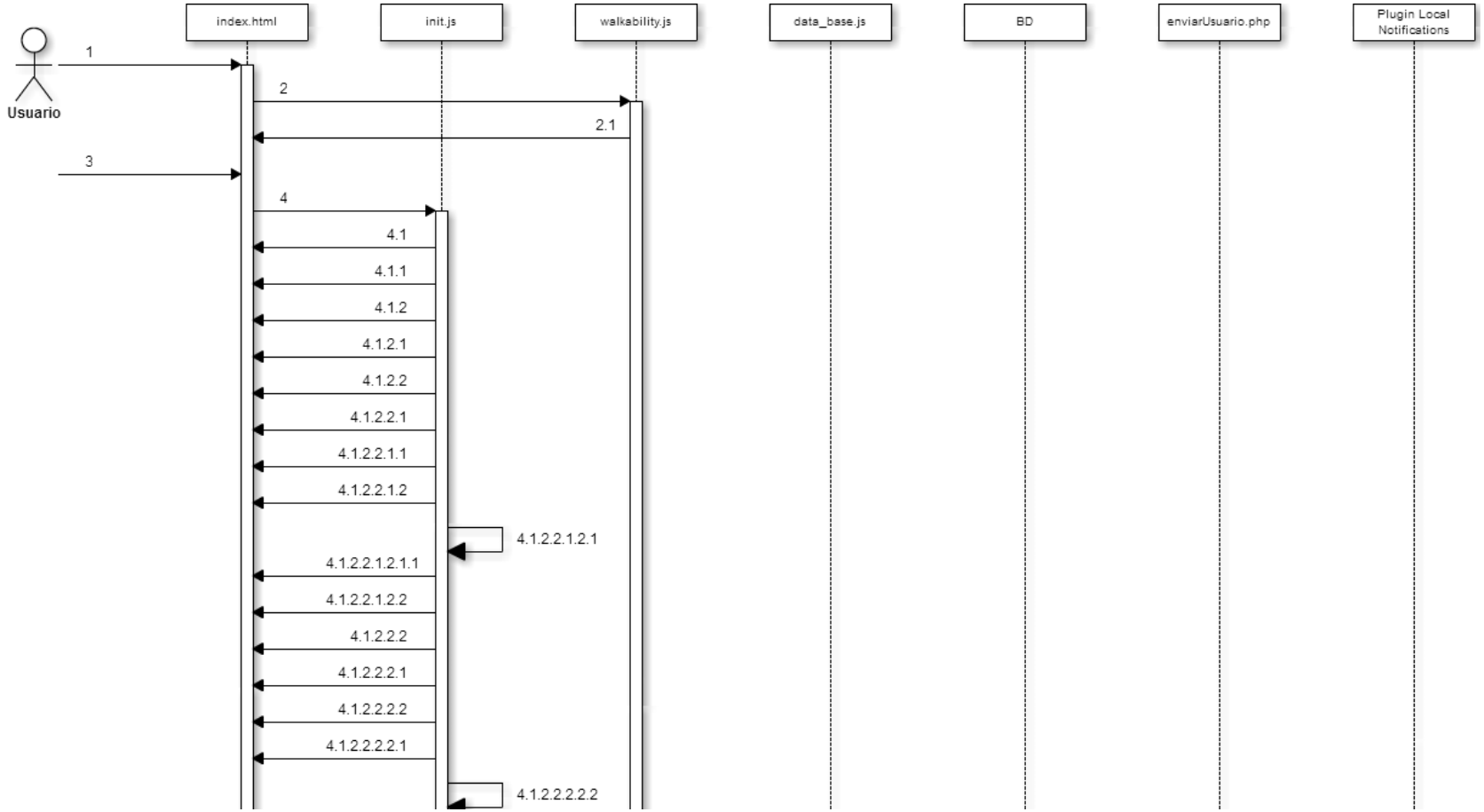

Ilustración 62 - Diagrama de secuencia: Configuración inicial (1/2)

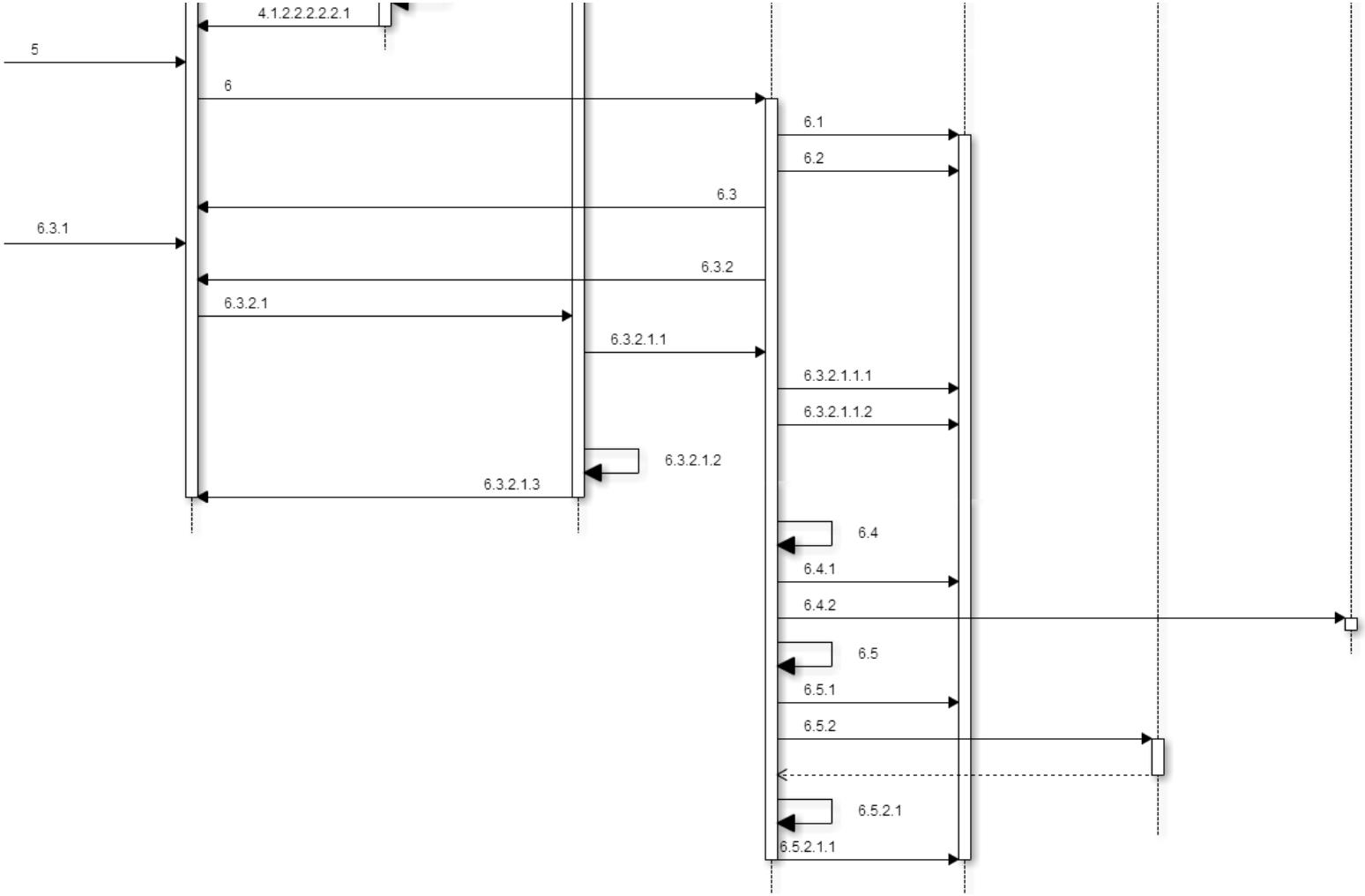

Ilustración 63 - Diagrama de secuencia: Configuración inicial (2/2)

1. El usuario pulsa el botón de la esquina izquierda
2. *$("#botonTexto").on("click", function() {...})*
   2.1. Se pasa al test inicial → *window.location.href="#testInicial"*
3. Se pulsa el botón "Crear usuario"
4. *$("#botonTest").on("click", function(){...})*
   4.1. Se comprueba que se haya seleccionado un año → *$("#anyoNacimiento option:selected").val()*
   [Si no se ha seleccionado]
      4.1.1. *alert("Por favor, seleccione un año")*
   [Si se ha seleccionado]
      4.1.2. Se comprueba que se haya seleccionado una hora → *$("#hora").val().length*
      [Si no se ha seleccionado]
         4.1.2.1. *alert("Por favor, seleccione una hora")*
      [Si se ha seleccionado]
         4.1.2.2. Se comprueba que se haya seleccionado el checkbox → *$("#check").prop('checked')*
         [Si se ha seleccionado]
            4.1.2.2.1. Se comprueba que no esté vacío → *$("#municipio").val()*
            [Si está vacío]
               4.1.2.2.1.1. *alert("Por favor, inserte un municipio")*
            [Si no está vacío]
               4.1.2.2.1.2. Se comprueba que se usen solo letras → *$("#municipio").val()*
               [Si se usan solo letras]
                  4.1.2.2.1.2.1. *confirmarUsuario()*
                     4.1.2.2.1.2.1.1. Se muestran al usuario los datos seleccionados → *navigator.notification.confirm(...)*
               [Si no]
                  4.1.2.2.1.2.2. *alert("Por favor, utilice letras nada más")*
               [Fin si]
            [Fin si]
         [Si no]
            4.1.2.2.2. Se comprueba que se haya seleccionado una provincia → *$(".ps-prov option:selected").val()*
            [Si no se ha seleccionado]
               4.1.2.2.2.1. *alert("Por favor, seleccione una provincia")*
            [Si se ha seleccionado]
               4.1.2.2.2.2. Se comprueba que se haya seleccionado un municipio → *$(".ps-mun option:selected").val()*
               [Si no se ha seleccionado]
                  4.1.2.2.2.2.1. *alert("Por favor, seleccione un municipio")*
               [Si se ha seleccionado]
                  4.1.2.2.2.2.2. *confirmarUsuario()*

4.1.2.2.2.2.2.1. Se muestran al usuario los datos seleccionados → *navigator.notification.confirm(…)*

[Fin si]

[Fin si]

[Fin si]

[Fin si]

[Fin si]

5. El usuario confirma los datos.
6. *anadirUsuario()*
   6.1. *'UPDATE configuracion SET realizar_cuestionario = ? WHERE idconfiguracion = 1'*
   6.2. *'INSERT INTO usuario (idusuario, anyo_nacimiento, genero, municipio_procedencia, movilidad_reducida, en_servidor) VALUES (?,?,?,?,?,?)'*
   6.3. Se da la bienvenida al usuario → *navigator.notification.alert(…)*
      6.3.1. El usuario pulsa aceptar.
      6.3.2. Se pasa a la pantalla principal → *window.location = "#app"*
         6.3.2.1. Se captura el evento generado al ocultarse la página del test → *$(document).on("pagehide", "#testInicial", function() {…})*
         6.3.2.1.1. *cargarConfig()*
            6.3.2.1.1.1. *'SELECT vel_max, dist_min_ruta, tiempo_parada, radio_parada, puntos_distintos, dist_puntos, dif_duracion FROM configuracion'*
            6.3.2.1.1.2. *'SELECT idusuario FROM usuario'*
         6.3.2.1.2. *initMap()*
         6.3.2.1.3. *$("#btnSeguimiento").trigger("click")*
   6.4. *activarNotificacionDiaria()*
      6.4.1. *'SELECT realizar_cuestionario FROM configuracion WHERE idconfiguracion = 1'*
      6.4.2. *cordova.plugins.notification.local.schedule({…})*
   6.5. *enviarUsuario()*
      6.5.1. *'SELECT idusuario, anyo_nacimiento, genero, municipio_procedencia, movilidad_reducida FROM usuario WHERE en_servidor = 0'*
      6.5.2. Se envían los datos al servidor → *$.ajax()*
      [Si no ocurre ningún problema]
         6.5.2.1. Se comprueba el valor de la variable insertado
         [Si es igual a 1]
            6.5.2.1.1. *'UPDATE usuario SET en_servidor = 1 WHERE idusuario = ?'*

CONFIGURAR SEGUIMIENTO

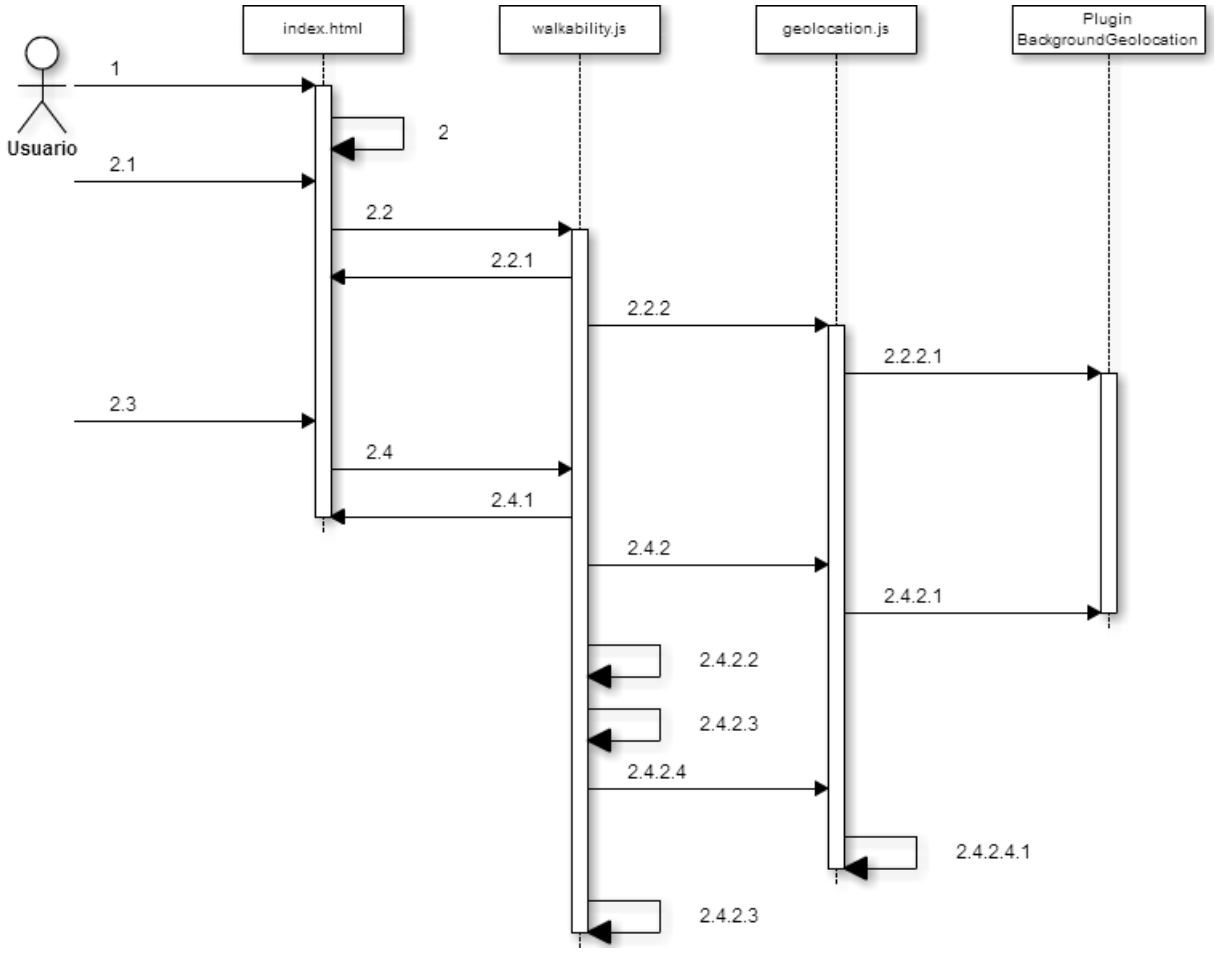

Ilustración 64 - Diagrama de secuencia: Configurar seguimiento

1. El usuario hace click en el botón superior izquierdo
2. Se abre el panel lateral → *href = "#panel"*

[Si el seguimiento está parado]
- 2.1. El usuario pulsa el botón "Iniciar Geo."
- 2.2. *$("#btnSeguimiento").on("click", function(){...})*
  - 2.2.1. *$("#btnSeguimiento").text("Parar Geo.")*
  - 2.2.2. *iniciarSeguimiento()*
    - 2.2.2.1. *backgroundGeoLocation.start()*

[Si está activado]
- 2.3. El usuario pulsa el botón "Parar Geo."
- 2.4. *$("#btnSeguimiento").on("click", function(){...})*
  - 2.4.1. *$("#btnSeguimiento").text("Iniciar Geo.")*
  - 2.4.2. *pararSeguimiento()*
    - 2.4.2.1. *backgroundGeoLocation.stop()*
  - [Si el temporizador de parada está activo]
    - 2.4.2.2. Parar temporizador de parada → *clearTimeout(temporizadorParada)*
  - [Fin si]
  - [Si el temporizador de espera de punto está activo]
    - 2.4.2.3. Parar temporizador de espera de punto → *clearTimeout(temporizadorPunto)*
  - [Fin si]
  - [Si estado = 3]
    - 2.4.2.4. Se pasa la ruta por el algoritmo de comparación de rutas → *recogerRutas(ruta, distanciaTotal)*
      - 2.4.2.4.1. Caso de uso Comparar y almacenar ruta
    - 2.4.2.5. *estado = 1*
  - [Fin si]

[Fin si]

CONFIGURAR COMPROBACIÓN DIARIA

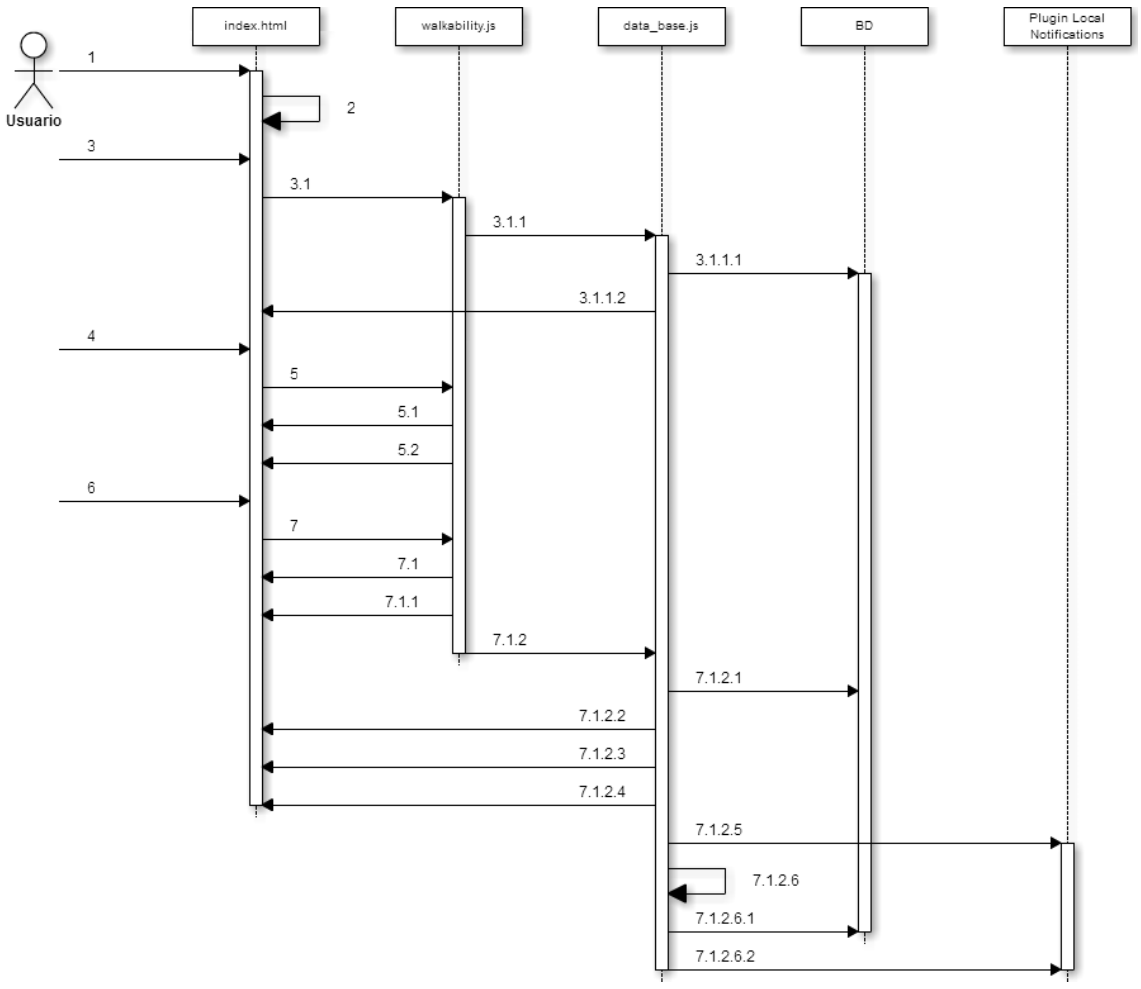

Ilustración 65 - Diagrama de secuencia: Configurar comprobación diaria

1. El usuario hace click en el botón superior izquierdo
2. Se abre el panel lateral → *href = "#panel"*
3. El usuario hace click en "Configuración"
   3.1. *$("#btnConfiguracion").on("click", function() {...})*
      3.1.1. *mostrarHoraCuestionario()*
         3.1.1.1. *'SELECT realizar_cuestionario FROM configuracion WHERE idconfiguracion = 1'*
         3.1.1.2. *$("#configActual").text("La hora definida para realizar el cuestionario es: " + horarioStr)*
4. El usuario pulsa el botón "Cambiar horario"
5. *$("#btnCambiarConfig").on("click", function() {...})*

[Si está oculto]
   5.1. *$('#cambiarConfig').show()*

[Si no está oculto]
   5.2. *$('#cambiarConfig').hide()*

[Fin si]

6. El usuario pulsa el botón "Guardar horario"
7. *$("#btnGuardarHoraNueva").on("click", function() {...})*
   7.1. Se comprueba que el área Time no esté vacía → *$("#horaNueva").val().length*

   [Si está vacía]
      7.1.1. *alert("Por favor, seleccione una hora")*

   [Si no]
      7.1.2. *actualizarHorarioCuestionario(horaInt, hora)*
         7.1.2.1. *'UPDATE configuracion SET realizar_cuestionario = ? WHERE idconfiguracion = 1'*
         7.1.2.2. *alert("Horario del cuestionario modificado")*
         7.1.2.3. Se oculta el contenedor donde se selecciona el nuevo horario → *$('#cambiarConfig').hide()*
         7.1.2.4. *$("#configActual").text("La hora definida para realizar el cuestionario es: " + horarioStr)*
         7.1.2.5. *cordova.plugins.notification.local.cancel(...)*
         7.1.2.6. *activarNotificacionDiaria()*
            7.1.2.6.1. *'SELECT realizar_cuestionario FROM configuracion WHERE idconfiguracion = 1'*
            7.1.2.6.2. *cordova.plugins.notification.local.schedule({...})*

   [Fin si]

REGISTRAR RUTA

Debido a que la práctica totalidad de la acción sucede en el archivo geolocation.js y para facilitar su comprensión, en el diagrama solo se muestran las funciones que implican una acción fuera del archivo.

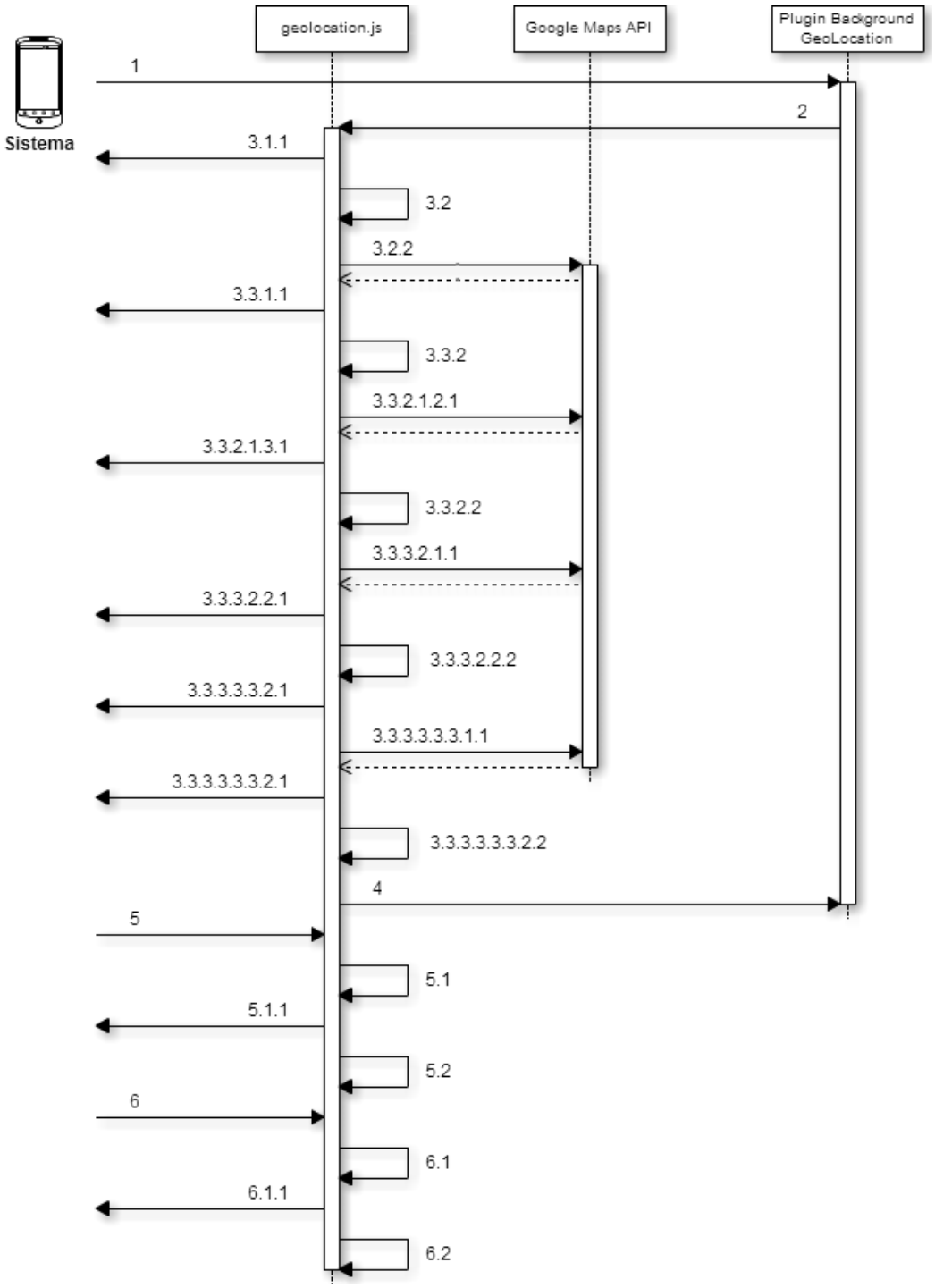

Ilustración 66 - Diagrama de secuencia: Registrar ruta

1. El sistema detecta movimiento y solicita un punto
2. Se recibe un punto nuevo → callbackFn (location){…}
3. Se comprueba la precisión

[Si es menor a 55]

3.1. Se comprueba el temporizador de espera de punto

[Si está activo]

3.1.1. clearTimeout(temporizadorPunto)

[Fin si]

3.2. Se comprueba que haya un punto anterior guardado

[Si no lo hay]

3.2.1. velocidad = 0

[Si lo hay]

3.2.2. Se calcula la separación entre los puntos guardados → google.maps.geometry.spherical.computeDistanceBetween(puntoAnterior, puntoNuevo).

3.2.3. Se calcula la velocidad

[Fin si]

3.3. Se comprueba el estado actual

[Si estado == 1]

3.3.1. Se comprueba el temporizador de parada

[Si está activo]

3.3.1.1. clearTimeout(temporizadorParada)

[Fin si]

3.3.2. Se comprueba la velocidad

[Si es mayor a 0 y menor que la máxima]

3.3.2.1. Se almacena el punto → crearYAlmacenarPunto(location.latitude, location.longitude, location.accuracy, latlon)

3.3.2.1.1. Se crea y almacena el punto

3.3.2.1.2. Se comprueba el estado

[Si el estado == 2]

3.3.2.1.2.1. Se mide la distancia de la ruta → google.maps.geometry.spherical.computeLength(pathRuta)

3.3.2.1.2.2. Se comprueba la distancia recorrida

[Si la distancia recorrida es mayor a la minima]

3.3.2.1.2.2.1. Se guarda el estado en la variable estadoAnterior

3.3.2.1.2.2.2. Se cambia a estado = 3

[Fin si]

[Fin si]

3.3.2.1.3. activarTempPunto()

3.3.2.1.3.1. setTimeout(function(){…})

3.3.2.2. estado = 2

[Fin si]

[Si no]

3.3.3. Se comprueba la velocidad
[Si es mayor a la velocidad máxima]
3.3.3.1. velocidadMaxSuperada(location.latitude, location.longitude, location.accuracy, latlon)
3.3.3.2. Se comprueba el temporizador de parada
[Si está activo]
3.3.3.2.1. Se mide la distancia entre el punto de parada y el recién recogido → punto_parada.distanciaConmigo(latitude, longitude)
3.3.3.2.1.1. google.maps.geometry.spherical.computeDistanceBetween(punto1, punto2)
3.3.3.2.2. Se comprueba la distancia entre los dos puntos
[Si la distancia es igual o superior al radio de parada]
3.3.3.2.2.1. Se para el temporizador de parada → clearTimeout(temporizadorParada)
3.3.3.2.2.2. comprobarVelSuperada(latitude, longitude, accuracy, latlon)
3.3.3.2.2.2.1. Se contabiliza las veces que se ha superado la velocidad
3.3.3.2.2.2.2. Se comprueban las veces que se ha superado la velocidad
[Si velocidadSuperada == 2]
3.3.3.2.2.2.2.1. Se comprueba el estado actual y el anterior
3.3.3.2.2.2.2.1.1. Se desecha el último punto y se almacena la ruta. Caso de uso Comparar y almacenar ruta
3.3.3.2.2.2.2.2. Se cambia de estado → estado = 1
[Si no]
3.3.3.2.2.2.2.3. crearYAlmacenarPunto(latitude, longitude, accuracy, latlon). Seguir 3.3.2.1
[Fin si]
[Si no]
3.3.3.2.2.3. activarTempPunto(). Seguir 3.3.2.1.3
[Fin si]
[Si no]
3.3.3.2.3. comprobarVelSuperada(latitude, longitude, accuracy, latlon). Seguir 3.3.3.2.2.2
[Fin si]
[Si no]
3.3.3.3. velocidadMaxNoSuperada(location.latitude, location.longitude, location.accuracy, latlon, velocidad)
3.3.3.3.1. Se pone a 0 la variable velocidadSuperada
3.3.3.3.2. Se almacena el estado en estadoAnterior
3.3.3.3.3. Se comprueba la velocidad
[Si la velocidad es menor a 1km/h]
3.3.3.3.3.1. Se comprueba el temporizador de parada
[Si está activo]
3.3.3.3.3.2. activarTempParada(latitude, longitude, accuracy, latlon)

3.3.3.3.3.2.1. setTimeout(function(){…})
3.3.3.3.3.2.2. crearYAlmacenarPunto(latitude, longitude, accuracy, latlon). Seguir 3.3.2.1
3.3.3.3.3.2.3. Se almacena el punto en la variable punto_parada
[Fin si]
[Si no]
3.3.3.3.3.3. Se comprueba el temporizador de parada
[Si está activo]
3.3.3.3.3.3.1. Se mide la distancia entre el punto de parada y el recién recogido → punto_parada.distanciaConmigo(latitude, longitude)
3.3.3.3.3.3.1.1. google.maps.geometry.spherical.computeDistanceBetween(punto1, punto2)
3.3.3.3.3.3.2. Se comprueba la distancia entre los dos puntos
[Si es mayor al radio de parada]
3.3.3.3.3.3.2.1. Se para el temporizador de parada → clearTimeout(temporizadorParada)
3.3.3.3.3.3.2.2. crearYAlmacenarPunto(latitude, longitude, accuracy, latlon). Seguir 3.3.2.1
[Si no]
3.3.3.3.3.3.2.3. activarTempPunto(). Seguir 3.3.2.1.3
[Fin si]
[Si no]
3.3.3.3.3.3.3. crearYAlmacenarPunto(latitude, longitude, accuracy, latlon). Seguir 3.3.2.1
[Fin si]
[Fin si]
[Fin si]
[Fin si]
[Fin si]
4. backgroundGeoLocation.finish()
5. El temporizador de espera de punto llega a 0
5.1. Se comprueba el temporizador de parada
[Si el temporizador está activo]
5.1.1. Se para → clearTimeout(temporizadorParada)
[Fin si]
5.2. Se comprueba el estado
[Si el estado es 3]
5.2.1. recogerRutas(ruta, distanciaTotal)
[Fin si]
5.3. Se cambia de estado → estado = 1
6. El temporizador de parada llega a 0
6.1. Se comprueba el temporizador de espera de punto
[Si el temporizador está activo]

6.1.1. Se para → clearTimeout(temporizadorPunto)
[Fin si]
6.2. Se comprueba el estado
[Si el estado es 3]
6.2.1. recogerRutas(ruta, distanciaTotal)
[Fin si]
6.3. Se cambia de estado → estado = 1

5.1 COMPARAR Y ALMACENAR RUTA

Para facilitar la lectura del diagrama, éste solo contiene las funciones que implican una acción fuera de los archivos geolocation.js o data_base.js. Cuando las acciones son dentro de los archivos solo se muestra la primera de la secuencia.

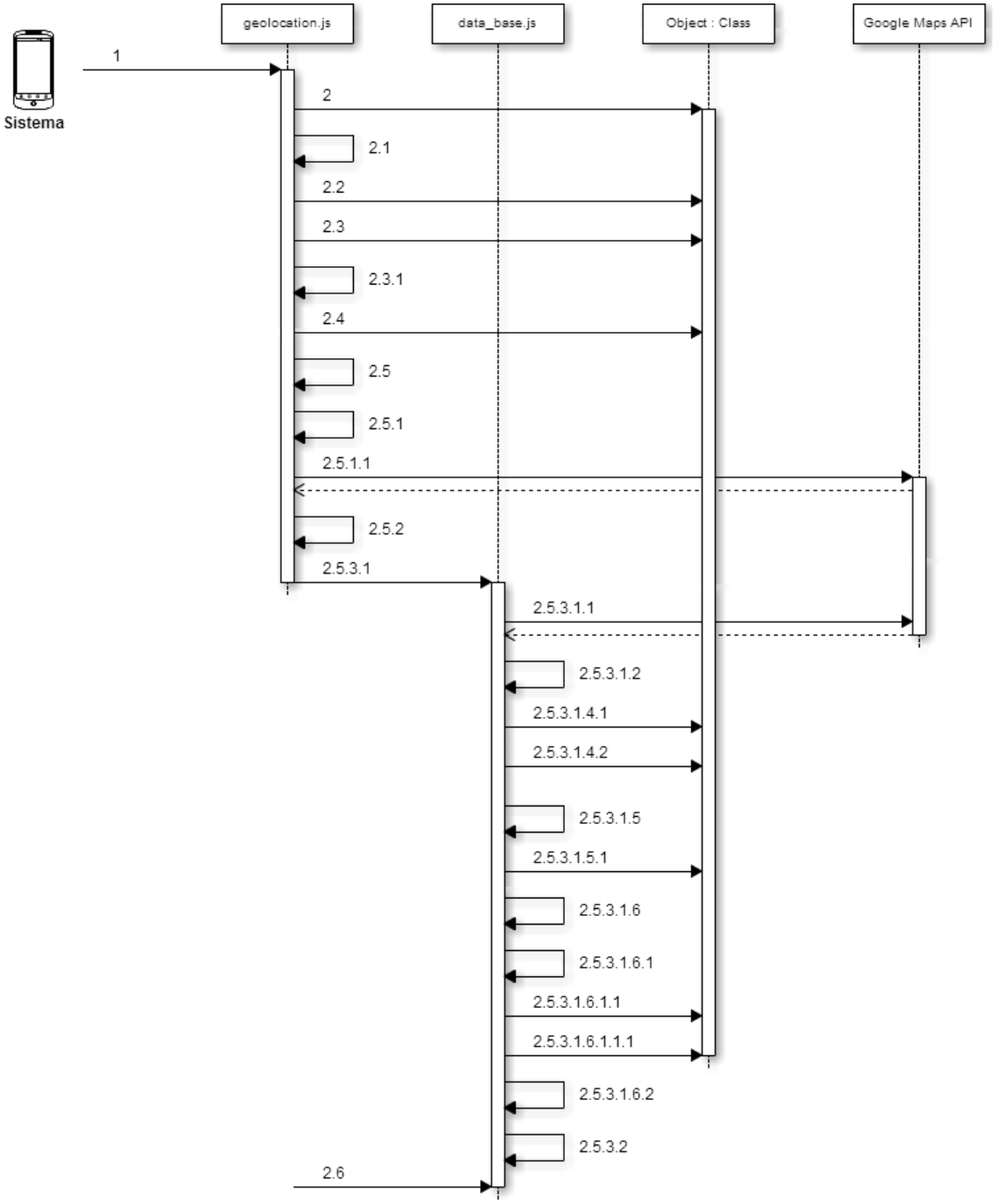

Ilustración 67 - Diagrama de secuencia: Comparar y almacenar ruta

1. Se comprueba que haya rutas originales almacenadas → *recogerRutas(rutaNueva, distanciaRuta)*
2. *'SELECT idruta, duracion, punto_inicio, punto_fin FROM ruta WHERE copia_de IS NULL'*

[Si hay]
[Mientras haya tuplas]
2.1. Se almacenan los datos en array rutasRecogidas
[Fin mientras]
[Mientras haya datos en el array]
2.2. Se seleccionan los datos del punto de inicio y se agregan al array → *'SELECT latitud, longitud, idruta FROM punto INNER JOIN ruta ON punto.idpunto = ruta.punto_inicio WHERE idruta = ?'*
2.3. Se seleccionan los puntos intermedios → *'SELECT latitud, longitud, idruta FROM punto INNER JOIN punto_intermedio ON punto.idpunto = punto_intermedio.idpunto WHERE idruta = ? ORDER BY orden ASC'*
[Mientras haya puntos]
2.3.1. Se agrega al array
[Fin mientras]
2.4. Se selecciona el punto final → *'SELECT latitud, longitud FROM punto INNER JOIN ruta ON punto.idpunto = ruta.punto_fin WHERE idruta = ?'*
[Fin mientras]
2.5. *compararRutas(rutasRecogidas, rutaNueva, distanciaRuta)*
[Mientras haya rutas]
2.5.1. Se mide la distancia entre los dos puntos de inicio y los dos de fin → *distanciaConmigo(punto_inicio_guardada.latitud, punto_inicio_guardada.longitud)*
2.5.1.1. *google.maps.geometry.spherical.computeDistanceBetween(punto1, punto2)*
2.5.2. Se comprueba la distancia
[Si es menor a la distancia máxima entre puntos]
2.5.2.1. Se compara la diferencia de duración
[Si está entre los límites permitidos]
2.5.2.1.1. Se identifica la ruta con menos puntos
[Si la ruta registrada tiene menos puntos]
2.5.2.1.1.1. *compararRutas(rutaNueva, rutasRecogidas[i].recorrido)*
[Mientras haya rutas]
2.5.2.1.1.1.1. Se compara punto a punto → *compararPuntos(rutaCorta[i], rutaLarga[j])*
[Si la distancia es mayor a la permitida]
2.5.2.1.1.1.1.1. Se compara con el siguiente
[Si no]
2.5.2.1.1.1.1.2. Se comparan los dos siguientes. Se almacena cuál ha sido la última coincidencia. Se contabiliza el número de puntos iguales
[Fin si]
2.5.2.1.1.1.1.3. Si se recorre la ruta larga sin encontrar similitud, se contabiliza el punto diferente y se continua desde la última similitud

[Fin mientras]
2.5.2.1.1.1.2. Se comprueba el número de puntos distintos
[Si el número de puntos distintos supera el máximo]
2.5.2.1.1.1.2.1. Se devuelve 0
[Si no]
2.5.2.1.1.1.2.2. Se devuelve el número de puntos iguales
[Fin si]
[Si no]
2.5.2.1.1.2. *compararRutas(rutasRecogidas[i].recorrido, rutaNueva)*
[Fin si]
2.5.2.1.2. Se comprueba que haya candidata a copia
[Si hay]
2.5.2.1.2.1. Si compararRutas devuelve valor distinto de 0, se almacena el punto como candidato
[Si no hay]
2.5.2.1.2.2. Si el valor devuelto es mayor que el almacenado, se almacena
[Fin si]
[Fin si]
[Fin si] $("#btnEnviarCuestionario").on("click", function() {
[Fin mientras]
2.5.3. Se comprueba que haya alguna ruta candidata a copia
[Si no hay candidata a copia]
2.5.3.1. Se almacena la ruta registrada como original → *anadirRuta(ruta, distancia, copia_de)*
2.5.3.1.1. Se solicita el municipio de inicio y de fin → *google.maps.Geocoder*
2.5.3.1.2. *calcularDuracion(inicio, fin)*
[Mientras haya puntos]
2.5.3.1.3. *insertarPunto(idpunto, latitud, longitud, precision)*
[Fin mientras]
2.5.3.1.4. Se comprueba que la ruta sea original
[Si es original]
2.5.3.1.4.1. *'INSERT INTO ruta (idruta, duracion, distancia_recorrida, cuantas, municipio_inicio, municipio_fin, punto_inicio, punto_fin, en_servidor) VALUES (?,?,?,?,?,?,?,?,?)'*
[Si no]
2.5.3.1.4.2. *'INSERT INTO ruta (idruta, duracion, distancia_recorrida, cuantas, municipio_inicio, municipio_fin, punto_inicio, punto_fin, copia_de, en_servidor) VALUES (?,?,?,?,?,?,?,?,?,?)'*
[Fin si]
[Mientras haya puntos]
2.5.3.1.5. *relacionarPuntoYRuta(idruta, idpunto, orden, hora)*
2.5.3.1.5.1. *'INSERT INTO punto_intermedio (idruta, idpunto, orden, hora) VALUES (?,?,?,?)'*

[Fin mientras]
2.5.3.1.6. Se comprueba que la ruta sea original
[Si es original]
2.5.3.1.6.1. *insertarFechaRuta(idruta)*
2.5.3.1.6.1.1. *'SELECT version_actual FROM configuracion WHERE idconfiguracion = 1'*
2.5.3.1.6.1.1.1. *'INSERT INTO fecha_ruta (idfecha_ruta, dia, hora, idruta, en_servidor, configuracion_version) VALUES (?,?,?,?,?,?)'*
[Si no]
2.5.3.1.6.2. *insertarFechaRuta(copia_de)*
[Fin si]
[Si hay candidata a copia]
2.5.3.2. Se almacena la ruta registrada como copia → *anadirRuta(ruta, distancia, copia_de)*
[Fin si]
[Si no hay]
2.6. *anadirRuta(ruta, distancia, copia_de)*
[Fin si]

VALORAR RUTA

Debido a que la práctica totalidad de la acción sucede en el archivo cuestionario.js y para facilitar su comprensión, en el diagrama solo se muestran las funciones que implican una acción fuera del archivo. También, como hay dos actores, el diagrama se ha dividido según el actor.

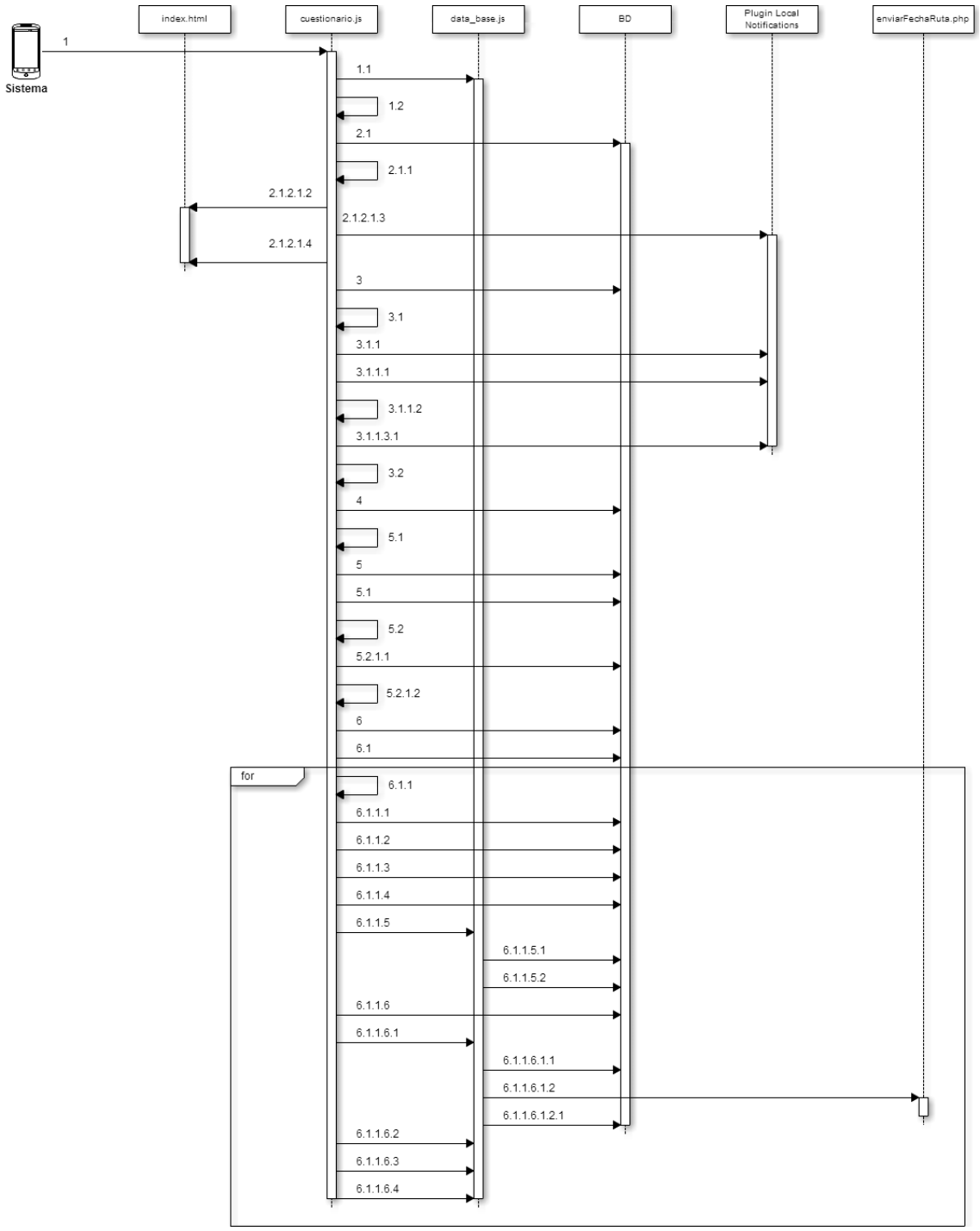

Ilustración 68 - Diagrama de secuencia: Valorar ruta Sistema

1. El sistema inicia la comprobación → obtenerHoraCuestionario()
   1.1. Se obtiene la hora elegida por el usuario para responder el cuestionario → 'SELECT realizar_cuestionario FROM configuracion'
   1.2. recuperarRutasRealizadas(hora) → **Sigue en 2**
2. Se calcula la fecha del día anterior → diaAnterior(dia, mes, anyo)
   2.1. Se obtienen las rutas realizadas las últimas 24 horas → 'SELECT idruta FROM fecha_ruta WHERE (dia = ? AND hora >= ?) OR (dia = ? AND hora <= ?)'
   [Si se ha realizado alguna]
      2.1.1. recuperarRutasRealizadasNuevas(fechaAyer, fechaHoy, hora) → **Sigue en 3**
   [Si no]
      2.1.2. Se comprueba si anteriormente se ha borrado alguna ruta
      [Si se ha borrado alguna]
         2.1.2.1. noHayMasRutas()
            2.1.2.1.1. Si se ha borrado alguna ruta anteriormente → enviarDatosDiarios()→ **Sigue en 6**
            2.1.2.1.2. alert("No hay más rutas pendientes de valoración")
            2.1.2.1.3. cordova.plugins.notification.local.cancel(…)
            2.1.2.1.4. Se pasa a la pantalla principal → window.location.href="#app"
      [Fin si]
   [Fin si]
3. 'SELECT fecha_ruta.idruta, fecha_ruta.idfecha_ruta FROM fecha_ruta INNER JOIN ruta ON fecha_ruta.idruta = ruta.idruta WHERE (dia = ? AND hora >= ? AND fecha_ruta.en_servidor = 0 AND ruta.copia_de IS NULL) OR (dia = ? AND hora <= ? AND fecha_ruta.en_servidor = 0 AND ruta.copia_de IS NULL)'

[Si se ha realizado alguna ruta nueva]
   [Si es una]
   3.1. comprobarNotificacion(idruta, idfecha_ruta, momento)
      3.1.1. cordova.plugins.notification.local.isScheduled(10, function(a){…})
      [Si la notificación está activada]
         3.1.1.1. cordova.plugins.notification.local.update({…})
         3.1.1.2. realizarCuestionario(idruta, idfecha_ruta)
      [Si no]
         3.1.1.3. activarNotificacionCuestionario(idruta, idfecha_ruta)
            3.1.1.3.1. cordova.plugins.notification.local.schedule({…})
      [Fin si]
   [Si son más]
   3.2. Calcular numero aleatorio → aleatorio = Math.round(Math.random()*rutasRecuperadas.length)
   3.3. comprobarNotificacion(idruta, idfecha_ruta, momento)
   [Fin si]
[Si no]
   3.4. recuperarRutasRealizadasYNoValoradas(fechaAyer, fechaHoy, hora) → **Sigue en 4**
[Fin si]

4. 'SELECT fecha_ruta.idruta, fecha_ruta.idfecha_ruta FROM fecha_ruta LEFT JOIN ruta_valorada ON fecha_ruta.idfecha_ruta = ruta_valorada.fecha_ruta_idfecha_ruta WHERE (dia = ? AND hora >= ? AND ruta_valorada.fecha_ruta_idfecha_ruta IS NULL) OR (dia = ? AND hora <= ? AND ruta_valorada.fecha_ruta_idfecha_ruta IS NULL)'

[Si se ha realizado alguna ruta no valorada]

[Si es una]

4.1. comprobarNotificacion(idruta, idfecha_ruta, momento)

[Si son más]

4.2. Calcular numero aleatorio → aleatorio = Math.round(Math.random()*rutasRecuperadas.length)

4.3. comprobarNotificacion(idruta, idfecha_ruta, momento)

[Fin si]

[Si no]

4.4. recuperarRutasRealizadasYValoradas(fechaAyer, fechaHoy, hora) → **Sigue en 5**

[Fin si]

5. 'SELECT ruta.copia_de FROM fecha_ruta INNER JOIN ruta ON fecha_ruta.idruta = ruta.idruta WHERE (dia = ? AND hora >= ?) OR (dia = ? AND hora <= ?)'

[Si se ha realizado alguna ruta valorada]

[Por cada ruta]

5.1. Se selecciona cuándo ha sido valorada → 'SELECT fecha_cuestionario.dia, fecha_ruta.idfecha_ruta, fecha_ruta.idruta FROM fecha_ruta INNER JOIN ruta_valorada ON fecha_ruta.idfecha_ruta = ruta_valorada.fecha_ruta_idfecha_ruta INNER JOIN fecha_cuestionario ON ruta_valorada.fecha_cuestionario_idfecha_cuestionario = fecha_cuestionario.idfecha_cuestionario WHERE fecha_ruta.idruta = ? ORDER BY fecha_cuestionario.dia DESC'

5.2. Si hace un mes o más que ha sido valorada, se guarda en un array

[Fin por cada ruta]

[Si hay alguna valorada hace más de un mes]

[Si hay una]

5.2.1.1. 'SELECT fecha_ruta.idruta, fecha_ruta.idfecha_ruta FROM fecha_ruta INNER JOIN ruta ON fecha_ruta.idruta = ruta.idruta WHERE ((dia = ? AND hora >= ?) OR (dia = ? AND hora <= ?)) AND ruta.copia_de = ?'

5.2.1.2. comprobarNotificacion(idruta, idfecha_ruta, momento)

[Si hay más]

5.2.1.3. Calcular numero aleatorio → aleatorio = Math.round(Math.random()*rutasRecuperadas.length)

5.2.1.4. comprobarNotificacion(idruta, idfecha_ruta, momento)

[Fin si]

[Si no]

5.2.2. noHayMasRutas()

[Fin si]

[Si no]

5.3. noHayMasRutas()

[Fin si]

6. Se obtiene la hora elegida por el usuario para responder el cuestionario → 'SELECT realizar_cuestionario FROM configuracion'
   6.1. Se obtienen las rutas realizadas en las últimas 24 horas y que no están en el servidor → 'SELECT idfecha_ruta FROM fecha_ruta WHERE (dia = ? AND hora >= ? AND en_servidor = 0) OR (dia = ? AND hora <= ? AND en_servidor = 0)'
   [Por cada ruta]
   6.1.1. reasignarFechaRuta(idfecha_ruta)
      6.1.1.1. Se obtiene la ruta original de la ruta copia → 'SELECT fecha_ruta.idruta, ruta.copia_de FROM fecha_ruta INNER JOIN ruta ON fecha_ruta.idruta = ruta.idruta WHERE idfecha_ruta = ? AND copia_de IS NOT NULL'
      6.1.1.2. Se asigna la fecha_ruta de la ruta copia a la ruta original → 'UPDATE fecha_ruta SET idruta = ? WHERE idfecha_ruta = ?'
      6.1.1.3. Se obtienen los datos de la fecha_ruta → 'SELECT dia, hora, en_servidor, configuracion_idconfiguracion FROM fecha_ruta WHERE idfecha_ruta = ?'
      6.1.1.4. Se crea una nueva fecha ruta con los mismos datos para la ruta copia → 'INSERT INTO fecha_ruta (idfecha_ruta, dia, hora, idruta, en_servidor, configuracion_version) VALUES (?,?,?,?,?,?)'
      6.1.1.5. Se aumenta el número de veces realizada la ruta original → aumentarCuantas(idruta)
         6.1.1.5.1. 'SELECT cuantas FROM ruta WHERE idruta = ?'
         6.1.1.5.2. 'UPDATE ruta SET cuantas = ? WHERE idruta = ?'
      6.1.1.6. Se consulta si la ruta original está en el servidor → 'SELECT idruta FROM ruta WHERE idruta = ?, en_servidor = 1'
         6.1.1.6.1. Se envía la fecha_ruta de la ruta original → enviarFechaRuta(idfecha_ruta)
            6.1.1.6.1.1. 'SELECT idfecha_ruta, dia, hora, idruta, configuracion_version FROM fecha_ruta WHERE idfecha_ruta = ?'
            6.1.1.6.1.2. Se envía al servidor → $.ajax({…})
            [Si se envía correctamente]
               6.1.1.6.1.2.1. 'UPDATE fecha_ruta SET en_servidor = 1 WHERE idfecha_ruta = ?'
         6.1.1.6.2. Se envía la ruta copia → enviarRuta(idfecha_ruta_nueva)
         6.1.1.6.3. Se envía la ruta original → enviarRuta(idfecha_ruta)
         6.1.1.6.4. Se envía la ruta copia → enviarRuta(idfecha_ruta_nueva)
   [Fin por cada ruta]

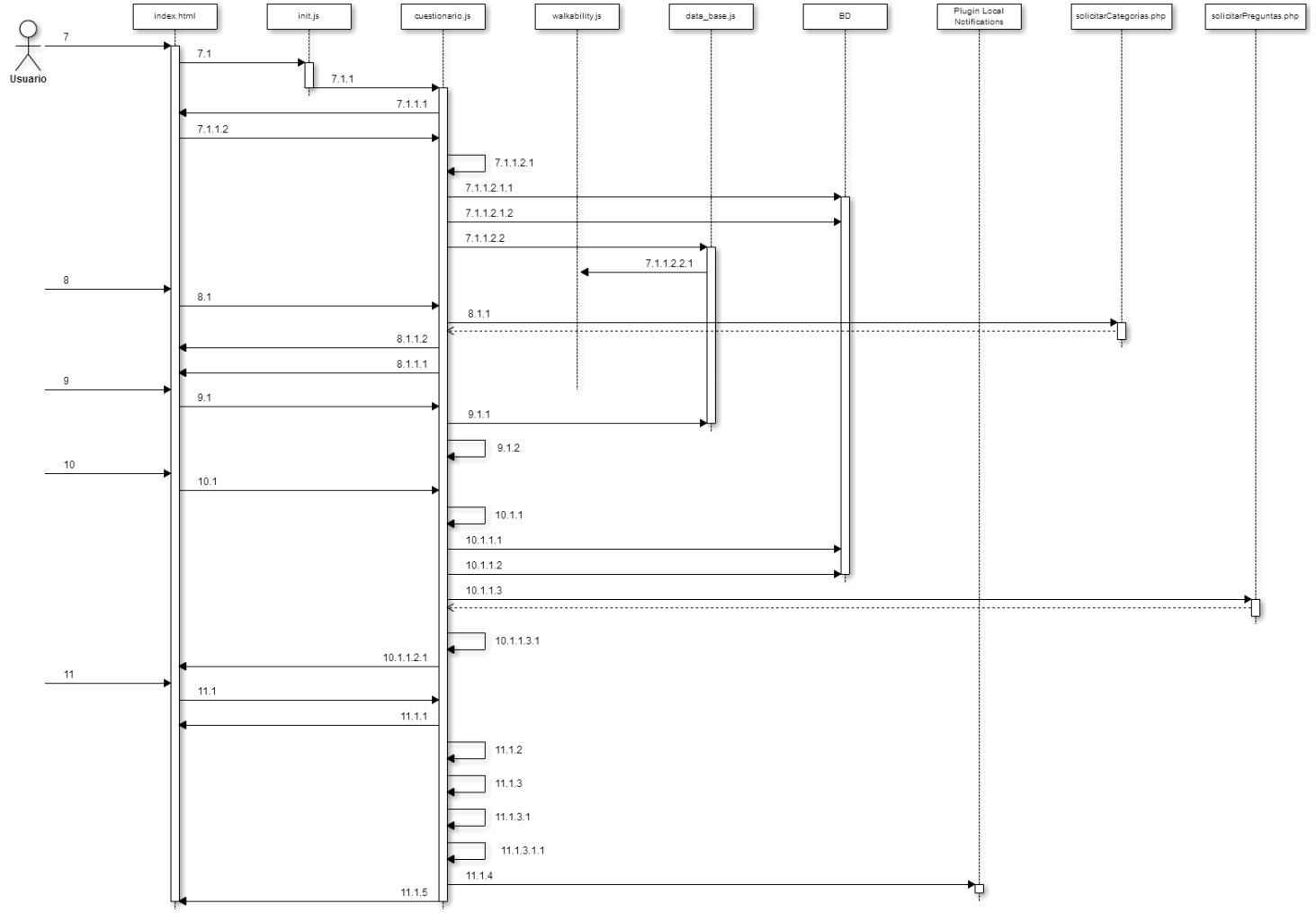

Ilustración 69 - Diagrama de secuencia: Valorar ruta Usuario

7. El sistema muestra la notificación y el usuario hace click en ella
    7.1. onNotificationClick(notification)
        7.1.1. realizarCuestionario(idruta, idfecha_ruta)
            7.1.1.1. window.location.href="#cuestionarioPage"
            7.1.1.2. $(document).on("pageshow", "#cuestionarioPage", function() {…})
                7.1.1.2.1. recuperarHoraInicioFin(idruta)
                    7.1.1.2.1.1. 'SELECT hora FROM punto_intermedio WHERE idruta = ? AND orden = 1'
                    7.1.1.2.1.2. 'SELECT hora FROM punto_intermedio WHERE idruta = ? ORDER BY orden DESC'
                7.1.1.2.2. recuperarRuta(idruta, "mapaCuestionario")
                    7.1.1.2.2.1. pintarMapa()
8. El usuario hace click en “Sí”
    8.1. $("#btnSi").on("click", function(){…})
        8.1.1. Se solicitan las categorías al servidor → $.ajax({…})
        [Si la conexión se realiza correctamente]
            8.1.1.1. Se muestran las categorías
        [Si no]
            8.1.1.2. alert("Fallo de conexión. Por favor, inténtelo de nuevo. Si el problema persiste, contáctenos")
        [Fin si]
9. El usuario hace click en “No”
    9.1. $("#btnNo").on("click", function(){…})
        9.1.1. borrarRuta(idfecha_ruta)
        9.1.2. obtenerHoraCuestionario()
10. El usuario hace click en “Siguiente”
    10.1. $("#btnSiguiente").on("click", function(){…})
        10.1.1. solicitarPreguntas(categoria, idfecha_ruta)
            10.1.1.1. 'SELECT genero FROM usuario'
            10.1.1.2. 'SELECT hora FROM fecha_ruta WHERE idfecha_ruta = ?'
            10.1.1.3. Se solicitan las preguntas al servidor → $.ajax({…})
            [Si la conexión se realiza correctamente]
                10.1.1.3.1. crearCuestionario(data)
            [Si no]
                10.1.1.3.2. alert("Fallo de conexión. Por favor, inténtelo de nuevo. Si el problema persiste, contáctenos")
            [Fin si]
11. El usuario hace click en “Enviar cuestionario”
    11.1. $("#btnEnviarCuestionario").on("click", function(){…})
        11.1.1. alert("Cuestionario guardado. Gracias por tu colaboración")
        11.1.2. reasignarFechaRuta(idfecha_ruta)
        11.1.3. almacenarRespuestas()
            11.1.3.1. enviarCuestionario()

11.1.3.1.1. enviarDatosDiarios()

11.1.4. cordova.plugins.notification.local.cancel(10, function(){…})

11.1.5. Se pasa a la pantalla principal → window.location.href="#app"