Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN SISTEMAS DE INFORMACIÓN

Generación de aplicaciones nativas en el dominio de Gestión del Trabajo en Movilidad

Estudante: María Ocampo Quintáns
Dirección: Alejandro Cortiñas Álvarez
Dirección: Miguel Ángel Rodríguez Luaces

A Coruña, setembro de 2020.

A mi familia. En especial a ti, Eva.

Agradecimientos

A los profesores que me han impartido clase durante el grado. En especial, a Alejandro Cortiñas Álvarez y Miguel Ángel Rodríguez Luaces, por todo lo que me enseñaron, por su paciencia y porque sin ellos esto no sería posible.

A mi familia, por el apoyo y la confianza que tuvieron en mí durante todos estos años.

A mis amigos, por todo el ánimo que me han dado.

Resumen

El objetivo de este trabajo es desarrollar una herramienta de generación del código fuente de una aplicación móvil para gestionar la movilidad de los empleados.

Para alcanzar este objetivo fue necesario, en primer lugar, desarrollar una aplicación móvil que permitiese a los empleados en movilidad la gestión de los eventos que se le asignaban, incluyendo todas las funcionalidades que pudieran ser necesarias en este contexto. En segundo lugar, se realizó un proceso de anotación del código fuente creado, que posteriormente se utilizará para generar productos diferentes. Por último, se llevó a cabo el desarrollo de la herramienta de generación del código fuente de dicha aplicación móvil.

En el desarrollo de la aplicación móvil se empleó el sistema de gestión de bases de datos PostgreSQL para el almacenamiento de información; el *framework* React Native para la implementación de la interfaz; y para el servidor web se integró Spring Boot con JPA, utilizando Hibernate para implementarlo. Por otro lado, para la implementación de la herramienta de generación se empleó el framework VueJS y el motor de derivación spl-js-engine para la generación del código fuente de los productos.

El trabajo de fin de grado se desarrolló siguiendo una metodología iterativa e incremental para el desarrollo del software.

Abstract

The objective of this end-of-degree project is to develop a tool for generating the source code of a mobile application to manage the employees' mobility.

In order to achieve this goal, it was necessary, first of all, to develope mobile application that allows employees in mobility to manage the tasks that they have to do, including all the functionalities that could be necessary in this context. Secondly, the necessary anotations were made in the code to then generate products. Finally, the tool for generating source code of the application was developed.

In the mobile application development, as database management system to information storage PostgreSQL was used; React Native framework to implement de interface and for the web server Spring Boot with JPA was integrated, using Hibernate for the implementation. On the other hand, Vue.js framework was used to implement the generating tool and spl-js-engine was used to generate the products' source code.

The end-of-degree work was managed following an iterative and incremental methodology for software development.

Palabras clave:

- Aplicación móvil
- Línea de Producto Software
- Generación de productos
- Árbol de características
- Gestión de eventos
- React Native
- Spring
- Hibernate
- PostgreSQL
- Vue.js
- spl-js-engine

Keywords:

- Mobile application
- Software Product Line
- Products generation
- Feature model
- Tasks management
- React Native
- Spring
- Hibernate
- PostgreSQL
- Vue.js
- spl-js-engine

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1 Motivación

Las mejoras en las tecnologías de la comunicación, junto con la mayor penetración del acceso a Internet, han expandido el abanico de posibilidades que tienen las empresas a la hora de utilizar la tecnología para mejorar su flujo de trabajo. Esto es especialmente notable para las empresas con trabajadores en movilidad (empresas que prestan servicios fuera de sus instalaciones, es decir, cuyos trabajadores tienen que desplazarse para atender a clientes), que actualmente tienen los medios para conocer la posición de sus trabajadores. En este ámbito surgieron las aplicaciones de Gestión de Trabajo en Movilidad (GTM).

En general, estas aplicaciones normalmente soportan un conjunto de funcionalidades parecidas, centradas en la gestión de agendas de los trabajadores en movilidad, pero cubriendo aspectos como la optimización de horarios, rutas y eventos, o la detección de actividades de cada trabajador.

El objetivo principal de este trabajo es crear una herramienta que permita la generación del código fuente de aplicaciones móvil para gestionar la movilidad de los empleados. El código fuente de la aplicación se generará en función de las características especificadas por el usuario. El propósito de esta herramienta es facilitar la generación de aplicaciones con funcionalidades similares incluyendo, únicamente, las solicitadas por el usuario. Los productos generados estarán adaptados en función de la especificación indicada.

El desarrollo de esta herramienta facilita al usuario la personalización de productos a diferentes clientes, provocando una reducción en los costes ya que no es necesario el desarrollo de un software para cada producto, sino que se generan a partir de artefactos comunes sin la necesidad de ser implementados desde cero. El desarrollador podrá generear los productos con mayor facilidad y en menor tiempo. Además, esta herramienta permite ampliaciones, es decir, si el cliente necesita incluir alguna característica que no esté implementada, el desarrollador solo tendrá que implementar dicha característica, el tiempo de producción seguirá

siendo menor que si tuviera que llevar a cabo la implementación del producto completo.

1.2 Objetivos

Para alcanzar el objetivo general descrito en la sección anterior se describen, a continuación, una serie de objetivos específicos:

- Crear una **aplicación nativa móvil** que permita, a los trabajadores en movilidad, gestionar sus eventos.
- Crear una **línea de producto software (LPS)** que permita seleccionar una serie de características y, a partir de estas, generar el producto final.
- Realizar un análisis de los requisitos de la aplicación. Será la LPS la que tendrá los requisitos disponibles para que sean seleccionados por el analista.
- El uso de la herramienta de generación debe ser sencillo y el código fuente generado por la misma debe ser de fácil compresión permitiendo, además, una fácil modificación de su estructura.
- El proceso de generación del código resultante se llevará a cabo usando un enfoque anotativo, utilizando un motor de derivación que permita flexibilidad de forma que cualquier tipo de código fuente permita incluir nuevas características que el cliente necesite para su producto y que aún no estén implementadas.

Capítulo 2

Fundamentos tecnológicos

2.1 Estado del arte

Existen una serie de herramientas de líneas de producto software (LPS) que realizan funciones similares y dan soporte a algunas de las necesidades especificadas. Entre ellas, se destacaron las siguientes:

- **FeatureHOUSE** [1]: composición automática de software, indpendiente del lenguaje: framework que permite la composición de artefactos de software. Estos artefactos software pueden estar escritos en diferentes lenguajes de programación ya que una de las características de FeatureHOUSE es la independencia del idioma. FeatureHOUSE se basa en un árbol de estructura de características (FST) donde los artefactos de software son representados con una estructura jerárquica. Los FST diseñan la estructura modular de los artefactos abstrayendo los detalles específicos.

```
1  package com.sleepycat;
2  public class Database {
3    private void acquireReadLock() throws DatabaseException { ... }
4    private void releaseReadLock() throws DatabaseException { ... }
5    protected void notifyTriggers(Locker locker, DatabaseEntry priKey,
6      DatabaseEntry oldData, DatabaseEntry newData) throws DatabaseException {
7      acquireReadLock();
8      original(locker,priKey,oldData,newData);
9      releaseReadLock();
10   } // 50 further lines of code...
11 }
```

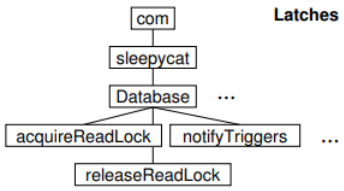


Figura 2.1: Ejemplo de código generado por FeatureHOUSE [1] a partir del árbol de características.

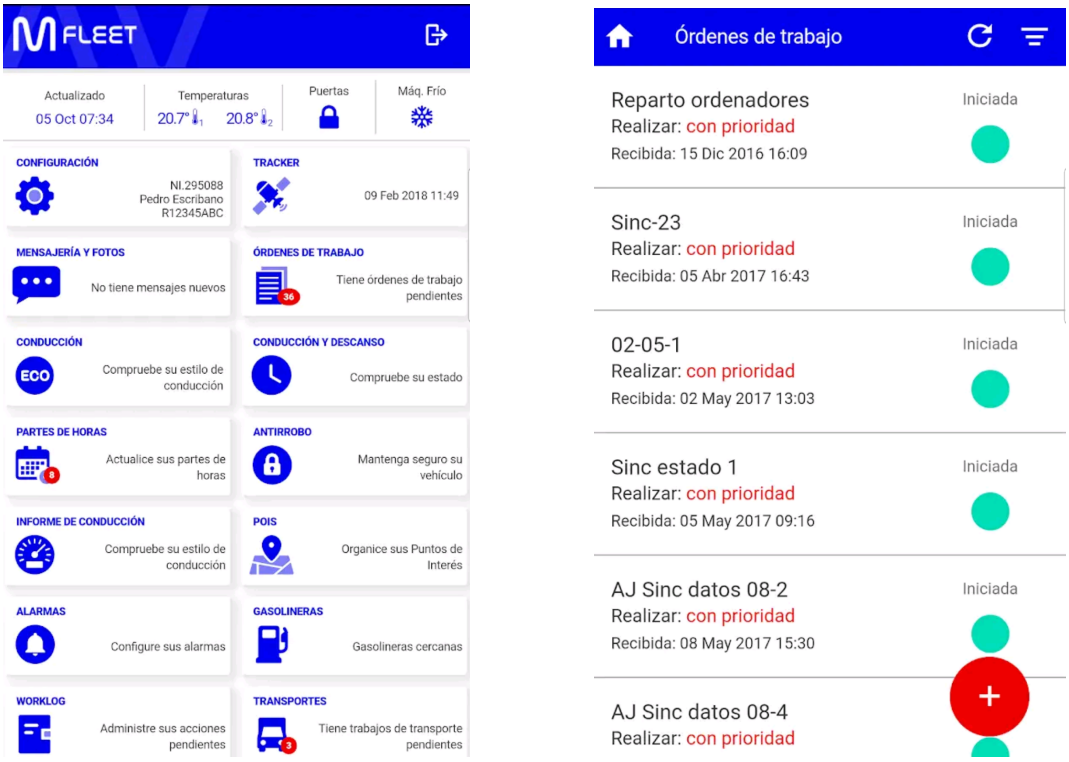
- **AHEAD Tool Suite (ATS)** [2]: conjunto de herramientas que dan soporte a la programación orientada a características (FOP, *Feature Oriented Programming*). AHEAD o *Algebraic Hierarchical Equations for Application Design* permite la representación de programas mediante expresiones algebraicas. Una característica fundamental de AHEAD

es que clasifica las características en constantes y funciones. Las constantes representan los programas base, que son implementados con clases e interfaces estándar. Las funciones representan las mejoras del programa, añaden una característica al programa recibido. Las características son implementadas como jerarquías de directorios y pueden contener múltiples artefactos distintos del código fuente. ATS provee herramientas personalizadas para diferentes artefactos los cuales son seleccionados por el compositor ATS, el elemento principal de AHEAD, de acuerdo con el tipo de artefacto. Todos los ficheros ATS son escritos en el lenguaje Jak (abreviatura de Jakarta); Jak es un lenguaje extendido de Java.

Tras el análisis realizado de las tecnologías existentes de herramientas de generación, se ha decido utilizar el entorno de generación spl-js-engine [3] ya que una de las desventajas tanto de AHEAD como de featureHOUSE es que el código generado es poco legible y no permite la fácil modificación de la estructura [4]. Con el motor de derivación spl-js-engine se genera un código claro para que sea extensible.

Por otro lado, en cuanto a aplicaciones de gestión de trabajadores en movilidad, hemos encontrado la siguiente aplicación:

- **MFleet**[5]: aplicación móvil que ofrece, por una parte, diferentes funcionalidades a los trabajadores en movilidad para la gestión de sus actividades y, por otra, al encargado de los trabajadores el control de su plantilla. A continuación, se detallan algunas de las funcionalidades que ofrece esta aplicación a los trabajadores:
  - Permite recibir órdenes de trabajo con los formularios correspondientes para cubrir los datos.
  - Permite la visualización de una lista con las órdenes de trabajo a realizar, indicando para cada una de ellas el estado de la tarea, la fecha y hora en la que fue recibida y la prioridad en la que debe ser realizada. Se puede ver un ejemplo en la Figura 2.2b.
  - Permite a los trabajadores responder a los mensajes enviados por el encargado además de confirmar la realización de las órdenes de trabajo.
  - Permite rellenar formularios para el registro de actividades y gastos.
  - Pueden consultar el tiempo que han estado conduciendo al igual que las próximas rutas y descansos que podrán realizar.
  - Permite a los trabajadores la visualización de puntos de interés cercanos a su ubicación actual.
  - Permite la consulta de datos sobre el tipo de conducción (uso de embrague, velocidad, etc.).



(a) Funcionalidades.

(b) Órdenes de trabajo

Figura 2.2: Pantallas de algunas de las funcionalidades de MFleet

  - Podrán cubrir datos correspondientes a su jornada laboral como el inicio y fin de esta.

MFleet proporciona, además de las funcionalidades mencionadas, otras como el reporte de gastos, mensajería, búsqueda de gasolineras cercanas, temperatura del remolque o tacógrafo, ver Figura 2.2a. Se ha considerado, por tanto, una aplicación compleja para el usuario ya que tiene demasiadas funcionalidades que para este proyecto no serían necesarias, ya que, nuestra aplicación se centra en proporcionar rapidez y facilidad de uso al usuario en la gestión de los eventos que este tenga que realizar.

2.2 Tecnologías utilizadas

A continuación, se enumeran las tecnologías utilizadas para la realización de este proyecto.

- **Vue.js** [6] es un framework progresivo para construir interfaces de usuario. Su núcleo es bastante pequeño y se escala a través de plugins. Engloba en un mismo archivo HTML, CSS y JavaScript.

- **Node.js** [7] es una plataforma de desarrollo para la creación de aplicaciones destinadas a la Web, orientada a redes y centrada en la velocidad y la escalabilidad.
- **React-Native** [8] es un framework que permite a los desarrolladores implementar aplicaciones nativas para dispositivos móviles utilizando JavaScript.
- **Axios** [9] es una librería JavaScript, basada en promesas, que permite la realización de peticiones como cliente HTTP de forma sencilla.
- **PostgreSQL** [10] es un sistema de gestión de bases de datos relacional orientado a objetos y de código abierto.
- **Expo** [11] es un framework y una plataforma para las aplicaciones universales de React. Es un conjunto de herramientas y servicios construidos entorno a las plataformas nativas y de React Native que ayudan a desarrollar, construir, desplegar e interar en iOS, Android y aplicaciones web con JavaScript y React.
- **spl-js-engine** [3] motor de derivación. Es una herramienta Javascript que se apoya en los formalismos de las Líneas de Producto Software proporcionando, además, ventajas de *scaffolding* generando código mantenible y permitiendo el uso de nuevas tecnologías.
- **Spring** [12] es un framework para el desarrollo de aplicaciones de código abierto basadas en Java.
- **Hibernate** [13] es un framework de mapeo objeto relacional, se encarga de la persistencia de los datos ya que se aplica a las bases de datos relacionales.

Capítulo 3

Metodología y planificación

3.1 Metodología de desarrollo

La metodología utilizada para el desarrollo de este proyecto es una metodología iterativa e incremental. Con esta metodología no es necesario realizar una implementación total del sistema global, sino que se pueden ir desarrollando implementaciones parciales con diversas funcionalidades e ir aumentando su implementación en cada una de las siguientes iteraciones o incrementos. Cada uno de estos incrementos, a su vez, puede consistir en la implementación de nuevas funcionalidades o en la mejora de los incrementos ya realizados. De esta forma, lo que se consigue es facilitar la incorporación de nuevos requisitos, en caso de que sean necesarios, al software ya implementado.

Algunas de las ventajas de esta metodología son:

- Obtención de incrementos operativos a lo largo del proceso de desarrollo.
- Reduce la posibilidad de que los requisitos del usuario/cliente cambien durante el desarrollo.
- La finalización de cada iteración permite decidir como mejorar el proceso de trabajo, dando lugar a la posibilidad de planificar qué cambios son necesarios para aumentar la productividad y calidad del sistema.
- Existe una mayor flexibilidad ante más funcionalidades.
- Permite reducir, desde el inicio, los riesgos del proyecto.
- La calidad y estabilidad del software progresan con las iteraciones, además, hay oportunidad para la mejora.

3.1.1 Scrum

Para el desarrollo de este proyecto no se ha escogido una metodología concreta sino que nos hemos basado en algunas de las buenas prácticas de Scrum [14], un marco de trabajo simple que promueve la colaboración en los equipos para lograr desarrollar productos complejos [15].

Scrum consta de un equipo, eventos y artefactos. A continuación, se realiza una breve descripción de lo que se ha llevado a cabo durante el desarrollo de este proyecto.

Equipo Scrum *(Scrum Team)*

En este proyecto, la alumna ha ejercido los roles de:

- ***Development Team*** o **equipo de desarrollo**: un proyecto real estaría formado por un grupo de expertos orientados al producto, pero en este caso, consta de una única persona ya que el proyecto se ha desarrollado de forma individual.

- **Scrum Master**: a medida que se desarrollaba el proyecto se ponían en práctica, y de forma correcta, los procesos de Scrum. En un proyecto que constase de un grupo de desarrolladores, esta tarea sería llevada a cabo por una única persona especialista de Scrum.

Por último, han sido tanto los directores del proyecto como la alumna los que han desempeñado el rol de ***Product Owner*** o **Dueño del producto**, es decir, los directores han fijado un proyecto con unas necesidades específicas y ha sido la alumna la encargada de redactar los elementos que formarían el Backlog del producto (*product backlog*) obtenidos de dichas necesidades.

Eventos Scrum

Para el desarrollo del proyecto, se han aplicado los siguientes eventos de Scrum:

- En primer lugar, se decidieron los **Sprints** en los que se dividiría el proyecto, es decir, cada una de las iteraciones de desarrollo. Para cada uno de estos sprints se establecía un tiempo de, aproximadamente, 4 semanas.

- En segundo lugar, después de haber establecido los sprints a realizar, se llevaba a cabo una **planificación del Sprint** (***Sprint Planning***) en la que se definían los objetivos que se debían alcanzar en el Sprint y cada una de las tareas del *product backlog* que se tendrían que realizar.

- Después de la finalización de cada uno de los sprints se acordaba con los directores del proyecto una **Revisión de Sprint** (***Sprint Review***) en la que se realizaba un análisis y se presentaba el incremento desarrollado. El tiempo máximo de estas reuniones era de 1 hora o 1 hora y media.

- Durante la presentación de los incrementos realizados en las reuniones, mencionadas anteriormente, tenía lugar la **Retrospectiva de Sprint** (***Sprint Retrospective***) en la que se identificaron las mejoras que se podían realizar en el desarrollo del siguiente Sprint.

Artefactos Scrum

Por último, se realiza una breve descripción de los elementos que se generaron tras la aplicación de Scrum.

- El primero de ellos, la **lista de productos** (***Product Backlog***) que contiene todos los requerimientos necesarios que se obtuvieron de las necesidades especificadas por los directores del proyecto, estos requerimientos se conocen como las historias de usuario.

- En segundo lugar, el ***Sprint Backlog***, este elemento contiene aquellas historias de usuario que se seleccionaron para llevar a cabo el desarrollo de un Sprint, lo que nos permitiría conseguir un incremento del producto terminado.

- Por último, el **Incremento** (***Increment***) que se basa en una suma con todos los elementos terminados que se han llevado a cabo durante el desarrollo de un Sprint.

3.1.2 Herramientas de soporte a la metodología

Las herramientas que se han utilizado, para la realización de las tareas, han sido las siguientes:

- **Gitlab** [16]: se ha utilizado la herramienta de control de versiones y desarrollo software.

- **Balsamiq Mockups**: herramienta para la creación de prototipos, bocetos o wireframes de un proyecto [17]. Utilizado para la realización de los prototipos de interfaces de la aplicación móvil.

- **Android Studio**: entorno de desarrollo integrado (IDE) oficial para el desarrollo de Android. Proporciona, entre otras cosas, un emulador rápido con múltiples funciones [18] utilizado para la ejecución del proyecto.

- **Eclipse**: entorno de desarrollo integrado (IDE) [19], utilizado para el desarrollo, en Java, del servicio REST de la aplicación.

- **Visual Studio Code**: editor de código [20], empleado para desarrollar, en React Native, la parte cliente de la aplicación.
- **PgAdmin**: plataforma de administración y desarrollo de bases de datos PostgreSQL [21]. Utilizada para almacenar los datos de prueba de la aplicación.
- **Postman**: plataforma de colaboración para el desarrollo de API [22]. Utilizada para acceder a los datos de prueba de la aplicación de forma fácil.
- **Npm**: sistema de gestión de paquetes [23]. Utilizado para la instalación de paquetes de la parte cliente.
- **Maven**: herramienta para la gestión de proyectos Java [24]. Utilizado para la implementación del servicio REST.
- **Overleaf**: editor colaborativo, online de LaTeX [25]. Utilizado para escribir este trabajo.
- **FeatureIDE**[26]: plugin de Eclipse utilizado para la creación del árbol de características o *feature model* de nuestro generador de aplicaciones.
- **Microsoft Teams**: plataforma que permite chatear, reunirse, colaborar y realizar llamadas entre personas de un equipo de trabajo[27]. Utilizada para la realización de reuniones con los directores del proyecto.
- **Microsoft Project**: software para la administración de proyectos [28]. Utilizado para la realización de diagramas de Gantt de planificación y seguimiento del proyecto.
- **Dia**: aplicación que permite la generación de diagramas [29]. Utilizada para el diseño del modelo de datos de la aplicación móvil.
- **draw.io**: aplicación online que permite la generación de diagramas [30]. Utilizada para el diseño de diagramas de paquetes y diagrama de componentes de la arquitectura del sistema.

3.2 Metodología de desarrollo de la Línea de Producto Software

El objetivo de una LPS es la generación de un conjunto de productos que comparten una serie de características o *features* comunes. El desarrollo de esta LPS permitirá la reutilización de aplicaciones [31]. La adaptación del producto a las necesidades indicadas por el usuario se verá afectada, en gran medida, por la variabilidad de la LPS.

El motor de derivación para la LPS utilizado en este proyecto es el spl-js-engine [3]. Este motor de derivación está basado en la técnica de *scaffolding*. El *scaffolding* se compone de diferentes plantillas que permiten la rápida compilación para la generación de código repetitivo.

El compilador utilizará, además, una especificación proporcionada por parte del desarrollador. Para el funcionamiento del motor de derivación son necesarios tres componentes: el gestor del modelo de variabilidad, el motor de plantillas y el gestor de ficheros [3].

Para el desarrollo de la LPS se llevaron a cabo las siguientes fases:

- **Estudio del dominio**: se realizó un análisis de las características comunes e identificación de las variabilidades.
- **Diseño del modelo de variabilidad**: las características definidas se representaron en una estructura jerárquica, con la herramienta FeatureIDE [26], permitiéndonos una representación de los productos posibles de una LPS en función de las características seleccionadas y las relaciones existentes entre ellas. Se puede ver el árbol de características realizado en la Figura 4.1.
- **Implementación de las características**: en esta fase se llevó a cabo la selección de las características que posteriormente serían implementadas en programas.
- **Análisis de producto**: se parte de un conjunto de requisitos concretos indicados por el cliente, que tienen que ver con los requisitos de la línea de producto software. Puede haber alguna parte nueva que tendrá que ser desarrollada, posteriormente, para el producto concreto.
- **Diseño del producto**: se realizó un diseño específico del producto concreto que cumpliera con los requisitos obtenidos durante el análisis.
- **Implementación del producto**: codificación del producto para la generación del producto final en función de las características seleccionadas.

Una vez desarrollado el producto, se llevó cabo el proceso de anotación del código fuente junto con la implementación de la interfaz de configuración de este, para la posterior selección de características y formularios que permitiesen al usuario la generación de productos.

La representación de esta metodología está reflejada en la Figura 3.1.

3.3 Planificación y seguimiento

En esta sección se realizará una breve descripción de las tareas y recursos necesarios para el desarrollo de este proyecto.

3.3.1 Planificación

La primera fase que se llevó a cabo fue un **estudio de las tecnologías**, mencionadas en el Apartado 2.2, que se iban a emplear en el desarrollo del proyecto. En la segunda fase se

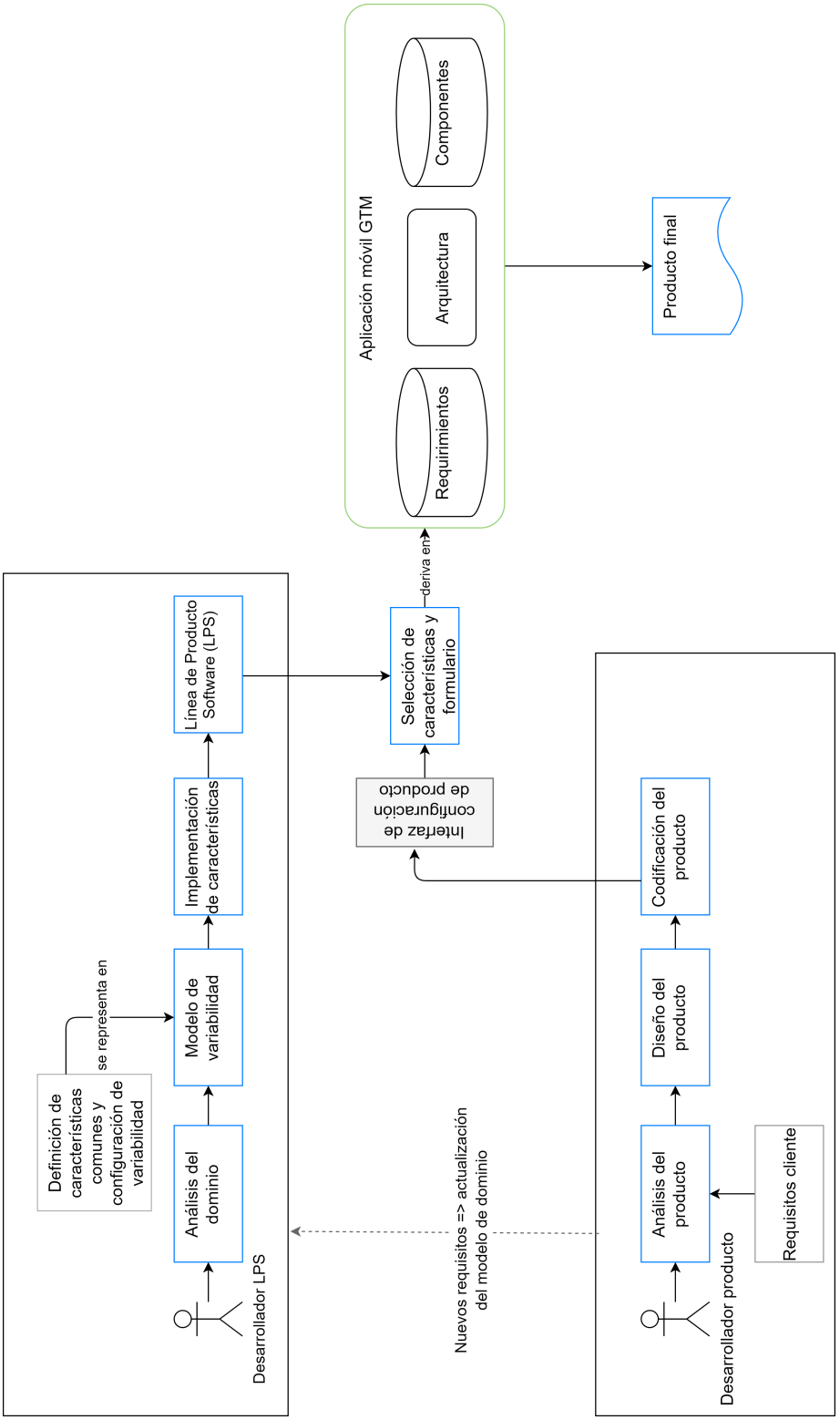


Figura 3.1: Metodología de desarrollo de la Línea de Producto Software

realizó un **análisis preliminar del dominio** donde se llevaron a cabo las siguientes tareas:

- Se determinaron las **historias de usuario** que formarían el *product backlog*, se han producido algunas modificaciones de las funcionalidades durante el desarrollo de la aplicación, pero el hecho de usar una metodología incremental da soporte a estos cambios.

- Se elaboraron los **mockups** o prototipos de pantalla que debería tener la aplicación, en función de las historias de usuario definidas.

- Se definió un **modelo de datos**, en base a las historias de usuario descritas, donde se establecían las entidades que se debían almacenar en la base de datos junto con los atributos correspondientes.

- Se implementó un **servicio web** que diera soporte a las necesidades descritas en las historias de usuario.

Después del análisis preliminar y realizada la estimación de la fecha de fin de proyecto, se dividió el tiempo en periodos similares para cada uno de los sprints que se tenían que llevar a cabo. En un principio, se hizo un estimación de cuatro sprints de, aproximadamente, tres semanas cada uno. Los dos primeros se establecieron para el desarrollo de la aplicación móvil, cada uno con sus correspondientes historias de usuarios y los dos siguientes para el desarrollo de la Línea de Producto Software dejando las tres semanas restantes para la elaboración de la memoria del proyecto. En la Figura 3.2a se muestra el diagrama de Gantt con esta planificación inicial.

3.3.2 Recursos

Para la elaboración de este proyecto distinguimos, en cuanto a recursos:

- **Humanos**: en este proyecto han intervenido los dos directores de este trabajo de fin grado y la alumna. Los directores desenvuelven el rol de jefes de proyecto, encargados de realizar la planificación y las revisiones del mismo asegurándose del cumplimiento de los requisitos y necesidades establecidas. Una vez conocida la planificación del proyecto, el desarrollador o analista programador, en este caso la alumna, se encarga del diseño, implementación y pruebas de la aplicación.

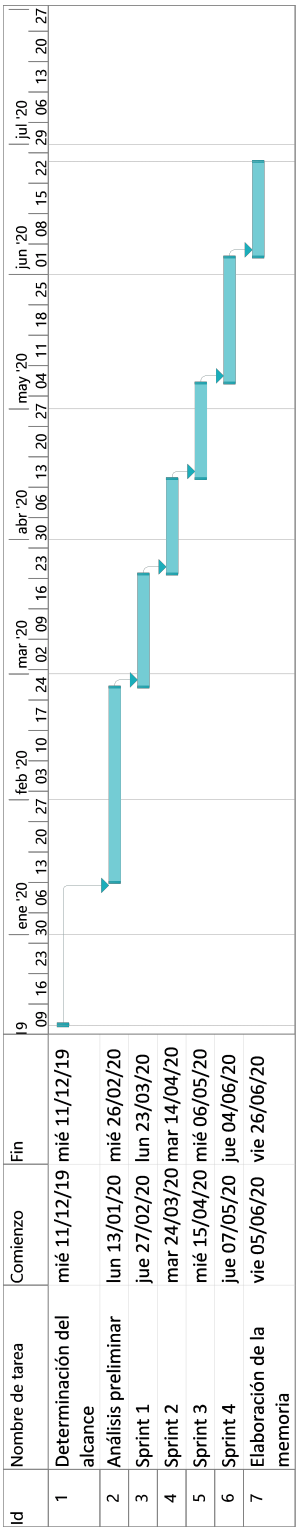
- **Técnicos**: se empleó un ordenador portátil y un dispositivo móvil personal para llevar a cabo el desarrollo del proyecto y la ejecución del mismo. Además de la utilización de las herramientas ya descritas en el Apartado 3.1.2.

3.3.3 Seguimiento

En esta sección se realiza la planificación real del proyecto, explicando las tareas que se llevaron a cabo en cada uno de los sprints representados en el diagrama de Gantt de la Figura 3.2b.

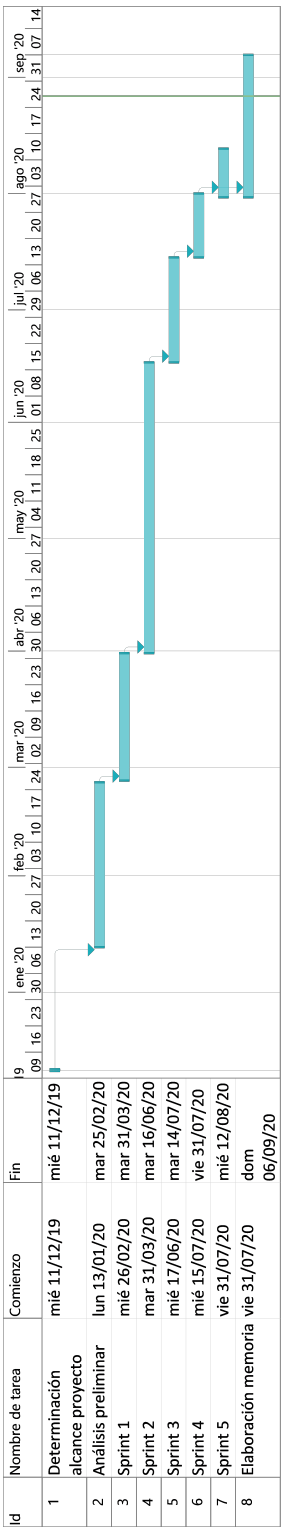
A continuación, se detallan las tareas establecidas para cada uno de los sprints.

- **Sprint 1**: periodo en el que tuvo lugar el desarrollo de las historias de usuario relacionadas con la visualización de los distintos tipos de listados de eventos, el detalle de un evento, de un cliente y el acceso a un mapa. Junto con el desarrollo de la parte del servidor.
- **Sprint 2**: periodo establecido para el desarrollo de las historias de usuario restantes, tanto para la parte cliente como la parte servidor.
- **Sprint 3**: realizar las anotaciones necesarias en el código e invocar el proceso de generación de productos utilizando un cliente terminal comprobando que se generaban de forma correcta.
- **Sprint 4**: desarrollo de la interfaz para la generación de productos permitiendo al usuario la descarga de un fichero *zip* con el código fuente del producto, la importación y exportación del fichero de especificación del producto generado y la selección de ficheros *json* con el esquema JSON de un formulario.
- **Sprint 5**: implementación de la interfaz para la generación de formularios, en vez de seleccionar un archivo que contenga el esquema JSON correspondiente, y almacenar dichos formularios en la especificación del producto.



Id	Nombre de tarea	Comienzo	Fin
1	Determinación del alcance	mié 11/12/19	mié 11/12/19
2	Análisis preliminar	lun 13/01/20	mié 26/02/20
3	Sprint 1	jue 27/02/20	lun 23/03/20
4	Sprint 2	mar 24/03/20	mar 14/04/20
5	Sprint 3	mié 15/04/20	mié 06/05/20
6	Sprint 4	jue 07/05/20	jue 04/06/20
7	Elaboración de la memoria	vie 05/06/20	vie 26/06/20

(a) Planificación inicial del proyecto



Id	Nombre de tarea	Comienzo	Fin
1	Determinación alcance proyecto	mié 11/12/19	mié 11/12/19
2	Análisis preliminar	lun 13/01/20	mar 25/02/20
3	Sprint 1	mié 26/02/20	mar 31/03/20
4	Sprint 2	mar 31/03/20	mar 16/06/20
5	Sprint 3	mié 17/06/20	mar 14/07/20
6	Sprint 4	mié 15/07/20	vie 31/07/20
7	Sprint 5	vie 31/07/20	mié 12/08/20
8	Elaboración memoria	vie 31/07/20	dom 06/09/20

(b) Seguimiento del proyecto

Figura 3.2: Diagramas de Gantt

Capítulo 4

Análisis de la línea de producto software

Para este proyecto no se ha realizado, únicamente, un análisis de un producto concreto sino de un conjunto de ellos, por eso se diferenciará entre el análisis de la Línea de Producto Software y el análisis de un producto concreto.

4.1 Requisitos del producto

4.1.1 Actores

Esta aplicación ha sido pensada para los trabajadores en movilidad de una empresa, para ello deben estar registrados en la aplicación, ya que solo tendrán acceso a la información de los eventos que le han sido asignados introduciendo sus credenciales. Por esto motivo, se ha decido que el único actor de esta aplicación sea el **usuario registrado**.

4.1.2 Requisitos

Este proyecto consta de una serie de características para satisfacer las necesidades del cliente. Se distinguen dos tipos, los requisitos funcionales y los no funcionales.

- Los **requisitos funcionales** son los que describen la funcionalidad de la aplicación y la interacción entre la misma y el entorno, independientemente de su implementación. En el Cuadro 4.1 se muestran las funcionalidades que constituyen el *product backlog* del proyecto.

- Los **requisitos no funcionales** no hacen referencia, directamente, a las funciones específicas que proporciona el sistema sino a las cualidades que debe tener el producto. Para esta aplicación se consideró necesario proporcionar al usuario **rapidez y facilidad**

de uso haciendo posible que el trabajador no tenga que cubrir en papel los datos correspondientes a un evento y las incidencias del mismo, ya que se ofrecen formularios para ello, esto deriva en un ahorro de tiempo para el trabajador. Otra de las cualidades es que se considera **intuitiva**, puede acceder a las múltiples funcionalidades de la aplicación sin ninguna complejidad.

ID	Nombre	Descripción
HU-1	Autenticarse	Proceso de certificación para acceder a la aplicación.
HU-2	Inicio	Permite al usuario acceder al menú de la aplicación.
HU-3	Cerrar sesión	Proceso de finalización de la sesión.
HU-4	Ver evento actual	Permite al usuario acceder a los datos del evento actual.
HU-5	Ver todos los eventos	Permite al usuario visualizar todos sus eventos, independientemente del estado de los mismos.
HU-6	Ver eventos planificados	Permite al usuario la opción de ver los eventos pendientes del día actual o los de una fecha concreta.
HU-7	Ver eventos realizados	Permite al usuario visualizar una lista con los eventos ya realizados.
HU-8	Ver eventos cancelados	El usuario tendrá acceso a una lista con los eventos que se le han cancelado.
HU-9	Acceder al mapa	Permite, al usuario, el acceso a un mapa con la ubicación de los eventos.
HU-10	Iniciar navegación	Proceso que permite al usuario obtener una ruta para llegar al destino de un evento.
HU-11	Visualizar detalle evento	Permite al usuario visualizar la información correspondiente a un evento.
HU-12	Visualizar detalle cliente	Permite al usuario acceder a la información del cliente de un evento.
HU-13	Acceder al mapa de un evento	Permite al usuario visualizar la ubicación de un evento concreto.
HU-14	Crear incidencia	Permite al usuario el acceso a un formulario para almacenar los datos de una incidencia.

HU-15	Cubrir formulario de datos	Permite al usuario el acceso a un formulario para almacenar la información correspondiente a un evento.
HU-16	Guardar evento	Proceso que permite al usuario almacenar la información del evento que haya realizado o modificado.
HU-17	Ver lista de incidencias	Permite al usuario la visualización de una lista que contiene todas las incidencias ocurridas.
HU-18	Ver lista de incidencias de un evento	Permite al usuario la visualización de una lista con las incidencias ocurridas en un evento concreto.
HU-19	Eliminar incidencia	Proceso que permite al usuario eliminar los datos de una incidencia concreta.
HU-20	Editar incidencia	Proceso que permite al usuario modificar los datos de una incidencia concreta.
HU-21	Ver detalle incidencia	Permite al usuario ver la información de una incidencia concreta.

Cuadro 4.1: Requisitos funcionales de la aplicación.

A continuación, se detallan cada una de las funcionalidades mencionadas en el Cuadro 4.1. Se hace una descripción de las historias de usuario en esta sección ya que es la LPS la que tienen todas las funcionalidades disponibles para que, posteriormente, sea el analista el que seleccione las necesarias.

- **HU-1: Autenticarse**. El usuario debe introducir su login y password. Una vez que el sistema compruebe que las credenciales son válidas, podrá acceder a la aplicación y a todas sus funciones. En caso contrario, se mostrará un mensaje de error indicando que las credenciales son incorrectas.

- **HU-2: Inicio**. El usuario tendrá acceso al menú de la aplicación, a través de él podrá acceder a las diferentes funcionalidades de la aplicación. Desde la pantalla principal tendrá acceso al detalle de los dos primeros eventos pendientes de la fecha actual, además de un enlace para ver la lista de los mismos.

- **HU-3: Cerrar sesión**. Una vez que el usuario haya terminado de utilizar la aplicación, tendrá la opción de terminar su sesión.

- **HU-4: Ver evento actual**. Desde el menú de inicio de la aplicación el usuario tendrá acceso directo al primer evento pendiente correspondiente a la fecha actual.
- **HU-5: Ver todos los eventos**. Desde el menú de inicio de la aplicación el empleado tendrá un acceso a una lista con todos los eventos, independientemente del estado del evento. Estos se mostrarán en una tabla indicando el nombre del evento, la fecha en la que tuvo lugar el evento y la hora de inicio del mismo. A través del nombre del evento se podrá visualizar el detalle del evento (HU-11).
- **HU-6: Ver eventos planificados**. El usuario podrá acceder a esta funcionalidad a través de la pantalla principal o del menú de inicio. Se mostrará, inicialmente, una lista con los eventos correspondientes a la fecha actual. Además, tendrá acceso a un calendario donde se marcarán las fechas con eventos asociados, pudiendo seleccionar la que le interese para visualizar una lista con dichos eventos.
- **HU-7: Ver eventos realizados**. Se mostrará al usuario una lista con los eventos que haya realizado donde se indicará la hora de inicio del mismo. Además de un enlace para poder visualizar la información correspondiente (HU-11 Visualizar detalle evento).
- **HU-8: Ver eventos cancelados**. Se mostrará al usuario una lista con los eventos que se le asignaron pero que han sido cancelados, podrá acceder a la información del mismo mediante un enlace (HU-11 Visualizar detalle evento). No podrá realizar modificaciones en los eventos con estado cancelado.
- **HU-9: Acceder al mapa**. En cada una de las lista de eventos, independientemente del estado del mismo, el usuario tendrá opción de visualizar su ubicación actual y la ubicación de los eventos en un mapa. Dependiendo del estado, el marcador de la ubicación se mostrará en un color diferente (rojo para los cancelados, amarillo para los pendientes y verde para los realizados), a través del popup del evento seleccionado podrá acceder a la información del mismo.
- **HU-10: Iniciar navegación**. En el mapa de cada uno de los eventos, el usuario tendrá la opción de iniciar la navegación. Para ello, se ha utilizado la navegación de Google Maps, es decir, en el momento en el que el usuario seleccione esta opción, se le abrirá la aplicación de Google Maps donde se mostrará la ruta del evento desde su ubicación.
- **HU-11: Visualizar detalle evento**. Una vez que el usuario accede a la información del evento se le mostrará la fecha en la se realizó o se debe realizar, un enlace para acceder a los datos del cliente, una descripción que indique en que consiste el evento, un acceso a un mapa que mostrará tanto su ubicación como la del evento, las horas en la que debería comenzar y finalizar el evento, dos campos para introducir las horas reales de iniciación

y finalización,el estado del evento (realizado, pendiente o cancelado) y, por último, tres accesos que permitirán al usuario la ejecución de tres historias de usuario. La primera de ellas, un acceso a un formulario para almacenar los datos correspondientes al evento, la segunda, un acceso a un formulario para almacenar las incidencias que le ocurran en la ejecución del evento y por último, un acceso a un listado de incidencias que le ocurrieron al usuario en la realización del evento. Finalmente, podrá ir visualizando uno a uno los eventos anteriores y siguientes al seleccionado.

- **HU-12: Visualizar detalle cliente**. Esta historia de usuario permitirá la visualización de la información del cliente correspondiente a un evento. Se mostrará el nombre completo, el email y el número de teléfono.

- **HU-13: Acceder al mapa de un evento concreto**. Se mostrará al usuario un mapa con su ubicación actual y la ubicación del evento concreto. En el popup del marcador se indicará el nombre del mismo. Además tendrá la posibilidad de iniciar la navegación (HU-10: Iniciar navegación).

- **HU-14: Crear incidencia**. Esta funcionalidad permitirá al usuario acceder a un formulario para almacenar la siguiente información: el tipo de incidencia (se mostrará un listado con los tipos de incidencia) y una descripción de la misma.

- **HU-15: Cubrir formulario de datos**. Esta historia de usuario permitirá al usuario almacenar los datos correspondientes al evento. El evento tendrá asignado un tipo de formulario, dependiendo del tipo, tendrá que almacenar unos datos u otro. Para las pruebas de la aplicación se emplearon tres tipos distintos de formularios.

- **HU-16: Guardar evento**. Proceso para almacenar la información de un evento. Cuando el empleado haya terminado de cubrir los datos del evento, podrán ser almacenados. El estado del evento se modificará de forma automática en el momento en el que se guarde el evento.

- **HU-17: Ver lista de incidencias**. En caso de que exista alguna incidencia almacenada en base de datos se mostrará al usuario una tabla con el siguiente contenido: el tipo de la incidencia que, a su vez, será un enlace para ver el detalle de la misma (HU-21: Ver detalle incidencia), la fecha en la que ocurrió, el evento al que está asociado dicha incidencia (enlace para acceder a su información, HU-11: Visualizar detalle evento) y las acciones que puede llevar a cabo que serán eliminar o editar incidencia. Si no hay ninguna incidencia almacenada, se mostrará una notificación indicándolo.

- **HU-18: Ver lista de incidencias de un evento**. Si el evento no tiene ninguna incidencia relacionada se mostrará una notificación. En caso contrario, se mostrará una tabla

con el tipo de incidencia que será un enlace para acceder a su información (HU-21: Ver detalle incidencia), el día en el que tuvo lugar la incidencia y las acciones que podrá realizar que serán eliminarla o editarla.

- **HU-19: Eliminar incidencia de un evento**. Con esta funcionalidad se eliminará de base de datos la información correspondiente a la incidencia de un evento concreto.
- **HU-20: Editar incidencia**. Esta funcionalidad permite al usuario modificar los datos correspondientes a una incidencia concreta, se mostrará un formulario con el tipo y descripción de la misma.
- **HU-21: Ver detalle incidencia**. Se mostrará al usuario un formulario con el tipo y la descripción de la incidencia.

4.2 Modelo de variabilidad

Para la descripción de una línea de producto software es necesario establecer un modelo de características o *feature model* en el que se definen las características comunes u obligatorias (*mandatory features*) y las variables u opcionales (*optional features*) del conjunto de productos pertenecientes a la LPS [32]. Este árbol de características es representado en una estructura jerárquica. Cada producto, de la LPS, estaría representado por un conjunto de características únicas que lo definen. Además de las características, el modelo de variabilidad permite la definición de restricciones y de los grupos de multiplicidad *or group* y *alternative group*.

- Las características obligatorias o ***mandatory features*** permiten la definición del dominio. Están presentes en todos los productos de la LPS.
- Las características variables u ***optional features*** definen la variabilidad de una LPS. No están presentes en todos los productos, varían de uno a otro.
- Los grupos de multiplicidad ***or group*** y ***alternative group*** indican el número, mínimo y máximo respectivamente, de características que debe tener un producto.

Para la construcción del modelo de variabilidad de este proyecto se utilizó la herramienta FeatureIDE [26] mencionada en el Apartado 3.1.2. Se puede ver la representación de nuestro modelo de características en la Figura 4.1. La característica que se consideró obligatoria para todos los productos fue la visualización del evento actual junto con el detalle del cliente. En cuanto a los grupos de multiplicidad, se definió un *or group* para la selección del listado con los tipos de eventos que se desea visualizar en la aplicación. Además, se estableció una restricción en cuanto a la visualización del mapa, es decir, para la selección de esta característica es necesario seleccionar algún tipo de listado de evento.

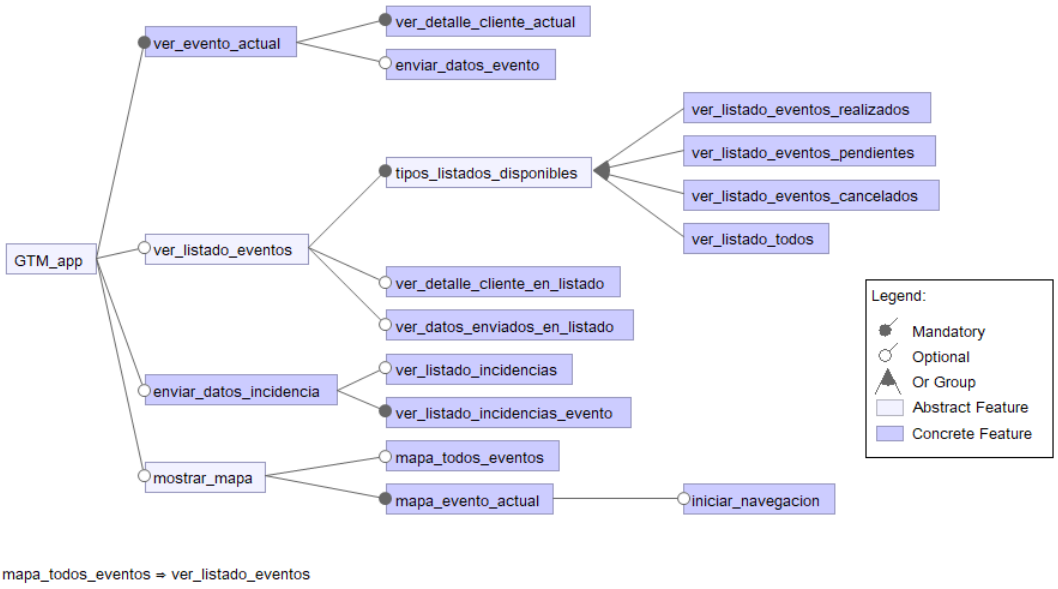


Figura 4.1: Árbol de características o *feature model* de nuestro generador de aplicaciones.

4.3 Requisitos de la herramienta de generación

4.3.1 Actores

usuario objetivo de esta aplicación: encargado de utilizar esta herramienta para la generación del producto.

4.3.2 Requisitos

En cuanto a los requisitos de la herramienta de generación también distinguimos entre requisitos funcionales y no funcionales.

- **Requisitos funcionales**. La herramienta de generación presenta las siguientes funcionalidades:
  - **Selección de características**. El usuario será el encargado de seleccionar las funcionalidades que considere necesarias.
  - **Descargar fichero *zip***. Se permite, al usuario de la aplicación, la descarga de un fichero *zip* con el código del producto final.
  - **Exportar e importar fichero de especificación**. El usuario podrá descargar o cargar un fichero con la especificación del producto.

  - **Generación de formularios**. Se ofrece al usuario una interfaz para la generación de formularios que se mostrarán en la aplicación para almacenar los datos correspondientes a un evento.
  - **Visualización de formularios**. El usuario podrá visualizar la lista de formularios que ha creado anteriormente junto con los que haya definidos en la especificación del producto, en caso de que se haya importado.

- **Requisitos no funcionales**. Una de las cualidades de la herramienta es la facilidad de uso, ya que se le permite al usuario la generación de formularios y la selección de características a través de la interfaz, pero también la importación o exportación del fichero de especificación si lo desea.

A continuación se detalla cada una de las funcionalidades mencionadas que forman el *product backlog*.

- **Selección de características**: se mostrará al usuario una lista con las características del árbol de características de la Figura 4.1. Podrá seleccionar las que considere necesarias para la generación del producto. Las características se mostrarán en forma jerárquica con su checkbox correspondiente para marcarla o desmarcarla si lo necesita. Para los grupos de multiplicidad, en nuestro caso para el *or group*, se mostrará una notificación indicando que al menos una de las características que contiene debe ser seleccionada. El gestor del modelo de variabilidad se encarga de la validación de la selección.

- **Descargar fichero *zip***: el usuario podrá descargar un fichero *zip* con el código del producto final que se ha generado a partir del código fuente con anotaciones y de la especificación indicada por el usuario.

- **Exportar e importar fichero de especificación**: se permite, al usuario de la aplicación, la descarga del fichero de especificación que se genera a partir de la selección de características para, posteriormente, volver a cargarlo en caso de que sea necesario. En esta especificación podrá indicar, además, una lista con el esquema JSON de cada uno de los formularios que desea que tenga la aplicación para almacenar los datos correspondientes a un evento. Este fichero se descargará en formato JSON.

- **Generación de formularios**: se ofrece al usuario una interfaz en la que podrá seleccionar las propiedades del esquema JSON de los formularios. El formulario tendrá que tener un título y una o varias propiedades, para todas ellas deberá indicar el tipo, que lo seleccionaría de una lista de tipos de propiedades (string, integer, real, etc.), el nombre (que actuaría como identificador de la propiedad) y la etiqueta (título de la propiedad

que se mostraría en el formulario visto desde la aplicación móvil). Por último, dependiendo del tipo de propiedad que seleccione tendrá que indicar otros campos a mayores. Los formularios creados se guardarán dentro de la especificación del producto.

- **Visualización de formularios**: se le proporciona al usuario una interfaz para la visualización de los formularios que haya realizado o que se hayan importado en la especificación. Para cada uno de ellos se mostrará el título y tendrá la opción de eliminarlos de la lista y, por tanto, de la especificación o editarlos, modificando el título y cada una de las propiedades existentes.

Capítulo 5

Construcción del producto

5.1 Análisis

5.1.1 Arquitectura del sistema

En nuestra arquitectura se diferencian la presentación al usuario o vista, la capa de lógica de negocio o controlador y la capa de acceso a datos o modelo. Esta arquitectura es conocida como la **arquitectura en tres capas**. El uso de esta arquitectura se encarga de obligarnos a ubicar cada una de las capas en un servidor independiente. Se trata de una arquitectura rígida, es decir, cada capa conoce, únicamente, la capa inmediatamente inferior. En la Figura 5.1 se puede ver la representación del diagrama de componentes de la arquitectura de nuestro sistema.

A continuación, se hará una breve descripción de cada una de las capas mencionadas anteriormente.

- **Presentación al usuario / vista**: se presentan los datos al usuario a través de una interfaz.
- **Capa lógica de negocio / controlador**: se encarga de gestionar la lógica de la aplicación. Recibe las peticiones realizadas por el cliente determinando las acciones que se deben llevar a cabo para cada petición. También se encarga de enviar la respuesta de la petición.
- **Capa de acceso a datos / modelo**: se encarga de almacenar los datos en base de datos o enviarlos a la capa de lógica de negocio cuando el usuario realice alguna petición.

Algunas de las ventajas de esta arquitectura son la escalabilidad y la flexibilidad debido a que cada una de las tres capas es asignada a un servidor independiente, esto facilita que si es necesario realizar alguna modificación no tiene por qué afectar al resto de capas.

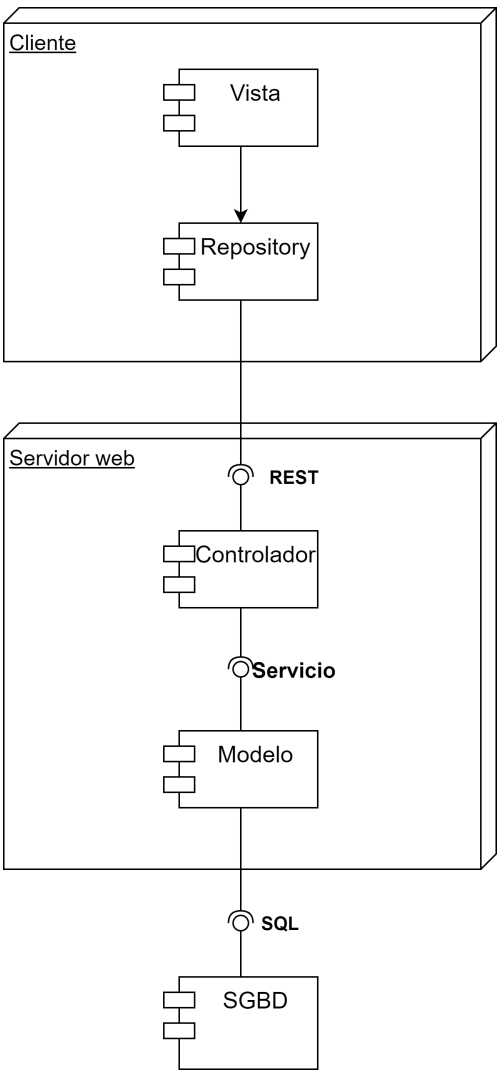


Figura 5.1: Diagrama de componentes de la arquitectura del sistema.

5.1.2 Interfaz de usuario

La interfaz de usuario permite al cliente la visualización de los datos y la interacción con la aplicación que se ha desarrollado.

A continuación, se mostrarán algunos de los mockups realizados explicando su funcionamiento. Ha de tenerse en cuenta que las figuras que se irán mostrando con los mockups son los prototipos de pantalla que se han realizado en el análisis preliminar del proyecto y, por tanto, probablemente se hayan realizado diversos cambios a medida que se avanzaba en el desarrollo de la aplicación. Se ha intentado, en todo momento, que los cambios no provocaran al usuario complejidad para utilizar la aplicación. Se pueden visualizar todos los prototipos realizados en el Apéndice B.

- **Inicio**. El primer prototipo de pantalla que se detallará será el de la pantalla de inicio, como se puede ver en la Figura 5.2 se muestra al usuario el evento actual, el siguiente y el número de eventos que tendrá que realizar en la fecha actual. Ambos eventos son un enlace para que el usuario pueda acceder, de forma rápida, a la información del mismo. Además, el número de eventos es un enlace a una lista con los eventos pendientes.

  También se muestra el menú de navegación de la aplicación con accesos directos a los diferentes tipos de listados disponibles, a la lista de todas las incidencias, independientemente del evento, al cierre de sesión (se mostrará una pantalla para confirmar que desea terminar su sesión) y a la página de inicio ya descrita.

- **Lista de eventos pendientes**. El usuario tendrá dos opciones para ver la lista de eventos planeados como se muestra en la Figura 5.3. Inicialmente, se mostrarán los eventos de la fecha actual (Figura 5.3a), pudiendo desplazarse a la lista de los eventos de las fechas anteriores y posteriores de una en una, mostrando dicha fecha. En la figura Figura 5.3b, se puede observar que la otra opción de la que dispone el usuario es la visualización de un calendario donde se mostrará el mes actual con el día seleccionado. Actualmente, se han marcado también las fechas para las que el usuario tiene algún evento asignado. A continuación del calendario, se mostrará un botón que le dará acceso a la visualización de los eventos en un mapa. Y, por último, se mostrará la lista de los eventos de la fecha que el usuario ha seleccionado.

- **Detalle del evento**. La Figura 5.4a muestra la información que debe contener un evento concreto. En primer lugar, dos botones para poder acceder a la información de los eventos anteriores y posteriores y un acceso a un mapa para visualizar la ubicación del evento. En segundo lugar, se mostrará el nombre del cliente al que está asociado dicho evento que, a su vez, será un enlace para que el usuario pueda ver los datos de este. A continuación, una descripción indicando en que consiste el evento, es decir, lo que debe realizar el usuario, después se mostrará el estado del evento que, en el análisis preliminar, debía ser modificado por el usuario antes de guardar los datos (actualmente se modifica automáticamente) y, posteriormente, se muestran dos campos numéricos para introducir las horas de iniciación y finalización del evento (actualmente podrá visualizar las horas planeadas, también). Por último, tendrá tres botones que le permitirán, acceder a la lista de incidencias de ese evento, la visualización de un formulario para cubrir la información de una nueva incidencia y otro formulario para almacenar los datos correspondientes al evento.

- **Lista de incidencias de un evento**. Muestra al usuario una tabla con las incidencias ocurridas en un evento determinado, indicando, para cada una de ellas, la fecha en la que

Figura 5.2: Mockup de la pantalla de inicio de la aplicación.

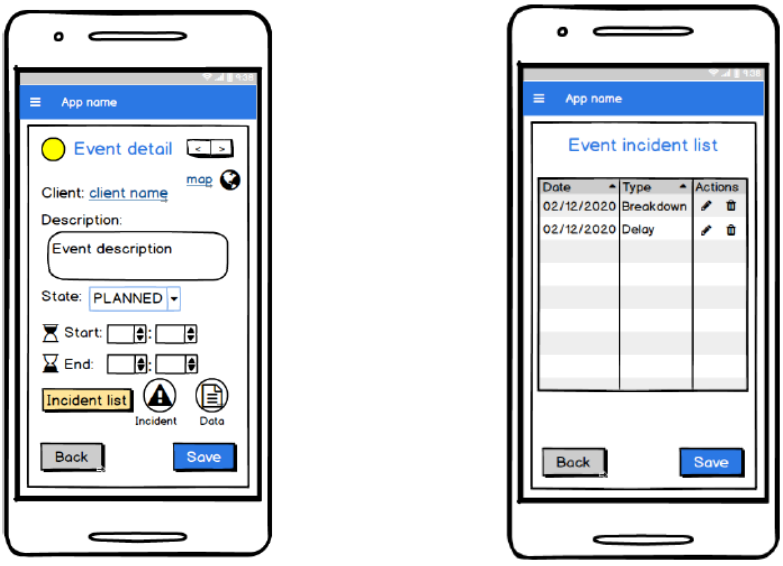
ocurrió y las acciones que puede llevar a cabo que son eliminar o editar dicha incidencia. Se puede ver el prototipo de esta funcionalidad en la Figura 5.4b.

- **Cubrir formulario de datos**. Para esta funcionalidad se ha decido mostrar dos de los tres ejemplos de formularios que se han realizado para esta aplicación.
  - El primero de ellos, se puede ver en la Figura 5.5a, se ha pensado para la entrega o recogida de algún tipo de producto, este formulario consta, por tanto, de dos campos de texto en los que debe indicar la actividad que ha llevado a cabo y el producto o productos que ha recogido/entregado y un campo numérico para indicar la cantidad de estos.
  - El segundo formulario, sin embargo, ha sido pensado para el ámbito de enfermería, el trabajador debe indicar el tipo de actividad que ha realizado, seleccionándola de una lista, (en este ejemplo se trata de una recogida de muestra de sangre), los tests requeridos (al menos uno) y la cantidad de tubos que ha sido necesario extraer. Se puede ver este ejemplo en la Figura 5.5b. Los datos que debe cubrir dependerán del tipo de actividad que tenga que llevar a cabo.

(a) Opción día.

(b) Opción mes.

Figura 5.3: Mockups lista de eventos pendientes.



(a) Información de un evento.

(b) Lista de incidencias de un evento.

Figura 5.4: Mockups con el detalle de un evento y sus incidencias.

(a) Mockups formularios de ejemplo 1

(b) Mockup formulario de ejemplo 2

Figura 5.5: Mockups con formularios de ejemplo para cubrir los datos correspondientes a un evento.

5.1.3 Modelo conceptual de datos

El modelo de datos nos permite realizar una estructura organizada de los datos que será necesario almacenar en nuestra base de datos.

A continuación, se hará una breve descripción de cada una de las entidades presentadas en el modelo de datos de la Figura 5.6.

- ***Employee***: entidad para representar a los usuarios de la aplicación, serán los trabajadores en movilidad. Cada empleado podrá tener asignado más de un evento. Se almacenarán los siguientes atributos:
  - *fullName*: nombre completo del empleado.
  - *email*: dirección de correo electrónico del empleado.
  - *phone*: número de teléfono móvil del empleado.
  - *label*: etiqueta de identificación del empleado.
  - *location*: coordenadas de ubicación del empleado.
  - *username*: nombre de usuario para autenticarse en el sistema.
  - *password*: contraseña para autenticarse, junto con el nombre de usuario en el sistema.

- ***Client***: entidad para representar a los clientes que tienen asociado o no algún evento. Atributos:
  - *fullName*: nombre completo del cliente.
  - *email*: dirección de correo electrónico del cliente.
  - *phone*: número de teléfono móvil del cliente.
  - *label*: etiqueta de identificación del cliente.
  - *location*: coordenadas de ubicación del cliente.
- ***Event***: entidad para representar cada uno de los eventos que tendrá que realizar un empleado. Un evento estará asignado a un único empleado.
  - *label*: etiqueta/título para identificar el evento.
  - *date*: fecha en la que se debe realizar el evento.
  - *description*: breve descripción de lo que debe realizar el empleado a la hora de llevar a cabo el evento.
  - *state*: estado en el que se encuentra el evento (planeado, cancelado o realizado).
  - *planned init*: hora de estimación en la que se debe iniciar la tarea.
  - *planned end*: hora de estimación en la que se debe haber finalizado la tarea.
  - *real init*: hora real en la que el empleado ha iniciado la tarea.
  - *real end*: hora real en la que el empleado ha terminado la tarea.
  - *location*: coordenadas de la ubicación del evento, lugar al que se tiene que desplazar el empleado. No son las mismas que las del cliente.
  - *data*: datos correspondientes a lo que ha tenido que llevar a cabo el empleado durante la realización del evento.
  - *type*: tipo de formulario que tiene asociado el evento. Se han definido tres para las pruebas. Serán indicados por el usuario de la herramienta de generación del producto.
- ***Incident***: entidad para representar las incidencias ocurridas durante la realización de un evento. Los datos que se tendrán que almacenar son:
  - *type*: tipo de incidencia ocurrida (*BREAKDOWN, DELAY, OTHER).*
  - *description*: breve descripción de lo sucedido.

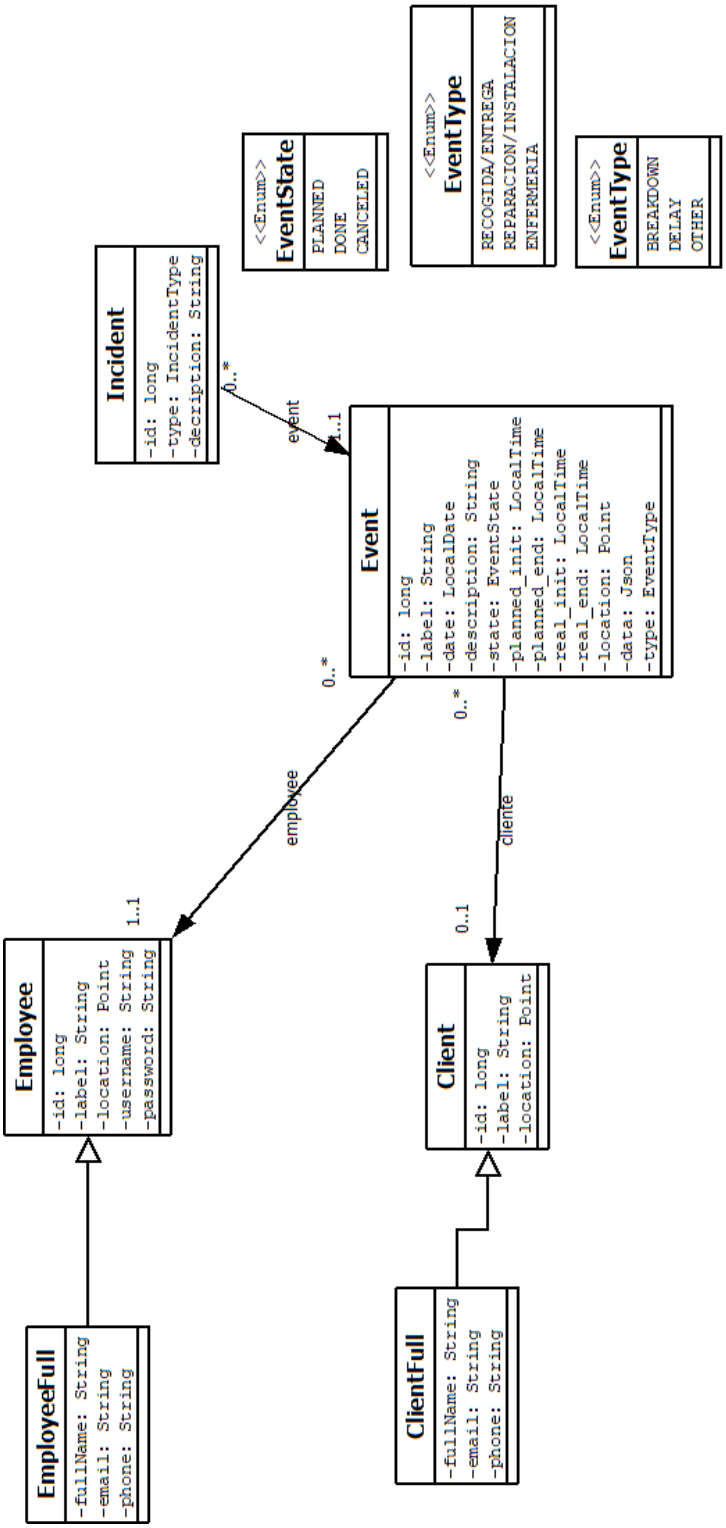


Figura 5.6: Modelo de datos de la aplicación

5.2 Diseño

5.2.1 Arquitectura tecnológica del sistema

En esta sección se mostrará un diagrama con la arquitectura de componentes del sistema junto con las tecnologías que se han empleado. Se hará una breve descripción de los componentes representados en la Figura 5.7 .

- **Cliente**: para la implementación de la aplicación móvil se ha utilizado la tecnología React Native [8], funciona con Node y Npm, es decir, utiliza JavaScript para la gestión de la capa de presentación. Las peticiones contra el servidor web se realizan con axios, una librería JavaScript basada en promesas, permitiendo la realización de solicitudes como cliente HTTP [9], ya que facilita el envío de peticiones a *endpoints* REST. La información obtenida en las respuestas la transforma, de manera automática, en formato JSON.

- **Servidor web**: proyecto Maven integrando Spring Boot con JPA (*Java Persistence API*, framework que maneja datos relacionales) y utilizando Hibernate [13] para la implementación ya que nos permite el procesamiento de peticiones accediendo a la base de datos para obtener la información correspondiente y mostrársela al usuario. Springboot nos permite centrarnos en el desarrollo de la aplicación simplificando la selección de las dependencias y el despliegue del servidor [12].

- **Sistema de Gestión de Base de Datos (SGBD)**: la implementación de la base de datos se realiza a partir de un gestor de base de datos, encargado de gestionar la información de la aplicación. En este proyecto se utilizó PostgreSQL [10] creando la base de datos a partir del modelo de datos descrito en el Apartado 5.1.3. Se hizo uso de la herramienta pgAdmin III debido a la facilidad que ofrece para la manipulación de datos a través de diferentes consultas como recuperación y actualización [21]. La comunicación entre la capa de acceso a datos y el servicio web se realiza mediante el lenguaje SQL.

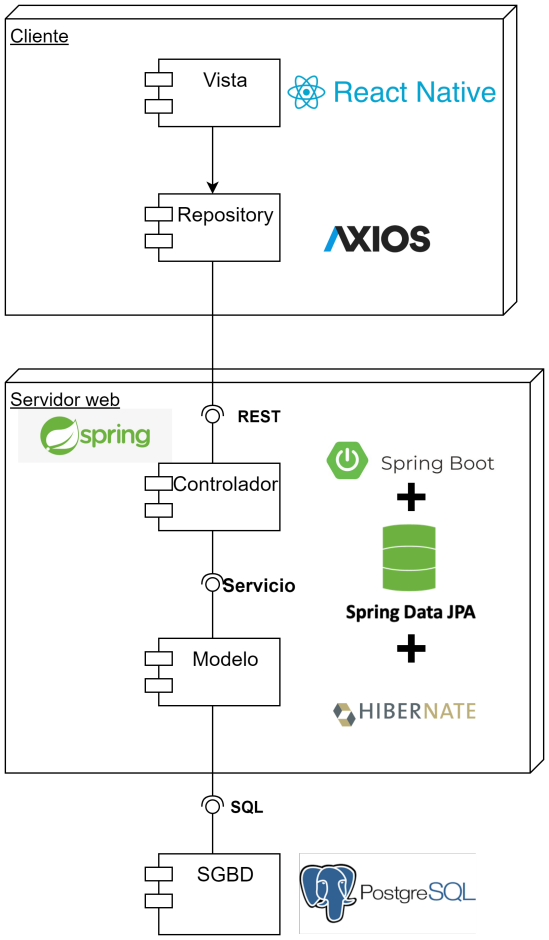


Figura 5.7: Diagrama de la arquitectura tecnológica del sistema.

5.2.2 Diseño de la aplicación

En esta sección se hará una descripción de la estructura del proyecto, tanto del *back-end* como del *front-end*.

Back-end

En la Figura 5.8 se puede ver la estructura del *back-end* del proyecto. En el servidor web definimos los siguientes paquetes clave:

- ***Resources***: cada clase implementa un *endpoint* del proyecto. El servicio REST va a estar definido en las clases *Resource* del proyecto, implementan las operaciones que van a ser utilizadas por la capa superior. Estas clases son lo que se denomina *Rest Controllers* o controladores. Los controladores trabajan siempre contra DTOs, nunca contra las entidades directamente. Se encarga de devolver la información de la petición

Petición	endpoint	Método
POST	/authenticate	authenticate
GET	/account	getAccount

Cuadro 5.1: *Endpoints* de *AccountResource*.

en formato JSON. Se puede encontrar la implementación de estas clases en el paquete *es.udc.lbd.tfg.restservice.web*. En el Cuadro 5.1, Cuadro 5.4, Cuadro 5.2 y Cuadro 5.3 se muestran los endpoints definidos para cada una de las clases *Resource*.

- ***Services***: servicios de la aplicación, implementan la lógica de negocio. Los servicios son los encargados de hacer la transformación de las entidades que recibe del DAO a DTO. Estas clases están implementadas en el paquete *es.udc.lbd.tfg.restservice.model.service*.

- **DAOs (*Data Access Object*)**: *Data Access Object* es un patrón de objetos de acceso a datos. En ellos se define como podemos acceder a la información almacenada, típicamente y, en nuestro caso, en una base de datos y como obtener esa información en un objeto que tenga las propiedades necesarias (en este caso los DTOs). Las interacciones con la base de datos están encapsuladas en las clases DAO. Se define un DAO por cada entidad del modelo y, para cada implementación del DAO, su interfaz correspondiente. Estes solo utilizan las entidades, no hacen uso de los DTOs. Las interfaces de los DAOs y su correpondiente implementación se encuentran en el paquete *es.udc.lbd.tfg.restservice.model.repository*.

- **DTOs (*Data Transfer Object*)**. El DTO es un patrón utilizado para la creación de objetos que sirven, únicamente, para ser transportados, permitiendo al servidor facilidad para ser enviados o recuperados. Son los encargados de transportar la información entre las capas de la aplicación. Objetos implementados en el paquete *es.udc.lbd.tfg.restservice.model.service.dto*.

- **Entidades**: entidades de la aplicación descritas en el Apartado 5.1.3. Representan las tablas o registros de la base de datos. Implementadas en el paquete *es.udc.lbd.tfg.restservice.model.domain*.

Front-end

Como se ha mencionado en el Apartado 5.2.1, la tecnología empleada para el desarrollo de la parte cliente ha sido React Native usando JavaScript. Se han incluido diversas librerías nativas para la implementación de las diferentes pantallas que nos permiten la construcción de la interfaz móvil. En la Figura 5.10 se muestra el diagrama de paquetes que componen la

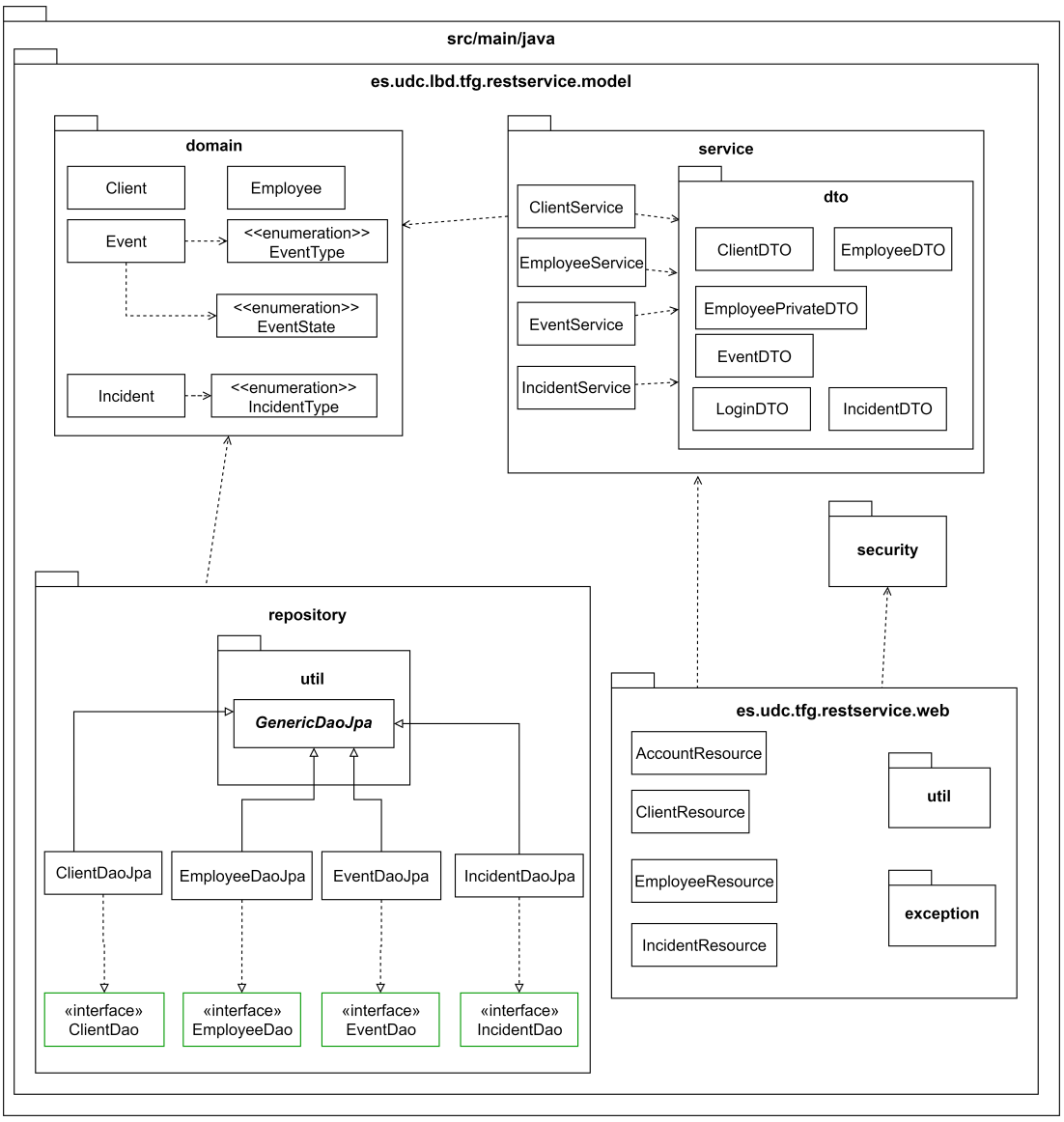


Figura 5.8: Diagrama de paquetes del servidor.

Petición	**endpoint**	**Método**
GET	/incidents/event/id	findAllByIdEvent
POST	/incidents/event/id/new	create
GET	/incidents/username	findAllByEmp
GET	/incidents/incidentType	findAllIncidentType
DELETE	/incidents/id	deleteById
PUT	/incidents/id	update

Cuadro 5.2: *Endpoints* de *IncidentResource*.

Petición	**endpoint**	**Método**
GET	/employees	findAll
GET	/employees/username	findByUsername

Cuadro 5.3: *Endpoints* de *EmployeesResource*.

Petición	**endpoint**	**Método**
GET	/events	findAll
GET	/events/id	findById
GET	/events/all/username	findAllByUsername
GET	/events/count-planned/username	countPlanned
POST	/events/planned/username	findPlannedByDate
GET	/events/done/username	findDoneByIdEmp
GET	/events/canceled/username	findCanceledByIdEmp
GET	/events/current/username	findCurrentByIdEmp
GET	/events/current/exist/username	findCurrentEventByIdEmp
GET	/events/current-next/username	findCurrentAndNextByEmp
PUT	/events/id	updateEvent
GET	/events/planned/dates/username	findDatesPlannedByIdEmp
POST	/events/id/new-data-event	createData

Cuadro 5.4: *Endpoints* de *EventResource*.

estrucura del proyecto de la parte cliente, a continuación se hace una breve descripción de los mismos.

- ***assets***: contiene ficheros públicos.
- **common**: contiene las utilidades comunes, en nuestro proyecto se pueden encontrar los ficheros:
  - ***http.js***: implementación de la configuración axios para realizar las peticiones.
  - ***auth.js***: se encarga de gestionar la autenticación del usuario.
  - ***store.js***: se encuentran los datos del usuario que se acaba de autenticar.

- **components**: en este directorio se encuentran los ficheros implementados que no están relacionados con las entidades.
- **json**: se almacenarán los formularios generados desde la Línea de Producto Software en formato JSON.
- **repositories**: se implementó un cliente REST por cada entidad necesaria. Se encargan de la comunicación con el servidor.
- **screens**: contiene todas las pantallas necesarias para la visualización de los datos en la interfaz móvil. Están relacionadas con las entidades del modelo.

En la Figura 5.9 se muestra un ejemplo de comportamiento de la arquitectura cliente-servidor en el momento en el que se accede a la visualización de un evento. A continuación, se realiza una descripción de la misma.

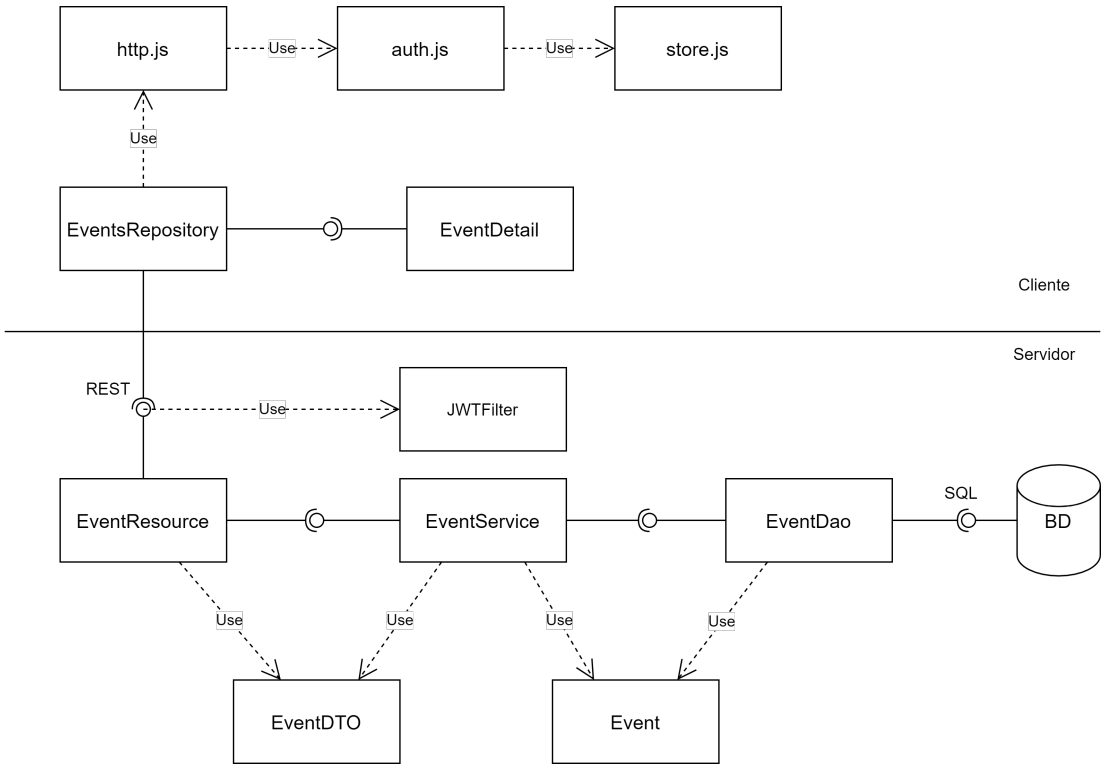


Figura 5.9: Ejemplo de arquitectura completa cliente-servidor de EventDetail.

En el momento en el que el cliente accede a la visualización del detalle de un evento, se realizará una petición al servicio REST que ofrece el controlador, la clase *JWTFilter* (en el paquete *es.udc.lbd.tfg.restservice.security*) realiza la validación del token. En caso de que el token sea válido, el controlador se encarga de gestionar la petición recibida por el cliente y hace

uso del DTO para enviar la información al servicio. Una vez que el servicio recibe el objeto, emplea la entidad *Event* del proyecto para llevar a cabo, contra el DAO correspondiente, las operaciones necesarias para la resolución de la petición. Los DAOs son los que acceden directamente a la base de datos para obtener la información que corresponda.

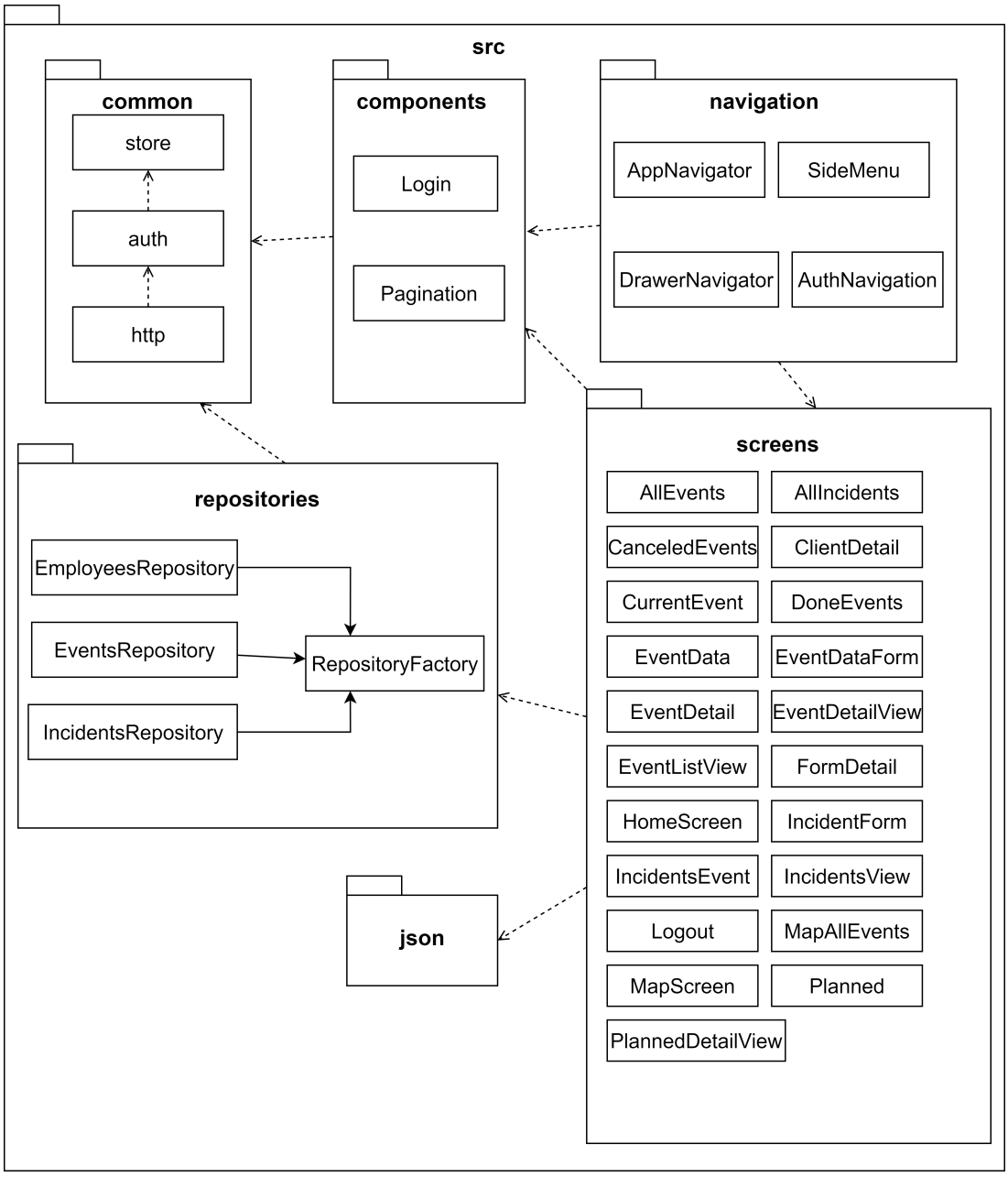


Figura 5.10: Diagrama de paquetes del cliente.

5.3 Implementación y pruebas

5.3.1 Implementación

Se destacan dos funcionalidades importantes en el desarrollo de esta aplicación. La primera de ellas es que el empleado puede cubrir en un formulario los datos correspondientes a un evento. La aplicación permite la definición de diferentes formularios en formato JSON, para cada uno de ellos se establecen unas propiedades para las cuales se generará un campo correspondiente en la interfaz de la aplicación. A cada evento se le asigna un formulario dependiendo de la tarea que se tenga que llevar a cabo. En segundo lugar, se destaca la visualización de un evento en un mapa, esta funcionalidad le mostrará al usuario su ubicación y la ubicación del evento concreto. Además, seleccionando el marcador de la ubicación del evento se abrirá un popup con el nombre correspondiente.

Cubrir formulario de datos

En primer lugar se explica la primera funcionalidad destacada, **cubrir formulario con los datos correspondientes a un evento**, disponible desde el detalle del evento. Para su implementación, principalmente se comprueba el tipo de formulario que tiene asignado el evento, dependiendo del tipo se utilizará un esquema JSON definido previamente. Un ejemplo de formulario es el siguiente:

```
{
    "title":"Recogida_Entrega",
    "properties":[
        {
            "name": "activity",
            "type":"string",
            "maxLength":20,
            "label":"Activity"
        },
        {
            "name": "merchandise",
            "type":"string",
            "maxLength":30,
            "label":"Merchandise"
        },
        {
            "type":"integer",
            "name":"quantity",
            "minimum":1,
            "label":"Quantity"
        }
```

```
    ],
    "required":["activity","merchandise","quantity"]
}
```

En este esquema se define un formulario llamado "Recogida_Entrega" al que se le asignan tres propiedades que se deben generar en la interfaz de la aplicación móvil. Para cada una de ellas se define el nombre, el tipo y la etiqueta. Dependiendo del tipo, tendrán que indicarse nuevos atributos. Para este caso, en las dos primeras propiedades, que son de tipo *string*, se define un atributo que indicará la longitud máxima que podrá tener la cadena de texto introducida por el usuario. Sin embargo, para la tercera propiedad, al ser de tipo *integer*, el atributo que se ha añadido es el valor mínimo que deberá introducir el usuario. Además, en este caso, las tres propiedades son obligatorias, es decir, para poder almacenar los datos en base de datos, el usuario no podrá dejar ninguna de las propiedades en blanco. Una vez que se ha definido el esquema JSON se obtiene el array con las propiedades (atributo *properties*) que se deberán generar en la interfaz de la aplicación (se puede ver en la Figura 5.11 el resultado del formulario generado). Para cada una de ellas se comprueba su tipo y se hace uso del componente *FormDetail* que contiene la implementación de cada tipo de propiedad. Por ejemplo, para la de tipo string se implementó lo siguiente:

```
<View style={{ flex: 1, flexDirection: 'column',
    alignItems:'flex-start',justifyContent:"flex-start", marginTop:30}}>
        <Text style={styles.label}>{props.prop.label}</Text>
        <TextInput
            defaultValue={data[props.prop.name]}
            maxLength={data[props.prop.maxLength]}
            onChangeText={(value) => {onChange_item(props.prop.name,value)}}
            style={styles.inputStyle}></TextInput>
 </View>
```

Este código generará un campo de texto con la etiqueta (*label*) indicada en la propiedad de tipo string del esquema JSON, en el momento en el que el usuario introduzca algún valor en el campo de texto se hace una llamada a una función en la que se asigna el valor indicado al nombre (*name*) de la propiedad y se almacena en un objeto. Se realiza, además, una llamada a la función *callback* que se ha definido previamente en el componente padre (desde el que se ha llamado a *FormDetail*), esta función lo que hace es que dicho componente obtenga un objeto con el nombre y el valor que acaba de indicar el usuario. Para cada una de las propiedades se hace uso de esta función para que finalmente, se obtenga un objeto con los pares clave:valor de cada una de ellas.

Además, si en el esquema JSON se ha definido el atributo *required*, que contiene los campos obligatorios, en el momento en el que el usuario quiera guardar los datos se comprobará que todas las propiedades están en el objeto definido. Si no lo están se mostrará una alerta con las

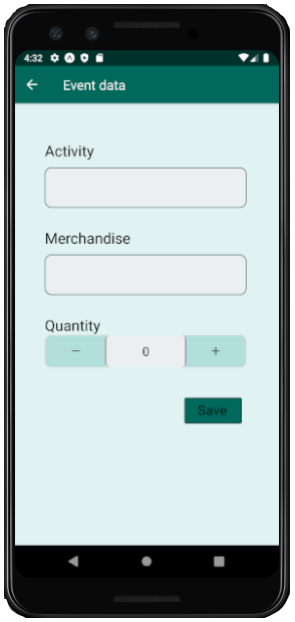


Figura 5.11: Pantalla de un formulario de ejemplo.

etiquetas a las que todavía no se le ha asignado ningún valor, en caso contrario, se guardarán los datos en base de datos en formato JSON.

De esta forma, lo que se consigue es realizar una implementación general de formularios.

Visualización del mapa

A continuación se explica la segunda funcionalidad destacada, la **visualización del evento en un mapa** a la que el usuario tendrá acceso desde el detalle del evento. Para su implementación se hizo uso del módulo 'react-native-maps' [33] de React Native ya que ofrece el componente mapa con el que se mostrará tanto la ubicación del evento como la del usuario. Para la visualización del evento se obtienen las coordenadas almacenadas en base de datos. Sin embargo, para la visualización de la posición actual se hizo uso de la API de geolocalización que existe de manera global en el objeto *navigator* [34] de React Native. Se realiza una llamada a *navigator.geolocation* para acceder a esta API que ofrece varios métodos, en nuestro caso, se ha utilizado la función *getCurrentPosition* que solicitará permiso al usuario para la obtención de la ubicación. La lógica de la función es la siguiente:

```
1 //ubicación empleado
2 navigator.geolocation.getCurrentPosition(
3     position => {
4         const currentP = {
5             latitude: position.coords.latitude,
6             longitude: position.coords.longitude,
```

```
            name:'Current position',
            state:'current position'
        }
        listCoordinates.push(currentP)
        this.setState(prevState => ({
                            currentPosition:{
                               ...prevState.currentPosition,
                               latitude: position.coords.latitude,
                               longitude: position.coords.longitude,
                               name:'Current position'
                            },
                            updateData:true,
                            list:listCoordinates,
                            loading:false,
                        }))

    },
    error => Alert.alert(error.message),
    { enableHighAccuracy: true, timeout: 20000, maximumAge: 0 }

);
```

En caso de que la operación transcurra con éxito se obtiene un argumento de posición (*position*), objeto que contiene una serie de propiedades de las que nos interesan la latitud y longitud que se almacenarán en un lista junto con las coordenadas del evento para, posteriormente, visualizarlas en el mapa. En nuestra función se definen tres propiedades opcionales [35]:

- *enableHighAccuracy*: indicador para que, si el dispositivo puede, obtenga el mejor resultado, es decir, que la posición sea lo más precisa posible.
- *timeout*: tiempo máximo, en milisegundos, para que el dispositivo obtenga un resultado.
- *maximumAge*: valor que indica la edad máxima en milisegundos de una posible posición de caché que es aceptable devolver. En nuestro caso, se definió con 0 para indicar al dispositivo que debe intentar recuperar la posición actual en vez de usar una posición de caché.

Se ha definido la variable *updateData* para que el mapa se muestre cuando se haya obtenido la ubicación del empleado. Una vez que se tenga la lista con las coordenadas tanto del usuario como del evento, se hará uso de los componentes *MapView* y *Marker*. Este último ofrece una propiedad (*pinColor*) que nos permite definir una función para establecer un marcador personalizado para cada posición; se definió el color azul para la posición actual y, para el

evento, los colores rojo, verde y amarillo para los estados cancelado, realizado y planeado, respectivamente.

El usuario también tendrá la opción de **iniciar la navegación** una vez que accede a la visualización del mapa. El cálculo de la ruta se delega en la aplicación de mapas por defecto que tenga instalada en el dispositivo, para ello, se ha implementado la siguiente función:

```
_callShowDirections = () => {
    const transportPlan='d';
    const endPoint = this.state.list[0]
    const startPoint = this.state.list[1]
    OpenMapDirections(startPoint,endPoint,transportPlan)
}
```

En primer lugar, se definen tres parámetros:

- *transportPlan*: indica el tipo de transporte para realizar el cálculo de la ruta. Disponemos de tres parámetros:
  - 'd': en coche.
  - 'w': a pie.
  - 'r': en transporte público.

  se ha decido establecer por defecto el cálculo en coche ya que posteriormente, el usuario podrá escoger el que desee.
- *endPoint*: se establecen las coordenadas del destino final (las del evento).
- *startPoint*: se establecen las coordenadas de inicio (las de usuario). Aunque este parámetro es opcional ya que en el momento en el que se abra el navegador predeterminado hará uso de la ubicación del dispositivo.

Una vez definidos estos tres parámetros, se hace una llamada a la función *OpenMapDirections*. Esta función, definida en el módulo ´react-native-navigation-directions' [36] de React Native, solicitará permiso al usuario para acceder a su ubicación y abrirá una aplicación de navegación predeterminada en nuestro dispositivo. En nuestro caso, como se ha utilizado el simulador de Android, se abrirá Google Maps.

5.3.2 Pruebas

Pruebas manuales

Una vez implementado el servidor, se han realizado peticiones con el uso de la herramienta Postman mencionada en el Apartado 3.1.2 para comprobar el correcto funcionamiento de las

operaciones implementadas, visualizando que el resultado de las peticiones era el esperado y, en caso de peticiones que provocasen cambios en los datos almacenados en base de datos, se comprobó que las actualizaciones se llevaban a cabo.

En la parte cliente se probó el correcto funcionamiento de la interfaz implementada, ejecutando las posibles funcionalidades a las que tiene acceso el trabajador y asegurándose que la información que se mostraba era legible desde dispositivos móviles.

Capítulo 6

Construcción de la línea de producto software

6.1 Análisis

6.1.1 Arquitectura del sistema

En la Figura 6.1 se muestra la arquitectura de la Línea de Producto Software, se hará una breve explicación de la misma.

- **Interfaz**: se desarrolló una interfaz web para la selección de características y generación de formularios.
- **Derivador**: motor de derivación empleado para la generación del producto final.
- **Componentes**: código fuente anotado de la aplicación móvil desarrollada que utilizará el motor de derivación para generar un producto en función de las características seleccionadas por el usuario de la herramienta.
- **Producto final**: código fuente de la aplicación generada según la especificación proporcionada por el usuario.

6.1.2 Interfaz de usuario

La interfaz web desarrollada permitirá al usuario de la herramienta la visualización de las diversas funcionalidades que puede presentar el producto final y la selección de las que necesite. Estas funcionalidades se presentarán en una estructura jerárquica. Además, se mostrará una interfaz que le permitirá la generación de los formularios que quiere tener en su aplicación final.

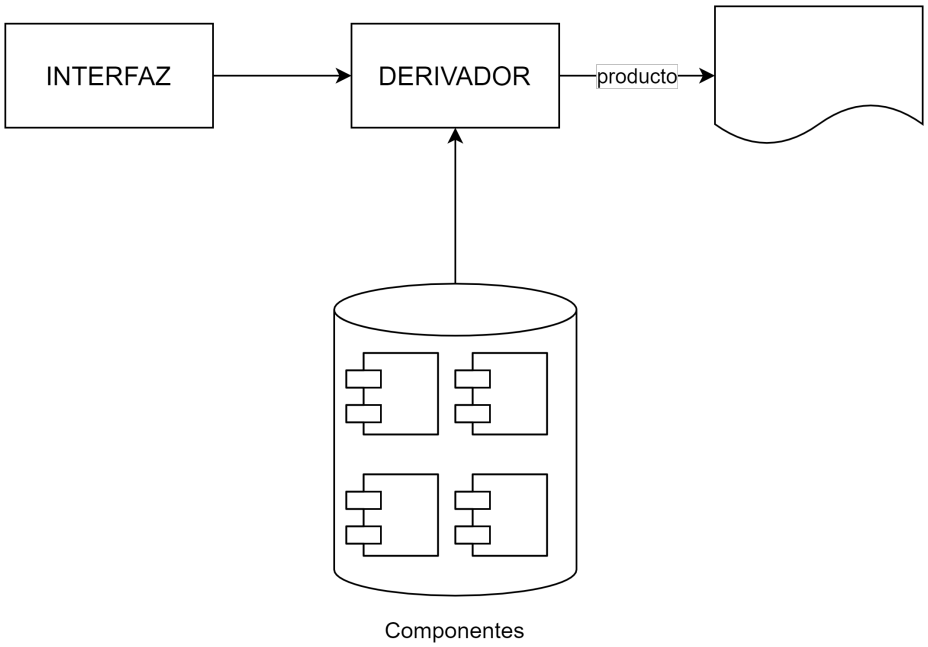


Figura 6.1: Diagrama de la arquitectura de la Línea de Producto Software.

- Se muestra al usuario una estructura jerárquica para marcar o desmarcar las características definidas en el modelo de variabilidad.
- En la barra de navegación de la interfaz se presentan tres botones con las siguientes funcionalidades:
  - Visualizar una lista con formularios. En esta lista se muestran tantos los formularios almacenados en una variable definida en la especificación importada del producto, como también los creados por el usuario. Para poder crearlos, se le presenta al usuario una interfaz en la que podrá indicar en un campo de texto el título de dicho formulario y un botón que le permitirá añadir las propiedades necesarias, cada propiedad nueva se muestra en una tabla con las opciones de editar o eliminar y poder marcar dicha propiedad como obligatoria. Finalmente, en el momento en el que el usuario cree el formulario, se mostrará de nuevo la lista inicial con el nombre y se generará un esquema JSON que se almacenará en la especificación del producto con el resto de formularios existentes.
  - Importar la especificación del producto que desea generar.
  - Exportar el fichero con la especificación con las características y formularios seleccionados para el producto que se va a generar.
  - Generar el producto final, permitirá la descarga de un fichero *zip* con el código fuente del producto.

6.2 Diseño

6.2.1 Arquitectura tecnológica del sistema

En la Figura 6.2 se puede ver la arquitectura tecnológica del sistema, en ella se presentan las tecnologías que se han empleado para el desarrollo de la Línea de Producto Software.

Como se ha mencionado en el Apartado 3.2, el motor de derivación utilizado ha sido desarrollado por uno de los directores de este proyecto.

La implementación de la interfaz se ha realizado con Vue.js.

Este motor de derivación se encarga de gestionar el proceso de generación de un producto. La arquitectura es la siguiente:

El motor de derivación recibe una serie de ficheros para su correcto funcionamiento, los ficheros que se deben utilizar son los siguientes:

- **model.xml** o **model.json**: en nuestro caso se ha utilizado el fichero XML generado a partir del árbol de características de la Figura 4.1 realizado con FeatureIDE. Es decir, contiene el modelo de variabilidad del sistema.
- **config.json**: fichero que contiene la configuración de la LPS. Indica al navegador cuales son los delimitadores de las anotaciones incluidas en el código fuente de la aplicación y qué ficheros y carpetas deberá ignorar a la hora de generar el producto final.
- **product.json**: fichero que contiene la especificación de un producto concreto.
- **code**: carpeta que contiene el código fuente de de la aplicación, tanto del cliente como del servidor, con sus correspondientes anotaciones, a partir de este código anotado se generará el código del producto final.

A continuación, se hará una breve explicación sobre el funcionamiento de este motor de derivación [3].

El motor de derivación se encarga de la generación del producto de la siguiente forma:

- Primero, delega en el **gestor de variabilidad** para la interpretación del modelo que ha proporcionado el usuario, selección de las características a partir de la interfaz.
- Segundo, comprueba el cumplimiento de todas las restricciones e interpreta el fichero de configuración de la LPS.
- Por último, se lleva a cabo la interpretación del fichero con la especificación que ha proporcionado el usuario y a continuación, el **motor de plantillas** y **gestor de ficheros** se encargan de aplicar el modelo de variabilidad a las plantillas y generar el código fuente del producto final.

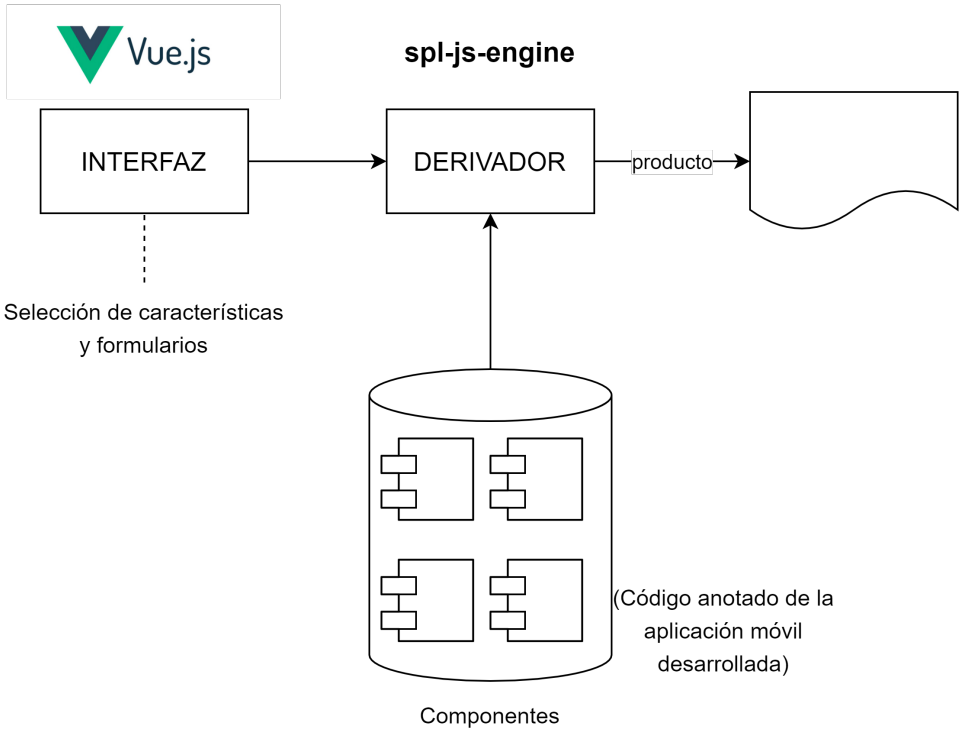


Figura 6.2: Diagrama de la arquitectura tecnológica de la Línea de Producto Software.

6.2.2 Diseño de la aplicación

En esta sección se realiza un breve descripción de los paquetes clave de la herramienta.

- **components**: contiene los ficheros no relacionados con las características y formularios que se generarán en la aplicación. Además de dos directorios donde se encuentra la implementación de las pantallas que mostrarán al usuario la estructura jerárquica con las características y las interfaces para la visualización de la lista de formularios y generación de estos.

- **services**: contiene la implementación de las funciones utilizadas por los componentes para la selección de características y generación de formularios que posteriormente se almacenarán en un fichero JSON (especificación del producto concreto). Estos ficheros harán uso del motor de derivación, encargado de la generación del producto validando que la especificación sea correcta.

- **platform**: contiene el código fuente anotado de la aplicación que se ha desarrollado junto con los ficheros necesarios para el funcionamiento del motor de derivación.

6.3 Implementación y pruebas

6.3.1 Implementación

En esta sección se describen dos de los aspectos fundamentales de la aplicación: la generación de formularios que permite al usuario la definición de diferentes formularios en formato JSON para, posteriormente, mostrarlos en la interfaz de la aplicación móvil y las anotaciones realizadas en el código para poder generar los productos solo con las funcionalidades especificadas por el usuario.

Generación de formularios

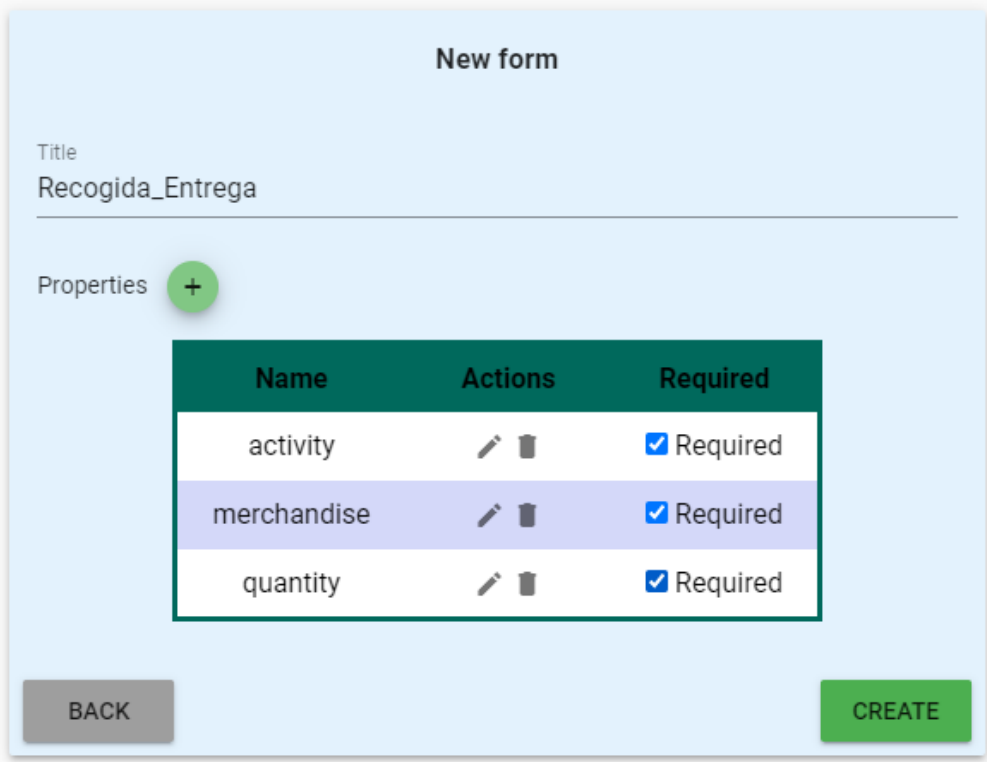
A continuación, se explica como un usuario puede generar un formulario. Como se ha mencionado en el Apartado 5.3.1, la aplicación permite la definición de diferentes formularios ya que está diseñada con la posibilidad de ser usada por diversas empresas, independientemente del ámbito. Por ello se consideró importante la idea de poder generarlos a partir de una interfaz.

En primer lugar, el usuario responsable de la herramienta podrá visualizar la lista de formularios que se hayan importado o creado. Para ello, se accede a una variable definida en la especificación del producto, esta variable es un array que contendrá los formularios que haya definido el usuario.

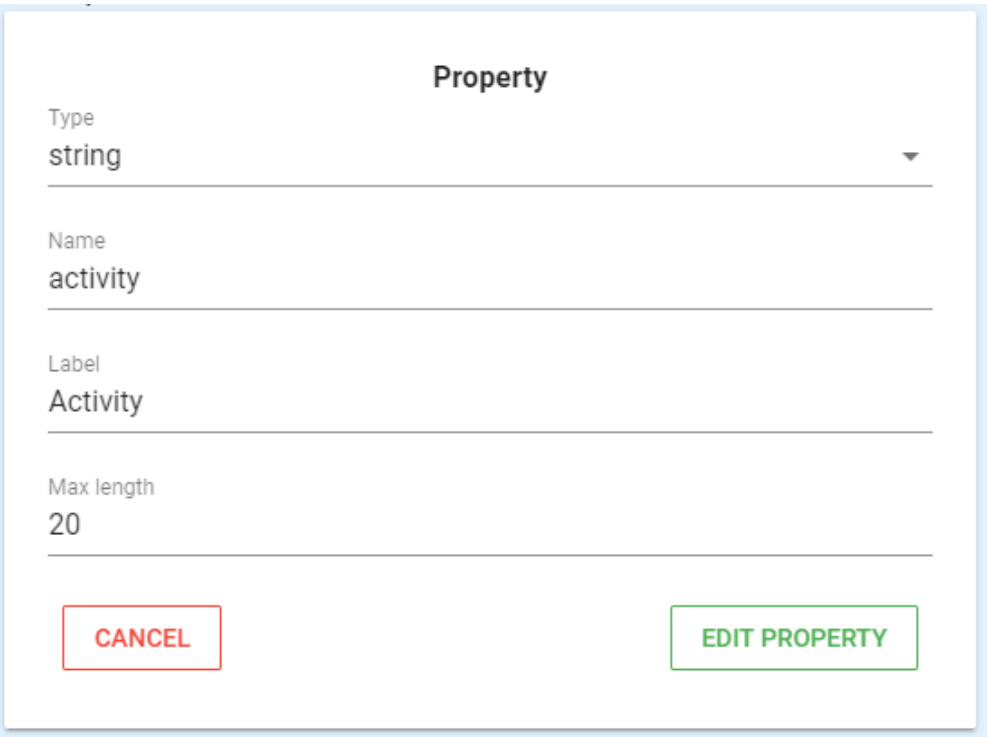
En caso de que la especificación del producto no sea importada, el usuario podrá generar los formularios, para los cuales se ha implementado una interfaz que facilite su generación. En la Figura 6.3 se puede ver un ejemplo de formulario realizado por el usuario desde el editor de formularios. En la imagen 6.3a se muestra el título asignado al formulario y una tabla con las propiedades. Todas las propiedades tendrán los atributos nombre, etiqueta y tipo, y dependiendo de este último se mostrarán nuevos atributos (en la figura 6.3b se puede ver un ejemplo de propiedad de tipo *string*). Cada una de las propiedades podrá ser modificada o eliminada de la tabla. Además, el usuario tendrá la opción de marcar dicha propiedad como obligatoria, es decir, en la interfaz de la aplicación, el empleado tendrá la obligación de asignarle algún valor para poder ser almacenada en la base de datos. Finalmente, esta interfaz se transformará en un esquema JSON, explicado en el Apartado 5.3.1, que se almacenará en una variable definida en la especificación el producto.

Anotaciones

El segundo de los aspectos destacados es la realización de **anotaciones** en el código fuente de la aplicación que se ha desarrollado, tanto en la parte cliente como en la parte servidor, que nos permitirá la generación de productos con unas funcionalidades específicas. Las anotacio-



(a) Título y lista de propiedades



(b) Atributos de una propiedad de tipo *string*

Figura 6.3: Pantalla de la interfaz para la generación de formularios

nes se indican mediante comentarios y contienen cualquier código JavaScript, en función de la extensión de los ficheros se establecen unos delimitadores de los comentarios. A continuación, se realiza una breve explicación de las anotaciones realizadas en nuestro código con un ejemplo correspondiente. Se distinguen entre las anotaciones genéricas, las de interpolación y las de generación de ficheros.

- **Anotaciones genéricas**: a continuación se muestra un ejemplo de este tipo de anotación realizada en la clase *HomeScreen* del cliente.

```
{/*% if (feature.ver_listado_eventos_pendientes) { %*/
<View style={{ flex: 1, flexDirection: 'row', alignItems:'flex-start',
    justifyContent:'flex-start'}}>
    <Text style={{fontSize:19, paddingLeft:10}}>You have</Text>
        <TouchableHighlight underlayColor='#e0f2f1'
        onPress={() =>
                this.props.navigation.navigate('PlannedEvents') }>
            <Text style={{ color:'blue', fontSize:19}}>
                {countEvents}
            </Text>
        </TouchableHighlight>
    <Text style={{fontSize:19}}> events today</Text>
</View>
/*% } %*/}
```

  Esta anotación nos indica que el texto con el enlace para acceder a la lista de eventos pendientes estará disponible, únicamente, si el usuario selecciona la característica que le permita visualizar dicha lista (en la Figura 6.4 se muestra con un círculo rojo el resultado al generar el producto con dicha funcionalidad) . En caso contrario, el código no se mostrará en el código fuente del producto final generado y, por tanto, el empleado no tendrá dicho enlace disponible.

- **Anotaciones de generación de ficheros**: el motor de derivación empleado nos permite la definición de variables en la especificación del producto. En este caso, se ha definido una variable para almacenar una lista con los formularios generados por el usuario. Con este tipo de anotaciones lo que se pretende es recorrer dicha variable para poder generar un fichero JSON por cada uno de los formularios definidos. Para el correcto funcionamiento de estas anotaciones es necesario:

  - que las anotaciones estén siempre al principio de los ficheros.
  - que contenga la implementación de una función con los datos que reciba en la especificación y devuelva un array con dos objetos *fileName* y *context*.

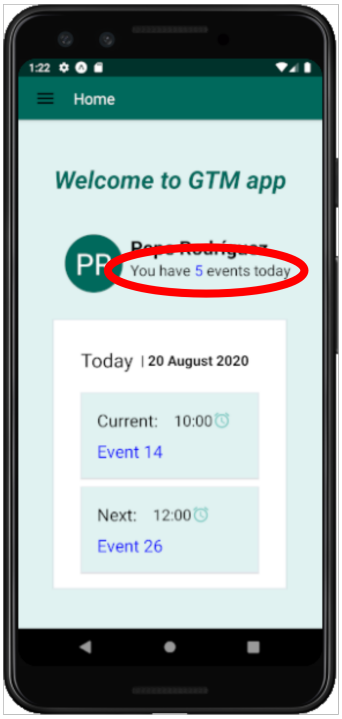


Figura 6.4: Resultado de anotación genérica en la interfaz de la aplicación al generar un producto.

En la Figura 6.5 se muestra la implementación del código utilizado para la generación de los ficheros. A continuación, se explica su funcionamiento. En primer lugar, una

```
###<@ return data.formularios.map(function(form) {
    return {
        fileName: normalize(form.title) + '.json',
        context: form
    };
})>###
###<= JSON.stringify(context,null,2) >###
```

Figura 6.5: Fichero con las anotaciones para la generación de formularios

vez que el usuario ha indicado la especificación del producto, se recorre la lista de los formularios almacenados en una variable de dicha especificación, por cada uno de los formularios existentes, se generará un fichero con la siguiente información:

  - ***fileName***: será el título del formulario que ha indicado el usuario (se ha utilizado la función *normalize* para evitar la presencia de caracteres extraños y espacios en blanco).
  - ***context***: será el contenido del formulario.

- **Anotaciones de interpolación**: un ejemplo de este tipo de anotaciones es el siguiente

(línea 7 de la Figura 6.5):

```
1    ###<= JSON.stringify(context,null,2) >###
```

Lo que se hace es interpolar el contenido de la variable context, es decir, con esta anotación se indica dónde deberá ser substituido el valor de la variable context. En este caso, es el contenido del formulario generado por el usuario en formato JSON.

6.3.2 Pruebas

Pruebas manuales

Las pruebas que se han realizado en el desarrollo de la herramienta han sido manuales. En primer lugar, se comprobó la correcta validación de las restricciones establecidas en el modelo de variabilidad. Se han realizado diferentes combinaciones posibles en la selección de características, desde el más básico, conteniendo solo las obligatorias, hasta el más completo, conteniendo todas las características posibles.

En segundo lugar, se ha comprobado la correcta generación de los formularios con su correspondiente esquema JSON, marcando las distintas propiedades que podían ser seleccionadas y el correcto almacenamiento de la lista de formularios en la especificación. Además, en el caso de que no exista ningún formulario en la lista, se mostrará un mensaje al usuario indicando que al menos debe haber uno a la hora de generar el producto.

Capítulo 7

Solución desarrollada

En esta sección se redacta una visita guiada por la herramienta de generación y por las principales funcionalidades de la aplicación móvil que se ha desarrollado.

7.1 Herramienta de generación

En esta herramienta no es necesario autenticarse para acceder a sus funcionalidades, ha sido desarrollada para uso interno, es decir, el usuario objetivo de la herramienta será personal interno a la empresa encargado de la generación de productos, no podrá ser utilizada por cualquier persona.

Suponemos al actor objetivo de la herramienta como usuario de la misma: accediendo al botón *Features* se mostrará la ventana de la Figura 7.1 donde puede visualizar una estructura jerárquica con todas las características de las que puede disponer el producto que desee generar, marcando o desmarcando las que necesite. Para la generación de los formularios puede acceder a través del botón *Forms* donde se le mostrará una lista con todos los formularios que haya creado como se puede ver en la Figura 7.2, si no hay ninguno se muestra una notificación. Al añadir un nuevo formulario, accede a la interfaz de la Figura 7.4 donde tendrá que rellenar un formulario con el título y las propiedades que desee.

Desde la barra de navegación tiene acceso a las opciones de importación, exportación y generación de producto. En el primer caso tendrá que cargar un fichero *json* con la especificación que desee, en el segundo se le descargará un fichero *json* con la especificación indicada y, en el último caso, un fichero *zip* con el producto final generado a partir de la especificación (en la Figura 7.3 están marcados con un círculo rojo las funcionalidades que se acaban de mencionar).

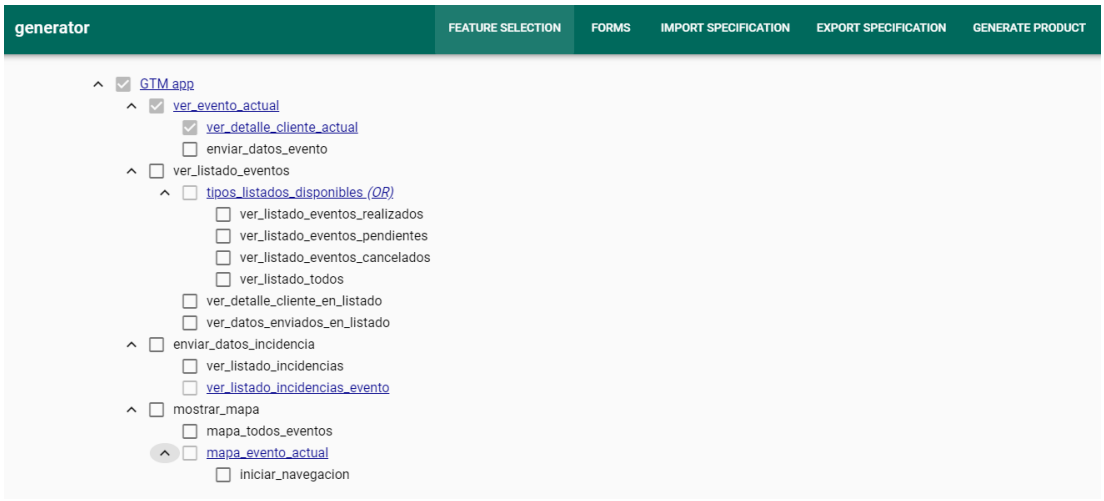


Figura 7.1: Estructura jerárquica con las características representadas en el *feature-model* 4.1.

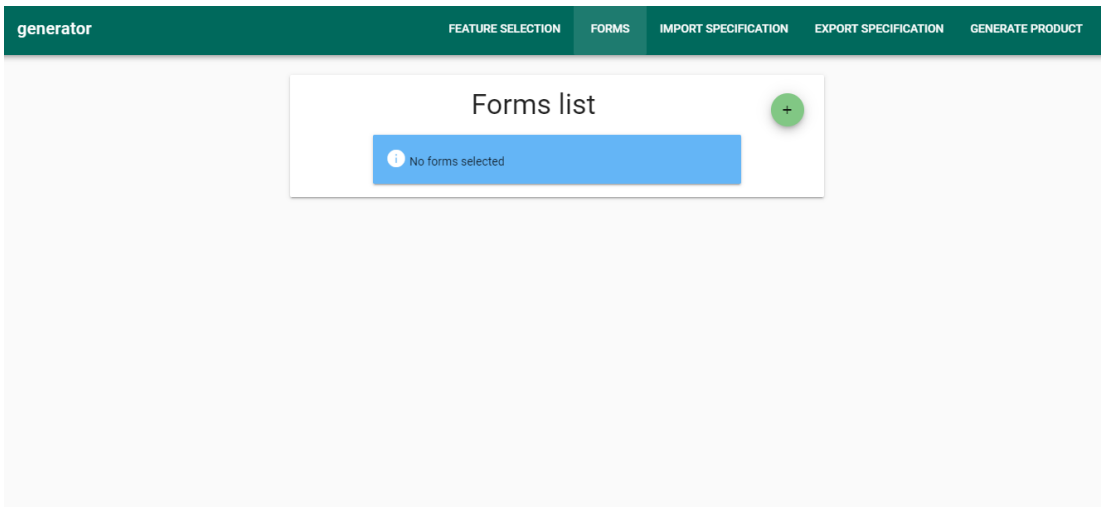


Figura 7.2: Lista de formularios creados o importados

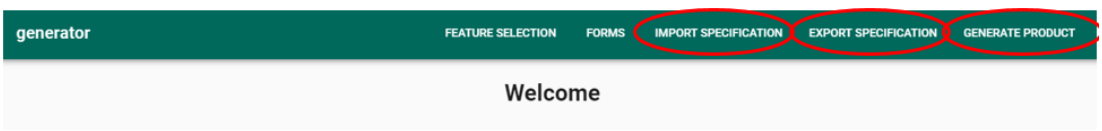


Figura 7.3: Barra de navegación de la herramienta de generación.

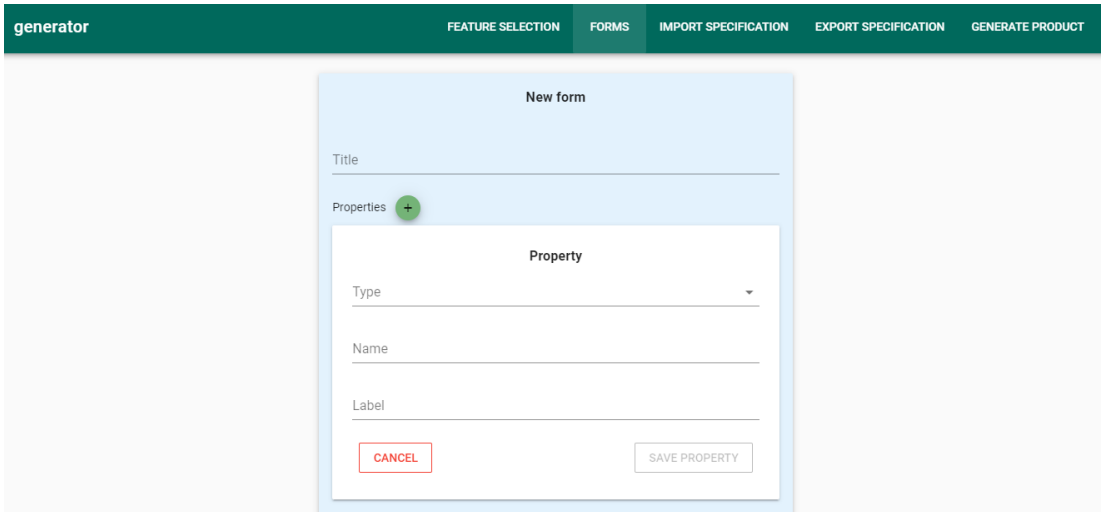


Figura 7.4: Formulario para la generación de nuevos formularios para la aplicación móvil.

7.2 Producto

Una vez que el usuario se autentica, la pantalla principal a la que tiene acceso es la Figura 7.5a donde se muestran los eventos actual y siguiente junto con la fecha actual y el número de eventos pendientes para el día. Para acceder a la información de alguno de los eventos lo hace a partir del nombre del evento, que es un enlace al mismo. Una vez seleccionado el enlace se muestra la pantalla de la Figura 7.9 con el detalle de dicho evento, se puede ver que las horas de inicio y fin reales del evento coinciden con las estimadas para que solo sea necesaria su modificación en caso de que exista alguna desviación, una vez que se guarden los datos del evento, se muestra un mensaje indicándolo y el evento pasa automáticamente a la lista de los realizados además de volver a la pantalla desde la que ha accedido al detalle del evento, en este caso, a la pantalla de inicio. En esta pantalla Figura 7.9a también tiene acceso a un mapa para la visualización de la ubicación del evento y de la suya propia, tal y como se muestra en la pantalla de la Figura 7.6a, pudiendo obtener la ruta para llegar a su destino donde se le abrirá la herramienta de Google Maps con el cálculo de las posibles rutas (ver Figura 7.6b). Desde el detalle del evento de la Figura 7.9b también puede acceder a un formulario que le permitirá cubrir los datos correspondientes a alguna incidencia que le haya ocurrido durante el trayecto o en la ejecución de un evento. Podrá visualizar todas las incidencias ocurridas en un evento desde el botón *Incident list* de la pantalla con la información del evento. En esta tabla se indicará la fecha, el tipo y las acciones de eliminar o editar incidencia. Una vez haya finalizado la realización del evento, tendrá acceso a un formulario que le permitirá almacenar los datos correspondientes a la tarea que acaba de llevar a cabo. Dependiendo de lo que tenga que realizar en cada evento se le mostrará un tipo de formulario que se le asignó previamente.

Un ejemplo sería el de la Figura 7.9c.

Como se ha mencionado anteriormente, en la pantalla de la Figura 7.5a se indica el número de eventos pendientes para el día actual, este número es un enlace que le lleva a la pantalla de la Figura 7.10a donde puede observar, inicialmente, la lista de los eventos, pero también tiene la opción de la visualización de un calendario que le permite seleccionar una fecha y visualizar sus eventos tal y como se ve en la Figura 7.10b. En ambos casos tiene acceso a un mapa con los eventos de la fecha indicada donde a partir del popup de un evento seleccionado en el mapa (ver la Figura 7.10c) puede ver su información, volviendo así a la pantalla de la Figura 7.9.

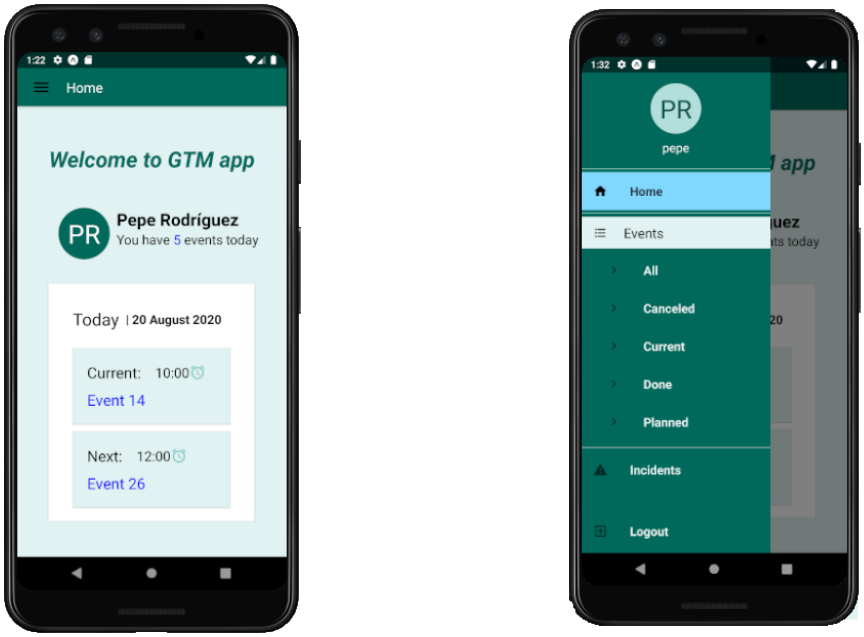
A partir del menú navegación de la Figura 7.5b tiene acceso a diferentes funcionalidades, entre ellas, están los diferentes tipos de listados de eventos. Si accede a la lista con todos los eventos se le mostrará la pantalla de la Figura 7.7a donde se muestra, en una tabla, el nombre de todos los eventos que le fueron asignados, su fecha correspondiente, resaltando la fecha de los del día actual, y la hora de inicio de cada uno de ellos. Además, tendrá acceso a un mapa donde podrá visualizar la ubicación de todos los eventos, como se puede ver en la Figura 7.7b, y dependiendo del estado del evento se mostrará el marcador de la ubicación en un color diferente. Para el estado:

- CANCELADO: color rojo.
- PLANEADO: color amarillo.
- REALIZADO: color verde

El marcador azul indica la posición actual del empleado.

En caso de acceder al listado de los eventos realizados se mostrará la pantalla de la Figura 7.10d donde tendrá una lista con cada unos de los eventos ya realizados. Para cada uno de ellos puede ver el nombre, que a su vez es un enlace para acceder a la pantalla Figura 7.9 con la información del evento, la hora de inicio y la fecha en la que se ha llevado a cabo el evento.

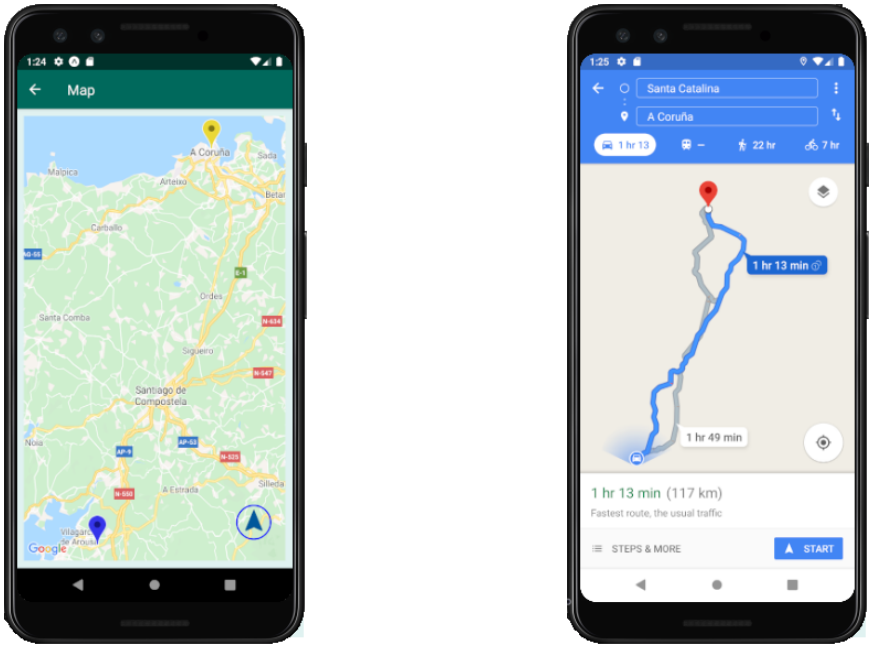
Otra de las opciones de acceso desde el menú de navegación de la Figura 7.5b de la aplicación, es la visualización de la tabla de la Figura 7.8a que contiene todas las incidencias registradas por el empleado. En ella se muestra el evento en el que ocurrió la incidencia, la fecha en que sucedió, el tipo de incidencia y las acciones que puede llevar a cabo, que serían editar y eliminar. Para editarla, se accede al formulario de la Figura 7.8b que le permite la modificación de los campos necesarios y para eliminarla se le mostrará una notificación de confirmación antes de su eliminación.



(a) Pantalla de inicio.

(b) Menú de navegación.

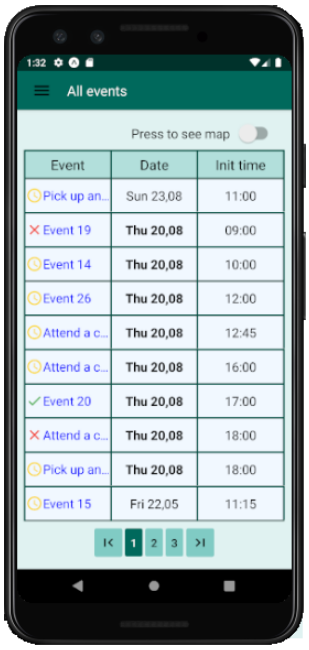
Figura 7.5: Pantalla de inicio y menú de navegación de la aplicación.



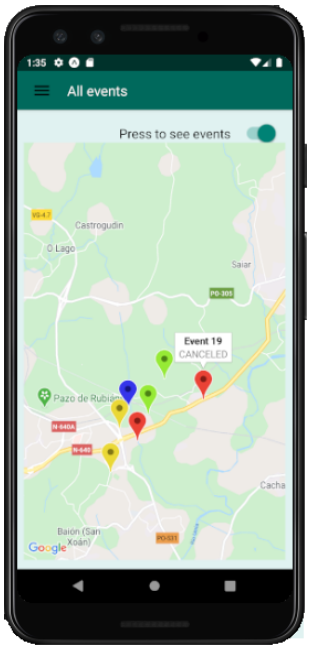
(a) Ubicación del empleado y del evento.

(b) Ruta obtenida en Google Maps.

Figura 7.6: Pantallas para la visualización del mapa de un evento.

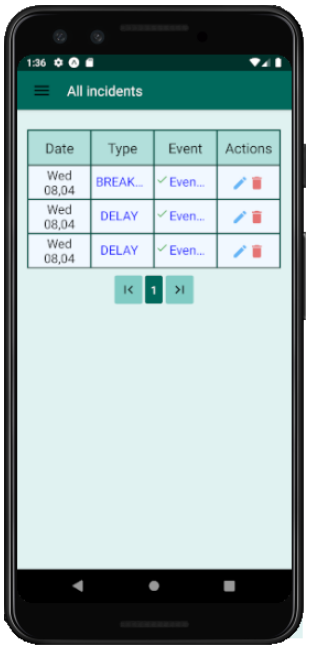


(a) Listado de eventos.

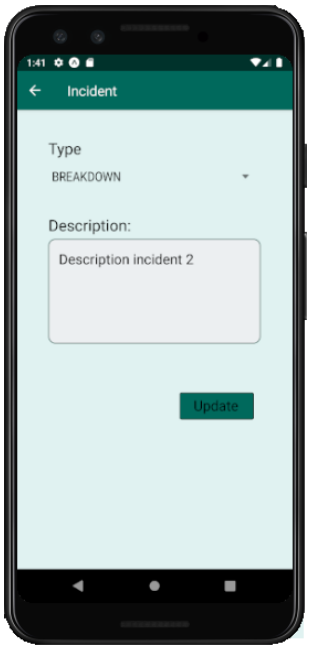


(b) Ubicación eventos y empleado.

Figura 7.7: Pantalla para la visualización de todos los eventos asignados a un empleado.

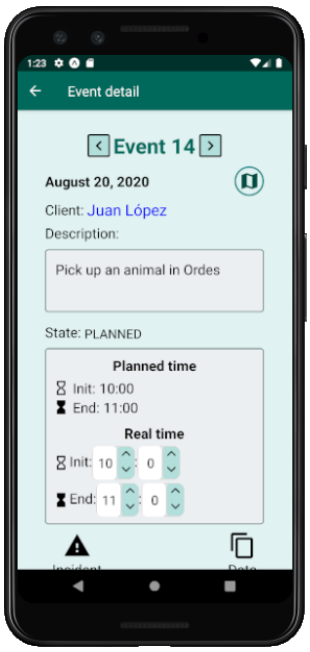


(a) Tabla de incidencias.



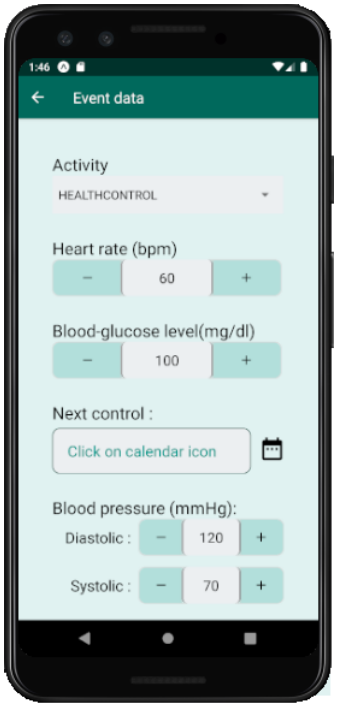
(b) Formulario incidencia.

Figura 7.8: Pantallas con las incidencias registradas por el usuario y el formulario para editar una incidencia.



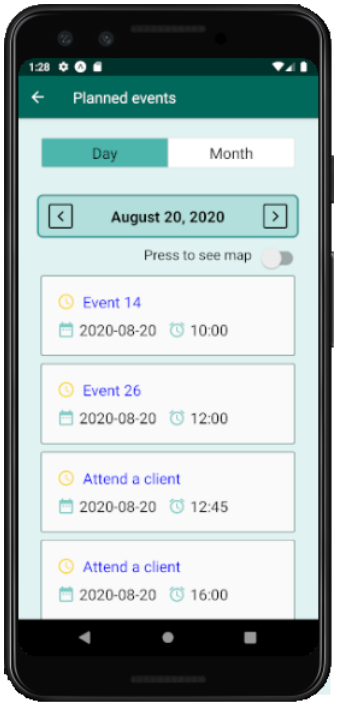
(a) Pantalla detalle evento (1parte)

(b) Pantalla detalle evento (2parte)

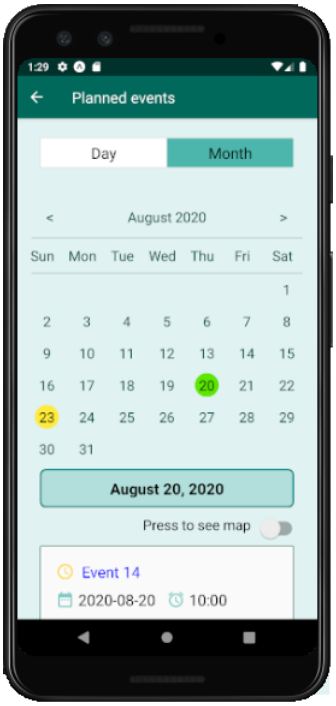


(c) Formulario de datos de ejemplo.

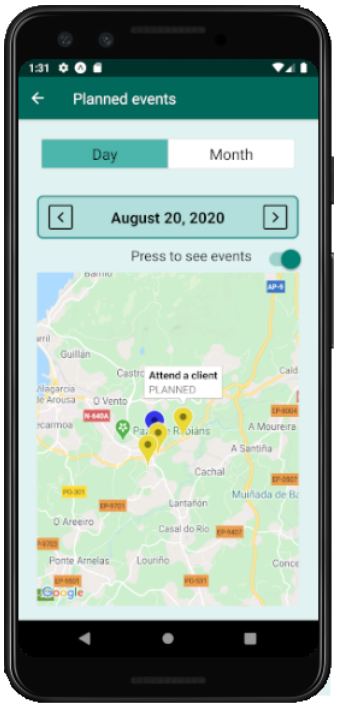
Figura 7.9: Pantallas con la información completa de un evento y formulario para almacenar los datos correspondientes.



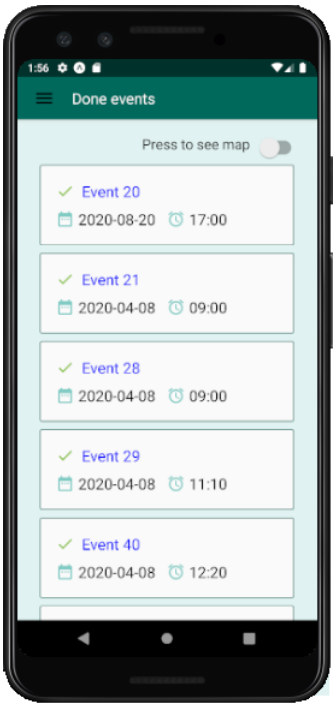
(a) Eventos planeados opción día.



(b) Eventos planeados opción mes.



(c) Ubicación de los eventos planeados.



(d) Eventos realizados.

Figura 7.10: Pantalla con diferentes listados de eventos.

Capítulo 8

Conclusiones y trabajo futuro

8.1 Conclusiones

Finalizado el proyecto, se han alcanzado los objetivos planteados inicialmente.

- **Herramienta de generación de productos**: se ha desarrollado una LPS que cumple con las funcionalidades descritas en el Apartado 4.3.2. Permite al usuario la generación de múltiples productos con diferentes características y formularios.

- **Aplicación móvil**: se ha llevado a cabo el desarrollo de una aplicación móvil que permite a los trabajadores en movilidad la gestión de sus eventos de manera eficaz y sencilla, cumpliendo con las funcionalidades descritas en el Apartado 4.1.2.

Las características que se consideraron más importantes en el desarrollo de este proyecto, fueron que la LPS facilitase al usuario la generación de productos con características similares sin necesidad de una nueva implementación para cada producto, de forma que, en caso de ser necesario, las modificaciones o nuevas implementaciones fuesen mínimas. Y la posibilidad de la generación de formularios mediante una interfaz web. En cuanto a la aplicación móvil se destaca la visualización de los eventos en un mapa junto con el acceso a Google Maps para iniciar la navegación y por último, que la aplicación disponga de formularios para almacenar la información de los eventos evitando, de esta forma, que tengan que ser completados en papel.

Se han aplicado conocimientos adquiridos durante el grado:

- Aplicación de algunas de las buenas prácticas de Scrum para poder obtener un buen resultado final del proyecto.

- Aplicación de conceptos básicos en el desarrollo del análisis (Capítulo 4).

- Uso de una metodología iterativa e incremental facilitando el desarrollo del proyecto de forma que se iban aumentando las funcionalidades a medida que se finalizaba cada iteración.

Se han adquirido nuevas competencias, entre ellas se destaca:

- El conocimiento de la Línea de Producto Software, la técnica de *scaffolding* aplicada y el uso del motor de derivación spl-js-engine [3] desarrollado por uno de los directores de este proyecto.
- Se han adquirido nuevos conocimientos en el uso de la tecnología React-Native.
- Se ha llevado a cabo la planificación y seguimiento de un proyecto más completo de los que habitualmente se realizan durante el grado.

8.2 Ampliaciones a futuro

A continuación, se describen una serie de ampliaciones no consideradas en los objetivos principales del proyecto.

- La internacionalización del proyecto, ya que el desarrollo en varios idiomas hará que aumente el porcentaje de gente que utilice la aplicación lo que podrá favorecer el alcance del éxito.
- Envío de notificaciones al dispositivo del empleado, por ejemplo:
  - al iniciar navegación, cuando exista algún tipo de obstáculo en la ruta que vaya a provocar retraso para llegar a su destino concreto.
  - cuando la hora actual se aproxime a la hora de inicio de un evento.
- Visualización de un mapa con una ruta que muestre la ubicación, no solo de un evento concreto, sino la de todos los eventos pendientes, haciendo el cálculo de esta ruta en función de la hora de inicio de los eventos.

Apéndices

Apéndice A

Manual de instalación

A.0.1 Producto

Requisitos previos

- Instalación de **Node.js** que nos permitirá utilizar npm para la gestión de paquetes, versión de **JDK** 11.0.4, **python** 3.8.1 y **Apache maven** 3.5.4.
- Si se quiere hacer uso de un emulador es necesaria la instalación de la herramienta **Android Studio** que nos permitirá la visualización de la aplicación en un simulador móvil. Para la ejecución de un simulador en ordenadores Mac se puede utilizar el entorno de desarrollo Xcode.
- Instalación del **CLI de Expo** [37]: 'npm install -g expo-cli'. Permitirá la ejecución de nuestra aplicación en un simulador.

Configuración del fichero */client/src/common/http.js* cambiar 'direcciónIP' de la Figura A.1 por la dirección IP del ordenador desde el que se ejecutará el servidor.

Antes de la ejecución de la aplicación es necesaria la configuración de la base de datos:

- Creación de una base de datos en PostgreSQL.
- Configuración del fichero */service/src/main/resources/application.yml*
- **Aplicación móvil**
  - **Cliente**: desde el directorio *client*
    - Instalar dependencias: 'npm install'
    - Ejecutar: 'expo start'

    Una vez ejecutada la aplicación, se puede escanear el código QR desde nuestro dispositivo móvil (aunque para ello sería necesaria la instalación de la aplicación

```
const HTTP = axios.create({
    baseURL: 'http://direccionIP:8080/api/'
});
```

Figura A.1: Base URL

```
To run the app with live reloading, choose one of:
• Scan the QR code above with the Expo app (Android) or the Camera app (iOS).
• Press a for Android emulator, or w to run on web.
• Press e to send a link to your phone with email.
• Press s to sign in and enable more options.

Expo  Press ? to show a list of all available commands.
```

Figura A.2: Opciones disponibles para desplegar la aplicación en un emulador

Expo en nuestro dispositivo). Sin embargo, si se quiere desplegar en un emulador, se nos ofrecen distintas posibilidades, se pueden ver en la Figura A.2

- **Servicio**: desde el directorio *service*
  - Instalar dependencias: 'mvn install'
  - Ejecutar: 'mvn spring-boot:run'

A.0.2 Herramienta de generación de producto

Requisitos previos

- Instalación de Node.js

Despliegue de la herramienta:

- Instalar dependencias: 'npm install'
- Ejecutar: 'npm run debug'

Apéndice B

Prototipos de pantalla de la aplicación móvil

En esta sección se muestran todos los prototipos de pantalla realizados en al análisis preliminar del proyecto.

Figura B.1: Prototipo de pantalla de inicio de sesión.

Figura B.2: Prototipo de la pantalla de inicio.

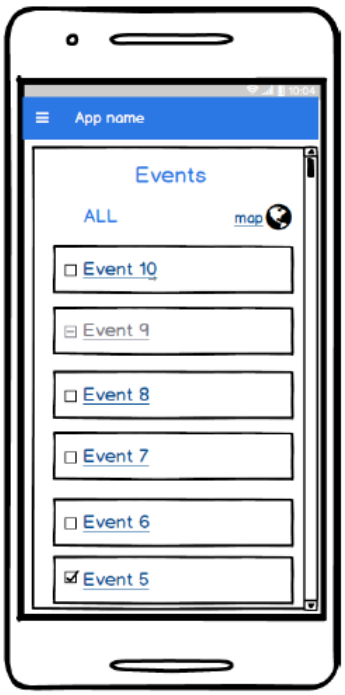


Figura B.3: Prototipo de pantalla con la lista de todos los eventos.

Figura B.4: Prototipos de pantalla con la lista de eventos pendientes.

Figura B.5: Prototipo de pantalla con la información de un evento.

Figura B.6: Prototipo de pantalla con la información del cliente de un evento.

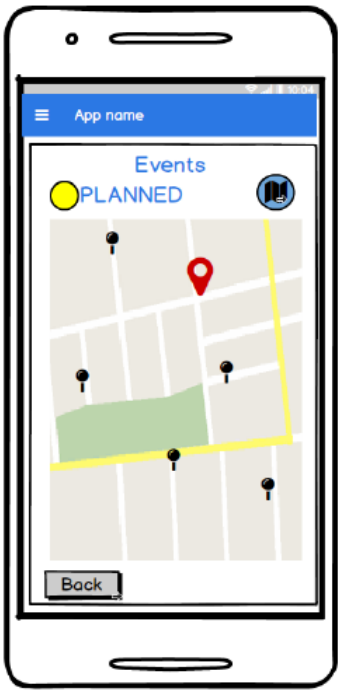


Figura B.7: Prototipo de pantalla del mapa con la ubicación de los eventos pendientes.

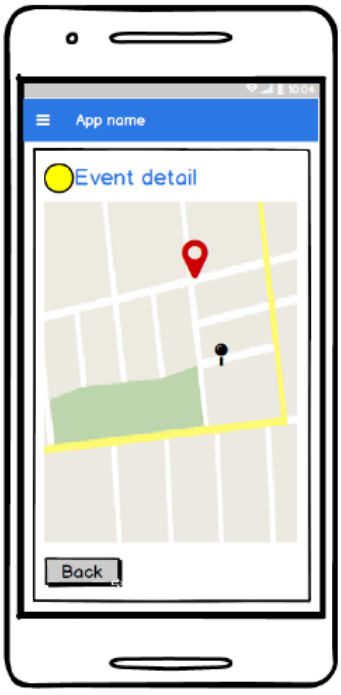


Figura B.8: Prototipo de pantalla del mapa con la ubicación de un evento concreto.

Figura B.9: Prototipo de pantalla con la lista de eventos realizados.

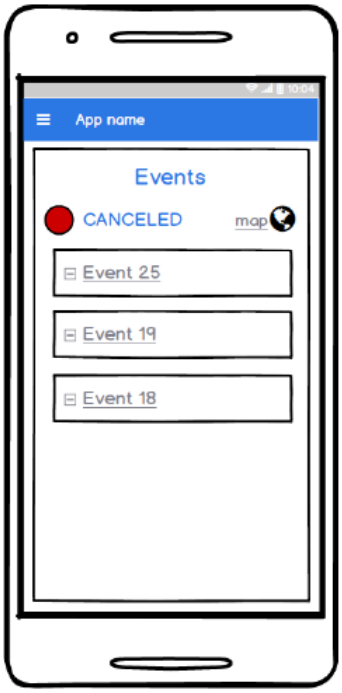


Figura B.10: Prototipo de pantalla con la lista de eventos cancelados.

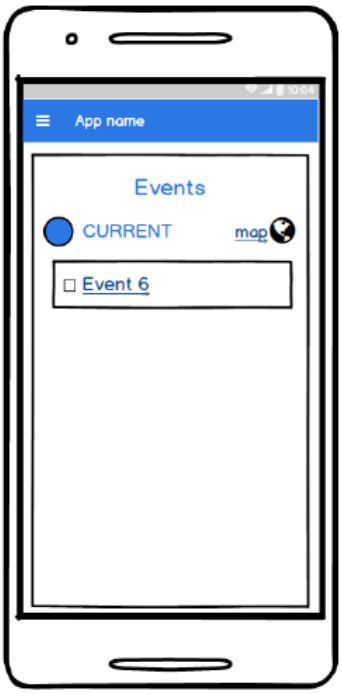


Figura B.11: Prototipo de pantalla con el acceso al evento actual.

Figura B.12: Prototipo de pantalla con la lista de incidencias de un evento.

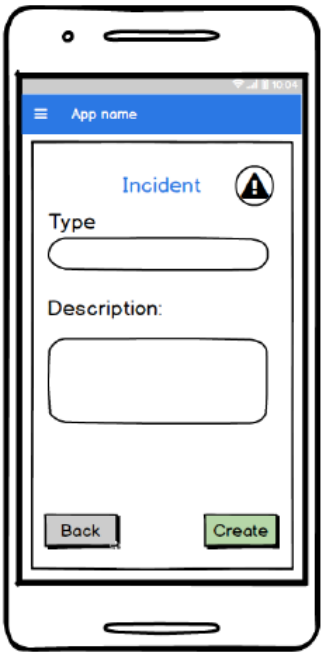


Figura B.13: Prototipo de pantalla con el formulario para crear una nueva incidencia.

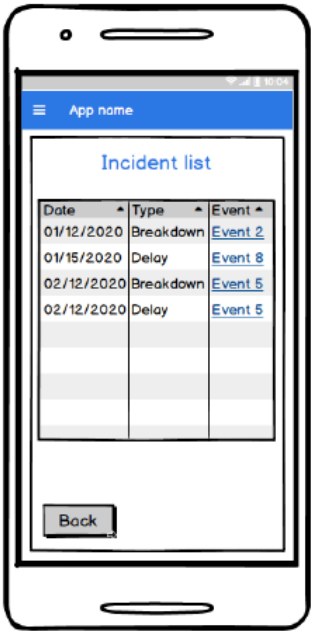


Figura B.14: Prototipo de pantalla con todas las incidencias registradas.

Figura B.15: Prototipo de pantalla de un formulario de ejemplo asociado a un evento.

Figura B.16: Prototipo de pantalla de un formulario de ejemplo asociado a un evento.

Figura B.17: Prototipo de pantalla de un formulario de ejemplo asociado a un evento.

Figura B.18: Prototipo de pantalla de un formulario de ejemplo asociado a un evento.

Apéndice C

Glosario de acrónimos

AHEAD *Algebraic Hierarchical Equations for Application Design.*

ATS *AHEAD Tool Suite.*

API *Aplication Programming Interface.*

DAO *Data Access Object.*

DTO *Data Transfer Object.*

FOP *Feature Oriented Programming.*

FST *Feature Structure Tree.*

GPS *Global Positioning System.*

GTM *Gestión de Trabajo en Movilidad.*

HTML *HyperText Markup Language*

HTTP *HyperText Transfer Protocol.*

IDE *Integrated Development Environment.*

JPA *Java Persistence API.*

JSON *JavaScript Object Notation.*

LPS *Línea de Producto Software.*

REST *Representational State Transfer.*

SGBD *Sistema de Gestión de Bases de Datos.*

SQL *Structured Query Languge.*

XML *Extensible Markup Language*

Apéndice D

Glosario de termos

Back-end Conjunto del desarrollo de una aplicación encargado del acceso a datos. No accesible, de forma directa, por los usuarios.

Endpoint URL del back-end, no interactúan con el usuario final sino que se encargan de responder a una petición realizada por este.

Front-end Interfaz web que permite a los usuarios la interacción con una aplicación.

Feature-model Representación de todos los posibles productos de la Línea de Producto Software en términos de características y relaciones entre ellas.

Línea de Producto Software Conjunto de sistemas software que comparten características comunes que satisfacen las necesidades específicas de un dominio.

Scaffolding Consiste en la generación de código a partir de plantillas predefinidas de una especificación.

Bibliografía

[1] "Featurehouse: Language-independet, automated software composition." [En línea]. Disponible en: http://www.fosd.de/featurehouse

[2] "Ahead tool suite." [En línea]. Disponible en: https://www.cs.utexas.edu/users/schwartz/ATS.html

[3] A.Cortiñas, "spl-js-engine." [En línea]. Disponible en: https://github.com/AlexCortinas/spl-js-engine

[4] N. R. Brisaboa, A. Cortiñas, M. R. Luaces, and Óscar Pedreira, "Aplicando *scaffolding* en el desarrollo de líneas de producto software," 2016.

[5] "Página web de mfleet." [En línea]. Disponible en: https://movildata.com/mfleet/

[6] "Página web de vue.js." [En línea]. Disponible en: https://vuejs.org/

[7] "Página web de node.js." [En línea]. Disponible en: https://nodejs.org/es/

[8] "Página web de react-native." [En línea]. Disponible en: https://reactnative.dev/

[9] "Repositorio github de axios." [En línea]. Disponible en: https://github.com/axios/axios

[10] "Página web de postgresql." [En línea]. Disponible en: https://www.postgresql.org/

[11] "Página web de expo." [En línea]. Disponible en: https://docs.expo.io/

[12] "Página web de spring." [En línea]. Disponible en: https://spring.io/projects/spring-boot

[13] "Página web de hibernate." [En línea]. Disponible en: https://hibernate.org/

[14] K. Schwaber and J. Sutherland, "The scrum guide™," November 2019. [En línea]. Disponible en: https://scrumguides.org/scrum-guide.html

[15] "Scrum." [En línea]. Disponible en: https://www.scrum.org/resources/blog/que-es-scrum

[16] "Gitlab." [En línea]. Disponible en: https://gitlab.lbd.org.es/users/sign_in

[17] "Página web de balsamiq wireframes." [En línea]. Disponible en: https://balsamiq.com/wireframes/

[18] "Página web de android studio." [En línea]. Disponible en: https://developer.android.com/studio/intro

[19] "Página web de eclipse ide." [En línea]. Disponible en: https://www.eclipse.org/

[20] "Página web de visual studio code." [En línea]. Disponible en: https://code.visualstudio.com/docs

[21] "Página web de pgadmin, postgresql tools." [En línea]. Disponible en: https://www.pgadmin.org/

[22] "Página web de postman." [En línea]. Disponible en: https://www.postman.com/

[23] "Página web de npm." [En línea]. Disponible en: https://docs.npmjs.com/about-npm/

[24] "Página web de maven (apache maven)." [En línea]. Disponible en: https://maven.apache.org/

[25] "Página web de overleaf." [En línea]. Disponible en: https://www.overleaf.com/

[26] "Página web de featureide." [En línea]. Disponible en: http://www.featureide.com/

[27] "Página web de microsoft teams." [En línea]. Disponible en: https://www.microsoft.com/es-ww/microsoft-365/microsoft-teams/group-chat-software

[28] "Página web de microsoft project." [En línea]. Disponible en: https://www.microsoft.com/es-es/microsoft-365/project/project-management-software

[29] "Página web del proyecto dia." [En línea]. Disponible en: https://wiki.gnome.org/Apps/Dia

[30] "Página web de draw.io." [En línea]. Disponible en: https://www.draw.io/

[31] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns.* Addison Wesley, August 2001.

[32] F. J. van der Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action. The Best Industrial Practice in Product Line Engineering.* Springer, 2007.

[33] "Repositorio de react-native-maps." [En línea]. Disponible en: https://github.com/react-native-community/react-native-maps

[34] “Página web de api navigator.” [En línea]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/API/Navigator

[35] “Página web ´position options’ getcurrentposition.” [En línea]. Disponible en: https://developer.mozilla.org/en-US/docs/Web/API/PositionOptions

[36] “Paquete react-native-navigation-directions.” [En línea]. Disponible en: https://www.npmjs.com/package/react-native-navigation-directions

[37] “Página web de expo cli.” [En línea]. Disponible en: https://docs.expo.io/workflow/expo-cli/