Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Sistema de gestión de recorridos grabados con GPS

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Miguel Sancho Chilet

Tutor: Mario Gómez Martínez

2015/2016

Resumen

El presente proyecto trata sobre el desarrollo de una aplicación de escritorio para monitorizar, analizar y gestionar actividades deportivas grabadas mediante GPS, así como la sincronización de estas actividades entre diferentes plataformas Web. Los recorridos son consumidos por el gestor de recorridos que se conecta a cada una de las plataformas web, y es capaz de centralizar tareas en las diferentes plataformas. Entre dichas tareas se incluye la posibilidad de importar/exportar archivos mediante el estándar GPX, visualizar recorridos en Google Maps, modificar recorridos, sincronizar y visualizar estadísticas.

Para ello se ha trabajado en la integración entre diferentes APIs, concretamente con las dos plataformas de recorridos GPS más utilizadas en el ámbito deportivo, Strava y Endomondo.

Con este proyecto se ha pretendido crear una herramienta para el usuario que permita centralizar la gestión y visualización de recorridos grabados con GPS entre diferentes plataformas web.

Como conclusión, indicar la motivación extra con la que se ha desarrollado el proyecto, al ser usuario habitual de estas dos plataformas.

Palabras clave: JavaFX, Java, Java Scene Builder, GPS, GPX, Strava, Endomondo, API, Integración de servicios, deportes.

Abstract

This project is about developing an application for monitoring, analyzing y management of sports activities recorded via GPS, synchronization of these activities between different platforms is also done. Routes are consumed by the manager which synchronizes tasks between platforms. Tasks as importing/exporting files using GPX standard, route tracking using Google Maps, routes modification and statistics visualization.

To achieve this, work has been focused on integrating several APIs, specifically two of the most used sports activities managers: Strava and Endomondo.

This project pretends to create a tool which let the user gather all their sports activities across all the platforms into a single one.

Concluding, project development had always an extra motivation because of being a usual user of these platforms.

Keywords: JavaFX, Java, Java Scene Builder, GPS, GPX, Strava, Endomondo, Services integration, Sports.

Tabla de contenidos

Tabla de figuras

1. Introducción

1.1. Contexto

El ámbito de esta aplicación se centra en la integración de servicios para la gestión de recorridos grabados mediante GPS. Existen diversos servicios que nos proporcionan la posibilidad de subir nuestras actividades para así compartirlas en la red, pudiendo así mantener nuestras actividades deportivas organizadas y disponibles en cualquier momento, de igual manera nosotros podemos navegar y descargar cualquier recorrido disponible de terceras personas que han decidido compartirlos.

En la actualidad estos servicios se han extendido de forma considerable, son muchas las plataformas en las que podemos compartir nuestras actividades, por esta razón el propósito de la aplicación es la integración de varios de estos servicios. Y de esta manera conseguir desarrollar una aplicación que nos proporcione la posibilidad de mantener una sincronización entre las distintas plataformas en las que guardamos nuestros recorridos, y a su vez permitir gestionar de forma individual cada uno de estos.

1.2. Objetivos

El objetivo principal de esta aplicación es conseguir una integración total entre diferentes servicios, proporcionando al usuario final una herramienta con la que pueda gestionar diversas plataformas, unificar todos sus recorridos y poder realizar diferentes tareas de gestión sobre ellos.

Un usuario de esta aplicación será capaz de descargar sus actividades deportivas de diferentes plataformas, visualizar el recorrido desde Google Maps, visualizar estadísticas detalladas y modificar cada una de ellas de forma totalmente transparente, es decir, sin importar de qué plataforma proviene dicha actividad. Además la herramienta proporciona al usuario la posibilidad de sincronizar todos sus recorridos, unificando aquellos que puedan están repetidos en varios servicios.

1.3. Definiciones, siglas y acrónimos

En el siguiente apartado se definirán los diferentes términos, siglas o acrónimos que se van a utilizar a lo largo de la especificación de requerimientos.

- **Token:** cadena de caracteres que identifica una conexión en la API de Strava.
- **JRE:** *Java Runtime Environment*, conjunto de utilidades que se proporcionan para la ejecución de programas Java.
- **API**: *Application programming interface*, interfaz de programación de aplicaciones. Es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- **Google Maps**: servidor de aplicaciones de mapas que ofrece imágenes de mapas con la posibilidad de desplazarse sobre ellos y visualizar diferentes ubicaciones.

- **XML**: *eXtensible Markup Language* o lenguaje de marcas Extensible, es un leguaje utilizado para almacenar datos en forma legible.

- **BBDD**: Siglas del término bases de datos.

- **GPS**: Sistema de posicionamiento global, es un sistema que permite determinar en la Tierra la posición de un objeto con una precisión de hasta centímetros.

- **GPX**: *GPS eXachange Format* o Formato de Intercambio de GPS es un esquema XML pensado para transferir datos GPS entre aplicaciones.

- **Punto de recorrido**: cada uno de los puntos que representa una localización por la que hemos pasado.

- **Recorrido**: sucesión de puntos concatenados que definen un trayecto.

2. Especificación de requerimientos

Para una más cómoda exposición de los requerimientos, vamos a utilizar de ahora en adelante las siglas RF para hacer referencia a los requerimientos funcionales, y por otro lado RNF para los requerimientos no funcionales.

2.1. Requerimientos funcionales

Los requerimientos funcionales son declaraciones de los servicios que proveerá la aplicación, es decir cómo se comportará el sistema ante diferentes entradas. Por lo tanto todos los servicios solicitados por el usuario deben estar definidos en los requisitos funcionales.

RF.01 Autenticación de la aplicación con las diferentes plataformas

Cada una de las plataformas utilizadas necesita de unos credenciales para acceder a los datos del usuario. Al iniciar la aplicación se solicitaran los credenciales correspondientes a cada uno de los servicios. Mediante los credenciales introducidos el sistema solicitará acceso a las distintas plataformas y generará un *token* que identificará la conexión, el cual permitirá realizar todas las comunicaciones futuras.

RF.02 Descarga de las actividades y perfil de usuario

La aplicación, una vez autenticado el usuario en las correspondientes plataformas, cargará el perfil de usuario y se encargará de consultar si existen cambios respecto a la anterior consulta. En el caso de que existan nuevas actividades en alguna de las plataformas, iniciará la descarga de las actividades disponibles.

RF.03 Conversión y unificación de datos

Durante la descarga de los diferentes datos existe un proceso dedicado a adaptar los datos que se descargan de las diferentes plataformas. Este proceso es uno de los más importantes, ya que para el algoritmo de sincronización es totalmente transparente la procedencia de las diferentes actividades. Los datos son unificados y adaptados antes de su sincronización.

RF.04 Sincronización de las actividades entre diferentes plataformas

Las actividades de las diferentes plataformas que han sido transformadas son procesadas para así encontrar aquellas repetidas. Tras el proceso de sincronización se dispondrá por un lado de todas las actividades unificadas y por otro lado las actividades que difieren.

RF.05 Persistencia de las actividades en base de datos

Al iniciar la aplicación el sistema comprueba si hay actividades en BBDD, en caso negativo añade las actividades, en caso afirmativo comprueba que las actividades están actualizadas, en el caso de no estarlo, se actualizarán únicamente aquellas actividades que falten. Además cualquier modificación posterior que pueda sufrir una actividad, será actualizada en base de datos.

RF.06 Recuperar actividades de base de datos

El sistema es capaz de recuperar las actividades guardadas en BBDD para realizar cualquier acción que solicite el usuario, en el caso de que alguna de las acciones solicitadas necesite datos que no están disponibles en BBDD el sistema recurrirá a la conexión a la plataforma web para obtenerlos.

RF.07 Visualización de lista de actividades

Al finalizar el proceso de sincronización de los datos, se muestra la pantalla principal de la aplicación. Aquí se muestra al usuario la lista final de todas las actividades unificadas entre las diferentes plataformas, así como detalles de las mismas. Se dispone a la vez de tres botones donde el usuario puede interaccionar con los diferentes recorridos.

RF.08 Edición de una actividad concreta

La aplicación permite al usuario editar tanto el nombre como la descripción de las actividades registradas en el sistema. Esta modificación conlleva una actualización de la actividad en la base de datos y en la plataforma remota, así como un refresco de la lista de actividades para visualizar los nuevos detalles.

RF.09 Actualizar una actividad remota

En el momento que una actividad sufre algún cambio, el sistema es capaz de mantener la integridad de los datos locales y los datos remotos, los datos modificados por el usuario son actualizados en su actividad correspondiente en la plataforma web.

RF.10 Visualización de una actividad en Google Maps

Todas las actividades pueden visualizarse sobre Google Maps, con la posibilidad de intercambiar entre diferentes vistas que muestras diferentes tipos de terrenos. El usuario dispone de un botón para abrir la visualización del mapa. En este momento el sistema descarga los puntos del recorrido seleccionado y es capaz de dibujar el recorrido.

RF.11 Visualización de estadísticas de una actividad

Cada actividad seleccionada ofrece la posibilidad de visualizar diferentes estadísticas y gráficos relacionados con dicha actividad. Se mostraran diferentes gráficas representando factores como son la altitud, la velocidad o la distancia del recorrido seleccionado.

RF.12 Visualización de estadísticas mensuales

El gestor dispone de un recuento de actividades, mediante el cual podemos visualizar un gráfico que muestra cuantas actividades de cada tipo se han practicado durante cada mes, diferenciando a su vez el año en el que fueron grabadas.

RF.13 Subir una actividad a las diferentes plataformas

El sistema dispone de la posibilidad de subir nuevas actividades a la plataforma web, si existen actividades en una plataforma web que no están presentes en la otra plataforma, el sistema realizara la subida de dicha actividad, manteniendo así las plataformas sincronizadas en todo momento.

RF.14 Exportar una actividad a un archivo GPX

Las actividades disponibles en la aplicación dispondrán de la opción de ser exportadas en formato GPX a nuestro sistema. El gestor de recorridos se encarga de adaptar la actividad para construir posteriormente el fichero GPX, mediante el explorador de archivos que se abrirá en nuestra aplicación seleccionaremos la ruta exacta donde exportar el recorrido.

2.2. Requerimientos no funcionales

Los requerimientos no funcionales son aquellos que no representan directamente las funciones que proporcionan el sistema, sino algunas propiedades del mismo como pueden ser el tiempo de respuesta o la fiabilidad entre otras. Se dice que los requerimientos no funcionales surgen de la necesidad del usuario, ya que van ligados a factores como pueden ser el presupuesto, a su vez los requerimientos no funcionales también pueden verse afectados por factores externos como puede ser la privacidad.

RNF.01 Factores comunicación Strava API

La API de Strava está limitada en cuanto al número de peticiones se refiere, su límite se encuentra en 600 solicitudes cada 15 minutos, y un total de 30.000 solicitudes por día.

RNF.02 Rendimiento al cargar de actividades

La descarga de las actividades en el sistema no debe sobrepasar los 5 segundos.

RNF.03 Rendimiento al cargar un recorrido en Google Maps

La visualización del recorrido sobre Google Maps no debe sobrepasar los 5 segundos, independientemente del número de puntos que compongan el recorrido.

3. Contexto tecnológico

En el siguiente apartado se va a describir las características y conceptos de los distintos lenguajes y tecnologías que se ha empleado para el desarrollo de la aplicación.

3.1. Entorno de desarrollo

Un entorno de desarrollo integrado o IDE es una aplicación que proporciona a los desarrolladores una serie de mecanismos para facilitar el desarrollo de software. Por norma general el IDE nos proporciona un editor de código fuente entre otras herramientas como la posibilidad de construcción automática de código. Además la mayoría suelen ir acompañados por un depurador que permite al usuario examinar, probar y eliminar errores. Otra de la característica que suelen incluir la mayoría de los IDE es el auto-completado inteligente de código, proporcionando una mayor agilidad y evitando un gran número de errores.

3.1.1. Eclipse

Eclipse es uno de los IDE más utilizados para el desarrollo de aplicaciones en Java. La razón principal es que se trata de una plataforma de código abierto y posee una gran cantidad de *plugins* compatibles.

Sus principales características son su editor de texto con analizador sintáctico, la capacidad de compilación en tiempo real, además de integrar diferentes herramientas para el control de versiones.

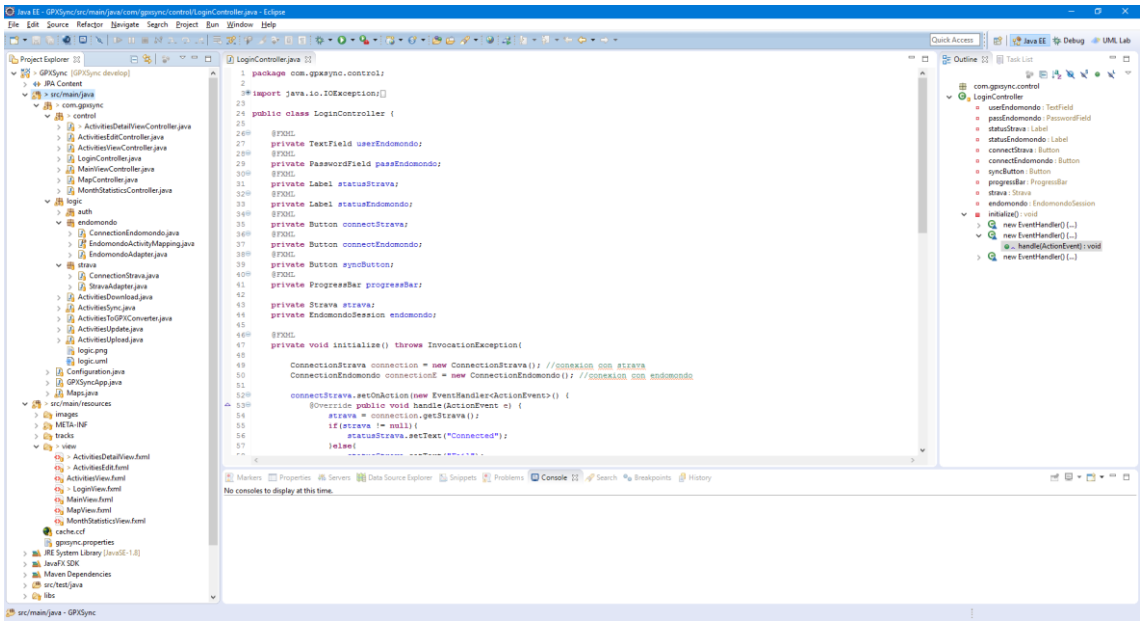

Ilustración 1 - Eclipse IDE

3.1.1.1. Maven

Maven es una herramienta para la construcción de proyectos Java, utiliza el formato XML para la configuración y definición del proyecto. Es decir, es la herramienta encargada de generar todas las estructuras de directorios para nuestro proyecto.

Cuando hablamos de Maven es imprescindible hablar del término POM o *Project Object Model* el cual es el fichero indicado para describir todas aquellas dependencias externas con las que se va a construir nuestro proyecto. Una de las principales características de Maven es la posibilidad de descargar dinámicamente *plugins* o librerías que nuestro proyecto necesite.

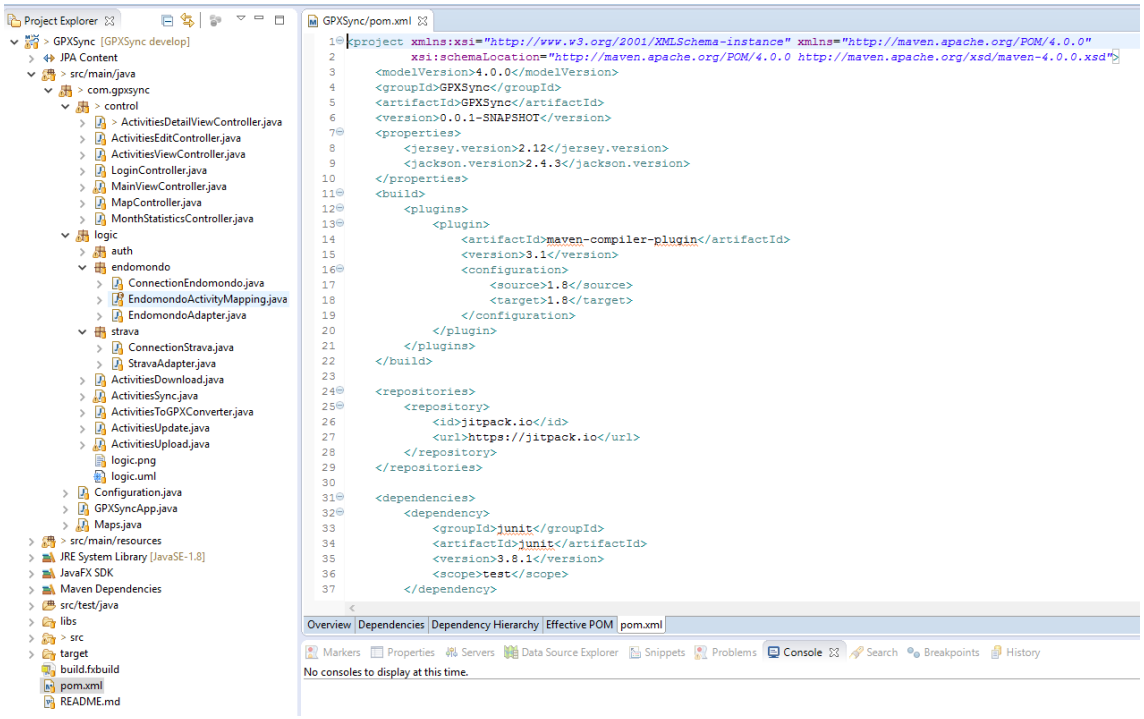

Ilustración 2 - Ejemplo fichero pom.xml

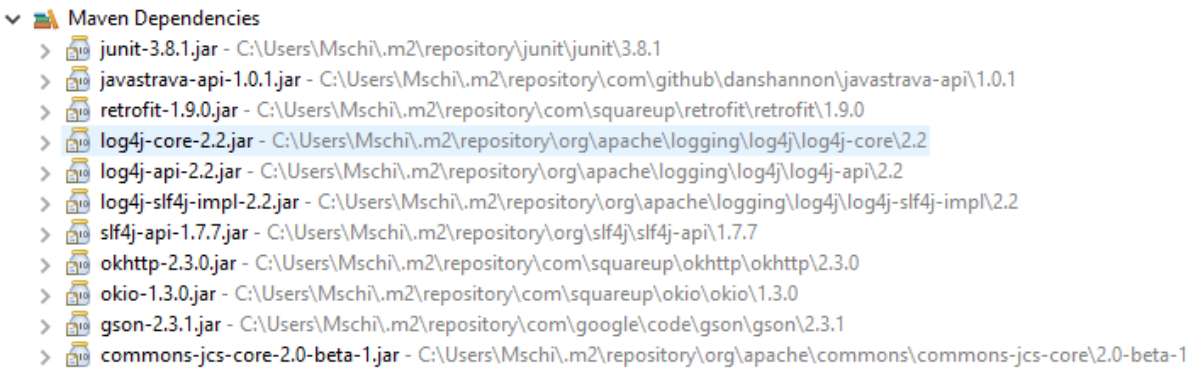

Ilustración 3 - Dependencias Maven

3.1.2. JavaFX Scene Builder 2.0

JavaFX Scene Builder es una herramienta desarrollada por Oracle para facilitar la creación de una interfaz gráfica en JavaFX de una forma visual y sin el uso directo de código fuente. Mediante esta herramienta el usuario es capaz de construir la interfaz utilizando diferentes componentes y además modificar las propiedades de los mismos. JavaFX Scene Builder trabaja con ficheros FXML una extensión de los ficheros XML convencionales, dicho fichero será la clase que represente a la vista dentro del patrón MVC.

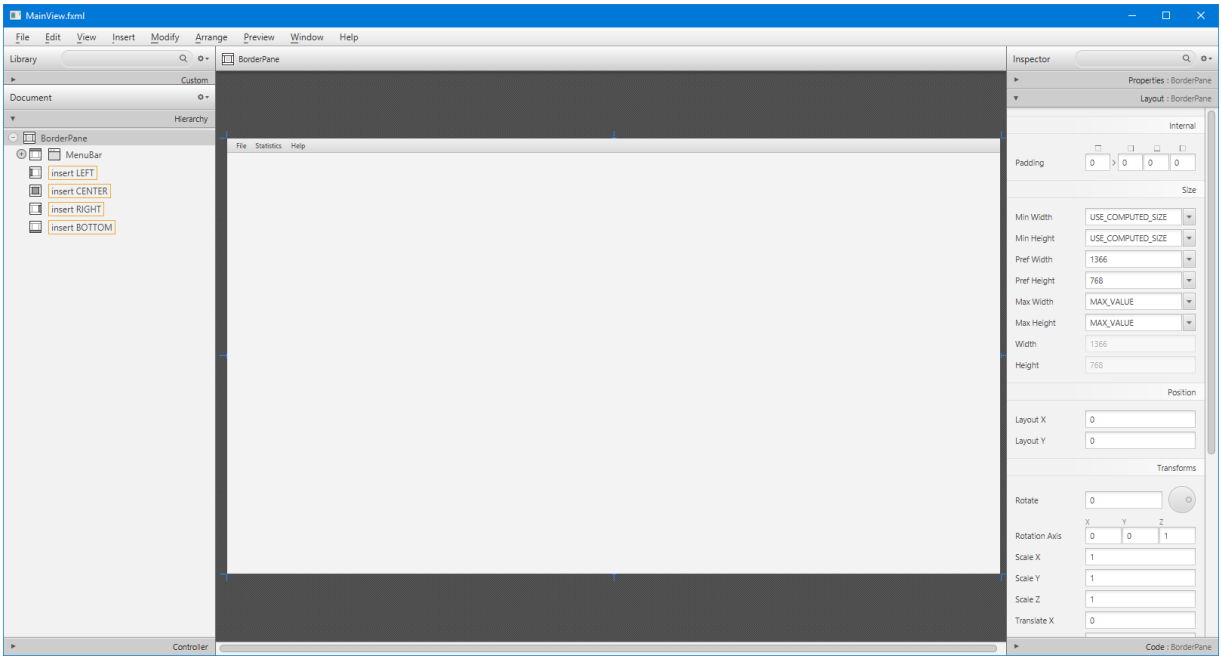

Ilustración 4 - JavaFX Scene Builder

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

<?import java.lang.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.BorderPane?>

<BorderPane fx:id="root" maxHeight="1.7976931348623157E308" maxWidth="1.7976931348623157E308" prefHeight="768.0" prefWidth="1366.0"
    <top>
        <MenuBar BorderPane.alignment="CENTER">
            <menus>
                <Menu mnemonicParsing="false" text="File">
                    <items>
                        <MenuItem mnemonicParsing="false" text="Close" />
                    </items>
                </Menu>
                <Menu mnemonicParsing="false" text="Statistics">
                    <items>
                        <MenuItem mnemonicParsing="false" onAction="#handleShowMonthStatistics" text="Show Statistics" />
                    </items>
                </Menu>
                <Menu mnemonicParsing="false" text="Help">
                    <items>
                        <MenuItem mnemonicParsing="false" text="About" />
                    </items>
                </Menu>
            </menus>
        </MenuBar>
    </top>
</BorderPane>

```

Ilustración 5 - Fichero FXML

3.2. Java

Java es un lenguaje de programación que fue diseñado con el objetivo de crear un lenguaje con el menor número de dependencias de implementación posibles.

Como características principales tenemos que se trata de un lenguaje de propósito general, es decir, puede ser utilizado para varios propósitos, una conexión a base de datos, comunicación entre dispositivos o diseño de imágenes entre otros. Además se trata de un lenguaje orientado a objetos, la programación orientada a objetos es una innovadora forma de obtener resultados mediante la creación de objetos que manipulan los datos de entrada para obtener datos de salida concretos.

3.3. Java FX

JavaFX es una tecnología software la cual al combinarse con Java permite al desarrollador crear aplicaciones que destacan por su aspecto visual. Como bien hemos dicho para poder trabajar con esta tecnología es imprescindible disponer de *Java Runtime Environment*.

Sus principales características son, la integración de contenidos multimedia como pueden ser gráficos, animaciones o videos. Además proporciona la posibilidad de trabajar de forma independiente entre el desarrollo y el diseño de una aplicación.

4. Arquitectura de la aplicación

Como referencia a la arquitectura de la aplicación, se van a describir en las siguientes líneas cada una de las tecnologías y lenguajes que se ha optado por utilizar en cada uno de los casos y a su vez se va a justificar la elección de cada una de ellas.

4.1. Entornos de desarrollo

En primera instancia una de las decisiones a tomar antes de empezar con el desarrollo de la aplicación ha sido la de elegir el entorno de desarrollo oportuno. Desde que empecé a cursar el Grado de Ingeniería Informática se nos hizo saber la existencia de diferentes IDE, pero por su carácter de código abierto y por su gran expansión en diferentes ámbitos empresariales, Eclipse fue el elegido. Durante mi último curso de grado, tuve la posibilidad de realizar prácticas en una de las empresas líder en desarrollo de software, Eclipse era el IDE utilizado para multitud de desarrollos dentro de la empresa, por lo que me he ido sintiendo cada vez más cómodo con Eclipse.

Concluyendo, gracias a la experiencia previa adquirida en ambos ámbitos, Eclipse ha sido el IDE que he decidido utilizar para el desarrollo de mi proyecto, ya que me proporciona las mecanismos necesarios para la implementación del mismo además de facilitarme más las tareas al estar ya familiarizado con sus diferentes herramientas.

Destacar a su vez el uso de la herramienta *JavaFX Scene Builder* que se ha utilizado para el diseño de las interfaces de usuario. Esta herramienta genera archivos en formato FXML. Al editar este fichero mediante *JavaFX Scene Buider* nos evitamos la tediosa tarea de construir la interfaz gráfica utilizando código.

4.1.1. Control de versiones

El proyecto ha sido desarrollado de forma individual, por lo que la necesidad de incorporar un software para el control de versiones no era imprescindible. Aun así he considerado que la utilización del mismo podría ayudarme para mantener un orden entre los diferentes cambios que iba realizando al proyecto, así como la supervisión del tutor mediante el gestor de incidencias que provee la herramienta utilizada. A su vez el hecho de mantener todo el trabajo realizado en un repositorio remoto donde poder descargarlo y reanudarlo en cualquier otro dispositivo ha sido otra de las razones por la que incorporar esta herramienta.

4.1.1.1. Bitbucket

El software para control de versiones elegido ha sido Bitbucket [7], al igual que Eclipse durante el grado de Ingeniería Informática he tenido la posibilidad de trabajar con esta herramienta en alguna de las asignaturas cursadas, por este motivo disponía de algunos conocimientos básicos sobre la misma, no obstante, la incorporación de esta herramienta me ha servido para profundizar mejor y aprender nuevas funcionalidades sobre Bitbucket.

Para utilizar el servicio que proporciona Bitbucket se podría haber utilizado el propio gestor que incorpora Eclipse para el control de versiones, pero finalmente me decidí a utilizar SourceTree, una aplicación de escritorio que se sincroniza directamente con la plataforma de control de versiones para así poder mantener el proyecto sincronizado en todo momento. El motivo de esta elección ha sido la interfaz gráfica que proporciona SourceTree en la que es posible visualizar todos los cambios realizados en cada una de las ramas, así como crear nuevas ramas o nuevas características en nuestra aplicación.

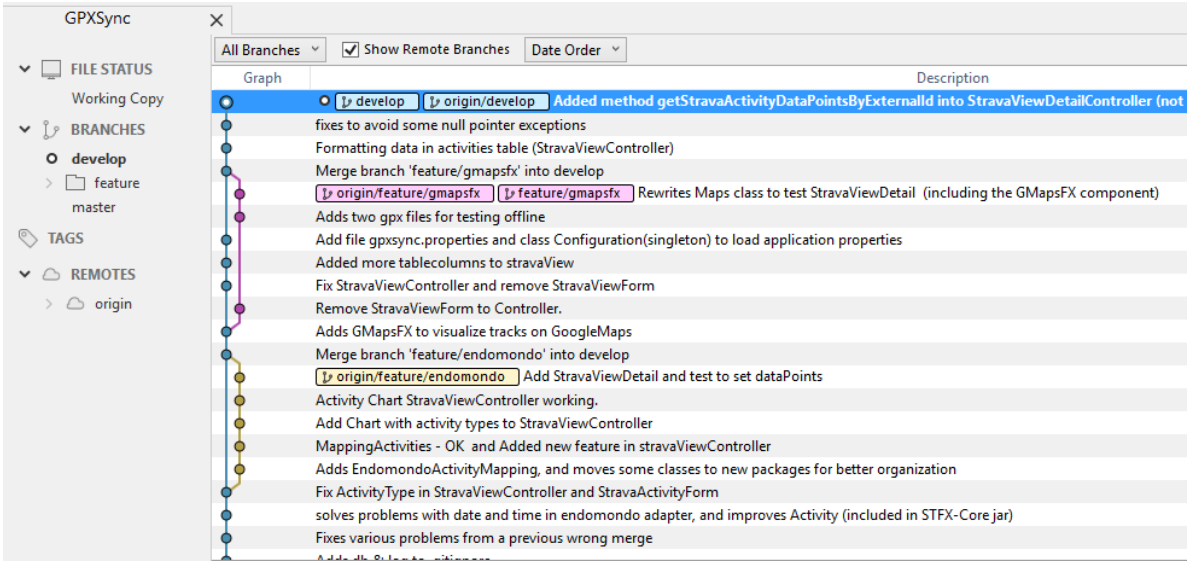

Ilustración 6 – SourceTree

Además de la posibilidad de observar todos y cada uno de los cambios realizados en el código en cada uno de los puntos.

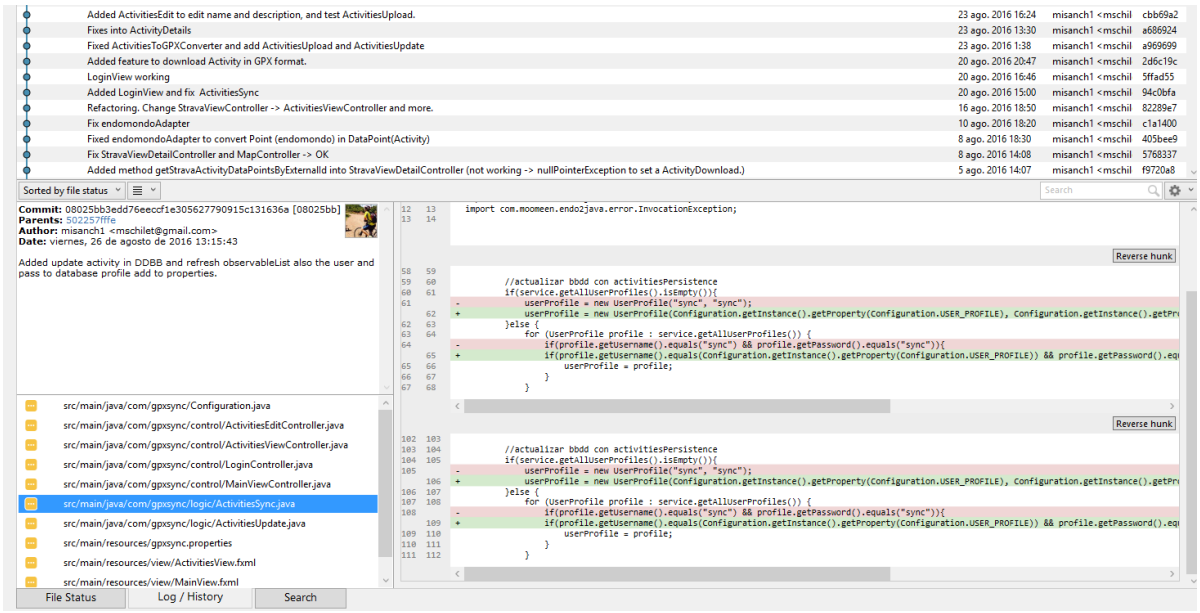

Ilustración 7 - Comparativa cambios repositorio

4.2. API

Para el desarrollo de la aplicación se han utilizado diferentes API de diferentes servicios, en el siguiente apartado voy a dedicar unas líneas a exponer el porqué de la utilización de estas, así como las ventajas e inconvenientes que han podido aportar a la aplicación.

4.2.1. Strava API v3 (javastravav3api)

La plataforma Strava dispone de diferentes API en diferentes lenguajes de programación, en este caso en particular para el desarrollo de nuestro proyecto se ha elegido Strava API v3 [8]. Su implementación basada en Java 8 y su completa y extensa funcionalidad me ha llevado a elegirla entre otras alternativas.

Además, otro de los motivos por el que se ha optado por esta alternativa es su integración con Maven. Gracias a esta última característica ha sido muy cómodo utilizar la API para el servicio Strava, ya que era Maven el encargado de recopilar las dependencias que iba necesitando mi proyecto a medida que avanzaba.

Cabe destacar como una de las grandes ventajas, la utilización del protocolo OAuth2 que utiliza para autenticarse con la plataforma Strava, este protocolo permite a terceros acceder a los contenidos de la plataforma sin la necesidad de que nuestra aplicación tenga que manejar ni conocer los credenciales del usuario.

En apartados futuros me gustaría profundizar sobre este protocolo, mostrando algunos detalles sobre su utilización, así como exponer como ha afectado su uso al proyecto, tanto positiva como negativamente.

4.2.2. Endo2Java

A diferencia de la anterior plataforma en la que disponíamos de diferentes API, el caso de Endomondo es totalmente opuesto. Desde hace años Endomondo revocó la posibilidad a los desarrolladores de utilizar los datos de su plataforma, por lo que no es posible de forma oficial conectarse a la plataforma desde aplicaciones de terceros y obtener información.

Después de un trabajo de investigación encontramos Endo2Java [9], una API no oficial de la que es posible extraer información de Endomondo, aunque bastante limitada en los detalles. Como principal inconveniente destaco que esta API no permite la escritura en la plataforma web, por lo que será una limitación a nuestra aplicación. Por otro lado destacar también la forma que utiliza para autenticarse con la plataforma Endomondo, ya que a diferencia de la API utilizada en Strava, esta utiliza el mecanismo tradicional de usuario y contraseña.

Finalmente, se elige la incorporación de esta API y se trabajará en mejorar algunos aspectos de la misma, para adaptarla a las necesidades de nuestra aplicación.

4.2.3. GMapsFx

GMapsFx [10] es una de las API más utilizadas para la integración de mapas de Google Maps en aplicaciones de escritorio desarrolladas en JavaFX. La elección de esta API ha sido una fácil elección ya que no existían muchas alternativas tan fiables y completas.

La librería nos proporciona la posibilidad de visualizar cada uno de los recorridos gestionados por nuestra aplicación. Pudiendo interaccionar con el mapa e intercalar entre diferentes modos de visualización.

4.2.4. GeoKarambolaLib

GeoKarambolaLib [11] es una API orientada al manejo de ficheros GPX, se trata de una librería java libre y de código abierto (GPLv3). La integración en nuestro proyecto nos proporciona grandes ventajas a la hora de manejar este tipo de fichero. Las utilidades que proporciona la librería son inmensas, pero para satisfacer nuestros requerimientos únicamente utilizaremos algunos de los módulos que nos proporciona.

Por estos motivos, se ha elegido la integración de dicha librería en el proyecto, proporcionando la posibilidad de mapear todos los datos con los que trabaja nuestro gestor, a un fichero GPX siguiendo el estándar GPX 1.1.

4.3. Persistencia

La aplicación desarrollada se comunica con diferentes plataformas web, existiendo una gran carga de comunicaciones entre la aplicación y los diferentes servicios. Por este motivo, se ha visto interesante hacer persistentes muchos de los datos que consume la aplicación. Con esto, conseguimos ahorrar comunicaciones innecesarias con las diferentes API, así como agilizar los procesamientos de datos del gestor de recorridos.

4.3.1. JPA

Java Persistence API [12] o más conocida por JPA, se trata de la API de persistencia que se ha elegido en nuestra aplicación para trabajar con datos persistentes. Uno de los motivos de esta elección ha sido la idea de no perder las ventajas de la orientación a objetos que nos proporciona Java en el momento que se interactúa con una base de datos.

Cuando hablamos de JPA es imprescindible conocer el término *Entity*, ya que cada entidad es una clase de nuestra aplicación, la cual va a persistir asociada a una tabla en base de datos, se conoce como un mapeo objeto/relacional, es decir como ya hemos dicho la relación entre entidades Java y tablas de la base de datos. Para identificar una relación se utilizan anotaciones en las propias clases Java.

4.3.1.1. ObjectDB

Por otro lado la base de datos elegida para trabajar con los datos ha sido *ObjectDB* [13]. Se trata de una base de datos orientada a objetos para Java. A diferencia de otras bases de datos orientadas a objetos, *ObjectDB* requiere de la API estándar de Java (JPA) *o Java Data Objects* (JDO) para la interacción con la aplicación.

La gran cantidad de documentación de la que dispone *ObjectDB* y el gran apoyo de su comunidad ha sido el requisito principal a la hora de elegir esta base de datos. Además, de ser una base de datos constatada y que se acoplaba perfectamente a nuestras necesidades.

4.3.1.1.1. Database Explorer

Database Explorer es una de las herramientas que proporciona *ObjectDB* para visualizar mediante una sencilla interfaz gráfica el contenido de la base de datos, así como realizar consultar y modificaciones sobre ella.

A continuación se visualiza la herramienta anteriormente descrita:

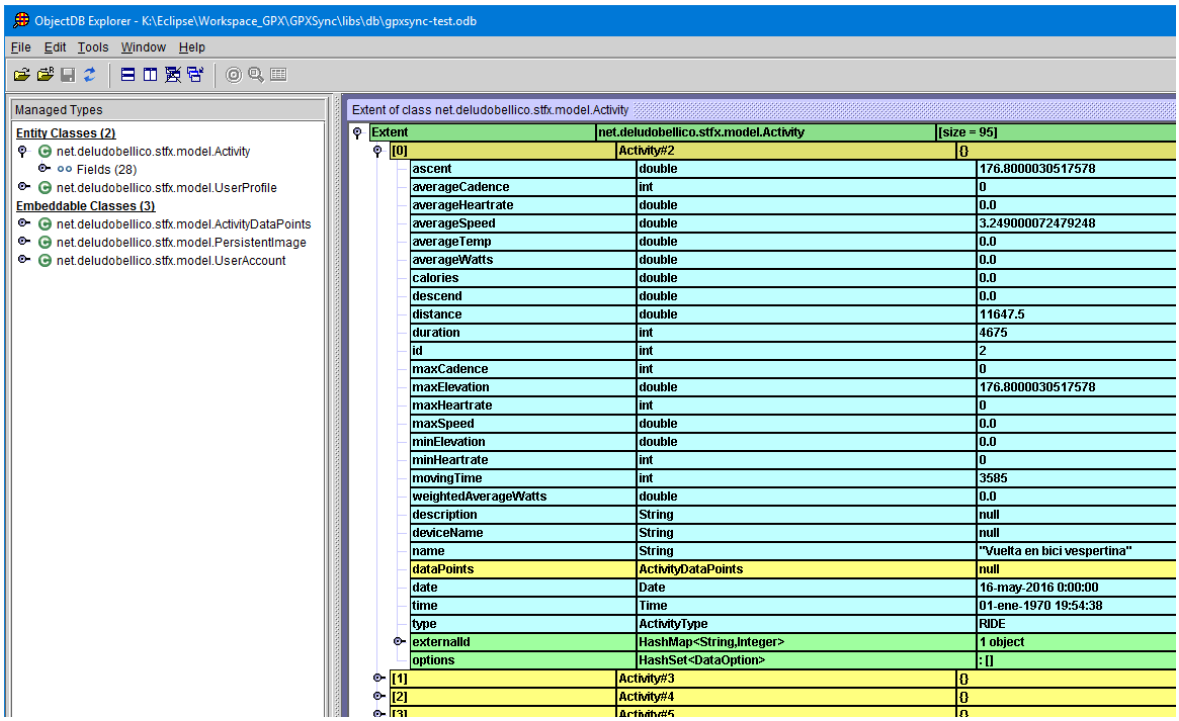

Ilustración 8 - ObjectDB Database Explorer

5. Análisis

En el siguiente apartado se va a realizar un análisis sobre el presente proyecto, utilizando para ello diferentes diagramas. Utilizaremos el lenguaje UML para describir métodos y procesos del sistema.

5.1. Diagrama de Casos de uso

Los diagramas de casos de uso son utilizados para describir de una manera simple y esquemática cada uno de los casos de uso del sistema que han sido detallados durante la especificación de requerimientos.

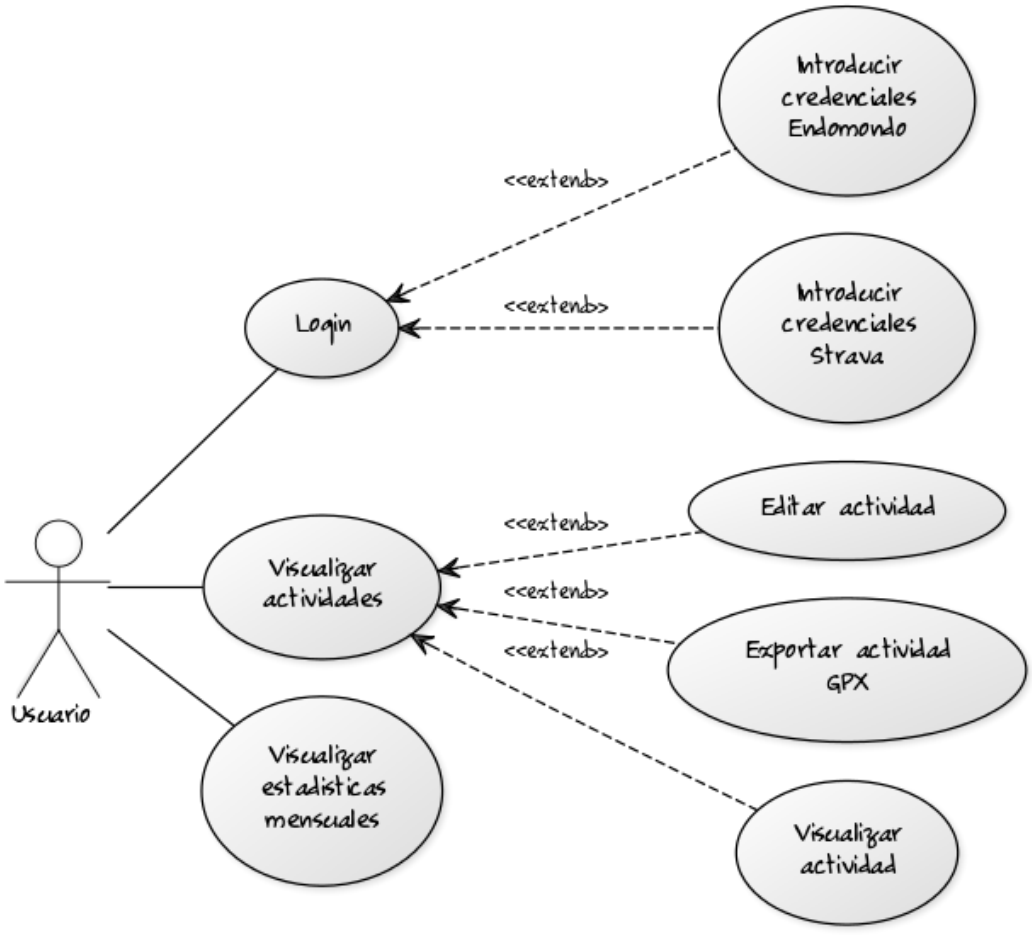

Ilustración 9 - Casos de uso (general)

5.1.1. Diagrama casos de uso detallado

Login

Actores	Usuario aplicación
Descripción	Se introducen los credenciales en las plataformas correspondientes
Precondición	Ninguna
Flujo Normal	▪ El sistema muestra la ventana para pedir los credenciales de las plataformas. ▪ El actor introduce los datos requeridos. ▪ El sistema comprueba los credenciales y muestra el mensaje *Connected*. ▪ El actor pulsa sobre el botón Synchronize para comenzar la sincronización de las plataformas.
Flujo excepcional	▪ El sistema comprueba los credenciales y muestra el mensaje *Fail*.

Visualizar actividades

Actores	Usuario aplicación
Descripción	Visualización del perfil de usuario y las actividades sincronizadas
Precondición	RF.01 Autenticación de la aplicación con las plataformas RF.02 Descarga de las actividades y perfil de usuario RF.03 Conversión y unificación de datos RF.04 Sincronización de las actividades entre plataformas RF.05 Persistencia de las actividades en base de datos RF.06 Recuperar actividades de base de datos RF.13 Subir una actividad a las diferentes plataformas
Flujo Normal	▪ El sistema muestra la ventana con el perfil de usuario y los datos del mismo. ▪ El sistema carga en la ventana todas las actividades sincronizadas correspondientes al perfil de usuario.
Flujo excepcional	Ninguno

Editar actividad

Actores	Usuario aplicación
Descripción	Se muestra la ventana para editar el nombre y descripción de una actividad.
Precondición	RF.07 Visualización de lista de actividades RF.08 Edición de una actividad concreta RF.09 Actualizar una actividad remota
Flujo Normal	▪ El actor pulsa sobre el botón *Edit* de una actividad. ▪ El sistema muestra la ventana para editar una actividad. ▪ El actor introduce el nuevo nombre y la nueva descripción. ▪ El actor pulsa sobre el botón *Update*. ▪ El sistema actualiza la actividad en base de datos y en la plataforma remota. ▪ El sistema muestra la lista con los datos actualizados.
Flujo excepcional	▪ El actor pulsa sobre el botón *Cancel*. ▪ El sistema no realiza ninguna modificación.

Visualizar actividad

Actores	Usuario aplicación
Descripción	Se visualizan los detalles de una actividad en concreto.
Precondición	RF.07 Visualización de lista de actividades RF.10 Visualización de una actividad en Google Maps RF.11 Visualización de estadísticas de una actividad
Flujo Normal	▪ El actor pulsa sobre el botón *Details* de una actividad. ▪ El sistema muestra la ventana donde se visualizan dos pestañas. ▪ El actor pulsa sobre *Map* y visualiza el recorrido de la actividad sobre el mapa. ▪ El actor pulsa sobre *Statistics* y visualiza las estadísticas del recorrido.
Flujo excepcional	Ninguno

Exportar actividad GPX

Actores	Usuario aplicación
Descripción	Se descarga la actividad seleccionada en formato GPX.
Precondición	RF.14 Exportar una actividad a un archivo GPX
Flujo Normal	▪ El actor pulsa sobre el botón *GPX* de una actividad. ▪ El sistema muestra el explorador de archivos para seleccionar donde se quiere guardar el fichero. ▪ El actor indica un nombre para el fichero GPX y selecciona guardar. ▪ El sistema genera el archivo GPX y lo guarda en la ruta seleccionada por el actor.
Flujo excepcional	▪ El sistema muestra el explorador de archivos para seleccionar donde se quiere guardar el fichero. ▪ El actor cancela el proceso de exportar el fichero.

Visualizar estadísticas mensuales

Actores	Usuario aplicación
Descripción	Visualización del perfil de usuario y las actividades sincronizadas
Precondición	RF.07 Visualización de lista de actividades RF.12 Visualización de estadísticas mensuales
Flujo Normal	▪ En la ventana de visualización de actividades el actor pulsa sobre el botón *Statistics*. ▪ El sistema muestra un gráfico con las estadísticas de cada mes.
Flujo excepcional	Ninguno

5.2. Diagrama de clases

A continuación se muestra el correspondiente diagrama de clases de la aplicación donde se puede visualizar la estructura del proyecto, así como cada una de las clases relacionadas.

En primer lugar la ilustración 10 muestra el diagrama de clases mostrando el nombre y las relaciones entre cada una de las clases. En segundo lugar en la ilustración 11 se muestra cada una de las clases con más detalle, especificando cada uno delos métodos que la contienen así como cuál es su salida esperada.

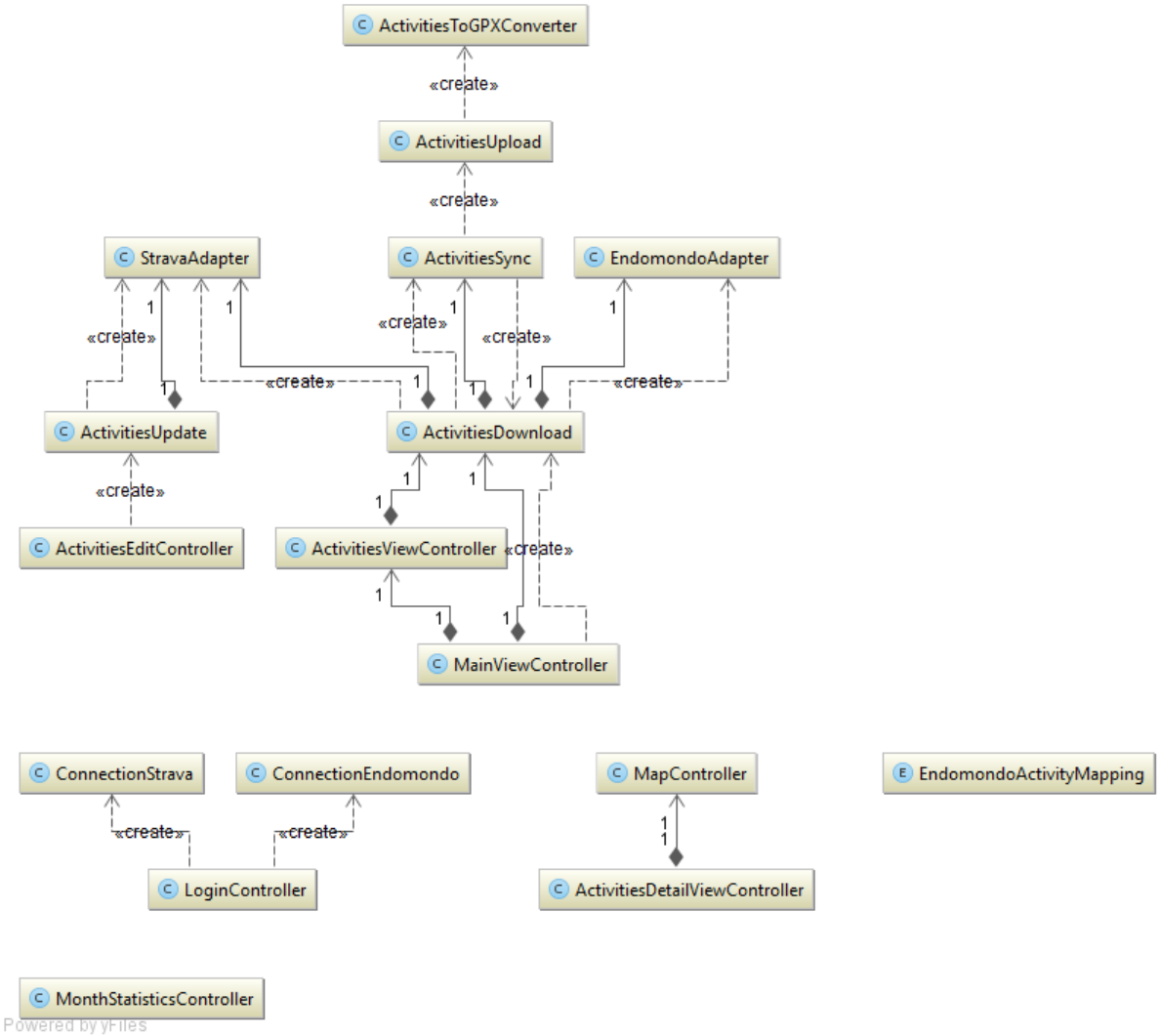

Ilustración 10 - Diagrama de clases

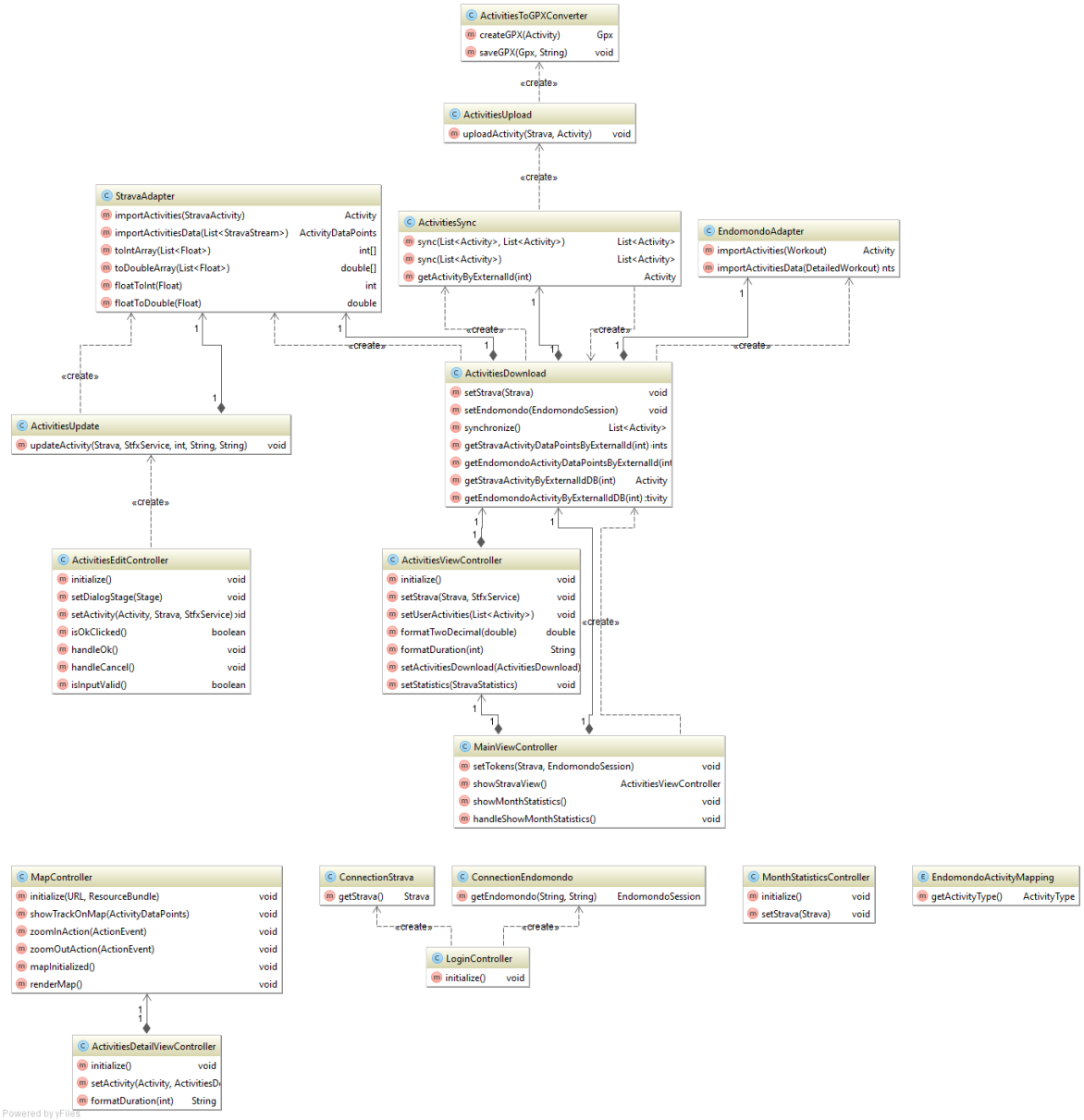

Ilustración 11 - Diagrama de clases (detallado)

6. Diseño

El proyecto ha sido diseñado siguiendo un patrón de arquitectura de software conocido como Modelo Vista Controlador (MVC). La principal característica de este patrón es la forma en la que se estructuran los diferentes datos. A grandes rasgos, podemos distinguir claramente entre los datos y la lógica de negocio de la parte visual, como es la interfaz de usuario y las diferentes comunicaciones con esta.

6.1. Patrón MVC

- **Modelo:** contiene los datos que maneja el sistema, es decir, la lógica de negocio y además en los casos que nuestra aplicación necesite interaccionar con una base de datos, el modelo será el encargado de su gestión.
- **Vista:** contendrá toda la vista, que actúa como interfaz de usuario, es decir toda la información que se muestra al usuario final de nuestra aplicación.
- **Controlador:** su papel es el de intermediario entre el Modelo y la Vista, se encarga de gestionar todo el intercambio de datos entre ellos, así como de transformar y adaptar los datos dependiendo de las necesidades de cada uno.

La utilización del patrón MVC nos proporciona una serie de ventajas durante la implementación de nuestra aplicación, entre las cuales podemos destacar las siguientes:

- ✓ Permite crear módulos de código que son reutilizables y a su vez pueden ser intercambiados por otros sin afectar a la estructura del proyecto.
- ✓ Proporciona a la aplicación escalabilidad.
- ✓ Permite abstraer a la vista totalmente del procesado de los datos, dedicándose esta únicamente a mostrar los datos al usuario.
- ✓ En el caso de realizar pruebas unitarias, la utilización del patrón MVC facilitará considerablemente dicha labor.

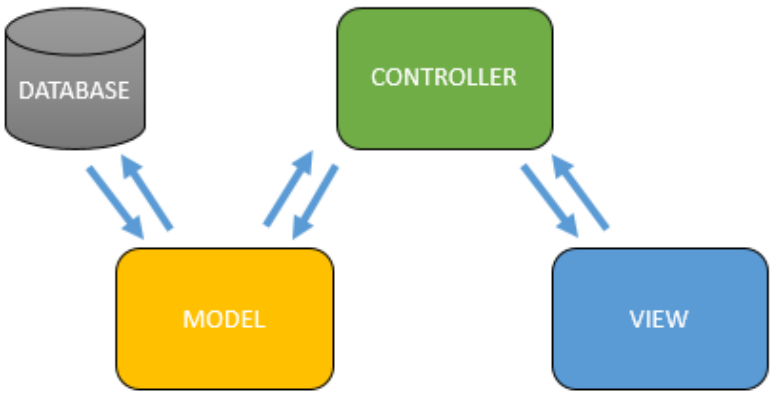

Ilustración 12 - Patrón MVC

6.1.1. Estructura del proyecto

Para una mejor visualización de las clases y de los recursos de la aplicación se he decidido utilizar una estructura de paquetes, con el fin de separar visualmente las diferentes capas.

Control

El paquete *control* está compuesto por todas las clases que van a encargarse de coordinar la capa *logic* y *view*, es decir se encargará de la interacción entre la lógica de negocio y la interfaz gráfica.

Logic

Contiene las clases con toda la lógica de la aplicación, se encargará del modelado de los datos, así como adaptarlos a las necesidades del usuario, además dentro del paquete encontramos también la parte de persistencia de datos.

View

Es el paquete encargado de contener todas las vistas que le serán presentadas al usuario final, las vistas estarán ubicadas en el apartado *resources* de nuestro proyecto. El motivo es la forma declarativa en la que se definen las interfaces, utilizando FXML, por esta razón es más correcto separar las vistas del código de la aplicación.

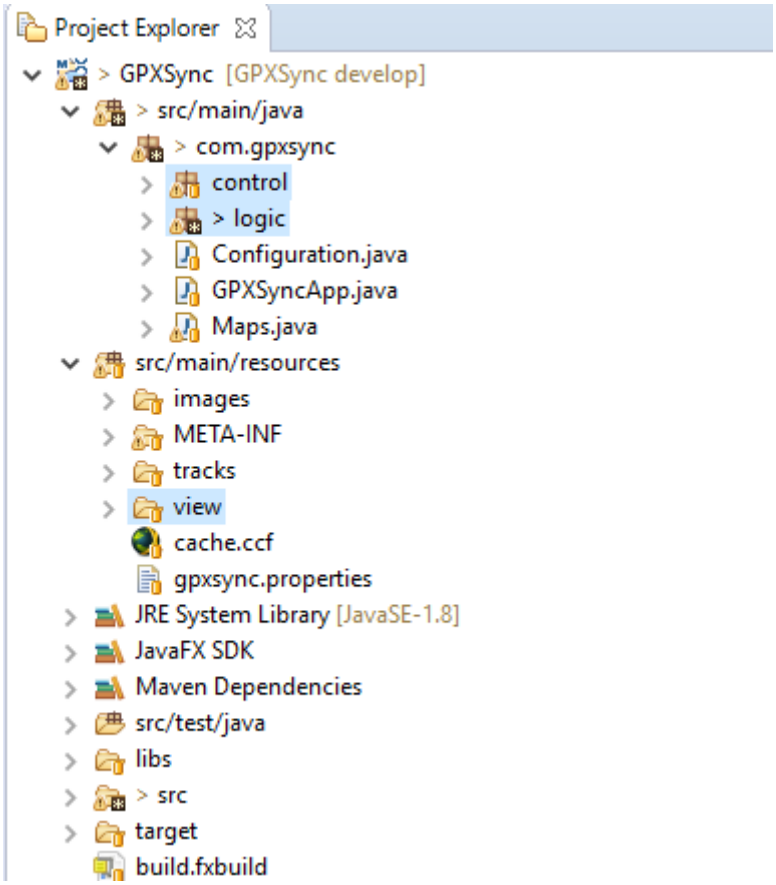

Ilustración 13 - Estructura del proyecto

6.1.2. Capa de controladores

El paquete ***control*** va a contener los diferentes controladores encargados de la interacción entre la campa vista y la capa lógica.

LoginController.java	Se encarga de la vista *LoginView* donde el usuario introduce sus credenciales, para así poder establecer las diferentes conexiones con las plataformas.
MainViewController.java	Se trata del controlador encargado de la vista principal, la cual va a contener a nuestra aplicación.
ActivitiesViewController.java	Es la clase encargada de comunicarle a su vista correspondiente todas las actividades disponibles en las plataformas después del procesado del sistema. Así como permitirle al usuario interactuar mediante una serie de entradas a las que debe de responder.
ActivitiesDetailViewController.java	Es el controlador encargado de interactuar con la vista para mostrar todos los detalles de una actividad seleccionada por el usuario.
MapController.java	Este controlador es el encargado de proporcionar a la vista *MapView* todos los puntos del recorrido para así mostrar al usuario el recorrido sobre Google Maps.
ActivitiesEditController.java	Es el controlador encargado de interaccionar con la vista *ActivitiesEdit*, la cual permite al usuario editar una actividad. Y que está actividad sea correctamente modificada en el sistema y la plataforma web.
MonthStatisticsController.java	Se encarga de proporcionar a la vista los datos necesarios para mostrar estadísticas sobre las actividades realizadas cada mes.

6.1.3. Capa modelo o lógica de negocio

Dentro del paquete ***logic*** se ha querido mantener una organización entre las diferentes plataformas utilizadas, separando en dos paquetes la parte lógica que corresponde a cada plataforma, y posteriormente dejando una parte común a las dos.

Endomondo *(package)*

ConnectionEndomondo.java	Define los parámetros necesarios para realizar la conexión con la plataforma Endomondo. Es la clase encargada de retornar el *token* de acceso a la plataforma.
EndomondoActivityMapping.java	Se trata de una clase de tipo *enum* encargada de mapear los diferentes tipos de actividades que encontramos en las distintas plataformas.
EndomondoAdapter.java	Es la clase encargada de la trasformación y el modelado de los datos. Su principal función es la de adaptar las diferentes actividades de Endomondo a una actividad común que el sistema pueda interpretar.

Strava *(package)*

ConnectionStrava.java	Define los parámetros necesarios para realizar la conexión con la plataforma Strava. Es la clase encargada de retornar el *token* de acceso a la plataforma.
StravaAdapter.java	Es la clase encargada de la trasformación y el modelado de los datos. Su principal función es la de adaptar las diferentes actividades de Strava a una actividad común para el sistema.

Además de los dos paquetes anteriores, el paquete ***logic*** contiene las siguientes clases comunes para las plataformas utilizadas en la aplicación.

ActivitiesDownload.java	La siguiente clase java es la encargada de descargar todos los datos de las distintas plataformas, centralizando así los procesos de descarga de las actividades y sus propiedades.
ActivitiesSync.java	Es la encargada del proceso de sincronización de la aplicación. Su principal función consiste en unificar las actividades que han sido procesadas por los adaptadores que hemos descrito anteriormente, *StravaAdapter* y *EndomondoAdapter*. Además se encarga de mantener la integridad de datos entre las plataformas y la base de datos.
ActivitiesToGPXConverter.java	Como su nombre indica, se trata de una clase capaz de convertir una actividad de nuestra aplicación, en un archivo GPX, además de proporcionarnos la posibilidad de guardar el fichero GPX en nuestro ordenador.
ActivitiesUpdate.java	La presente clase Java nos proporciona la posibilidad de actualizar una actividad guardada en el sistema, y actualizar su nuevo estado en la plataforma web.
ActivitiesUpload.java	Es la encargada de subir nuevas actividades a la plataforma web. Permite tras la importación de nuevas actividades al sistema, que estas se actualicen en la plataforma correspondiente.

6.1.4. Capa vistas

En cuanto al paquete ***View*** contiene las diferentes vistas de nuestra aplicación, utilizando el formato FXML.

LoginView.fxml	Muestra la vista donde el usuario debe logarse para acceder a la aplicación.
MainView.fxml	Se trata de la vista principal donde se va a visualizar la aplicación.
ActivitiesView.fxml	Muestra la lista de actividades final al usuario, además de una serie de botones para visualizar detalles, exportar actividad en GPX y editar.
ActivitiesDetailView.fxml	Muestra diferentes estadísticas en forma de gráficas, además de contener la vista *MapView*.
MapView.fxml	Permite la visualización de una actividad en Google Maps.
ActivitiesEdit.fxml	Proporciona la posibilidad de editar el nombre y la descripción de una actividad.
MonthStatisticsView.fxml	Permite visualizar al usuario una gráfica con las actividades que se han realizado todos los meses de cada año.

7. Detalles de implementación

En el siguiente apartado se van a abarcar algunos aspectos más detallados que he considerado que son importante profundizar sobre ellos.

7.1. Conversión y unificación de datos

Uno de los punto más importantes durante la implementación de la aplicación es el de adaptar los datos. Cada plataforma utiliza de forma diferente los datos que muestra en sus respectivas webs, la mayoría de los tipos de datos no coinciden entre las plataformas, por lo que es necesaria la adaptación y construcción de nuevos tipos de datos comunes, que la aplicación sea capaz de interpretar.

Para ello, es necesario procesar y convertir todos los datos descargados de cada una de las plataformas. La siguiente ilustración, muestra de una forma resumida como se transforman los diferentes datos.

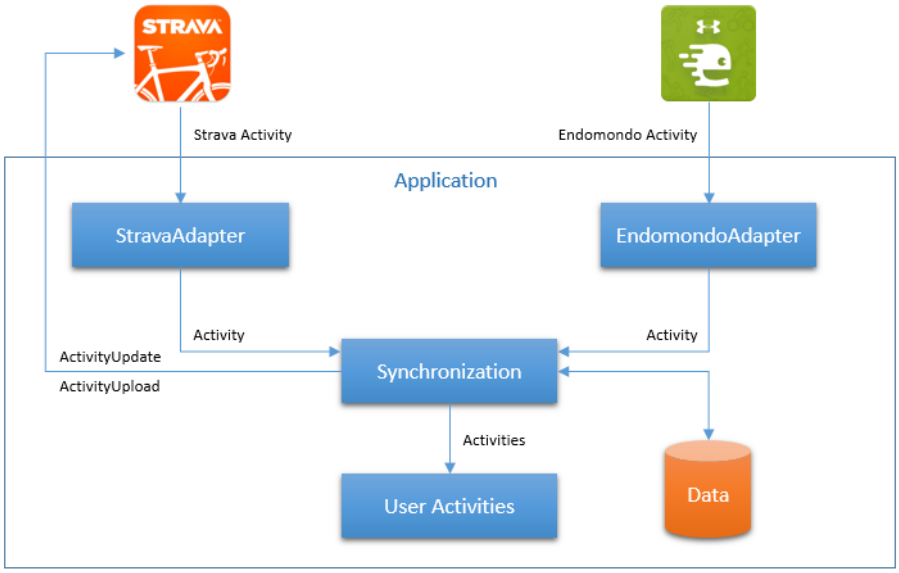

Ilustración 14 - Adaptación de datos

En la siguiente imagen se muestra un pequeño fragmento de StravaAdapter.

```
44          activity.setDescription(stravaActivity.getDescription());
45          activity.setDistance(stravaActivity.getDistance());
46          activity.setDuration(stravaActivity.getElapsedTime());
47          activity.setMaxElevation(stravaActivity.getTotalElevationGain());
48          activity.setMovingTime(stravaActivity.getMovingTime());
49          activity.setName(stravaActivity.getName());
50          activity.setTime(stravaActivity.getStartDate().toLocalTime());
51          if (null != stravaActivity.getWeightedAverageWatts()) {
52              activity.setAverageWatts(stravaActivity.getAverageWatts());
53              activity.setWeightedAverageWatts(stravaActivity.getWeightedAverageWatts());
54              activity.getOptions().add(DataOption.POWER);
55          }
```

Ilustración 15 - Fragmento StravaAdapter

7.2. Protocolo de autenticación OAuth2

Como ya comentamos con anterioridad, Strava hace uso del protocolo OAuth2 para autenticar a los usuarios que acceden desde aplicaciones de terceros. OAuth2 nos permite acceder a contenidos externos a nuestra aplicación, sin necesidad de que la aplicación cliente conozca los credenciales del usuario.

7.2.1. Introducción Strava OAuth2

Un escenario claro para explicar nuestro objetivo utilizando este protocolo es el siguiente:

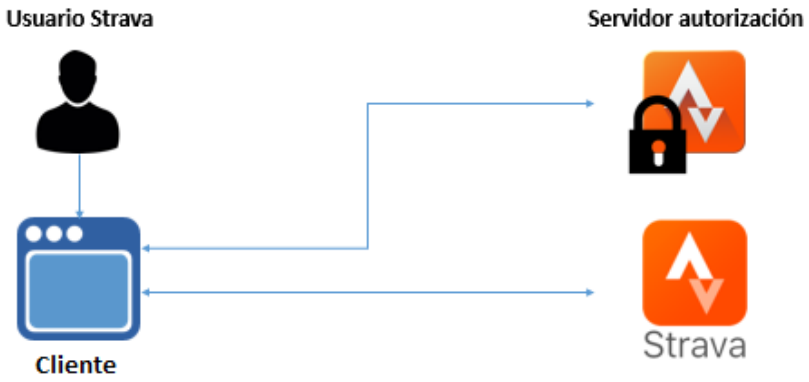

Ilustración 16 - OAuth2

- **Usuario Strava**: es el propietario de los recursos que se alojan en la plataforma Strava.
- **Cliente:** En este escenario nuestra aplicación tendrá el rol de cliente, es decir la aplicación de terceros desde la cual se hacen peticiones a los recursos protegidos del usuario Strava con su autorización.
- **Servidor autorización:** es el responsable de generar los *tokens* de acceso y validar a los usuarios con sus credenciales.
- **Servidor de recursos (Strava):** donde se alojan los recursos del usuario. Acepta y responde peticiones utilizando el *token* de acceso incluido en el cuerpo de la petición.

7.2.2. Integración con el gestor de recorridos

Para la utilización del protocolo OAuth2 en nuestra aplicación ha sido necesario registrar el gestor de recorrido en la web de Strava, para así obtener los datos que identificaran a nuestra aplicación dentro de la plataforma. Los datos que se nos han proporcionado son los siguientes:

- **Client Id**: se trata del ID que identifica a nuestra aplicación dentro de la plataforma Strava, como ya hemos dicho se nos proporciona al registrar la aplicación en Strava.

- **Client Secret**: es un valor secreto que se obtiene al registrar la aplicación, y será necesario para generar el *token* de acceso posteriormente.

El proceso de autenticación de la aplicación con la plataforma Strava se inicia con la redirección a un navegador mediante la URL de Strava donde el usuario será autenticado. El usuario una vez introducidos los credenciales en la web de Strava, deberá autorizar a la aplicación para que esta pueda acceder a sus datos.

Si el usuario autoriza a la aplicación, Strava devolverá un código de autorización que será necesario junto al **Client Id** y el **Client Secret** para generar el *token* de acceso necesario.

En la siguiente ilustración se observa la utilización de los parámetros anteriores para la creación del *token de acceso*:

```
37          if (credentials.getClientToken() != null) {
38              AuthorisationService service = new AuthorisationServiceImpl();
39              token = service.tokenExchange(Integer.valueOf(OAuth2Credentials.clientId),
40                      OAuth2Credentials.clientSecret,
41                      credentials.getClientToken(), scopes);
42          }
43          if (token != null) {
44              strava = new Strava(token);
45          }
46          return strava;
```

Ilustración 17 - Token de acceso Strava

7.3. Archivo de Propiedades

Durante la implementación del proyecto se ha decidido la utilización de un fichero de propiedades con el que poder agrupar las configuraciones con las que arrancará la aplicación. En nuestro caso, el fichero de propiedades nos es útil para activar o desactivar las diferentes plataformas sobre las que estamos trabajando además de la configuración de usuario y contraseña de nuestra base de datos.

A continuación mostramos un fragmento del fichero *properties* de nuestra aplicación.

```
1 ####################################GPXSync Properties#####################################
2
3 #ActivitiesDownload
4 ## Enable/Disable strava activity download
5 stravaDownload=true
6 ## Enable/Disable endomondo activity download
7 endomondoDownload=true
8 ## Enable/Disable strava DataPoints download
9 downloadDataPoint=false
10
11 ##BBDD Profile
12 userProfile=sync
13 passProfile=sync
14
```

Ilustración 18 - Fichero Properties

Para la utilización del fichero *properties*, se ha creado una clase *Configuration.java* que va a implementar el patrón de diseño *singleton*, con esto garantizamos que la clase *Configuration* solo tenga una instancia con la que acceder al fichero *properties*. A continuación un pequeño fragmento de la clase *Configuration*.

```
28    private Configuration() {
29        this.properties = new Properties();
30        try {
31            properties.load(Configuration.class.getClassLoader().getResourceAsStream(CONFIG_FILE_NAME));
32        } catch (IOException ex) {
33            ex.printStackTrace();
34        }
35    }//Configuration
36
37    /**
38     * Implementando Singleton
39     *
40     * @return
41     */
42    public static Configuration getInstance() {
43        return ConfigurationHolder.INSTANCE;
44    }
45
46    private static class ConfigurationHolder {
47        private static final Configuration INSTANCE = new Configuration();
48    }
49
50    /**
51     * Retorna la propiedad de configuración solicitada
52     *
53     * @param key
54     * @return
55     */
56    public String getProperty(String key) {
57        return this.properties.getProperty(key);
58    }//getProperty
```

Ilustración 19 - Configuration.java (Singleton)

7.4. Mapeo de actividades

Durante el proceso de unificación de los datos, nos hemos encontrado con el problema de los diferentes tipos de actividades que existen en cada una de las plataformas. Para su resolución se ha optado por la utilización de un *enum*, con el cual realizamos el mapeo directo de cada una de las actividades. Un *enum* es un tipo de dato que consiste en un conjunto de valores con nombre, a los cuales se les conoce como enumeradores. Los nombres de los elementos son como identificadores que se comportan como una constantes. En nuestro caso para el mapeo de las actividades el uso que se le ha dado a los *enum* es el siguiente:

Los enumeradores referencian los tipos de actividades que tenemos en Strava, y su atributo corresponde al tipo de actividad equivalente en la plataforma Endomondo.

```
1  package com.gpxsync.logic.endomondo;
2
3  import net.deludobellico.stfx.model.enums.ActivityType;
4
5  public enum EndomondoActivityMapping {
6      RUNNING(ActivityType.RUN),
7      CYCLING_TRANSPORT(ActivityType.RIDE),
8      CYCLING_SPORT(ActivityType.RIDE),
9      MOUNTAIN_BIKING(ActivityType.RIDE),
10     SKATING(ActivityType.INLINE_SKATE),
11     ROLLER_SKIING(ActivityType.ROLLERSKI),
12     SKIING_CROSS_COUNTRY(ActivityType.BACKCOUNTRY_SKI),
13     SKIING_DOWNHILL(ActivityType.INLINE_SKATE),
14     SNOWBOARDING(ActivityType.SNOWBOARD),
15     KAYAKING(ActivityType.KAYAKING),
16     KITE_SURFING(ActivityType.KITESURF),
17     ROWING(ActivityType.ROWING),
18     SAILING(ActivityType.UNKNOWN),
19     WINDSURFING(ActivityType.WINDSURF),
20     FITNESS_WALKING(ActivityType.WALK),
21     GOLFING(ActivityType.UNKNOWN),
22     HIKING(ActivityType.HIKE),
23     ORIENTEERING(ActivityType.UNKNOWN),
24     WALKING(ActivityType.WALK),
25     RIDING(ActivityType.RIDE),
26     SWIMMING(ActivityType.SWIM),
27     INDOOR_CYCLING(ActivityType.VIRTUAL_RIDE),
28     OTHER(ActivityType.UNKNOWN),
```

Ilustración 20 - Activity Type Mapping

```
49         if(endomondoActivity.getStartTime() != null){
50             ZonedDateTime zoneDate = endomondoActivity.getStartTime();
51             activity.setDate(zoneDate.toLocalDate());
52             activity.setTime(zoneDate.toLocalTime());
53         }
54
55
56         activity.setType(EndomondoActivityMapping.valueOf(endomondoActivity.getSport().name()).getActivityType());
57
58         return activity;
59     }
```

Ilustración 21 - Ejemplo mapeo ActivityType

8. Aplicación final

En el siguiente apartado se pretende mostrar algunos de los resultados finales de la aplicación en lo que a la interfaz gráfica se refiere.

8.1. Ventana de *Login*

Al iniciar la aplicación, se muestra una primera ventana (Ilustración 22) donde el usuario indicara los credenciales necesarios para realizar las diferentes conexiones con las plataformas web. Como ya se ha visto durante el documento, la autenticación a Strava se realiza al pulsar sobre el botón de la parte izquierda *Connect with Strava*, si se trata de la primera vez que iniciamos, la aplicación abrirá un nuevo navegador para que en primer caso autoricemos a la aplicación (Ilustración 23) y en segundo caso introduzcamos los credenciales (Ilustración 24). Si los credenciales son correctos aparecerá el mensaje *Connected*. Si no son correctos se mostrará el mensaje *Fail*.

Por parte de Endomondo, los credenciales son introducidos por el usuario directamente en la ventana de nuestra aplicación. Existen dos apartados en la parte derecha, en la que se debe indicar el usuario y contraseña. Al pulsar sobre *Login* si los credenciales son correctos aparecerá el mensaje *Connected*. Si no son correctos se mostrará el mensaje *Fail*.

Ilustración 22 – Login

8.2. Autorizando a la aplicación (Strava)

Como se ha visto en el apartado anterior, es necesario que el usuario de la aplicación autorice a la misma para que pueda acceder a los datos personales almacenados en la plataforma web Strava. Al pulsar sobre Autorizar el usuario permite que nuestra aplicación pueda acceder a su perfil y sus datos de actividades, ademas de tener también la posibilidad de cargar nuevas actividades:

Ilustración 23 - Autorización Strava

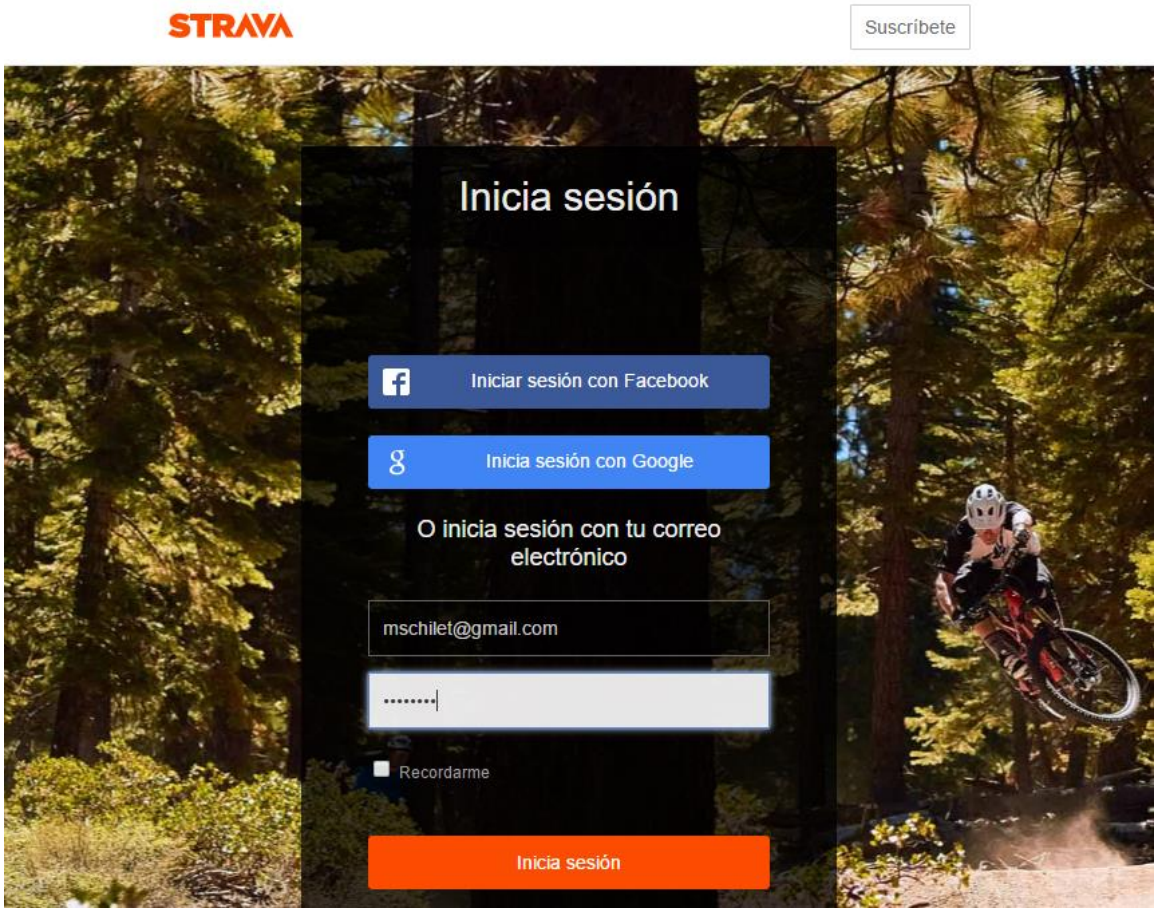

Ilustración 24 - Login Strava

8.3. Visualización de actividades sincronizadas

Una vez autenticados en las plataformas, se muestra la ventana con todas las actividades resultado de todo el proceso de sincronización que posee la aplicación. El usuario final puede visualizar en la parte izquierda de la ventana información detallada sobre su perfil y datos sobre sus actividades, además de un pequeño gráfico donde visualizar la cantidad de actividades de cada tipo de las que dispone.

Cada una de las actividades dispone de tres botones en la parte derecha de la imagen, que proporciona al usuario la posibilidad de visualizar, descargar y editar cualquier actividad.

Permite visualizar el recorrido sobre Google Maps y mostrar estadísticas.

Permite la descarga de una actividad en formato GPX.

Permite editar una actividad de forma local y remota.

En los siguientes aparatados se van a dar más detalles sobre la utilización de cada uno de los botones, así como el resultado de cada uno de estos.

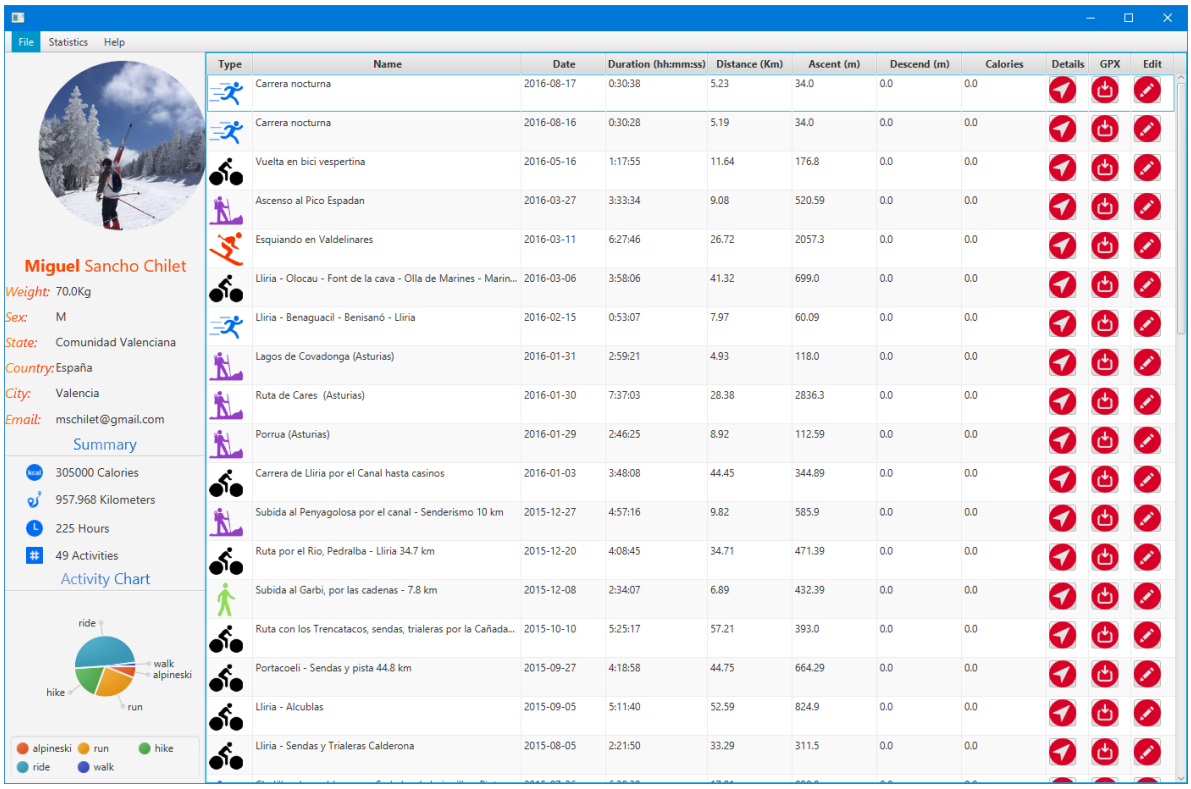

Ilustración 25 - Lista de actividades sincronizadas

8.4. Visualización recorrido Google Maps

Para la visualización del recorrido sobre Google Maps el usuario puede pulsar el botón  situado en la parte derecha de la anterior ventana. Como resultado se abre una nueva ventana en la que el sistema dibujara el recorrido seleccionado sobre el mapa. Dentro de esta vista se dispone a su vez la opción de visualizar estadísticas que se verá en el siguiente apartado. El usuario puede moverse a través del mapa y utilizar el zoom en cualquier parte.

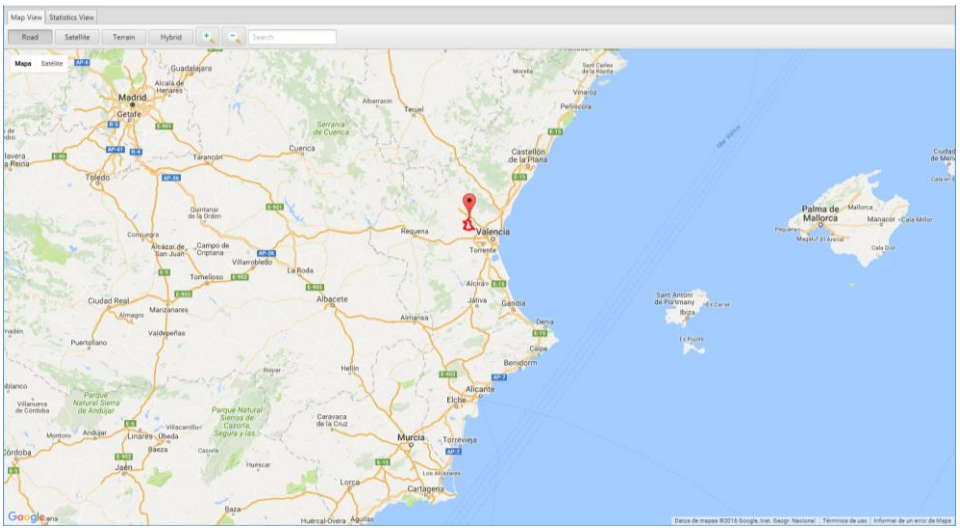

Ilustración 26 - Visualizar recorrido (Zoom)

Además existe la posibilidad de intercambiar el tipo de mapa, para esto el usuario deberá pulsar sobre los diferentes botones que se muestra a continuación.

- ***Road***: Muestra la vista del mapa de carreteras.

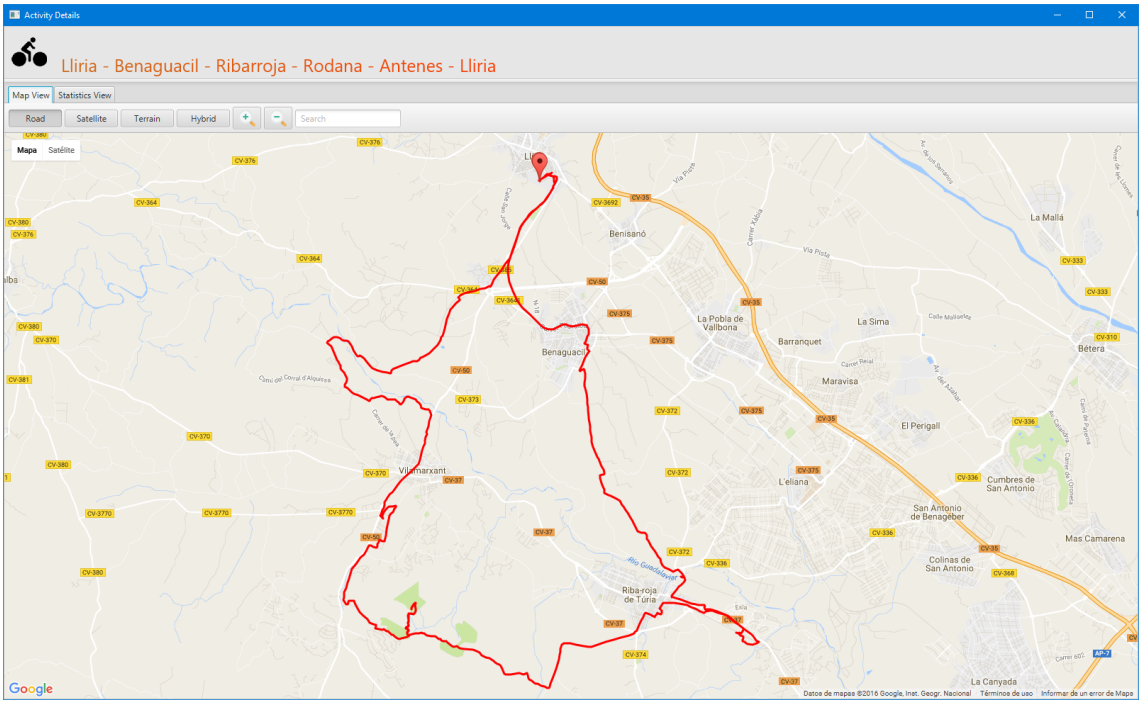

Ilustración 27 - Visualizar recorrido (Road)

- ***Satellite***: Este tipo de mapa muestra imágenes de satélite de Google Earth.

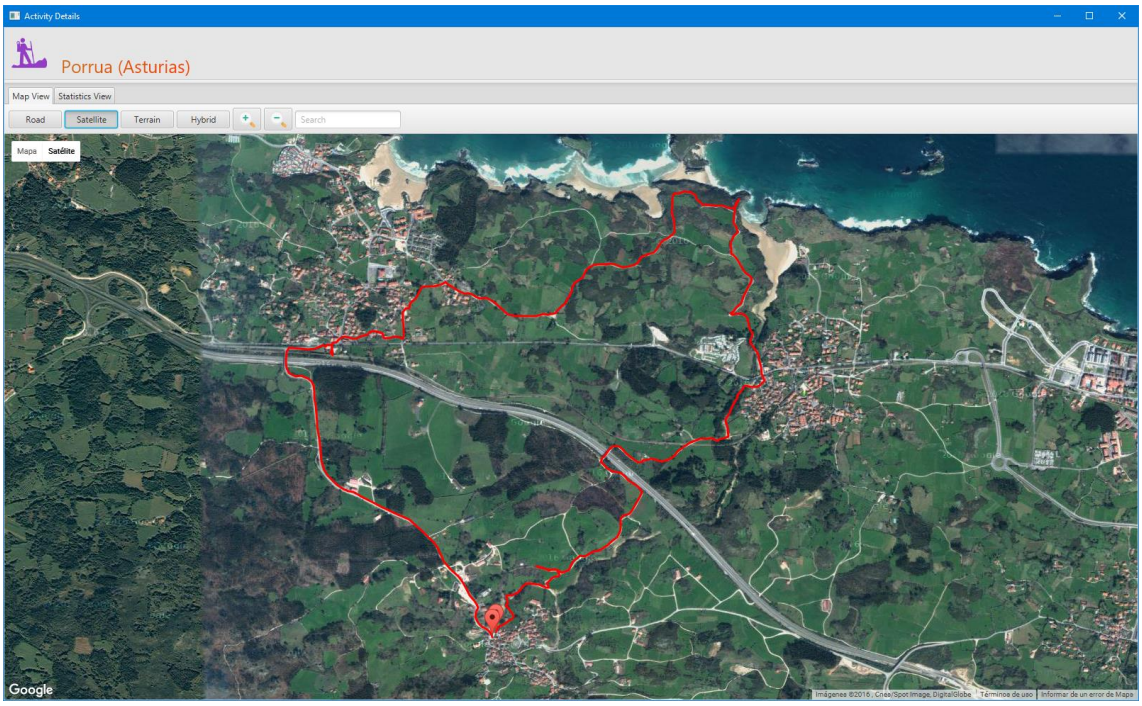

Ilustración 28 - Visualizar recorrido (Satellite)

- ***Terrain***: Es un tipo de mapa físico que muestra información terrestre.

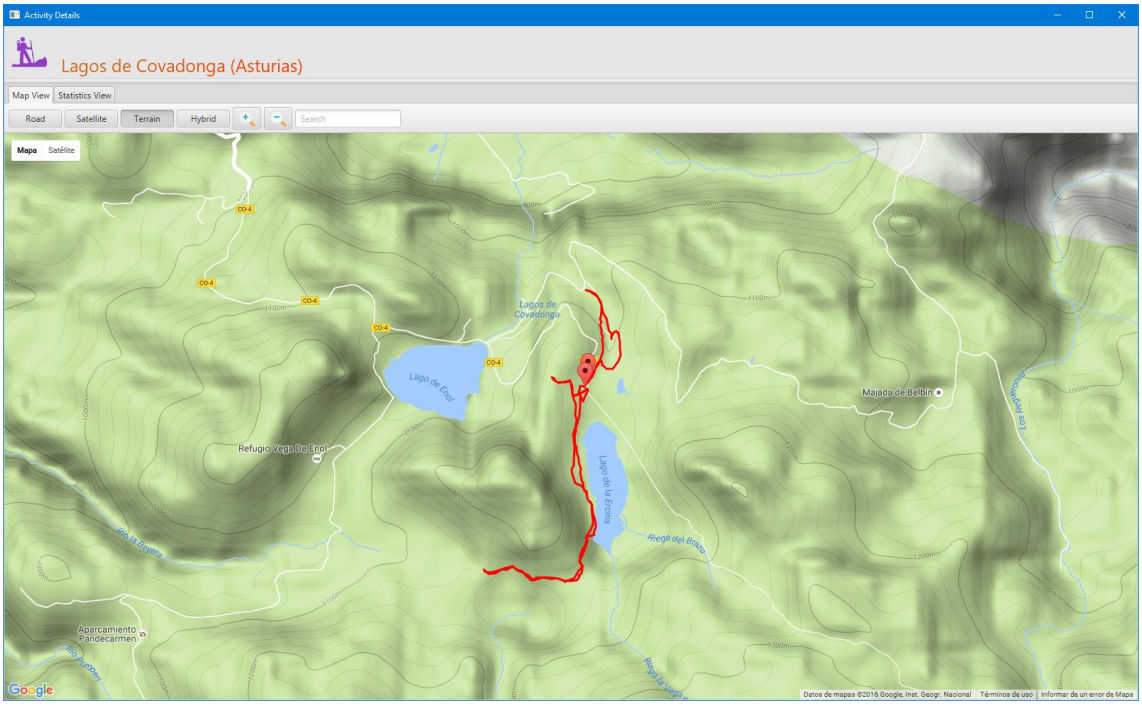

Ilustración 29 - Visualizar recorrido (Terrain)

- ***Hybrid***: El tipo de mapa Hibrido muestra una combinación de vistas normales con las vistas satélite.

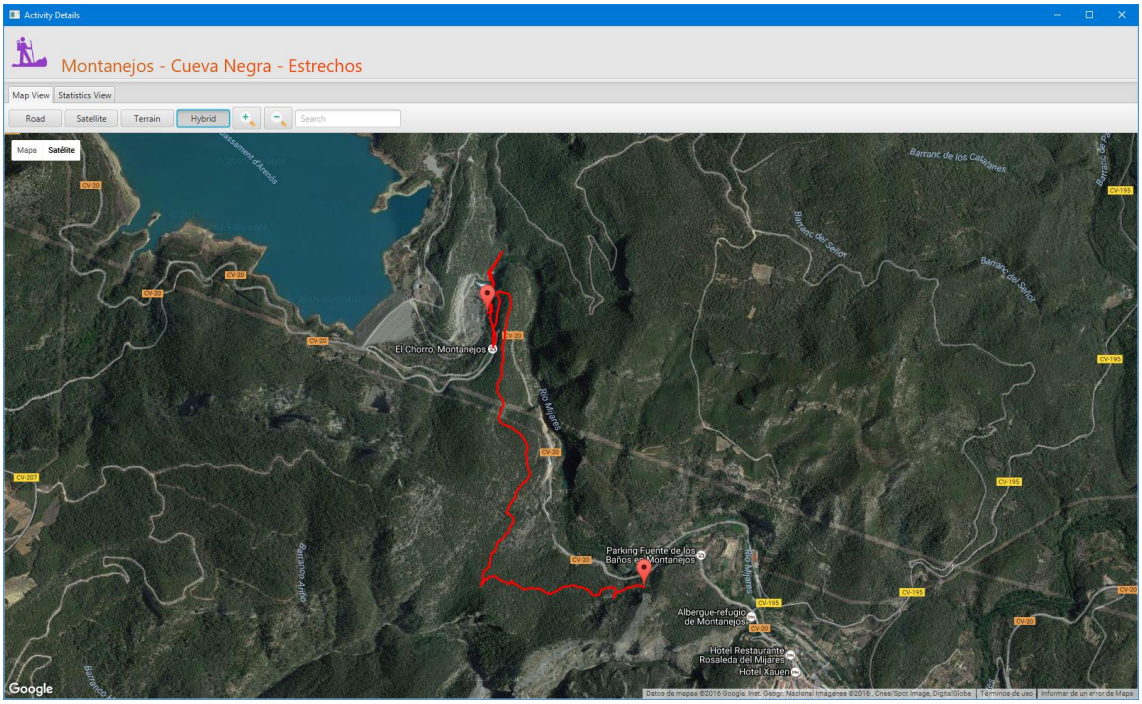

Ilustración 30 - Visualizar recorrido(Hybrid)

8.5. Visualización estadística de recorrido

Como se comentaba en el apartado anterior, desde la ventana de visualización de Google Maps es posible acceder a las estadísticas de la actividad seleccionada. Desde la ventana se dispone en la parte superior de dos pestañas con las que es posible cambiar entre visualización en mapa o estadísticas.

En referencia a las estadísticas encontramos para cada actividad, dos gráficas donde se representan diferentes factores referentes a nuestra actividad.

En primer lugar encontramos una gráfica de líneas que representa mediante la distancia recorrida y los metros de altitud los diferentes desniveles que se han presentado durante el recorrido.

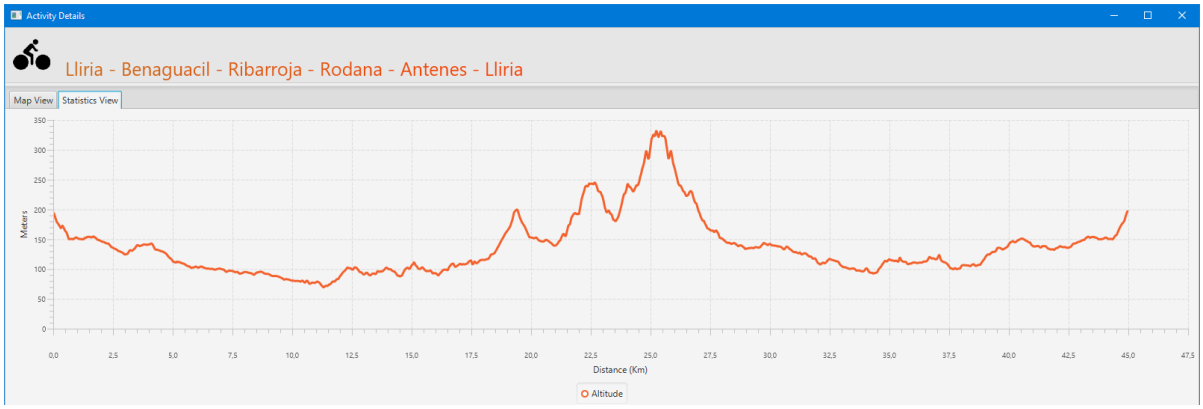

Ilustración 31 - Estadísticas actividad (Atitude)

En segundo lugar se presenta una gráfica que representa la velocidad en funcion de los quilometros recorridos. Pudiendo observar en ella las pequeños cambios de velocidad durante la ejecución de una actividad.

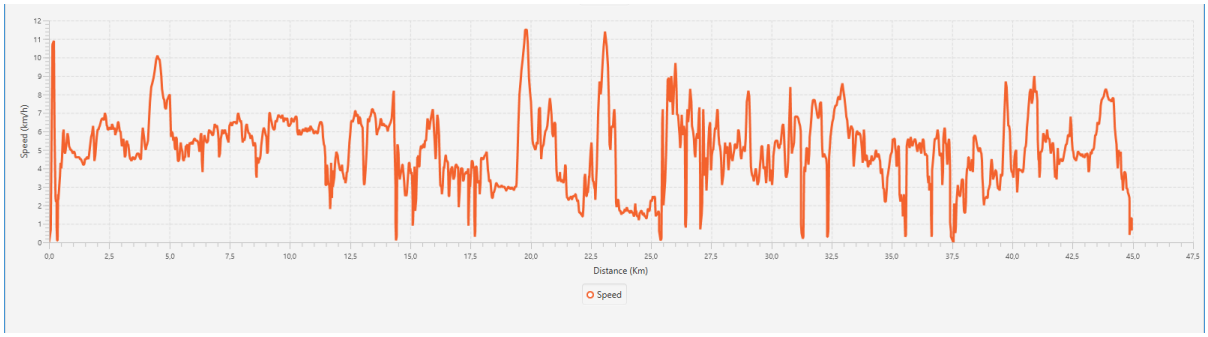

Ilustración 32 - Estadísticas actividad (Speed)

8.6. Descarga actividad en formato GPX

Al pulsar sobre el botón  el sistema permite al usuario final descargar el recorrido correspondiente. Cualquier actividad del sistema dispone de la posibilidad de ser transformado al formato GPX y ser exportado a nuestro ordenador. El sistema muestra un explorador de archivos en el que es posible seleccionar donde queremos guardar el archivo GPX seleccionado.

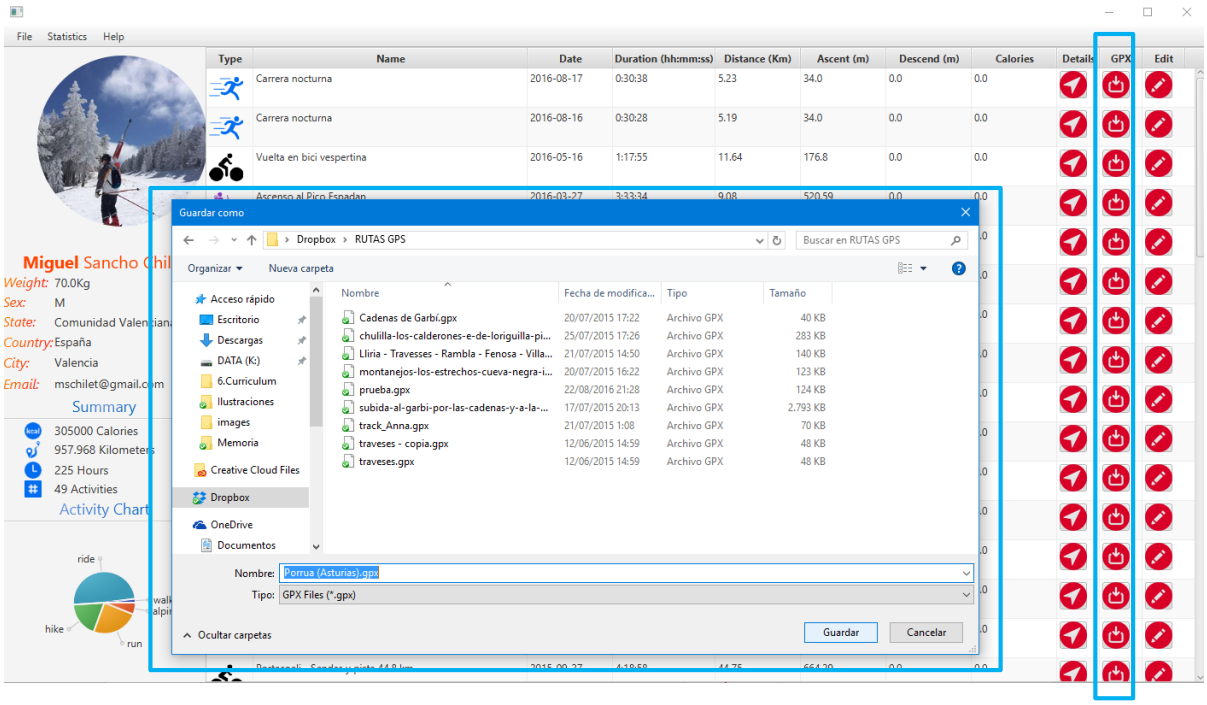

Ilustración 33 - Guardar actividad (GPX)

A continuación editaremos el fichero GPX que se ha guardado para observar cómo ha sido formateado y ver un pequeño fragmento de su contendido.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<gpx  version="1.1">
<trk>
<trkseg>
<trkpt lat="43.412620544433594" lon="-4.801962852478027">
<ele>47.29999923706055</ele>
</trkpt>
<trkpt lat="43.412620544433594" lon="-4.801962852478027">
<ele>47.29999923706055</ele>
</trkpt>
<trkpt lat="43.412803649902344" lon="-4.801835060119629">
<ele>46.79999923706055</ele>
</trkpt>
<trkpt lat="43.412872314453125" lon="-4.801815986633301">
<ele>46.5</ele>
</trkpt>
</trkseg>
</trk>
</gpx>
```

Ilustración 34 - Fragmento GPX

8.7. Edición de actividad

El usuario dispone de la posibilidad de editar cualquier actividad, para ello, debe pulsar sobre el botón  situado en el lateral derecho.

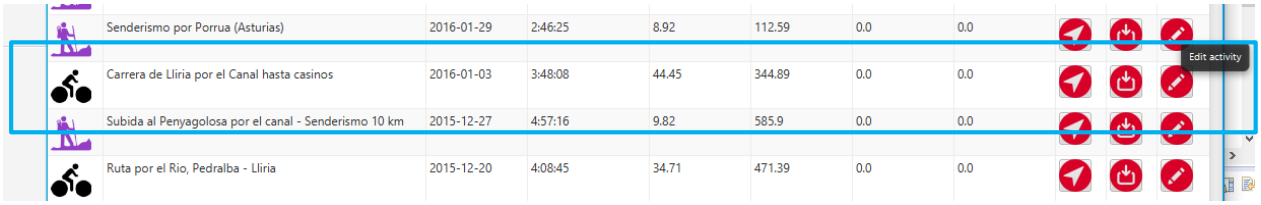

Ilustración 35 - Editar actividad

Al pulsar sobre el botón, se abrirá una nueva ventana en la que el usuario dispone de la posibilidad de modificar el nombre y la descripción de una actividad. Por defecto, la aplicación cargara los datos actuales en los campos correspondientes para así poder ser modificados. Una vez hechas las modificaciones, el usuario debe pulsar sobre el botón *Update* para que los cambios queden guardados. Si el usuario pulsa sobre el botón Cancel los datos no tendrán ningún efecto sobre la ampliación manteniendo los anteriores valores.

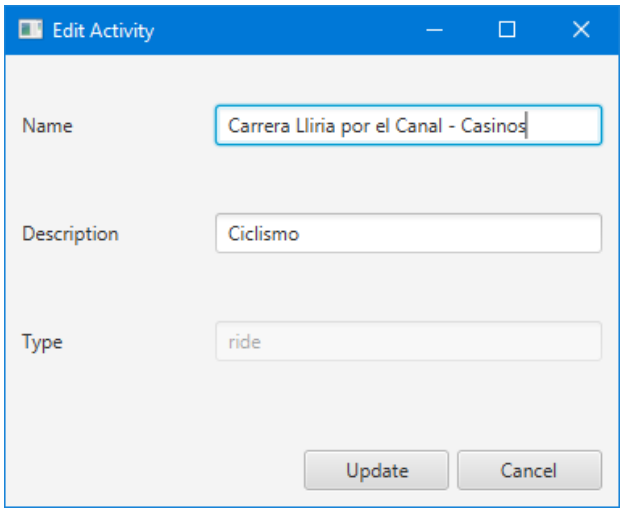

Ilustración 36 - Editar actividad

Si la opción finalmente ha sido la de actualizar los valores, se observara el cambio inmediatamente en la actividad correspondiente. Desde la aplicación únicamente se visualizará el nombre.

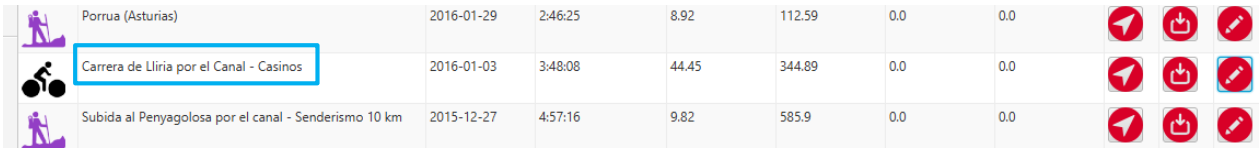

Ilustración 37 - Actividad editada (local)

De igual manera, los cambios serán actualizados inmediatamente en la plataforma web Strava, pudiéndose visualizar tanto el nombre como la descripción que el usuario ha introducido desde la aplicación.

Ilustración 38 - Actividad editada (remoto)

8.8. Visualización de estadísticas mensuales

Desde la ventana principal de la aplicación es posible visualizar un gráfico que representa las actividades realizadas cada mes durante los diferentes años. En la parte inferior del gráfico se puede observar una leyenda que muestra los diferentes años en los que el usuario ha registrado actividades.

Cada una de las barras representan mediante el color el año que se realizaron las actividades, mediante la altitud de la barra se representa el número de actividades, y mediante la posición del eje X se visualiza el mes en el que se realizaron las actividades.

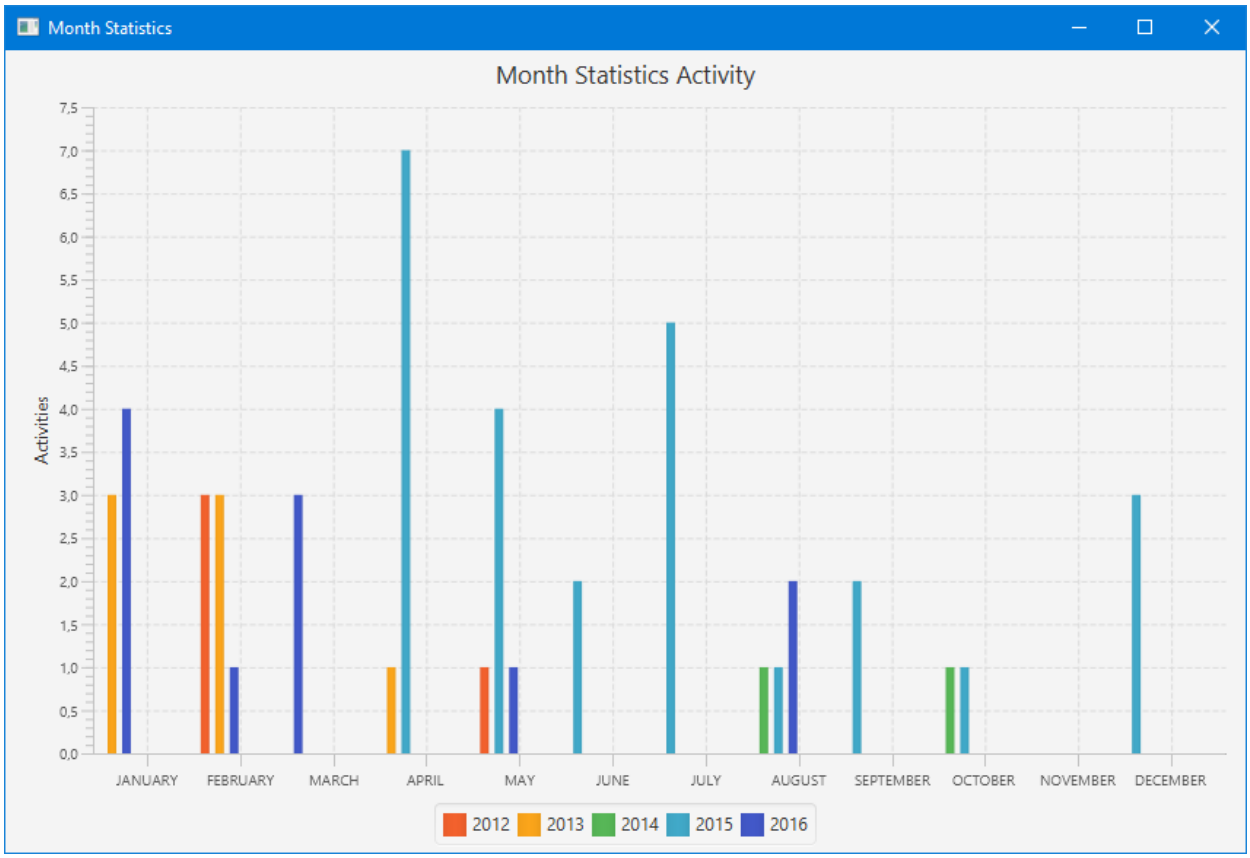

Ilustración 39 - Estadísticas Mensuales

9. Conclusión

El presente proyecto ha tratado sobre el desarrollo de una aplicación de escritorio para la gestión de recorridos grabados mediante dispositivos GPS. Para conseguir el objetivo y la funcionalidad deseada de la aplicación, se ha seguido en todo momento una serie de pautas y se han utilizado una serie de herramientas que nos han facilitado conseguir los objetivos marcados. El uso de un software para el control de versiones así como la estructura con la que fue definida la aplicación desde un primer momento, han sido algunos de los puntos clave que han permitido mejorar el proceso de desarrollo de la aplicación.

Respecto a la estructura, se decidió seguir el patrón MVC, con ello nuestro código ha quedado claramente separado entre diferentes capas de implementación. Ha sido una elección clave a la hora de desarrollar la aplicación, ya que el código queda organizado, y nos ha proporcionado en todo momento una visión más clara, además de proporcionar mayor flexibilidad a la hora de incorporar nuevas funcionalidades.

Aun así, a lo largo de un proceso de desarrollo siempre pueden surgir problemas o impedimentos, en nuestro caso fueron varias las limitaciones que se han ido interponiendo en el camino, por ejemplo, no todas las plataformas nos proporcionaban las mismas funcionalidades ni los mismos datos, por lo que hemos tenido que adaptar la aplicación en función de estas limitaciones.

El proyecto ha supuesto trabajar con diferentes plataformas, las cuales habían de ser integradas en la aplicación para así poder interaccionar con la misma. Como paso previo a la integración, todas las plataformas han sido previamente analizadas para conocer debidamente cada uno de sus aspectos y características. Este punto de análisis ha sido el más importante a la hora de posteriormente realizar una integración con la aplicación, además de que nos ha permitido encontrarnos con los primeros obstáculos, y con la necesidad de realizar las tareas para solventarlos.

Cabe destacar que durante el desarrollo e integración de las distintas plataforma han ido apareciendo nuevos términos, protocolos desconocidos hasta el momento por mí, así como la utilización de nuevos mecanismos de desarrollo, los cuales me han permitido extender los conocimientos adquiridos durante el grado cursado, así como mejorar y afianzar considerablemente cantidad de aspectos vistos durante cada uno de los cursos académicos.

En resumen, y como pequeña reflexión destacar el progreso personal que he podido observar a lo largo del desarrollo del proyecto. El comienzo fue una de las etapas más difíciles del proyecto, nuevas tecnologías, nuevos mecanismos y momentos de mucha documentación donde asimilar nuevos conceptos sin observar grandes cambios en el proyecto, sin duda una de las etapas más frustrantes en cuanto a la velocidad en la que evolucionaba el proyecto. Pero este aspecto, como era de esperar, iba a mejorar con el paso del tiempo, en el cual he ido experimentando más soltura progresivamente y he sido capaz de afrontar y resolver nuevas etapas del proyecto. Finalmente, puntualizar que se ha conseguido alcanzar los conocimientos necesarios de los que se requería para la implementación del proyecto, además de destacar la experiencia que se ha adquirido en la integración y desarrollo de aplicaciones de escritorio.

10. Bibliografía

[1] OAuth - http://oauth.net/2/

[2] Strava API Reference - https://strava.github.io/api/

[3] Code Makery - http://code.makery.ch/library/javafx-8-tutorial/es/part1/

[4] JavaFX API - http://docs.oracle.com/javase/8/javafx/api/toc.htm

[5] Java API - https://docs.oracle.com/javase/7/docs/api/

[6] JavaFX Scene Builder - https://docs.oracle.com/javase/8/scene-builder-2/get-started-tutorial/jfxsb-get_started.htm

[7] Bitbucket - https://bitbucket.org/

[8] Javastravav3api - http://danshannon.github.io/javastravav3api/

[9] Endo2Java API - https://github.com/MoOmEeN/endo2java

[10] Github GMapsFX - https://github.com/rterp/GMapsFX

[11] GeoKaramolaLib API - https://plus.google.com/u/0/communities/110606810455751902142

[12] JPA - http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[13] ObjectDB - http://www.objectdb.com/