Escuela Politécnica Superior de Linares

Grado en Ingeniería Telemática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

SERVICIO DE ALERTAS DE PRODUCTOS EXISTENTES EN TIENDAS ON LINE

Alumno: **María del Carmen Llobregat Jiménez**

Tutor 1: José Enrique Muñoz Expósito
Depto.: Ingeniería de Telecomunicación

Tutor 2: Antonio Troyano Gómez

Junio, 2020

UNIVERSIDAD DE JAÉN

ESCUELA POLITÉCNICA SUPERIOR DE LINARES

Trabajo Fin de Grado

Curso 2019-2020

Servicio de alertas de productos existentes en tiendas on line

Alumna: Llobregat Jiménez, María del Carmen

Tutor 1: Muñoz Expósito, José Enrique

Departamento: Ingeniería de Telecomunicación

Tutor 2: Troyano Gómez, Antonio Manuel

Firma de la autora

LLOBREGAT JIMENEZ MARIA DEL CARMEN - 26506357F

Firmado digitalmente por LLOBREGAT JIMENEZ MARIA DEL CARMEN - 26506357F
Fecha: 2020.06.16 09:14:33 +02'00'

Firma del Tutor 1

MUÑOZ EXPOSITO JOSE ENRIQUE - 27386513E

Firmado digitalmente por MUÑOZ EXPOSITO JOSE ENRIQUE - 27386513E
Nombre de reconocimiento (DN): c=ES, serialNumber=IDCES-27386513E, givenName=JOSE ENRIQUE, sn=MUÑOZ EXPOSITO, cn=MUÑOZ EXPOSITO JOSE ENRIQUE - 27386513E
Fecha: 2020.06.16 09:25:12 +02'00'

Firma del Tutor 2

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

INDICE DE TABLAS

ABREVIATURAS

A	
API	Application Programming Interface
C	
CSS	Cascading Style Sheets
D	
DOM	Document Object Model
F	
FIFO	First In, First Out
G	
GCP	Google Cloud Platform
H	
HTTP	HyperText Transfer Protocol
I	
IaaS	Infrastructure as a Service
ID	Inyección de Dependencias
IoC	Inversión del Control
J	
JPA	Java Persistence API
JSON	JavaScript Object Notation
O	
ORM	Object-Relational mapping
R	
REST	Representational State Transfer
S	
SQL	Structured query language
U	
URI	Uniform Resource Identifier
V	
VM	Virtual Machine
X	
XML	Extensible Markup Language

1. RESUMEN

1.1. RESUMEN

El objetivo de este Trabajo de Fin de Grado (TFG) es diseñar y desarrollar un servicio capaz de rastrear precios de productos seleccionados por los usuarios en páginas de ventas como Amazon o eBay.

Se podrá consultar la evolución de precios a través de históricos y se realizará un intento de predicción y extracción de conclusiones de la evolución de precios.

Los usuarios podrán configurar diferentes tipos de notificaciones por correo electrónico por la variación de los precios.

El servidor se preparará para ser desplegado en la nube. Los usuarios accederán al servicio a través de una aplicación Android que les permitirá utilizar todas las funciones anteriormente mencionadas.

1.2. ABSTRACT

The objective of this Capstone Project is the design and implementation of a service capable of tracking prices of articles selected by users from online shops, as Amazon or eBay.

Prices progress will be available to consult through charts. An attempt will be made to predict and draw conclusions from prices evolution.

Users will be able to configurate different types of email notifications because of price variations.

The server will be prepared to be deployed in the cloud. Users will access the service through an Android application that will allow the access to all the functions mentioned above.

2. INTRODUCCIÓN

En este capítulo se introduce el contexto en el que se desarrolla este trabajo y la motivación de su desarrollo. Además, se plantea la estructura del presente documento y se detalla el contenido de los capítulos del mismo.

2.1. JUSTIFICACIÓN Y CONTEXTO

En los últimos años el comercio electrónico o e-commerce ha aumentado de forma considerable, de forma que, en el tercer trimestre de 2019 en España, la facturación del comercio electrónico rozó los 12.500 millones de euros, un 23,5% más que el año anterior (1).

Sin duda, uno de los motivos principales que lleva a los consumidores a decantarse por las tiendas online en detrimento de las tiendas físicas es el precio. Los productos suelen tener precios menores en las tiendas online ya que se eliminan intermediarios de la cadena de suministro. Además, muchas tiendas online utilizan la fijación dinámica de precios o dynamic pricing, que consiste en ajustar el precio del producto según diferentes factores como la demanda del mismo, la solvencia del cliente, la singularidad del servicio a prestar o la planificación temporal del proveedor. Los usuarios podrían aprovechar esta metodología de las tiendas digitales para obtener los artículos a un menor precio.

Este TFG pretende desarrollar un servicio que permita aprovechar esta característica de las tiendas online para extraer los precios de los productos de forma periódica, guardar un histórico, notificar a los usuarios de variaciones y dar información sobre la tendencia general de dicho producto.

Dicho servicio se ha diseñado mediante una aplicación servidor que se desplegará en un entorno cloud con la funcionalidad principal y una aplicación Android para el acceso del usuario. Estas aplicaciones se comunicarán mediante una API REST, de forma que facilita el futuro desarrollo de aplicaciones para el acceso de usuario desde otras plataformas (iOS, macOS, Windows, web, etc).

Se ha escogido una aplicación Android para el acceso de usuario ya que es el sistema operativo móvil más utilizado actualmente, con un 72.52% del mercado en mayo de 2020 a nivel mundial (2) y un 78.44% en España (3).

Al servicio se le ha dado el nombre de Price Hunt (del inglés: Caza de precios), nombre que simboliza el objetivo principal del mismo. También se ha diseñado un logo y una interfaz de usuario con intención de hacerlo comercial.

2.2. ANTECEDENTES Y ESTADO DEL ARTE

En este TFG nos vamos a centrar en dos de las tiendas digitales más utilizadas, Amazon y eBay, que utilizan la metodología de fijación de precios dinámica introducida en el apartado anterior. Ambas webs proporcionan algunas facilidades al usuario para seguir los productos que les interesen: Amazon permite añadir artículos a una cesta e indica si su precio ha variado al volver a visitarla y eBay nos permite añadir hasta 200 productos a una lista de seguimiento. Ninguna de las webs nos permite ver un histórico de los precios ni configurar notificaciones personalizadas.

Respecto servicios externos a las propias webs, es posible encontrar diferentes herramientas que realicen algunas funcionalidades similares a las propuestas para el servicio de este TFG.

Un ejemplo podría ser la extensión para Google Chrome de espiaprecios.com. Esta extensión te permite visualizar un histórico de los precios de los productos de Amazon dentro de la misma web de Amazon (4). En esta misma línea también se encuentra camelcamelcamel, también es un rastreador de precios de Amazon a través de extensiones para los navegadores (5).

También es posible encontrar aplicaciones móviles que te permitan hacer el seguimiento de productos de Amazon, como Price Tracker para Amazon (6). Esta aplicación permite buscar productos de este sitio web desde dentro de la misma aplicación con un navegador embebido. Al encontrar el artículo deseado se selecciona y se añade a una lista con aquellos que resulten interesantes. Es posible hacer una configuración de notificaciones sencilla y visualizar la página de ofertas desde el navegador de la misma anteriormente mencionado. Un punto flojo de esta aplicación es la interfaz gráfica, la cual es bastante tosca y en ocasiones resulta antiestética.

En contraste con la anterior, Fluctuate (7) nos permite seguir el precio de productos de diferentes sitios web, pero es imprescindible la colaboración activa del usuario final. Al abrir la aplicación detecta si hay algún enlace web en el portapapeles y lo abre en un navegador embebido en la aplicación. Posteriormente, es necesario que el usuario seleccione el título y el precio del producto para que la aplicación sea capaz de detectarlos. Este servicio permite configurar un precio por debajo del cual se nos notificará, pero para acceder a la opción de visualización del histórico de precios es necesario pagar un paquete de 3,09€.

Las dos aplicaciones anteriores están exclusivamente disponibles para Android. Un ejemplo de aplicación disponible para Android y iOS es Idealo (8). Esta aplicación permite comparar precios de un producto entre diferentes páginas web de venta online. Para algunos productos permite ver un histórico con precios anteriores. La aplicación es más

compleja que las anteriores y esto se refleja en su interfaz de usuario, que en ocasiones resulta poco intuitiva.

De los servicios expuestos, las extensiones para los navegadores proporcionan información sobre los productos de forma inmediata y sin necesidad de hacer un seguimiento de los mismos. Por otro lado, las aplicaciones móviles son más cómodas de utilizar y ubicuas, aportando a demás, funcionalidades más completas.

Es muy común que estas aplicaciones permitan configurar notificaciones para avisar a los usuarios de las variaciones de precios de los productos, aunque generalmente hay pocas opciones de configuración. Sin embargo, no es tan común que permitan ver un histórico de los precios ni que te proporcionen información sobre la tendencia del mismo. En algún caso es necesaria la intervención del usuario final para la obtención de información del producto, no siendo un proceso automático. Además, en muchas ocasiones las interfaces gráficas de las aplicaciones son relativamente complejas o demasiado simples y poco refinadas.

En este TFG se va a tratar de desarrollar un servicio lo más completo posible, que no requiera la intervención del usuario para la extracción de la información y con una interfaz de usuario sencilla e intuitiva pero vistosa y agradable. Se va a estructurar de forma que permita ampliar las plataformas desde las que se hace uso del mismo.

2.3. ESTRUCTURA DEL DOCUMENTO

El presente documento está estructurado en los siguientes capítulos:

- **Capítulo 1. Resumen:** Se introducen de forma muy esquemática los objetivos del TFG.
- **Capítulo 2. Introducción:** Se introduce el contexto en el que se desarrolla el trabajo y los motivos de su desarrollo.
- **Capítulo 3. Objetivos:** Recoge los diferentes fines que intentan conseguirse con este TFG.
- **Capítulo 4. Tecnologías, análisis y diseño:** Detalla la información relativa al desarrollo del trabajo, las tecnologías empleadas y el diseño llevado a cabo.
- **Capítulo 5. Resultado y discusión:** Expone los resultados obtenidos con este trabajo y se hace un pequeño análisis de la consecución de objetivos.
- **Capítulo 6. Conclusiones:** Se detallan las conclusiones obtenidas con la realización de este TFG.
- **Capítulo 7. Líneas futuras:** Se enumeran posibles líneas de trabajo más allá de este TFG

- **Capítulo 8. Bibliografía:** Referencias a los artículos, documentos online y páginas webs que se han empleado como consulta durante el desarrollo del presente trabajo.
- **Capítulo 9 Anexos:** En este capítulo se añaden los manuales de instalación de las aplicaciones, los archivos de configuración para Docker e información sobre el protocolo OAuth 2.0 que utiliza Google Sign In.

3. OBJETIVOS

El objetivo principal de este TFG, como ya se ha expuesto anteriormente, es desarrollar un servicio que permita el seguimiento de precios de artículos seleccionados por usuarios. La consecución de dicho objetivo está supeditada a la consecución de los siguientes objetivos secundarios:

- **Implementación de la gestión de usuarios.**

La gestión de usuarios no es un objetivo principal en este TFG. Sin embargo, es necesaria una mínima gestión de los usuarios para guardar cada artículo en relación con el usuario o usuarios que lo estén siguiendo y mostrar a cada usuario sus artículos.

- **Extracción del precio de los productos y demás información de las webs de compras.**

Es necesario extraer la información más relevante de los productos introducidos por los usuarios de las diferentes tiendas webs. Se considera información relevante el título del producto, el precio (del producto y de envío), la categoría del producto, la marca, la calificación de los usuarios, y cualquier información adicional en forma de clave y valor sobre el producto. Esta información será almacenada en la base de datos.

- **Extracción periódica del precio y actualización de la información del producto.**

El servicio requiere que se recopile de forma periódica la información del precio de los diferentes productos que hayan sido introducidos por los usuarios con anterioridad. Asimismo, es necesaria la actualización en la base de datos de cualquier información sobre el producto que haya podido variar.

- **Implementación de la visualización del histórico.**

Los datos sobre los precios de los productos almacenados en la base de datos se mostrarán al usuario de manera que le resulte sencillo visualizarlos y compararlos a lo largo del tiempo.

- **Intento de predicción de precios y extracción de conclusiones.**

Recogidos un número suficiente de datos respecto al precio a lo largo del tiempo de un producto, se intentará realizar una predicción de precios futuros y extracción de conclusiones sobre la tendencia general de dicho producto a lo largo del tiempo de seguimiento.

- **Diseño e implementación de un sistema que notifique a los usuarios la variación de los precios según sus preferencias.**

Tras cada actualización de los datos se notificará a los usuarios si se ha producido variaciones de los precios. Los usuarios tendrán que tener opción de configurar estas notificaciones.

- **Implementación de la infraestructura en un entorno cloud.**

Toda la infraestructura necesaria para la parte servidor se implementará en un entorno en la nube.

4. MATERIALES Y MÉTODOS

En este capítulo se detallan las distintas tecnologías que se han utilizado para la implementación del servicio, detallando en qué consisten y exponiendo su utilidad en el mismo.

También se realiza un análisis del problema a resolver y de cada una de sus partes para, posteriormente, especificar el diseño de las soluciones adoptadas y las conclusiones respecto a las mismas.

4.1. TECNOLOGÍAS UTILIZADAS

En este apartado se enumeran las tecnologías y herramientas utilizadas para la implementación de todo el servicio y su aplicación en el mismo.

4.1.1. FRAMEWORK SPRING Y SPRING BOOT

Spring es un framework para el desarrollo de aplicaciones empresariales complejas en lenguajes de programación basados en Java. Proporciona un modelo de programación y configuración cuyo objetivo es simplificar el desarrollo de aplicaciones para aumentar el rendimiento. Algunas de las características más destacadas de este framework son la Inyección de dependencias (DI) como forma de Inversión del Control (IoC) o la capa de acceso a los datos con gestión de la transaccionalidad (9).

Utilizando como base el framework de Spring, el objetivo de Spring Boot es hacer más sencilla la creación y desarrollo de aplicaciones Spring.

A la hora de realizar cualquier aplicación es necesario realizar una configuración inicial, desarrollar la aplicación y, finalmente, desplegarla en un servidor. Spring Boot facilita enormemente la configuración inicial de las dependencias del proyecto. Además, Spring nos proporciona Spring initializr (10) para facilitar la creación de los proyectos. Asimismo, simplifica la configuración de la aplicación, ya que no requiere la generación de archivos XML, y nos proporciona un servidor embebido para ejecutarla. Esto facilita las dos etapas dedicadas a la infraestructura de la aplicación, dejando más tiempo para el desarrollo de la misma (11).

4.1.2. API REST

Una API REST (Representational State Transfer – Transferencia de Estado Representacional) es una interfaz para la comunicación entre sistemas que fue definida en el año 2000 por Roy T. Fielding, creador de HTTP (12).

REST utiliza el protocolo sin estados HTTP para la comunicación entre cliente y servidor. Los recursos están definidos por una URI (Universal Resource Identifier) y sobre ellos se pueden realizar las siguientes operaciones: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar). Los datos dentro de las peticiones pueden viajar en diferentes formatos (12), seleccionando JSON (Java Script Object Notation) en este caso en concreto por ser el que utiliza por defecto el módulo REST de Spring.

Se ha escogido esta API para la comunicación entre cliente y servidor por ser un paradigma muy extendido y por la facilidad de su implementación gracias a la dependencia de Spring Web, que nos permite desarrollar una aplicación REST a través de anotaciones.

4.1.3. SCRAPING

El scraping web es una técnica muy utilizada a día de hoy para extraer datos de sitios web. Consiste en programar un software que navega automáticamente por la web extrayendo información de ella. El software que realiza esta función suele conocerse como crawler (13).

Esta técnica es utilizada en este proyecto para la extracción de la información de los artículos de las tiendas online. Para programar nuestro crawler se hará uso del navegador PhantomJS y los selectores CSS, tecnologías explicadas en las siguientes secciones.

4.1.4. PHANTOMJS

PhantomJS es un navegador sin interfaz gráfica pensado para la automatización de las interacciones con páginas web a través de una API en JavaScript (14). Gracias al proyecto GhostDriver, disponemos de un driver para poder correrlo en lenguaje Java, llamado PhantomJSDriver (15). También es posible obtener drivers para otros lenguajes como C# o Python del Proyecto Selenium (16).

Se ha escogido este navegador en concreto por ser de los navegadores más utilizados para este tipo de técnicas. Será el encargado de visitar las páginas webs de los productos que han introducido los usuarios para recoger información de los mismos.

4.1.5. JSOUP Y LOS SELECTORES CSS

Jsoup es una librería en Java para trabajar con páginas HTML que nos permite manipular sus elementos y atributos (17). Una vez hemos recuperado una página web con PhantomJS, esta librería nos permite pasarla a un objeto DOM (Document Object Model –

Modelo de objetos del documento) para, a través de selectores CSS (Cascading Style Sheets), buscar la información que nos interesa dentro de la misma.

Los selectores CSS son patrones que se utilizan para seleccionar elementos determinados dentro de una página HTML según su estilo. Las páginas web de un determinado sitio web, como puede ser Amazon o eBay, suelen crearse siguiendo una estructura y estilo parecidos, lo que nos permite encontrar en ella la información que deseamos mediante estos selectores.

4.1.6. MYSQL

Como sistema gestor de base de datos se ha escogido MySQL, un sistema gestor de bases de datos relacional de código abierto basado en SQL (Structured Query Language - lenguaje de consulta estructurado) que ha sido escogido como sistema gestor de bases de datos del año 2019 (18).

Spring Boot, en el momento de inicializar el proyecto, te permite añadir todas las dependencias necesarias para conectar la aplicación con una base de datos MySQL, facilitando enormemente la integración de este sistema con nuestro servidor.

4.1.7. JPA E HIBERNATE

JPA (Java Persistence API) es un framework del lenguaje Java para manejar datos relacionales. Por defecto utiliza Hibernate como herramienta para el ORM (Object-Relational mapping – Mapeo objeto-relacional). Ambos vienen integrados en el módulo de Spring Data (19).

Con estas herramientas, la interacción con la base de datos MySQL es muy sencilla, haciendo innecesario programar el mapeo de los resultados de las sentencias o escribir código SQL, todo se programa en Java y mediante el uso de etiquetas. Únicamente es necesario definir objetos que van a representar nuestras entidades, indicando las características necesarias a través de etiquetas. Para realizar las consultas es suficiente con extender la interfaz del repositorio de JPA. Con este repositorio podemos realizar funciones básicas como buscar por identificador o guardar una entidad en la base de datos. Algunas operaciones algo más elaboradas se pueden construir definiendo métodos con una estructura específica en esa interfaz, que JPA se encarga de traducir en sentencias SQL.

4.1.8. SPRING BATCH Y SPRING SCHEDULER

Spring Batch es un modulo ligero de Spring que nos permite desarrollar aplicaciones que realicen tareas “batch” (20). El procesamiento batch o por lotes consiste en la ejecución de un programa sin la interacción necesaria del usuario para, generalmente, realizar tareas repetitivas sobre una gran cantidad de información.

Spring Scheduler es una herramienta de Spring que nos permite programar una tarea para realizarla de forma periódica a través de etiquetas (21).

Con estas herramientas se diseñará un proceso que actualice la información de los productos que ya existen en base de datos, programándolo para su ejecución periódica.

4.1.9. SPRING EMAIL Y THYMELEAF

Spring Email es un módulo de Spring que nos facilita el envío de correos electrónicos desde nuestra aplicación. Thymeleaf, por otro lado, es una biblioteca en Java que proporciona un motor de plantillas de HTML.

La primera nos permite enviar los correos a los usuarios para informarlos de las variaciones de los precios, sin perder una interfaz amigable y reconocible proporcionada por la segunda.

4.1.10. COMMONS MATH Y LA REGRESIÓN SIMPLE

Commons Math es una librería autocontenida y ligera de componentes matemáticos y estadísticos. Ha sido desarrollada por Apache Commons en lenguaje Java (22).

Utilizando esta librería se va visualizar la tendencia general de los precios de los productos y calcular una estimación de precios futuros mediante un modelo de regresión simple. Este modelo utiliza el método de los mínimos cuadrados con una variable independiente para crear un modelo lineal, que se ajusta a la fórmula Y = m + n*X.

4.1.11. ANDROID STUDIO

Android estudio es el entorno de desarrollo oficial para la plataforma Android. Se ha utilizado este entrono y el lenguaje de programación Java para el desarrollo de la aplicación que va a servir de interfaz entre nuestro servicio y el usuario.

4.1.12. GOOGLE SIGN IN

Google Sign In permite a los usuarios registrarse en aplicaciones de terceros utilizando su cuenta Google. Cada vez más aplicaciones están haciendo uso de esta funcionalidad beneficiando a los usuarios al simplificar la gestión de sus cuentas en las diversas aplicaciones.

En Price Hunt se va a hacer uso de esta herramienta para la autenticación de los usuarios a través de las APIs en Java que nos proporciona Google para Android y servidores de back-end (23).

4.1.13. MPANDROIDCHART

MPAndroidChart es una librería en Java de software libre para Android que nos permite representar potentes gráficos interactivos y fácilmente configurables (24). También tiene una versión disponible para iOS llamada Charts (25).

Se ha utilizado esta librería para representar el histórico y la estimación de los precios de cada producto, en caso de tener datos suficientes. El usuario puede interactuar con el gráfico para visualizar la información de forma sencilla y cómoda.

4.1.14. GLIDE V4

Glide v4 es una librería para Android que, entre otras funcionalidades, nos permite cargar imágenes de forma rápida y eficiente (26).

Se ha escogido esta librería para mejorar el rendimiento de la aplicación a la hora de cargar las imágenes de los productos. Asimismo, resulta muy práctico poder cargar una imagen por defecto si la imagen a cargar no se encuentra disponible.

4.1.15. DOCKER, DOCKER HUB Y DOCKER COMPOSE

Docker es una plataforma que nos permite crear imágenes de nuestras aplicaciones para, posteriormente, desplegarlas como contenedores en otros sistemas que también corran Docker, sin importar el sistema operativo anfitrión. De esta forma, la imagen de nuestra aplicación es autocontenida y portable. El concepto es similar al de las máquinas virtuales pero los contenedores son muchos más ligeros (27).

Docker Hub es un repositorio en la nube de imágenes Docker, del que podemos descargar y utilizar imágenes. Además, creando una cuenta, podemos subir imágenes propias en repositorios públicos y privados. Docker Hub es una forma muy rápida y sencilla de compartir imágenes (28).

Docker Compose es una herramienta que nos permite definir y desplegar aplicaciones multi-contenedor en Docker (29).

Se han utilizado estas tecnologías en el presente proyecto para preparar la aplicación servidor que posteriormente es desplegada en la nube.

4.1.16. GOOGLE COMPUTE ENGINE

Google Compute Engine es el módulo de infraestructura como servicio (IaaS) de la nube de Google o Google Cloud Platform (GCP). Te permite crear máquinas virtuales utilizando la infraestructura de la nube de Google.

Se ha escogido esta tecnología para el despliegue en la nube de la aplicación ya que la nube de Google está entre las tecnologías cloud más utilizadas junto con Amazon Web Services y Microsoft Azure. Aunque todas estas plataformas son de pago, Google proporciona un saldo de 300$ gratuitos para aprendizaje y realización de pruebas.

4.2. ANÁLISIS Y DISEÑO

En este apartado se realiza un análisis del servicio a desarrollar. En primer lugar, se identifica a los individuos que van a interactuar con el servicio y se definen los requisitos del mismo. A continuación, se realiza un análisis de la estructura del servicio y se expone detalladamente el diseño e implementación de las funcionalidades del mismo.

4.2.1. IDENTIFICACIÓN DE LOS USUARIOS

Podemos definir dos tipos de usuarios que van a interactuar con el servicio desarrollado:

- El usuario al que va destinado el servicio, el que va a hacer un uso del mismo.
- Un usuario administrador que se encargará de mantener y actualizar el servicio. Dicho administrador, como se explicará en el apartado 5.2, es de gran importancia para el correcto funcionamiento del servicio a largo plazo.

4.2.2. DEFINICIÓN DE LOS REQUISITOS

Para el correcto funcionamiento y desarrollo del servicio se pueden definir dos tipos de requisitos:

4.2.2.1. Requisitos funcionales

Los propios asociados a los objetivos esperados por el servicio.

- Gestión de usuarios. Aunque necesaria deberá de ser la mínima posible, con objetivo de simplificar una parte no intrínseca a nuestro servicio.
- Extracción de información de productos introducidos por usuarios de determinadas tiendas online.
- Actualización periódica de dicha información.
- Visualización de histórico de precios.
- Extracción de predicciones y conclusiones sobre los productos.
- Implementación de un sistema de notificaciones a los usuarios.
- Implementación del servicio en un entorno en la nube.

4.2.2.2. Requisitos no funcionales

Atributos de calidad o características generales del servicio.

- Interfaz de usuario atractiva, funcional y amigable.
- Utilización de software libre o de pruebas gratuitas para la minimización del coste del TFG.
- Utilización de normas y estándares en la medida de lo posible para facilitar la interoperabilidad del servicio y la comprensión del mismo.

4.2.3. ESTRUCTURA GENERAL DEL SERVICIO

El servicio desarrollado consta de dos partes principales que se muestran en el siguiente diagrama:

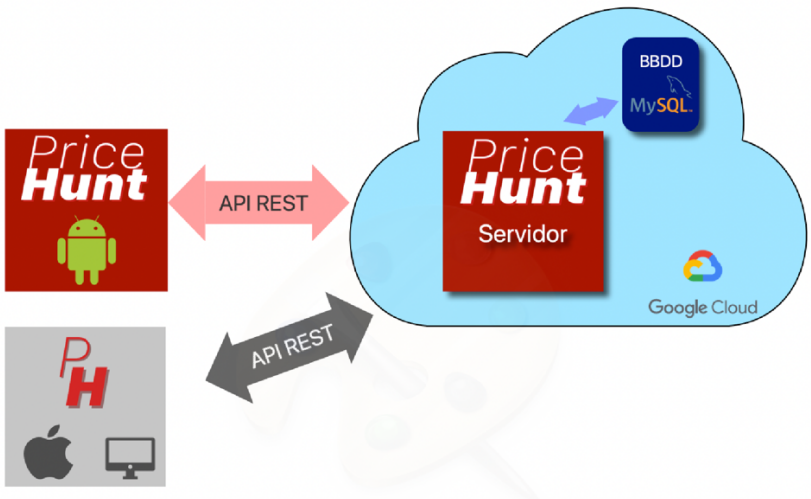

Figura 1. Estructura general del servicio

- **Back-end o aplicación servidor**: Es la aplicación que lleva a cabo la funcionalidad principal del servicio: se encarga de extraer la información de los productos, realizar

las actualizaciones periódicas y las predicciones y enviar los correos electrónicos a los usuarios. Una vez completado el desarrollo de la misma se desplegará en la nube.

- **Front-end o aplicación cliente**: Proporciona la interfaz con la que va a interactuar el usuario. Se ha implementado para tal fin una aplicación Android, la cual se ha denominado igual que el servicio, Price Hunt.

La comunicación entre ambos componentes de la aplicación se establece mediante una API REST, haciendo uso del protocolo HTTP. La interfaz de esta comunicación se detallará más adelante, en el apartado 4.2.8.

La intención tras esta estructura es la de separar front-end y back-end con una comunicación bien establecida de forma que posteriormente sea sencillo la implementación de aplicaciones front-end para otras plataformas, como iOS o web.

4.2.4. ESTRUCTURA DE LA APLICACIÓN SERVIDOR

La aplicación servidor se ha desarrollado con el framework Spring-Boot, utilizando Maven como software gestor de proyectos y Java como lenguaje de programación.

Dicha aplicación tiene funcionalidades diferenciadas, pero con bastante relación entre ellas, por lo que se ha implementado como una aplicación multi-módulo. En el siguiente diagrama se muestran los módulos que tiene la aplicación y las dependencias que hay entre los mismos.

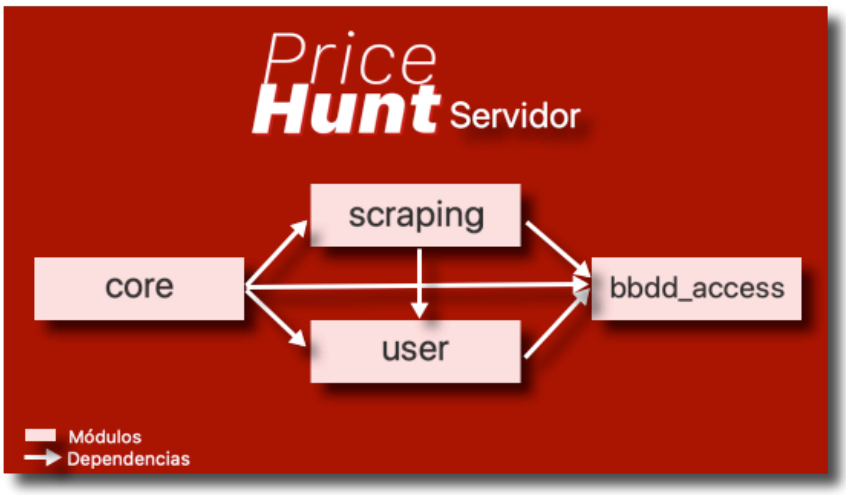

Figura 2. Estructura general de la aplicación servidor

4.2.4.1. Core

Se trata del módulo principal de la aplicación y el punto de entrada a la misma. Tiene la estructura que se muestra a continuación.

Figura 3. Estructura de paquetes del módulo core

4.2.4.1.1. Epsl.telematica.tfg.core

En el paquete **epsl.telemtica.tfg.core** tenemos la clase de inicio de la aplicación. En esta clase realizamos alguna configuración importante de la aplicación mediante etiquetas:

```
@SpringBootApplication
@ComponentScan(basePackages = {
        "epsl.telematica.tfg.scraping",
        "epsl.telematica.tfg.core",
        "epsl.telematica.tfg.bbdd_access",
        "epsl.telematica.tfg.user"
        })
@EnableJpaRepositories(basePackages ={
        "epsl.telematica.tfg.bbdd.product.repositories",
        "epsl.telematica.tfg.bbdd.user.repositories"
        })
@EntityScan(basePackages = {
        "epsl.telematica.tfg.bbdd.product.model",
        "epsl.telematica.tfg.bbdd.user.model"
        })
@EnableAsync
public class CoreApplication {
```

Figura 4. Configuración de la aplicación mediante etiquetas

- **@SpringBootApplication** realiza la función conjunta de otras tres etiquetas que le indica a Spring Boot que puede autoconfigurar la aplicación (@EnableAutoConfiguration), que debe activar el escaneo de componentes para la inyección de dependencias en el paquete de la aplicación (@ComponentScan) y permite el uso de clases de configuración adicionales (@Configuration) (30).
- **@ComponentScan** sobrescribe la configuración anterior en este aspecto indicándole a Spring Boot que debe buscar componentes en los paquetes de todos los módulos.
- **@EnableJpaRepositories** permite el uso de repositorios JPA y le indica a Spring Boot los paquetes donde debe buscarlos.
- **@EntityScan** indica a Spring Boot los paquetes donde se encuentran las entidades.

4.2.4.1.2. Epsl.telematica.tfg.core.controller

Este paquete contiene las clases que forman los controladores de la API REST de la aplicación. Los métodos se han dividido en tres clases para clasificarlos según uso. Se detalla más información sobre los endpoints de la API en la sección 4.2.8 DISEÑO DE LA API DE COMUNICACIÓN.

- **NotificationController**: contiene un método que establece el endpoint para la modificación de la configuración de las notificaciones por parte del usuario.
- **ProductController**: contiene todos los métodos que establecen endpoints relacionados con el control de los productos: añadir un producto nuevo, eliminar un producto, obtener una lista de productos de un usuario, obtener toda la información de un producto o enviar un reporte de error sobre un producto.
- **UserController**: contiene todos los métodos que establecen endpoints relacionados con el control de los usuarios: inicio de sesión y borrar la cuenta.

4.2.4.1.3. Epsl.telematica.tfg.core.controller.management

Este paquete contiene la clase **ScrapingController**, un controlador de gestión con un método que permite al administrador activar el proceso batch de actualización de los productos. De esta forma no es necesario esperar a la hora programada para que se realice la actualización.

4.2.4.1.4. Epsl.telematica.tfg.core.controller.exception_handler

La mayoría de errores que se producen en el servidor se han controlado con excepciones propias. Cuando estas se producen y llegan hasta los controladores, las excepciones son controladas por la clase **GlobalErrorExceptionHandler** de forma centralizada. Esto nos permite evitar que se duplique código que controle estas excepciones en cada uno de los controladores o incluso en cada uno de los métodos.

Esta clase se configura con las etiquetas @ControllerAdvice, que indica que la clase es un interceptor de excepciones, y @ExceptionHandler, que nos permite indicar qué excepciones intercepta cada método de los programados.

Estos métodos devuelven una respuesta HTTP adecuada a la excepción producida. Además, para una gestión de las mismas mientras se desarrolla, en el cuerpo de la respuesta se incluye un objeto **ApiError** que indica detalles de la excepción, como la fecha completa o el mensaje del error.

Para ver las posibles respuestas a cada una de las peticiones ir a la sección 4.2.8 DISEÑO DE LA API DE COMUNICACIÓN.

4.2.4.2. User

Este módulo lleva a cabo toda la funcionalidad relacionada con la gestión de usuarios y el envío de notificaciones. A demás de la estructura de paquetes que se muestra a continuación, en el directorio src/main/resources de este módulo se encuentran la plantilla y los recursos necesarios para los correos de las notificaciones.

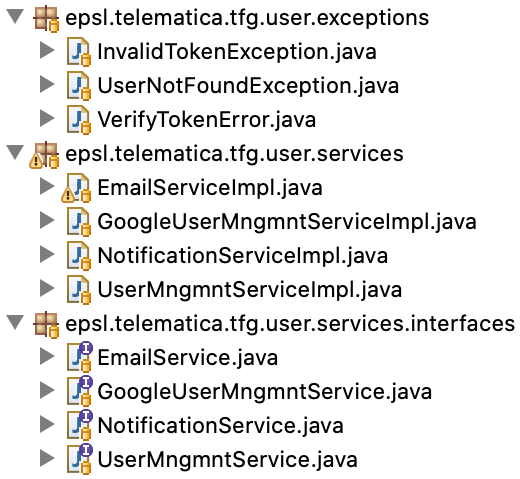

Figura 4.X Estructura de paquetes del módulo user

4.2.4.2.1. Epsl.telematica.tfg.user.exceptions

Las excepciones propias de este modulo están relacionadas con la gestión de usuarios que se detalla más adelante en el apartado 4.2.9 IMPLEMENTACION DE LA AUTENTICACIÓN DE USUARIOS.

- **InvalidTokenException**: Es lanzada cuando el token de autenticación del usuario no es válido. Esto se debe generalmente a que el token ha caducado.
- **UserNotFoundException**: Esta excepción es lanzada cuando hay algún problema encontrando a un usuario, bien porque no se encuentre en nuestra base de datos o porque no coincida con el usuario que nos indica Google al verificar el token.
- **VerifyTokenException**: Se lanza al producirse un error en la verificación del token.

4.2.4.2.2. Epsl.telematica.tfg.user.services y epsl.telematica.tfg.user.services.interfaces

En estos paquetes encontramos las interfaces y las implementaciones de los servicios que están relacionados con la gestión de usuarios y las notificaciones.

Ambas funcionalidades se explican más detalladamente en los apartados 4.2.9 IMPLEMENTACION DE LA AUTENTICACIÓN DE USUARIOS y 4.2.12 DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE NOTIFICACIÓN A USUARIOS, respectivamente.

- **UserMngmntService(Impl):** Es el servicio que se encarga de la creación y eliminación de las cuentas de usuario en el sistema. Hace uso del siguiente servicio para la autenticación de los usuarios.

- **GoogleUserMngmntService(Impl):** Este servicio hace uso de la API en java de Google Sign In para servidores. Con esta API se verifica el token Id que recibimos en el servidor con cada petición del front-end y se obtiene los datos del usuario que la ha realizado (31).
- **NotificationService(Impl):** Este servicio se encarga de forma asíncrona de comprobar, dado un producto, si es necesario notificar a un usuario de la variación de precio. Este servicio se utiliza cuando se actualiza la información de los productos y hace uso del siguiente servicio para enviar las notificaciones.
- **EmailService(Impl):** Este servicio es el encargado de enviar los correos a los usuarios. Utiliza una plantilla HTML para darle estilo a los correos.

4.2.4.3. Scraping

Este módulo se encarga de la gestión de los productos, de realizar la extracción de información de los artículos de las webs y sus actualizaciones. Las funcionalidades relacionadas con este módulo se explican con mayor detalle en las secciones 4.2.10 DISEÑO E IMPLEMENTACIÓN DE LA EXTRACCIÓN DE LA INFORMACIÓN DE LOS PRODUCTOS, 4.2.11DISEÑO E IMPLEMENTACIÓN DE LA ACTUALIZACIÓN PERIÓDICA DE LA INFORMACIÓN y 4.2.13 DISEÑO E IMPLEMENTACIÓN DE PREDICCIÓN DE PRECIOS Y EXTRACCIÓN DE CONCLUSIONES.

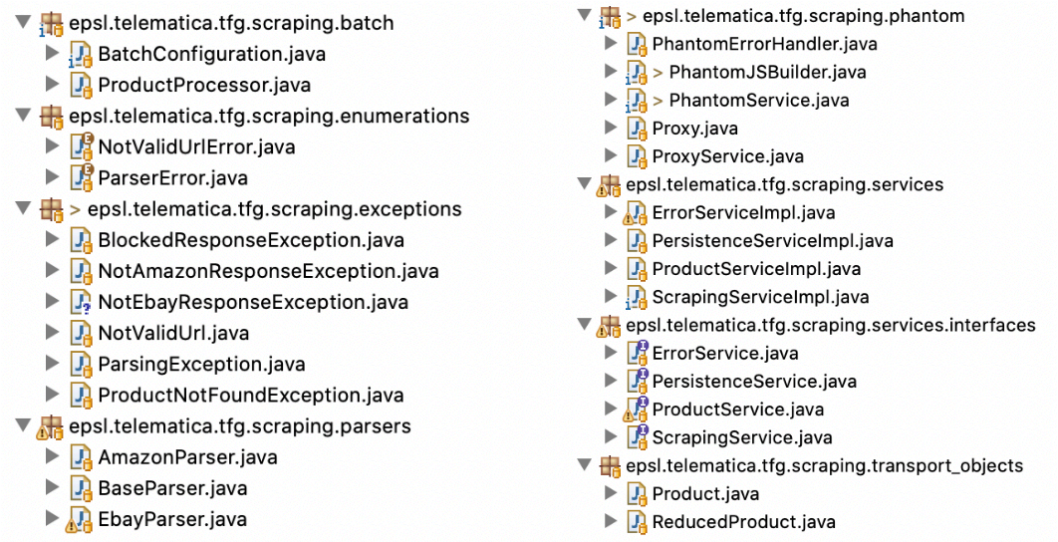

Figura 5. Estructura de paquetes del módulo scraping

4.2.4.3.1. Epsl.telematica.tfg.scraping.batch

Este paquete contiene las clases encargadas de configurar la tarea batch que actualiza la información de los productos y extrae precios actualizados.

Esta funcionalidad se explica en más detalle en el apartado 4.2.11 DISEÑO E IMPLEMENTACIÓN DE LA ACTUALIZACIÓN PERIÓDICA DE LA INFORMACIÓN.

- **BatchConfiguration**: Contiene toda la configuración del trabajo batch e inicia el proceso asíncrono de envío de notificaciones. Dicho trabajo está programado todos los días mediante la etiqueta @Scheduled y una expresión cron. Para que esto funcione es necesario que esta clase esté marcada con las etiquetas @Configuration, @EnableBatchProcessing y @EnableScheduling.
- **ProductProcessor**: Implementa el procesado que se hace en cada paso del trabajo, es decir, la extracción de información y actualización de cada producto.

4.2.4.3.2. Epsl.telematica.tfg.scraping.enumerations

Este paquete contiene dos enumeraciones que exponen tipos de errores que pueden producirse asociados a una excepción.

- **NotValidUrlError** indica errores de la excepción NotValidUrl, como son que la URL no esté bien formada, que no corresponda con un sito válido o que no tenga la parte del path.
- **ParserError:** indica errores de la excepción ParsingException, como son título o precio no encontrado, múltiples precios encontrados o moneda diferente al euro.

Cuando estas excepciones se producen, se indica el motivo de entre los que encontramos en estas enumeraciones. Cuando el interceptor de excepciones del módulo core detecta alguna de estas excepciones, devuelve en la respuesta un mensaje explicativo de la causa del error.

4.2.4.3.3. Epsl.telematica.tfg.scraping.exceptions

Las excepciones incluidas en este módulo están relacionadas con la recuperación de las webs de los artículos o el parseo de las mismas.

- **BlockedResponseException**: Esta excepción se lanza cuando la respuesta que devuelve phantomjs es una página que rechaza el scraping. Este problema y su solución se detallan más adelante, en el apartado 4.2.10.1 Bloqueo de PhantomJS por Amazon.
- **NotAmazon/EbayResponseException**: Excepción lanzada cuando la página que devuelve phantomjs no es una página de Amazon o eBay como se esperaba. Esto puede darse con el uso del servicio de proxies, ya que algunos de ellos no funcionan correctamente y devuelven otras páginas propias. Este servicio se explica en más detalle en el apartado 4.2.10.2 ProxyService.

- **NotValidUrl**: Se lanza cuando la URL introducida por el usuario no es válida por diversos motivos que se especifican en la enumeración correspondiente, NotValidUrlError.
- **ParsingException**: Se lanza cuando hay un problema con el parseo que impide formar un producto íntegro. Hay propiedades del producto que son secundarias, como las imágenes o la información adicional, y que por tanto no son esenciales. Sin embargo, un producto sin título o precio no es válido para la aplicación. El error específico se detalla de entre los de la enumeración correspondiente, ParserError.
- **ProductNotFoundException**: Excepción lanzada cuando no se encuentra en la base de datos un artículo requerido por el usuario.

4.2.4.3.4. Epsl.telematica.tfg.scraping.parsers

Este paquete contiene las clases parseadoras. El parsing o parseo consiste en transformar la página HTML en objetos java. **BaseParser** es una clase con métodos genéricos que ayudan a extraer información de las páginas. **AmazonParser** y **EbayParser** son clases que heredan de BaseParser y que realizan el parseo específico de páginas de Amazon y eBay respectivamente. El resultado del proceso en ambos casos es un objeto Java de la clase epsl.telematica.tfg.bbdd.product.model.Product.

4.2.4.3.5. Epsl.telematica.tfg.scraping.phantom

PhantomJS es la herramienta utilizada como navegador sin interfaz para el scraping de las páginas webs. Este paquete contiene las clases relacionadas con esta herramienta.

- **PhantomErrorHandler**: Intercepta errores de PhantomJS para lanzar una excepción del tipo TimeoutException en el caso pertinente. Esta excepción nos va a indicar generalmente que el proxy utilizado no es válido, ya que tarda demasiado en responder.
- **PhantomJSBuilder**: Clase configuración del driver de phantomjs.
- **PhantomService**: Clase que implementa el servicio de recuperación de páginas web.
- **Proxy:** Clase que va a representar a los proxies del servicio de proxies. Tiene parámetros que nos permite discernir los proxies activos (enabled) y los válidos (valid).
- **ProxyService**: Servicio que nos permite utilizar proxies desde una lista en la configuración de PhantomJS.

4.2.4.3.6. Epsl.telematica.tfg.scraping.services y epsl.telematica.tfg.scrpaing.services.interfaces

En estos paquetes encontramos las interfaces y las implementaciones de los servicios encargados de la gestión de los productos, de obtener la información de los artículos y de llevar un registro de los errores ocurridos para que el gestor del sistema pueda mejorarlo.

Estas funcionalidades se explican con más detalle en los apartados 4.2.10 DISEÑO E IMPLEMENTACIÓN DE LA EXTRACCIÓN DE LA INFORMACIÓN DE LOS PRODUCTOS y 4.2.13 DISEÑO E IMPLEMENTACIÓN DE PREDICCIÓN DE PRECIOS Y EXTRACCIÓN DE CONCLUSIONES .

- **ErrorService(Impl):** Servicio que guarda en un fichero de texto los errores ocurridos durante el parseo y los errores reportados por los usuarios. Este fichero se detalla en el apartado4.2.4.5 Ficheros externos.
- **PersistenceService(Impl):** Servicio con las funcionalidades para guardar u obtener productos de la base de datos.
- **ProductService(Impl):** Servicio que se hace cargo de toda la gestión de los productos, haciendo uso entre otros de los servicios ErrorService y ScrapingService. También es el servicio que realiza la predicción de los precios.
- **ScrapingService(Impl):** Servicio que se encarga del scraping de las páginas webs, haciendo uso del servicio de PhantomJS para obtener las páginas y de los parseadores para extraer la información.

4.2.4.3.7. Epsl.telematica.tfg.scraping.transport_objects

La información que es necesario comunicar a la aplicación Android sobre un recurso puede no ser exactamente la misma que mantenemos en la base de datos sobre el mismo. Las clases de este paquete representan algunos de esos recursos, pero con las características que sean necesarias para la comunicación.

- **Product**: trasporta la información de un producto y la configuración de notificaciones que tiene sobre el mismo el usuario que realiza la petición.
- **ReducedProduct**: incluye únicamente la información del producto que se muestra en la lista de productos.

4.2.4.4. Bbdd_access

Este módulo contiene entidades, repositorios y algunos elementos auxiliares, es decir, todas las clases relacionadas con el acceso a base de datos.

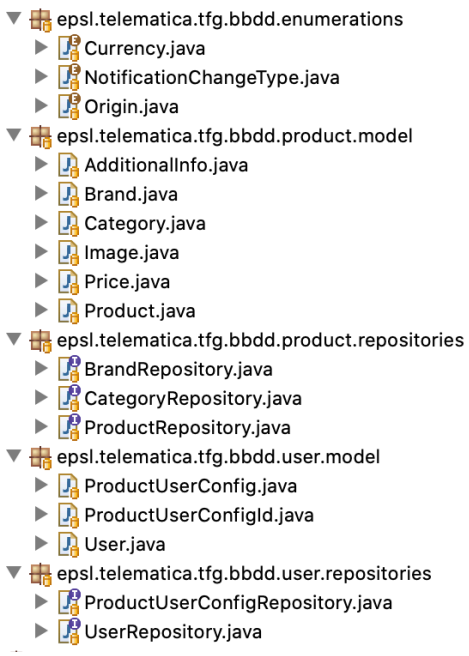

Figura 6. Estructura de paquetes del módulo bbdd_access

4.2.4.4.1. Epsl.telematica.tfg.bbdd.enumerations

El paquete contiene enumeraciones que representan los valores que pueden tomar determinados atributos de objetos.

- **Currency:** representa el tipo de divisa siguiendo la normativa ISO 4217, que utiliza tres letras para representar todas las divisas (32). Actualmente el servicio solo se ha desarrollado para el euro, por lo que esta enumeración únicamente tiene un valor.
- **NotificationChangeType:** contiene los tipos de configuraciones existentes para el cambio de valor de los artículos. Es posible elegir que se nos notifiquen todos los cambios, ninguno, todas las bajadas de precio o las bajadas de precio que no superen un valor concreto.
- **Origin:** indica el origen de un producto, refiriéndose al sitio web del que procede. Actualmente el servicio está desarrollado para acoger artículos de Amazon y eBay.

4.2.4.4.2. Epsl.telematica.tfg.bbdd.product.model y epsl.telematica.tfg.user.model

Recogen todas las entidades relacionadas con los productos de la aplicación y los usuarios respectivamente. Las entidades son clases java que representan a tablas de la base de datos y sus relaciones mediante el uso de anotaciones.

Se detallará más sobre la estructura de la base de datos en el apartado 4.2.7 ESTRUCTURA DE LA BASE DE DATOS.

Cabe destacar las clases **ProductUserConfig** y **ProductUserConfigId**. La primera recoge los parámetros de la configuración de notificaciones de un usuario para un producto

determinado. La segunda representa la clave primaria compuesta de la primera. La configuración depende del usuario y del producto, parámetros que forman la clase ProductUserConfigId, marcada como @Embeddable. En ProductUserConfig el id es un id "incrustado" formado por un objeto de la case ProductUserConfigId. Este id no está señalado con la etiqueta @Id sino con @EmbeddedId.

4.2.4.4.3. Epsl.telematica.tfg.bbdd.product.repositories y epsl.telematica.tfg.bbdd.user.repositories

Estos paquetes contienen los repositorios de los objetos que es necesario obtener de la base de datos de forma individual en algún momento. Los repositorios están clasificados en dos paquetes dependiendo de si están relacionados con los productos o con los usuarios.

Los repositorios son interfaces que extienden la interfaz JpaRepository y están marcados con la anotación @Repository. En ellos se definen métodos para realizar queries a la base de datos. JPA nos permite definir con un lenguaje muy similar al humano muchos tipos de query diferentes que permiten realizar muchas operaciones básicas, como la que se muestra en la siguiente figura. A demás, hay operaciones básicas como findById que no es necesario definirlas ya que vienen predefinidas en la interfaz JpaRepository.

```
@Repository
public interface BrandRepository extends JpaRepository<Brand, Long>{
    public Brand findByTitleKey(String titleKey);
}
```

Figura 7. Ejemplo de repositorio, BrandRepository

4.2.4.5. Ficheros externos

Al desplegar la aplicación servidor y al hacer uso de ella se generan una serie de ficheros que son de utilidad para algunas funciones del servicio o para la gestión del mismo.

Estos ficheros durante el desarrollo se encuentran dentro del directorio del módulo core. Sin embargo, una vez construido el fichero JAR y desplegada la aplicación, los ficheros se encuentran en el mismo directorio que el archivo ejecutable.

- **proxyList.txt**: Este es el fichero que debe de rellenarse con las direcciones de los proxies si se quiere implementar el servicio que los usa. Se introducen con el formato "<dirección ip>:<puerto>", uno por línea. Esta funcionalidad se detalla más profundamente en el apartado 4.2.10.2 ProxyService. Por el momento esta función está desactivada.
- **Phantomjsdriver.log**: Fichero log del navegador Phantomjs.
- **Pricehunt.log**: Fichero log de la aplicación servidor.
- **Error_AAAA_MM_DD.txt**: Fichero generado por el servicio ErrorService(Impl) que contiene información sobre errores de parseo y reportes de error de los usuarios. Si es necesario introducir información, se

genera un archivo por día indicando en el nombre la fecha a la que pertenece. El servicio se detalla más en profundidad en el apartado 4.2.10.3 ErrorService.

4.2.5. ESTRUCTURA DE LA APLICACIÓN ANDROID

La aplicación Android se ha estructurado con la intención de facilitar la comprensión de la aplicación. A continuación, se expone dicha estructura y se muestra un diagrama del flujo entre las actividades que componen la aplicación.

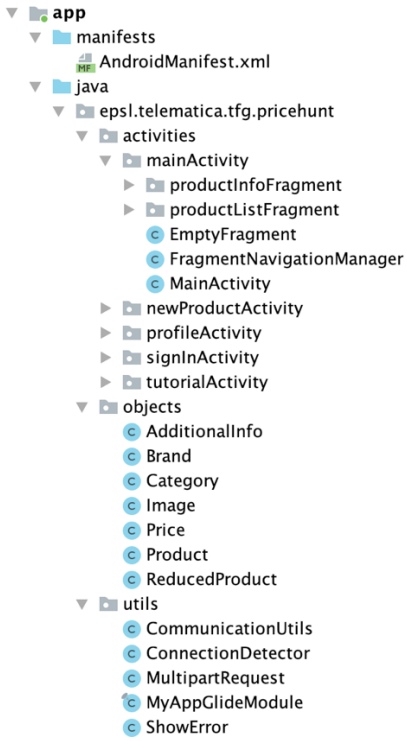

Figura 8. Estructura de paquetes de la aplicación Android

- **Activities**: Contiene los diferentes paquetes de las actividades que forman la aplicación. Estos subpaquetes a su vez están formados por los paquetes y clases de los fragmentos y componentes necesarios para el correcto funcionamiento de las actividades. Todas las actividades se comunican con el servidor de back-end haciendo uso de tareas asíncronas, ya que las funciones de red no deben realizarse en el hilo principal de la aplicación. A continuación, se muestra un diagrama con las distintas actividades que forman la aplicación y el flujo de transiciones entre ellas.

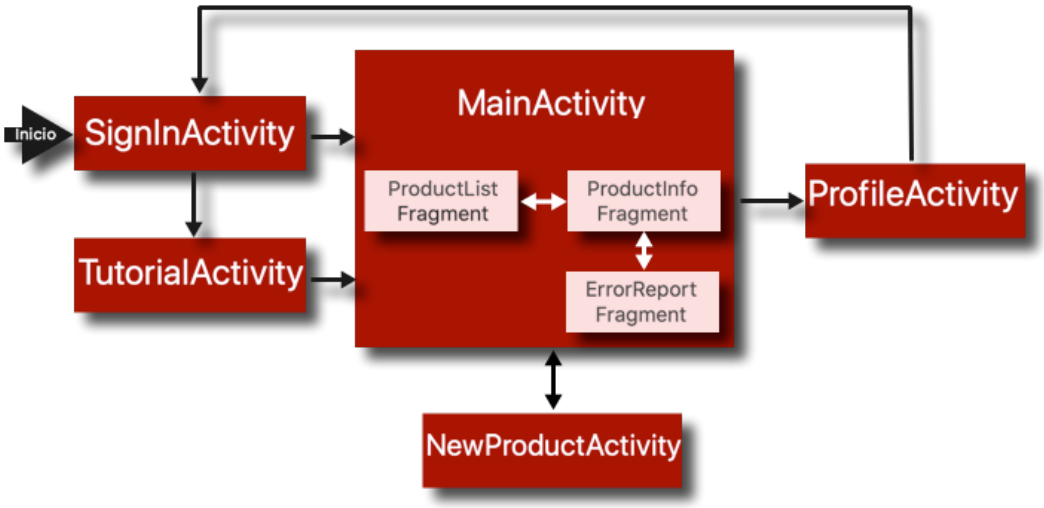

Figura 9. Diagrama de flujo de actividades de la aplicación Android

- **SignInActivity:** se encarga del inicio de sesión.
- **TutorialActivity:** es una actividad que muestra un tutorial introductorio para que el usuario conozca el funcionamiento de la aplicación. Se accede a esta actividad únicamente cuando el inicio de sesión se realiza con una cuenta que no está registrada en nuestra base de datos.
- **MainActivity:** es la clase principal de la aplicación. Desde ella se puede acceder a la actividad que nos muestra nuestro perfil y a la actividad para introducir un nuevo artículo. Esta actividad puede mostrar varios fragmentos: uno vacío si el usuario no tiene artículos asociados, una lista con la información básica de los artículos asociados al usuario, un fragmento con información más detallada sobre un producto o un fragmento para reportar un error en la información que se nos muestra del producto.
- **NewProductActivity:** es la actividad que nos permite añadir productos nuevos pegando del portapapeles o introduciendo su URL.
- **ProfileActivity:** nos permite ver la información de nuestro perfil, cerrar sesión o borrar nuestra cuenta de Price Hunt.

- **Objects**: Este paquete está formado por las clases java que representan los objetos intercambiados con el servidor: los productos y los elementos derivados de este.
- **Utils**: Aquí se congregan las clases que suponen utilidades o herramientas utilizadas en la aplicación.
  - **CommunicationUtils**: Centraliza los parámetros necesarios para la comunicación con el servidor: dirección y puerto del servidor, endpoints, nombres de los parámetros y los diferentes códigos de respuesta. Esto facilita cualquier modificación que haya que hacer respecto a las comunicaciones.

- **ConnectionDetector**: Esta clase se encarga de detectar si el dispositivo tiene conexión con el servidor, informándonos en caso contrario de cuál es el problema: si el dispositivo no está conectado a la red, si no está conectado a internet o si no tiene acceso al servidor.
- **MultipartRequest**: Clase que representa una petición multiparte y que facilita la construcción de las diferentes peticiones al servidor.
- **MyAppGlideModule**: Una clase necesaria para la configuración de la herramienta Glide, introducida en el apartado 4.1.14.
- **ShowError**: Clase que centraliza el mostrar, en forma de Toast o mensaje emergente, todos los diferentes errores que se pueden producir en toda la aplicación.

4.2.6. DISEÑO DE LA INTERFAZ GRÁFICA DE LA APLICACIÓN

La interfaz de usuario se ha diseñado con la intención de que fuera sencilla, fácil de usar y atrayente. Se han implementado, en la medida de lo posible, elementos visuales o gestos que sigan la tendencia de las aplicaciones actuales, de forma que la interacción con la aplicación sea bastante intuitiva. Además, Se ha diseñado un logo en dos colores y un icono para el servicio que son utilizados tanto en la aplicación Android como en los correos.

Figura 10. Logo de Price Hunt en Rojo

Figura 11. Logo de Price Hunt en Blanco

Figura 12. Icono de Price Hunt

Todas las cadenas de texto de la aplicación se encuentran extraídas en un fichero **strings.xml** para facilitar una hipotética traducción de la aplicación. Sin embargo, la aplicación solo se ha desarrollado en español ya que solamente se ha programado el scraping para webs de España. Misma razón por la que sólo se ha considerado el euro como divisa, a pesar de pensar en futuras ampliaciones.

4.2.6.1. Inicio de sesión

Al abrir la aplicación lo primero que se nos muestra es la pantalla de bienvenida o Splash de la Figura 13. Esta pantalla dura unos instantes y muestra el logo de la aplicación. Si tenemos alguna cuenta abierta en la aplicación el inicio de sesión se realiza automáticamente durante los instantes en los que vemos el Splash, en caso contrario se nos muestra la pantalla de inicio de sesión de la Figura 14.

Figura 13. Pantalla de Splash de la app

Figura 14. Pantalla de inicio de sesión

La pantalla de inicio de sesión se encuentra formada únicamente por el logo y el botón de Google Sign In sobre una imagen de unas tiendas de fondo. Al presionar este botón, el usuario podrá iniciar sesión con su cuenta de Google siguiendo un flujo de pantallas que no dependen de nuestra aplicación, como el que se muestra en la siguiente figura.

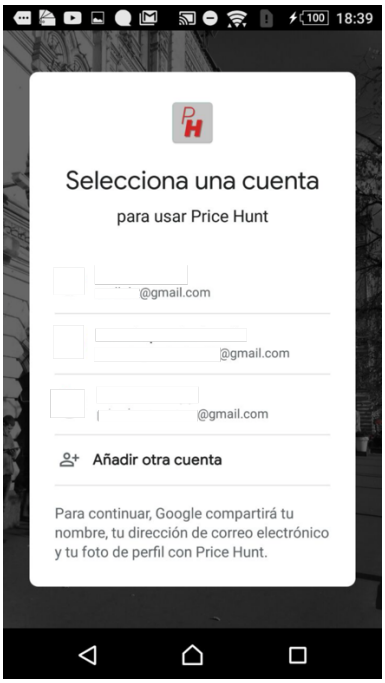

Figura 15. Pantalla del flujo de inicio de sesión con Google

4.2.6.2. Tutorial

Al iniciar sesión por primera vez en el sistema se muestra un pequeño tutorial de la aplicación. En él, el usuario puede desplazarse por varias pantallas que le indican las funcionalidades de la aplicación, como se muestra en las siguientes figuras.

Figura 16. Pantalla del tutorial

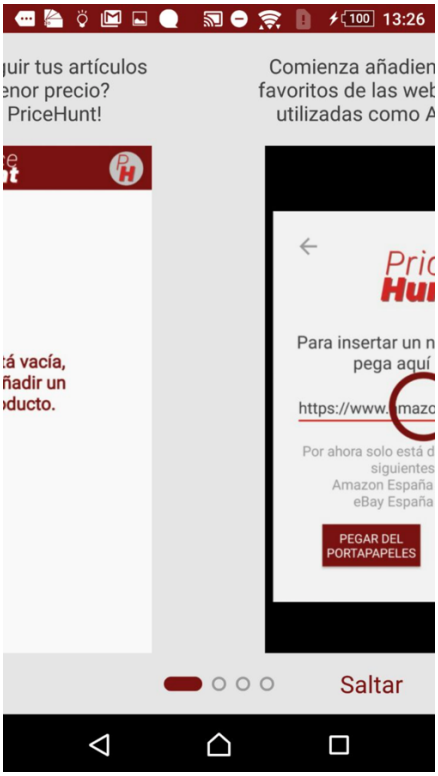

Figura 17. Paso de pantalla del tutorial

Estas pantallas se han diseñado utilizando **ViewPager2** (33), que permite implementar las páginas que se desplazan. El indicador de paso de página de la parte inferior es el objeto **WormDotsIndicator** de la librería de software libre "Material View Pager Dots Indicator" (34).

4.2.6.3. Añadir nuevo producto

Una vez se ha iniciado sesión, se entra en la actividad principal de la aplicación. Si es la primera vez que el usuario entra en el sistema, y por tanto no tiene artículos asociados a su cuenta, la aplicación se muestra como en la Figura 18. Para añadir un producto nuevo, se debe pulsar sobre el icono de la esquina superior izquierda y se conducirá al usuario a la actividad NewProductActivity, cuyo layout es el que se muestra en la Figura 19.

Figura 18. Pantalla principal sin artículos

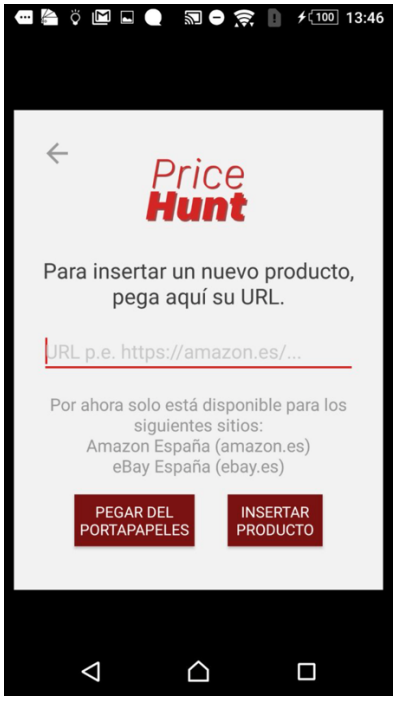

Figura 19. Pantalla para añadir un producto nuevo

Para añadir un producto nuevo es necesario introducir la dirección URL del mismo en el espacio disponible y pulsar en “Insertar producto”. Si tenemos la dirección copiada en el portapapeles podemos introducirla pulsando en “Pegar del portapapeles”.

Una vez introducida la dirección, mientras se realiza el scraping del producto por parte del servidor, aparece una barra de progreso circular (35) para indicar que se está procesando. Al acabar, si todo ha ido correctamente, se redirige automáticamente a la actividad principal para mostrarnos el artículo nuevo en la lista de artículos.

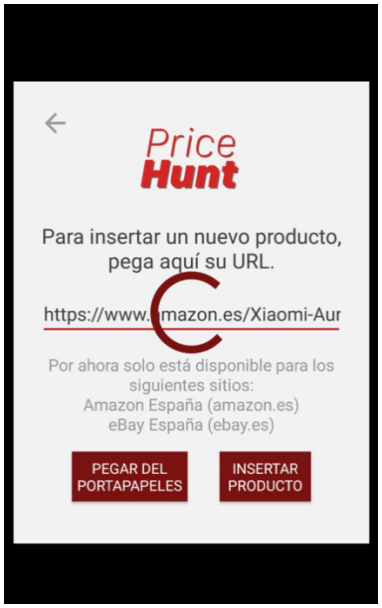

Figura 20. Barra de progreso circular

4.2.6.4. Consulta de información de productos y configuración de notificaciones

En la actividad principal, si el usuario tiene artículos asociados a su cuenta, aparece un listado de los mismos y la información más relevante, como se ve en la siguiente figura.

La lista de productos se ha implementado mediante un **RecyclerView**. Con este elemento se programa cómo debe visualizarse y actuar cada ítem de la lista o **ViewHolder**. Finalmente, la información de los elementos se introduce en un adaptador y es pasada al RecyclerView. Esta vista va creando los elementos conforme los va necesitando para visualizarlos en pantalla y recicla los que ya no son necesarios para crear los nuevos a medida que se hace scroll hacia abajo. El adaptador puede modificarse y, al notificarlo, se modifica la vista en consecuencia (36).

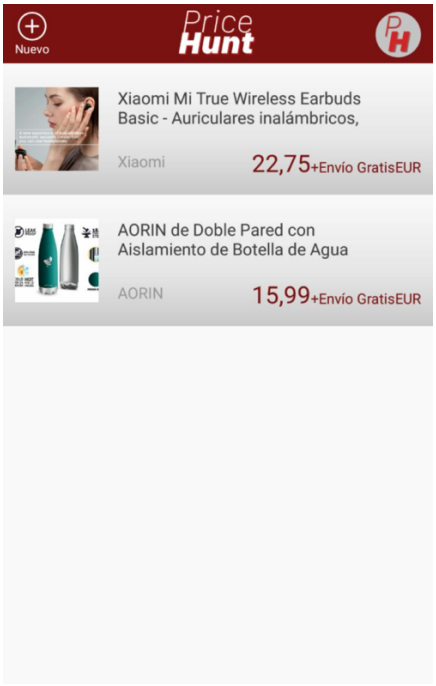

Figura 21. Pantalla principal de la aplicación con listado de productos

Es posible eliminar productos que ya no sean de interés arrastrando la fila del producto hacia la izquierda. Para ello se han definido dos capas en la vista del ítem, la superior con la información y la inferior con el icono de eliminar. Finalmente, utilizando un **ItemTouchHelper** se puede definir la dirección del desplazamiento y la funcionalidad a implementar una vez se realice la acción (37). Al completar el deslizado se ha implementado el SnackBar de la Figura 23, que da la opción al usuario de deshacer el borrado.

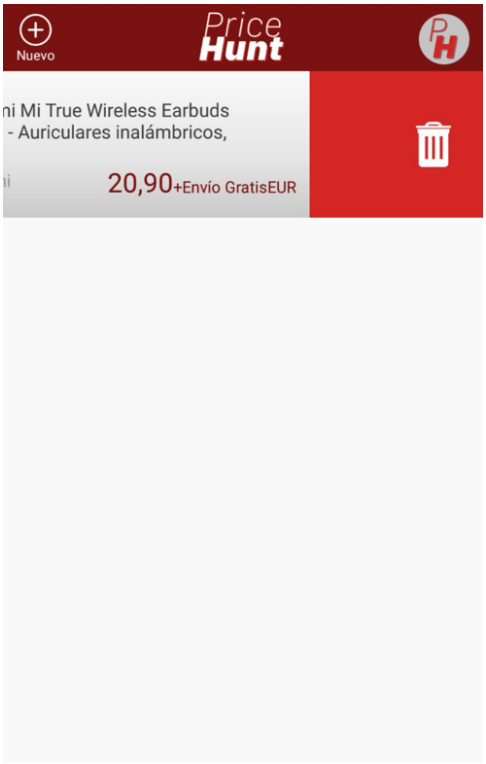

Figura 22. Pantalla principal al eliminar un producto

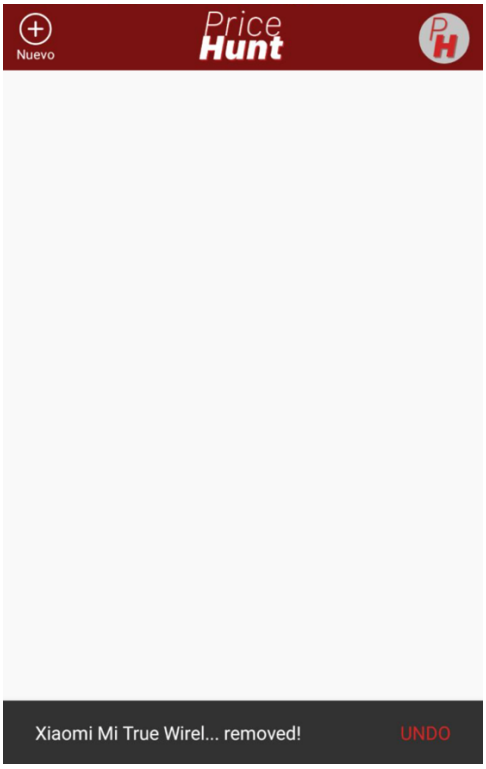

Figura 23. Snack Bar para deshacer el borrado de un producto

Podemos consultar más información de algún producto si pulsamos encima de él. El fragmento de la lista es sustituido por otro que nos muestra más información sobre el artículo seleccionado, pudiendo así ver las imágenes, la puntuación de los usuarios, información adicional sobre el mismo, etc.

Este fragmento de la información se ha implementado con una **NestedScrollView** que permite realizar scroll sobre la pantalla y visualizar toda la información del producto.

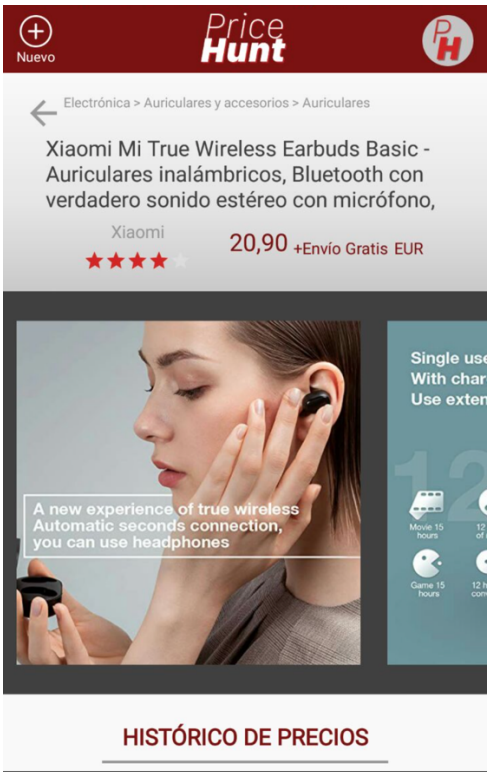

Figura 24. Pantalla de información de producto (cabecera)

En este fragmento encontramos diversas partes con diferente información sobre el producto. En la cabecera, podemos encontrar el título, la marca, las categorías, la puntuación de los usuarios y el precio actual del producto.

Tras la **cabecera**, tenemos una pequeña galería con las imágenes del producto que se encontraban en la web. Nos podemos desplazar haciendo scroll lateral para ver las imágenes. Esta pequeña galería se carga de forma dinámica gracias a la herramienta Glide v4 introducida en la sección 4.1.14. Si alguna de las imágenes no estuviera disponible o hubiera algún error al acceder a ella, Glide introduciría la siguiente imagen en su lugar.

Figura 25. Imagen de sustitución

A continuación, si hay más de un precio para el artículo, podemos encontrarnos un gráfico con el **histórico de precios** implementado con la librería MPAndroidChart introducida en el apartado 4.1.13.

Figura 26. Pantalla de información de producto (histórico)

El gráfico está configurado para visualizar los precios de una forma cómoda permitiendo al usuario interactuar con el mismo para ampliarlo o desplazarse por él. El gráfico se ajustará para que la visualización de los datos seleccionados sea óptima de forma automática.

En este gráfico, si el producto dispone de más de diez precios en la base de datos, se podrá visualizar una **estimación de la evolución** del mismo. Esta funcionalidad se explicará en profundidad más adelante en el apartado 4.2.13 DISEÑO E IMPLEMENTACIÓN DE PREDICCIÓN DE PRECIOS Y EXTRACCIÓN DE CONCLUSIONES.

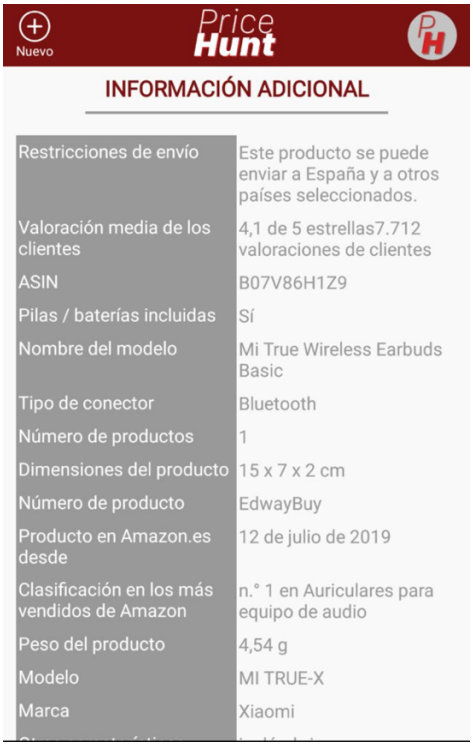

Figura 27. Pantalla de información de producto (Información adicional)

A demás de lo ya mencionado, en el fragmento podemos visualizar una tabla con la **información adicional** sobre el mismo, modificar la **configuración de las notificaciones** asociadas a este producto en concreto, redirigirnos a la tienda para comprarlo o reportar un error información mostrada.

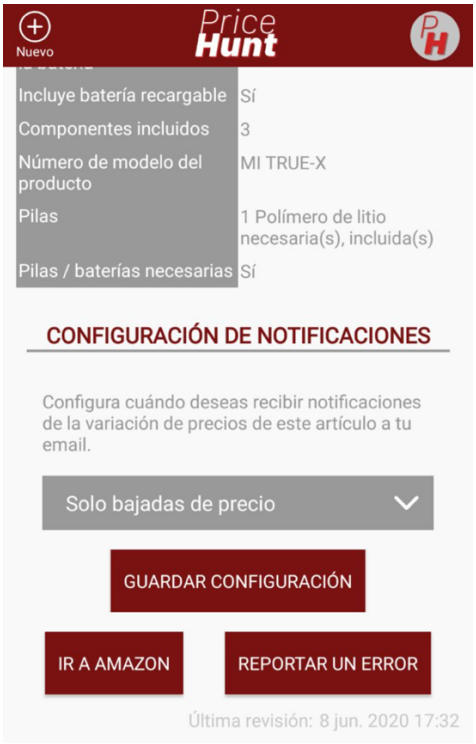

Figura 28. Pantalla de información de producto (Configuración de notificaciones)

Por defecto, todos los productos se crean con la misma configuración, la cual hace que se notifiquen todas las disminuciones de precio. También se puede configurar que no se notifique ninguna, que se notifiquen todas las variaciones o que se notifiquen todas las variaciones por debajo de un valor que introduzca el usuario.

Si la información mostrada es errónea, el usuario puede **reportar el error** pulsando en el botón inferior. La pantalla que se muestra a continuación permite aportar información adicional que ayudará en la corrección del error por parte del gestor del servicio.

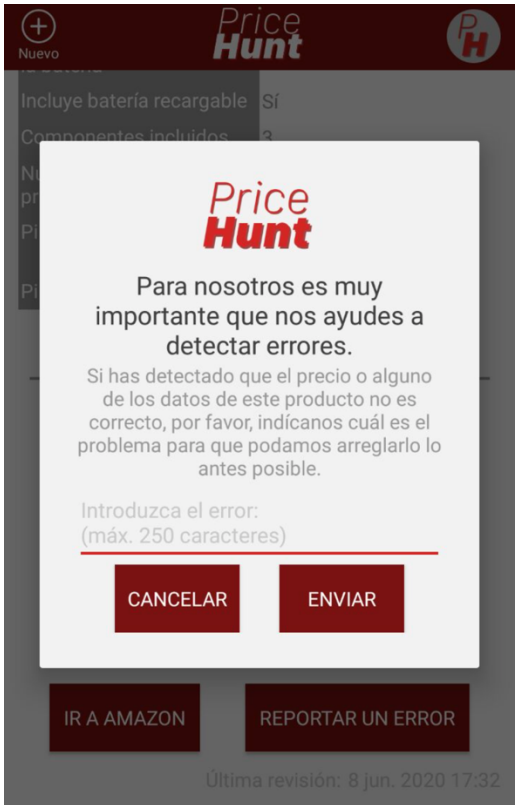

Figura 29. Pantalla de reporte de errores

4.2.6.5. Visualización de la cuenta de usuario y gestión de la misma

En la actividad principal se encuentra en todo momento la imagen de perfil de la cuenta activa en la esquina superior derecha. Al pulsar sobre la imagen se muestra la información de perfil de la cuenta, como se puede ver en la siguiente figura.

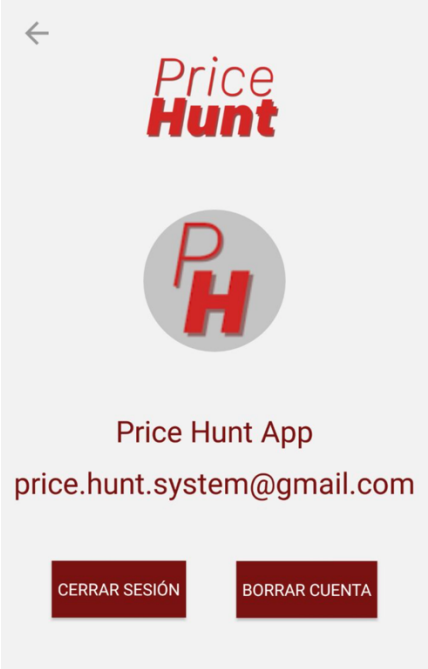

Figura 30. Pantalla de la información de perfil

La imagen de perfil también se carga gracias a la librería Glide v4. Si esta imagen no estuviera disponible o hubiera un error al cargarla, Glide mostraría la siguiente imagen en su lugar.

Figura 31. Imagen de sustitución de foto de perfil

Desde esta pantalla el usuario puede cerrar sesión de su cuenta en el dispositivo o eliminar por completo su cuenta de Price Hunt. En ambos casos se hace una doble comprobación mediante los siguientes **DialogFragments**.

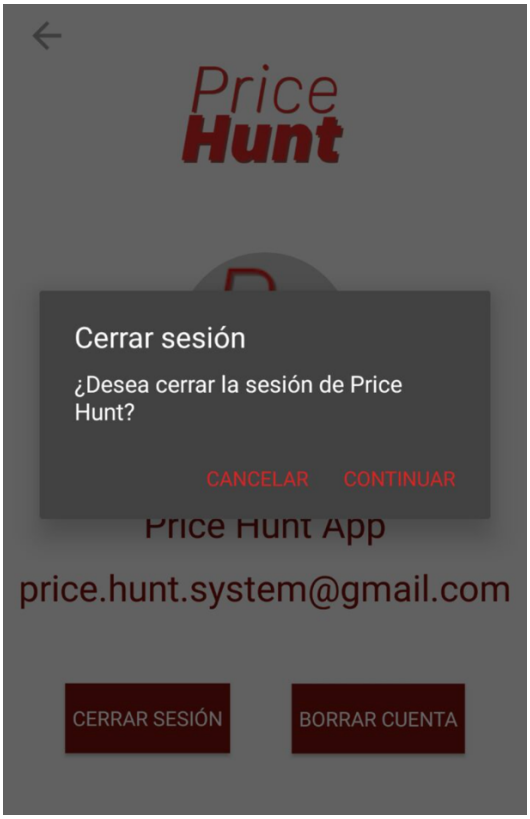

Figura 32. Diálogo de cierre de sesión

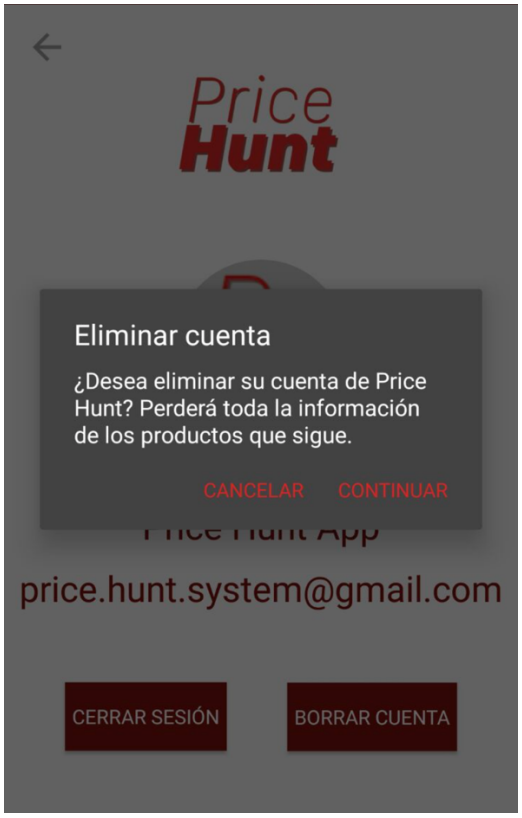

Figura 33. Diálogo de borrado de cuenta

4.2.6.6. Muestra de errores

Cualquier error que pueda producirse durante el uso de la aplicación se controla y se notifica al usuario mediante la clase **ShowError**, introducida en la sección 4.2.5. Esta clase muestra un mensaje de error explicativo adecuado a cada incidencia haciendo uso de un Toast o mensaje emergente. La aplicación puede lanzar hasta 13 mensajes de error diferentes.

Figura 34. Toast de error de conexión al servidor

4.2.6.7. Interfaz de los correos electrónicos

Los correos electrónicos han sido diseñados con una plantilla HTML sencilla, pero en sintonía con la interfaz de la aplicación. Con la misma plantilla HTML se generan hasta 5 tipos de correos diferentes: bajada de precio, bajada de precio por debajo de un valor, subida de precio, artículo no disponible y artículo disponible de nuevo.

A continuación, se muestran dos ejemplos de estos correos.

Figura 35. Correo de bajada de precio por debajo de una cantidad

Figura 36. Correo de artículo no disponible

4.2.7. ESTRUCTURA DE LA BASE DE DATOS

En las siguientes secciones se va a detallar la estructura de la base de datos PriceHunt utilizada para el servicio. Las tablas que se detallan a continuación son aquellas creadas para el funcionamiento del servicio, sin embargo, el procesamiento batch crea 9 tablas más para el control de sus procesos que no se detallarán en esta sección.

4.2.7.1. Diagrama entidad-relación

Para almacenar la información de los artículos y los usuarios de forma lógica se ha utilizado un esquema **relacional**, el cual ha sido implementado en una base de datos **MySQL**.

A continuación, se introducen las diferentes tablas que componen la base de datos y su diagrama entidad – relación.

- **Product**: tabla que contiene la información básica de un artículo o producto de una tienda online.
- **Brand**: entidad que representa una marca. Esta entidad tiene una relación uno a muchos (1:N) con la entidad Product.
- **Category**: entidad que representa una categoría. Las categorías se agrupan formando una jerarquía ya que cada categoría puede tener varias categorías

que dependan de ella. Por ello esta entidad tiene una relación reflexiva muchos a uno (N:1), ya que cada categoría puede tener una referencia a su categoría padre, que puede compartir con otras categorías. También tiene una relación 1:N con la entidad producto ya que a cada producto le asociamos la categoría inferior en la jerarquía.

- **Image**: representa una imagen de un producto. Un producto puede tener varias imágenes que a su vez solo pueden pertenecer a dicho producto, por lo que constata una relación N:1.
- **Additional_info**: tabla que representa información adicional sobre los productos en forma de pares clave – valor. Tiene una relación N:1 con el producto.
- **Price**: entidad que representa un precio de un producto en un momento determinado, incluyendo los gastos de envío. Cada producto va a tener varios precios a lo largo del tiempo, aunque cada precio únicamente está relacionado con un producto por lo que mantienen una relación N:1.
- **User**: representa la información básica de un usuario.
- **Product_user_config**: contiene la información de la configuración de las notificaciones de un usuario para un producto determinado. Tiene relaciones N:1 con la entidad producto y usuario.

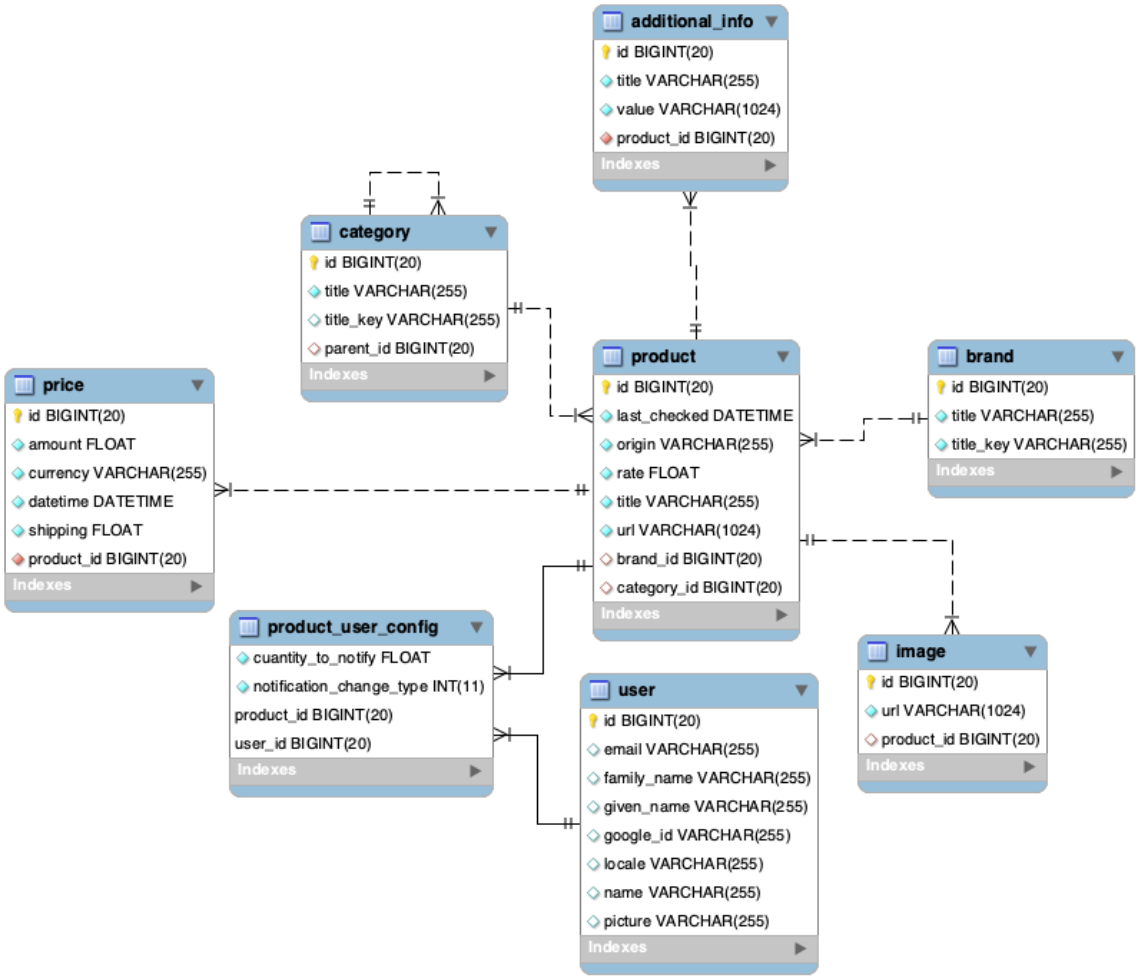

Figura 37. Diagrama Entidad-Relación de PriceHunt

4.2.7.2. Modelo relacional

En las tablas siguientes se describen las entidades anteriormente mencionadas: sus atributos, tipo de dato, propiedades, descripción y valor predeterminado.

Tabla 1. Entidad Product

Atributo	Tipo de dato	Propiedades	Descripción	Valor por defecto
id	BIGINT (20)	Clave Primaria (PK), Autoincrementar (AI)	Identificador del producto	-
Last_checked	DATETIME	No Nulo (NN)	Fecha de última actualización	-
origin	VARCHAR (255)	NN	Sitio web de origen (por ahora Amazon o eBay)	-
rate	FLOAT	NN	Puntuación asignada por los clientes del 0 al 5	-
title	VARCHAR (255)	NN	Título del artículo	-
url	VARCHAR (1024)	NN	Dirección web del artículo	-
brand_id	BIGINT (20)	-	Identificador de la marca	Null
category_id	BIGINT (20)	-	Identificador de la categoría inferior de la jerarquía	Null

Tabla 2. Entidad Brand

Atributo	Tipo de dato	Propiedades	Descripción	Valor por defecto
id	BIGINT (20)	PK, AI	Identificador de la marca	-
title	VARCHAR (255)	NN	Título de la marca	-
title_key	VARCHAR (255)	NN, UNIQUE(UQ)	Clave de la marca creada con el título en minúscula y sustituyendo espacios por guiones bajos (_)	-

Tabla 3. Entidad Price

Atributo	Tipo de dato	Propiedades	Descripción	Valor por defecto
id	BIGINT (20)	PK, AI	Identificador del precio	-
amount	FLOAT	NN	Cantidad a pagar por el producto o -2 si no está disponible	-
currency	VARCHAR (255)	NN	Divisa	-
datetime	DATETIME	NN	Fecha en la que se extrae el precio	-
shipping	FLOAT	NN	Gastos de envío o -2 si no está disponible	-
product_id	BIGINT (20)	NN	Identificador del producto al que corresponde	-

Tabla 4. Entidad Image

Atributo	Tipo de dato	Propiedades	Descripción	Valor por defecto
id	BIGINT (20)	PK, AI	Identificador de la imagen	-
url	VARCHAR (1024)	NN	Dirección web de la imagen	-
product_id	BIGINT (20)	NN	Identificador del producto al que pertenece.	-

Tabla 5. Entidad Category

Atributo	Tipo de dato	Propiedades	Descripción	Valor por defecto
id	BIGINT (20)	PK, AI	Identificador de la categoría	-
title	VARCHAR (255)	NN	Título de la categoría	-
title_key	VARCHAR (255)	NN, UQ	Clave de la categoría formada por el título en minúscula y sustituyendo espacios por guiones bajos (_)	-
parent_id	BIGINT (20)	-	Identificador de la categoría padre	Null

Tabla 6. Entidad Additional_info

Atributo	Tipo de dato	Propiedades	Descripción	Valor por defecto
id	BIGINT (20)	PK, AI	Identificador de la tupla de información	-
title	VARCHAR (255)	NN	Clave de la dupla de información	-
value	VARCHAR (1024)	NN	Valor de la dupla de información	-
product_id	BIGINT (20)	NN	Identificador del producto al que pertenece	-

Tabla 7. Entidad User

Atributo	Tipo de dato	Propiedades	Descripción	Valor por defecto
id	BIGINT (20)	PK, AI	Identificador del usuario en nuestro sistema	-
email	VARCHAR (255)	NN	Email del usuario	-
family_name	VARCHAR (255)	-	Apellido del usuario	Null
given_name	VARCHAR (255)	-	Nombre del usuario	Null
google_id	VARCHAR (255)	UQ	Identificador del usuario en el sistema de Google	Null
locale	VARCHAR (255)	-	Región del usuario	Null
name	VARCHAR (255)	-	Nombre completo del usuario	Null
picture	VARCHAR (255)	-	URL de la imagen del usuario	Null

Tabla 8. Entidad Product_user_config

Atributo	Tipo de dato	Propiedades	Descripción	Valor por defecto
cuantity_to_notify	FLOAT	NN	Cantidad a partir de la que notificar cambios	0
notification_change_type	INT (11)	NN	Tipo de notificación: 0 -> Ninguna 1-> Todas 2-> Bajadas de precio 3-> Bajadas por debajo de un precio	-
product_id	BIGINT (20)	PK, NN	Identificador del producto al que hace referencia	-
user_id	BIGINT (20)	PK, NN	Identificador del usuario al que hace referencia	-

4.2.8. DISEÑO DE LA API DE COMUNICACIÓN

En las siguientes secciones se describe la implementación de la interfaz diseñada entre la aplicación servidor y la aplicación Android.

4.2.8.1. User endpoints

Estos endpoints exponen los servicios de gestión de usuarios.

Tabla 9. Endpoint POST /user/{id}

POST /USER/{ID}	
Descripción	Comprobación del usuario en el sistema y crearlo en caso de no existir
Variables en el Path	ID: Identificador del servicio de google del usuario
Cabeceras obligatorias	Authentication – adjuntar el token ID
Campos en cuerpo	-
Respuestas	200 OK – Usuario existente Cuerpo de la respuesta: Información sobre el usuario
	201 Created – Usuario nuevo creado en el sistema Cuerpo de la respuesta: Información sobre el usuario
	401 Unauthorized – Token no válido Cuerpo de la respuesta: Información sobre el error
	404 Not Found – El id proporcionado no coincide con el del token. Cuerpo de la respuesta: Información sobre el error
	500 Internal Server Error – Error verificando el token Cuerpo de la respuesta: Información sobre el error

Tabla 10. Endpoint DELETE /user/{id}

DELETE /USER/{ID}	
Descripción	Eliminar un usuario del sistema
Variables en el Path	ID: Identificador del servicio de google del usuario
Cabeceras obligatorias	Authentication – adjuntar el token ID
Campos en cuerpo	-
Respuestas	204 No Content – Usuario eliminado
	401 Unauthorized – Token no válido Cuerpo de la respuesta: Información sobre el error
	404 Not Found – El id proporcionado no coincide con el del token. Cuerpo de la respuesta: Información sobre el error
	500 Internal Server Error – Error verificando el token Cuerpo de la respuesta: Información sobre el error

4.2.8.2. Notification endpoint

El endpoint de esta sección permite modificar la configuración de las notificaciones.

Tabla 11. Endpoint PUT /notification/{productId}

PUT /NOTIFICATION/{PRODUCTID}	
Descripción	Cambiar la configuración sobre las notificaciones de un producto.
Variables en el Path	ProductId: Identificador del producto
Cabeceras obligatorias	Authentication – adjuntar el token ID
Campos en cuerpo	notificationType (integer) [obligatorio] – Tipo de notificación (0-Ninguna, 1-Todas, 2-Bajadas de precio, 3-Bajadas por debajo de un valor)
	NotificationCuantity (float) – Valor a partir del cual notificar. Solo se aplica si el campo anterior es 3.
Respuestas	201 Created – Usuario nuevo creado en el sistema
	204 No Content – Configuración cambiada
	401 Unauthorized – Token no válido Cuerpo de la respuesta: Información sobre el error
	404 Not Found – Usuario no encontrado en el sistema Cuerpo de la respuesta: Información sobre el error
	500 Internal Server Error – Error verificando el token Cuerpo de la respuesta: Información sobre el error

4.2.8.3. Product endpoints

En esta sección se exponen todos los endpoints relacionados con la gestión de los productos.

Tabla 12. Endpoint POST /product

POST /PRODUCT	
Descripción	Añadir un producto al sistema
Variables en el Path	-
Cabeceras obligatorias	Authentication – adjuntar el token ID
Campos en cuerpo	url (String) [obligatorio] – URL de la página web del artículo
Respuestas	200 OK – Producto añadido correctamente Cuerpo de la respuesta: Información reducida del producto.
	204 No Content – Producto ya existente.
	400 Bad Request – La url no es válida. Cuerpo de la respuesta: Información sobre el error
	401 Unauthorized – Token no válido Cuerpo de la respuesta: Información sobre el error
	500 Internal Server Error – Error verificando el token o realizando la extracción de información de la web. Cuerpo de la respuesta: Información sobre el error

Tabla 13. Endpoint GET /product/list

GET /PRODUCT/LIST	
Descripción	Obtener la lista de productos de un usuario
Variables en el Path	-
Cabeceras obligatorias	Authentication – adjuntar el token ID
Campos en cuerpo	-
Respuestas	200 OK – Lista disponible Cuerpo de la respuesta: Lista de información reducida de los productos. Vacía si el usuario no tiene productos.
	404 Not Found – Usuario no encontrado en el sistema. Cuerpo de la respuesta: Información sobre el error
	401 Unauthorized – Token no válido Cuerpo de la respuesta: Información sobre el error
	500 Internal Server Error – Error verificando el token o realizando la extracción de información de la web. Cuerpo de la respuesta: Información sobre el error

Tabla 14. Endpoint GET /product/{id}

GET /PRODUCT/{ID}	
Descripción	Obtener toda la información de un producto
Variables en el Path	ID: Identificador del producto del que se quiere obtener la información.
Cabeceras obligatorias	Authentication – adjuntar el token ID
Campos en cuerpo	-
Respuestas	200 OK – Información disponible Cuerpo de la respuesta: Información completa del producto.
	404 Not Found – Producto no encontrado en el sistema. Cuerpo de la respuesta: Información sobre el error
	401 Unauthorized – Token no válido Cuerpo de la respuesta: Información sobre el error
	500 Internal Server Error – Error verificando el token o realizando la extracción de información de la web. Cuerpo de la respuesta: Información sobre el error

Tabla 15. Endpoint DELETE /product/{id}

DELETE /PRODUCT/{ID}	
Descripción	Eliminar un producto
Variables en el Path	ID: Identificador del producto que se desea eliminar
Cabeceras obligatorias	Authentication – adjuntar el token ID
Campos en cuerpo	-
Respuestas	204 No Content – Producto eliminado*. *Nota: El producto sigue en el sistema, aunque no asociado al usuario.
	404 Not Found – Producto o usuario no encontrado en el sistema. Cuerpo de la respuesta: Información sobre el error
	401 Unauthorized – Token no válido Cuerpo de la respuesta: Información sobre el error
	500 Internal Server Error – Error verificando el token o realizando la extracción de información de la web. Cuerpo de la respuesta: Información sobre el error

Tabla 16. Endpoint POST /product/errorReport/{id}

POST /PRODUCT/ERRORREPORT/{ID}	
Descripción	Reportar un error en la información de un producto
Variables en el Path	ID: Identificador del producto.
Cabeceras obligatorias	Authentication – adjuntar el token ID
Campos en cuerpo	msg (String) [obligatorio] – Texto explicativo introducido por el usuario.
Respuestas	200 OK – Reporte satisfactorio
	400 Bad Request – La url no es válida. Cuerpo de la respuesta: Información sobre el error
	401 Unauthorized – Token no válido Cuerpo de la respuesta: Información sobre el error
	500 Internal Server Error – Error verificando el token o realizando la extracción de información de la web. Cuerpo de la respuesta: Información sobre el error

4.2.8.4. Management endpoint

Se ha implementado un endpoint de gestión para iniciar el proceso batch que actualiza la información de los productos de forma manual.

Tabla 17. Endpoint GET /scraping/start

GET /SCRAPING/START	
Descripción	Iniciar el proceso batch de actualización
Variables en el Path	-
Cabeceras obligatorias	-
Campos en cuerpo	-
Respuestas	200 OK – Proceso iniciado correctamente
	500 Internal Server Error – Error verificando el token o realizando la extracción de información de la web. Cuerpo de la respuesta: Información sobre el error

4.2.9. IMPLEMENTACION DE LA AUTENTICACIÓN DE USUARIOS

La gestión de usuarios no es una cuestión principal en el servicio, pero sí es necesaria. Por esto se ha intentado reducir la gestión de usuarios por parte de nuestro servicio, delegando la autenticación de los mismos en la herramienta de Google Sign In, mencionada en el apartado 4.1.12.

Para hacer uso de esta tecnología, es necesario que creemos un proyecto como desarrolladores en la consola de Google y que configuremos la API para obtener los identificadores cliente a utilizar para el protocolo OAuth 2.0. Este protocolo y cómo obtener los identificadores se explica más detalladamente en el ANEXO IV. OBTENCIÓN DE LOS IDENTIFICADORES DE CLIENTE PARA EL PROTOCOLO OAUTH 2.0.

4.2.9.1. Autenticación de peticiones con el servidor

A demás de para iniciar sesión, esta herramienta nos permite autenticar todas las peticiones que la aplicación Android realiza a nuestro servidor (pedir lista de productos, pedir toda la información sobre un producto, añadir un producto nuevo, etc.).

Como se muestra en el siguiente diagrama, En todas las peticiones se envía la cabecera authorization con el TokenID que nos proporciona Google. En el servidor se verifica el token y, si es válido, se procesa la petición y responde lo que corresponda. Si el token no es válido, se responde un 401 Unauthorized. Al llegar la respuesta a la aplicación, si es un 401 generalmente se debe a que el token está caducado por lo que se realiza un sign in silencioso, se renueva el token sin intervención del usuario y se repite la petición.

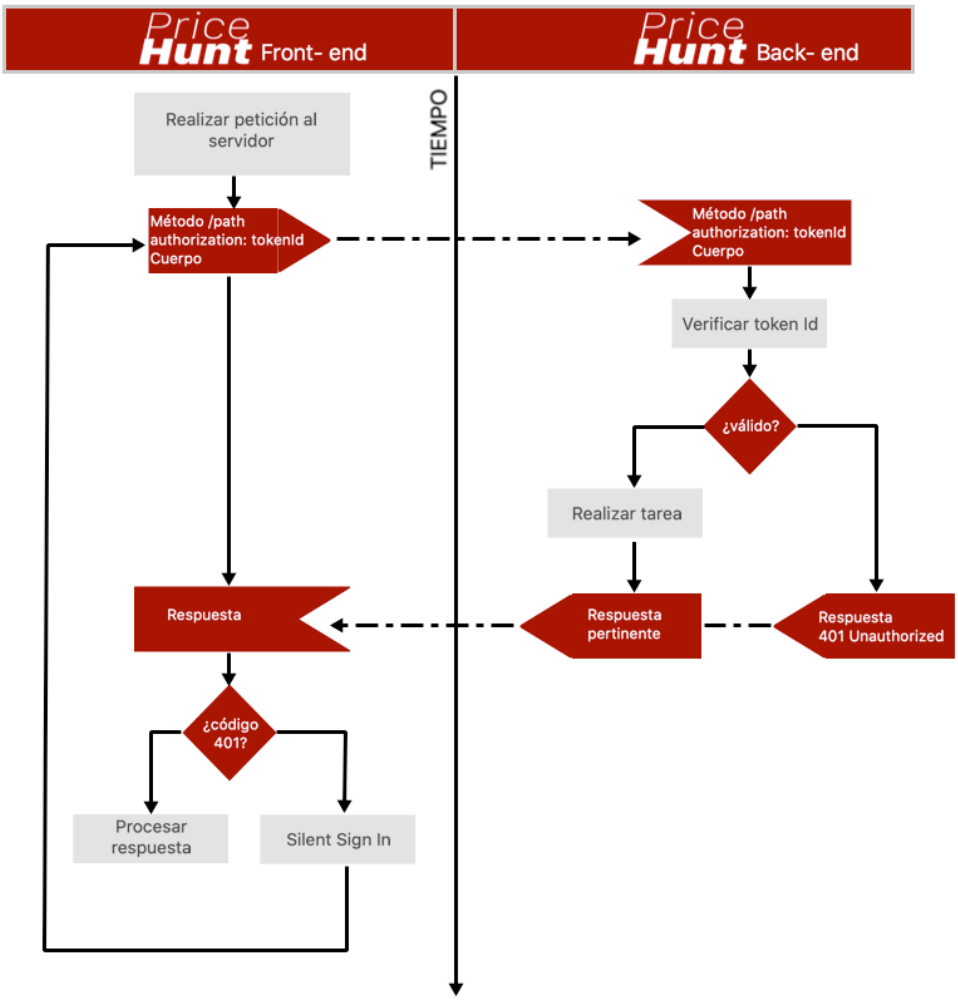

Figura 38. Flujo de autenticación de peticiones

4.2.9.2. API Google Sign In para Android.

Google nos proporciona una API en Java para poder utilizar este servicio de forma sencilla y rápida. Para hacer uso de ella, debemos añadir el repositorio maven de Google y la dependencia de Google Play Services (38).

```
//Google Sign-in
implementation 'com.google.android.gms:play-services-auth:18.0.0'
```

Figura 39. Configuración gradle de la aplicación, dependencia de Google Play Services

```
allprojects {
    repositories {
        google()
        jcenter()
        mavenCentral()
        maven { url 'https://jitpack.io' }

    }
}
```

Figura 40. Configuración gradle del proyecto, repositorio maven de Google

Para utilizar la API debemos crear una configuración con los parámetros que vamos a requerir a Google. En este caso nos interesa obtener el email y el identificador del token para la autenticación. Para ello, debemos pasarle el identificador del cliente web de OAuth 2.0.

Finalmente, con esta configuración obtenemos el cliente de Google Sign In para realizar las siguientes funcionalidades:

```
GoogleSignInOptions gso = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
        .requestIdToken(CLIENT_ID)
        .requestEmail()
        .build();

mGoogleSignInClient = GoogleSignIn.getClient( activity: this, gso);
```

Figura 41. Configuración del cliente Android de Google Sign In

- **Obtener último usuario identificado**: El cliente busca si el usuario ya ha iniciado sesión en nuestra aplicación con anterioridad. En caso positivo, podemos actualizar la interfaz de usuario y no es necesario mostrar la pantalla de la Figura 14 (38).

```
GoogleSignInAccount account = GoogleSignIn.getLastSignedInAccount( context: this);
```

Figura 42. Google Sign In Android. Obtener última cuenta identificada

- **Iniciar sesión:** La API nos proporciona el botón para iniciar el flujo de inicio de sesión. Una vez implementado, debemos configurarlo para que inicie el flujo de sesión de la Figura 15 y controlar el resultado que nos devuelva. En las siguientes figuras se puede ver este proceso (23).

```
<com.google.android.gms.common.SignInButton
    app:layout_constraintTop_toBottomOf="@id/iv_pricehunt"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:id="@+id/sign_in_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

Figura 43. Google Sign In Android. Botón de inicio de sesión.

```
SignInButton signInButton = findViewById(R.id.sign_in_button);
signInButton.setSize(SignInButton.SIZE_STANDARD);
signInButton.setOnClickListener(btn -> signIn());
```

Figura 44. Google Sign In Android. Configurar botón de inicio de sesión.

```
public void signIn(){
    Intent signInIntent = mGoogleSignInClient.getSignInIntent();
    startActivityForResult(signInIntent, RC_SIGN_IN);
}
```

Figura 45. Google Sign In Android. Iniciar el flujo de inicio de sesión.

```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == RC_SIGN_IN) {
        Task<GoogleSignInAccount> task = GoogleSignIn.getSignedInAccountFromIntent(data);
        handleSignInResult(task);
    }
}
```

Figura 46. Google Sign In Android. Resultado del flujo de inicio de sesión.

```
private void handleSignInResult(Task<GoogleSignInAccount> completedTask) {
    try {
        GoogleSignInAccount account = completedTask.getResult(ApiException.class);
        updateUI(account);
    } catch (ApiException e) {
        Log.w(TAG,  msg: "signInResult:failed code=" + GoogleSignInStatusCodes.getStatusCodeString(e.getStatusCode()));
        switch(e.getStatusCode()){
            case GoogleSignInStatusCodes.SIGN_IN_FAILED:
                new ShowError( mContext: SignInActivity.this).showToast(ShowError.SIGN_IN_ERROR,GoogleSignInStatusCodes.SIGN_IN_FAILED);
            case GoogleSignInStatusCodes.SIGN_IN_REQUIRED:
                showSignInLayout();
                break;
            case GoogleSignInStatusCodes.SIGN_IN_CANCELLED:
                break;
                default:
                    updateUI( account: null);
        }
    }
}
```

Figura 47. Google Sign In Android. Control del resultado de inicio de sesión.

- **Realizar un inicio de sesión silencioso**: Intenta hacer un inicio de sesión sin intervención del usuario. Esto es útil cuando el usuario ha iniciado sesión nuestro servicio pero la sesión ha caducado (39).

```
private void silentSignIn(){
    mGoogleSignInClient.silentSignIn()
            .addOnCompleteListener(
                    mActivity,
                    new OnCompleteListener<GoogleSignInAccount>() {
                        @Override
                        public void onComplete(@NonNull Task<GoogleSignInAccount> task) {
                            handleSignInResult(task);
                        }
                    });
}
```

Figura 48. Google sign In Android. Silent Sign In

- **Cerrar sesión**: Desde la pantalla de la información del perfil de la Figura 30, se le ofrece al usuario la posibilidad de cerrar sesión en nuestra aplicación. Tras esto, el usuario deberá iniciar sesión manualmente siguiendo el flujo de inicio de sesión de la Figura 15 para volver a acceder a nuestra aplicación.

```
private void signOut(){
    getGoogleSignInClient().signOut()
            .addOnCompleteListener( activity: ProfileActivity.this,
                    new OnCompleteListener<Void>() {
                        @Override
                        public void onComplete(@NonNull Task<Void> task) {
                            exitToSignInActivity();
                        }
                    });
}
```

Figura 49. Google Sign In Android. Cerrar sesión

- **Revocar los permisos de acceso**: Desde la pantalla de la información de perfil de la Figura 30 se le ofrece al usuario la posibilidad de eliminar su cuenta de Price Hunt. Tras eliminar la información del usuario de nuestro servidor debemos desconectar nuestra aplicación de su cuenta (40).

```
private void revokeAccess(){
    getGoogleSignInClient().revokeAccess()
            .addOnCompleteListener( activity: this, new OnCompleteListener<Void>() {
                @Override
                public void onComplete(@NonNull Task<Void> task) { exitToSignInActivity(); }
            });
}
```

Figura 50. Google Sign In Android. Revocar permisos de acceso.

4.2.9.3. API Google Sign In para Servidor en Java

Google nos proporciona librerías cliente de la API de Google Sign In para la autenticación contra servidores de back-end en Java, Node.js, PHP y Python. En este proyecto se ha utilizado la librería en Java para realizar la verificación de los tokens (39).

Para hacer uso de ella es necesario insertar la dependencia de la siguiente figura en el archivo pom.xml del módulo de user, que es el que va a hacer uso de esta funcionalidad.

```xml
<!-- Google sign in api -->
<dependency>
    <groupId>com.google.api-client</groupId>
    <artifactId>google-api-client</artifactId>
    <version>1.30.8</version>
</dependency>
```

Figura 51. Google Sign In Servidor. Dependencia

El cliente se va a utilizar en nuestro servicio GoogleUserMngmtServiceImpl introducido en el apartado 4.2.4.2.2.

Para hacer uso del cliente debemos obtener el GoogleNetHttpTransport que se encargará del trasporte de las peticiones y una instancia de JacksonFactory para el mapeo del JSON.

```
public GoogleUserMngmntServiceImpl() throws GeneralSecurityException, IOException {
    transport = GoogleNetHttpTransport.newTrustedTransport();
    jsonFactory = JacksonFactory.getDefaultInstance();
}
```

Figura 52. Google Sign In Servidor. Obtener objetos necesarios.

A partir de los objetos recuperados construimos el verificador del token, al que le pasamos el identificador de cliente OAuth 2.0 del servidor. Verificar el token es tan sencillo como llamar a la función “verify” del verificador, pasándole el identificador del token como parámetro. En caso de verificación positiva, el verificador nos devolverá el token, del que podemos extraer la información del usuario. Por el contrario, si el token no es válido, generalmente por haber caducado, el verificador nos devolverá un objeto nulo.

```
public User verifyTokenForUser(String tokenIdString)
        throws GeneralSecurityException, IOException, InvalidTokenException {
    GoogleIdTokenVerifier verifier = new GoogleIdTokenVerifier.Builder(transport, jsonFactory)
            .setAudience(Collections.singletonList(CLIENT_ID)).build();

    GoogleIdToken idToken = verifier.verify(tokenIdString);
    if (idToken != null) {
        Payload payload = idToken.getPayload();

        return new User(payload.getSubject(), payload.getEmail(), (String) payload.get("name"),
                (String) payload.get("given_name"), (String) payload.get("family_name"),
                (String) payload.get("picture"), (String) payload.get("locale"));

    } else {
        throw new InvalidTokenException();
    }
}
```

Figura 53. Google Sign In Servidor. Verificar token y obtener información de usuario.

4.2.10. DISEÑO E IMPLEMENTACIÓN DE LA EXTRACCIÓN DE LA INFORMACIÓN DE LOS PRODUCTOS

La extracción de la información de los productos se realiza mediante la técnica del scraping, introducida en la sección 4.1.3. A continuación, se muestra un diagrama del funcionamiento del scraping en nuestro servidor. También se va a exponer el problema que surgió durante su implementación y las soluciones aportadas.

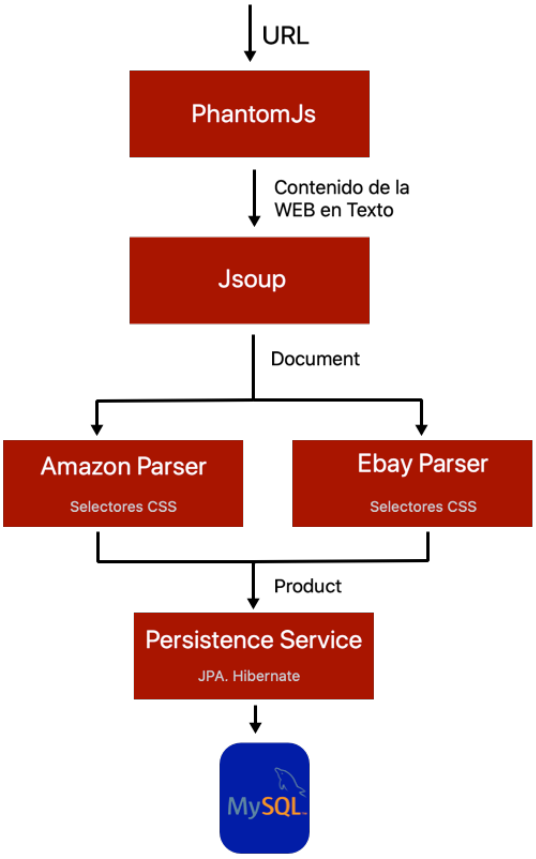

Figura 54. Diagrama de funcionamiento del scraping

En primer lugar, se comprueba que la URL que ha introducido el usuario a través de la aplicación es válida, es decir, que esté bien formada y que pertenezca a alguno de los sitios webs para los que hemos preparado nuestro crawler.

A continuación, se recupera la página web del producto con el navegador PhantomJS (introducido en la sección 4.1.4) en forma de texto plano. Para poder aprovechar las ventajas que ofrece que HTML sea un lenguaje estructurado y jerárquico, se utiliza la librería JSOUP (introducida en la sección 4.1.5) para transformar ese texto plano en un objeto Java, Document.

Finalmente, el documento es pasado a un parseador según el origen del artículo. Haciendo uso de los selectores CSS (introducidos en la sección 4.1.5) se extrae la información que nos interesa de la página y se procesa, para almacenarla de la forma más óptima para el uso futuro.

Para ilustrar la diferencia en las estructuras de las páginas dentro de un mismo sitio web, se muestra en la siguiente figura todos los selectores CSS utilizados por el parseador de Amazon hasta el momento. Como se ve en la imagen, dentro del mismo sitio podemos necesitar buscar algún elemento, como los gastos de envío, en múltiples elementos.

```
private static final String TITLE = "#title";
private static final String PRICE = "#priceblock_ourprice";
private static final String PRICE2 = "#price_inside_buybox";
private static final String PRICE3 = "#soldByThirdParty";
private static final String PRICE4 = "tr.kindle-price > td.a-color-price.a-size-medium.a-align-bottom > span";
private static final String AVAILABILITY = "#availability";
private static final String PRICESHIPPING = "#price-shipping-message";
private static final String PRICESHIPPING2 = "span.buyboxShippingLabel";
private static final String PRICESHIPPING3 = "#ourprice_shippingmessage";
private static final String PRICESHIPPING4 = "#shippingMessageInsideBuyBox_feature_div";
private static final String CATEGORIES = "#wayfinding-breadcrumbs_feature_div > ul > li > span.a-list-item";
private static final String BRAND = "a#bylineInfo";
private static final String PRODUCT_INFO = "#prodDetails > div.wrapper.ESlocale > div.column.col1 > div > div.
private static final String ADDITIONAL_INFO = "#prodDetails > div.wrapper.ESlocale > div.column.col2 > div:nth-
private static final String ADDITIONAL_INFO2 = "#detail_bullets_id > table > tbody > tr > td > div.content > u
private static final String ADDITIONAL_INFO3 = "div.tsRow";
private static final String RATE = "#reviewsMedley > div > div.a-fixed-left-grid-col.a-col-left > div.a-sectio
private static final String IMAGES = "#altImages > ul > li.imageThumbnail";
private static final String IMAGES2 = "#ebooks-img-canvas > img";
```

Figura 55. Selectores CSS utilizados en el parseador de Amazon

4.2.10.1. Bloqueo de PhantomJS por Amazon

Muchos sitios webs intentan **bloquear** el uso de estas técnicas porque introducen datos de visitas falsos que no favorecen a sus algoritmos. Para ello puede emplearse el uso de diferentes técnicas, como puede ser el bloquear una dirección IP que realice peticiones masivas.

Durante el desarrollo del parseador de Amazon surgió un problema que impedía recuperar las páginas de los productos. El servidor de Amazon bloqueaba nuestra petición y nos respondía con el siguiente mensaje:

> *To discuss automated access to Amazon data please contact api-services-support@amazon.com.*
>
> *For information about migrating to our APIs refer to our Marketplace APIs at https://developer.amazonservices.es/ref=rm_c_sv, or our Product Advertising API at https://afiliados.amazon.es/gp/advertising/api/detail/main.html/ref=rm_c_ac for advertising use cases.*

Esto no sucedía todas las veces que se intentaba recuperar un producto, pero sí que era recurrente durante un periodo de tiempo prolongado una vez ocurría. Este funcionamiento parecía indicar que Amazon estaba bloqueando peticiones recurrentes desde mi origen.

La solución parecía ser utilizar un **proxy intermedio** que ocultara la dirección de origen de mis peticiones. Este servicio se detallará en mayor medida en el siguiente apartado. El uso de este tipo de proxy suele ser de pago, y el rendimiento de aquellos gratuitos disponibles por internet es bastante bajo.

Sin embargo, esta solución no evitó que Amazon siguiera bloqueando las peticiones. Finalmente se descubrió que la solución era configurar PhantomJS para que utilizara en las peticiones la **cabecera UserAgent** con un valor que imitara al de otros navegadores de uso humano. De esta forma, Amazon dejó de bloquear las peticiones asumiendo que venían de un humano y no de un bot.

```
desiredCapabilities.setCapability("phantomjs.page.settings.userAgent",
        "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36");
```

Figura 56. Configuración cabecera UserAgent en PhantomJS

Una vez implementada la solución definitiva se desactivó el servicio de proxies, ya que introducía un gran retardo en la recuperación de las páginas (a veces de varios minutos) y complejidad en la gestión del servicio. Sin embargo, no se ha eliminado de la aplicación porque podría ser necesario utilizarlo cuando el volumen de peticiones fuera mayor o al añadir nuevos sitios webs al crawler.

4.2.10.2. ProxyService

Con la intención de utilizar proxies intermedios se diseñó e implementó el servicio **ProxyService** (se encuentra en el paquete de la sección 4.2.4.3.5) que utilizara una lista de proxies gratuitos para realizar las peticiones.

El funcionamiento era el siguiente:

I. El servicio leía la lista de el archivo proxyList.txt (introducido en la sección 4.2.4.5) y cargaba los proxies en memoria. Todos los proxies eran marcados como activos (“enabled = true”) y no válidos (“valid = false”).

II. En cada iteración, el servicio cogía primer proxy marcado como activo (“enabled = true”) y lo configura en PhantomJS para que lo utilizara como intermediario en la petición.
    a. Si este devolvía correctamente al menos una página, era marcado como valido (“valid = true”).
    b. Cuando un proxy devolvía una página incorrecta era marcado como no activo (“enabled = false”), pasando así al siguiente.

III. Al acabar la lista o cuando sea necesario actualizar el archivo, se eliminan los proxys marcados como no activos (“enabled = false”) y no válidos (“valid = false”), aquellos que habiendo sido usados no han devuelto ninguna página correctamente.

IV. El archivo se sobrescribe para guardar los proxies válidos para futuras ejecuciones del crawler.

Los proxies gratuitos disponibles por internet suelen dar un rendimiento muy bajo y su disponibilidad suele ser bastante corta. Para que este servicio funcione es necesario que

el gestor de la aplicación añada periódicamente proxies nuevos al fichero. Si el fichero se encontrara vacío, las peticiones se realizarían sin ningún proxy intermedio.

4.2.10.3. ErrorService

Cada sitio web construye su página con una estructura diferente, variando incluso entre páginas del mismo sitio web. Es por esto que el scraping puede definirse como una técnica iterativa. Partiendo de un análisis previo de las páginas de un sitio web lo más extenso posible, se diseña un parseador de ese sitio web que intente ajustarse a todas las páginas. Sin embargo, esto no quiere decir que no puedan surgir páginas con estructuras diferentes o que las páginas ya existentes varíen su estructura con el paso del tiempo.

Debido a esto, se ha implementado un servicio en la aplicación servidor para ayudar al gestor a corregir los errores del scraping que vayan surgiendo. Este servicio, introducido en el apartado 4.2.4.3.6, recopila los errores producidos de dos fuentes diferentes: los errores críticos del parseo de un producto de forma automática y los errores de la información ya existente en la base de datos a través de los reportes que los usuarios introducen a través de la aplicación Android (véase Figura 29. Pantalla de reporte de errores).

Estos errores son volcados a un archivo de texto con información básica sobre el mismo de forma que puedan ser identificados y corregidos. Entre otras cosas, se informa la causa del error, el origen del reporte, la dirección URL del producto.

El servicio genera un fichero de errores diario, como se introdujo en el apartado 4.2.4.5 Ficheros externos.

4.2.11. DISEÑO E IMPLEMENTACIÓN DE LA ACTUALIZACIÓN PERIÓDICA DE LA INFORMACIÓN

La actualización de la información de los productos supone el procesamiento de una cantidad de información considerable en forma de una tarea sencilla y repetitiva. El objetivo es programar la ejecución de esta tarea para que no necesite intervención humana y se realice automáticamente de forma periódica. Esto se ha llevado a cabo utilizando las tecnologías introducidas en el apartado 4.1.8, **Spring Batch** y **Spring Scheduler.**

La implementación del **proceso batch** que se encarga de la actualización de la información de todos los productos se lleva a cabo en la clase BatchConfiguration, introducida en la sección 4.2.4.3.1. El proceso batch definido tiene la siguiente estructura:

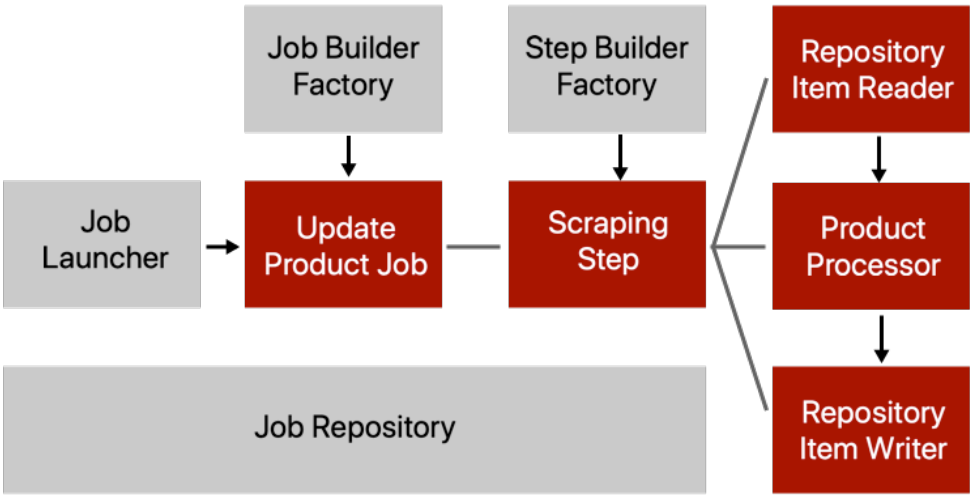

Figura 57. Estructura proceso Batch de actualización de productos

- **Update Product Job**: Un trabajo (Job) es la representación del proceso a llevar a cabo, en este caso, la actualización de la información de un producto. Este trabajo es un contenedor de un único paso (Step), aunque podría contener varios (41).
- **Scraping Step**: Un paso representa una fase independiente dentro de un proceso. El proceso de actualizar los productos sólo tiene un paso, el scraping web. Este paso, aunque no sería obligatorio, está compuesto de un item reader (elemento lector), un item processor (elemento procesador) y un item writer (elemento escritor). Los pasos siguen un procesamiento Chunk-oriented. Si definimos un chunk (pedazo) como un elemento que es leído, procesado y persistido, se puede definir un número de “chunks” (o intervalo de commit) que serán llevados a cabo en una misma transacción (41).
- **Repository Item Reader**: Un item reader es un elemento encargado de la lectura de los datos de una fuente, en este caso un repositorio de la base de datos MySQL. Este elemento recupera los productos almacenados en la base de datos para actualizarlos (41).
- **Product Processor**: El item procesor es el encargado de realizar el procesamiento del elemento recuperado. En esta implementación, se utiliza la técnica de scraping web explicada en el apartado 4.2.10 para actualizar los datos del producto y recuperar el precio de ese momento (41).
- **Repository Item Writer**: Un item writer se encarga de guardar la información procesada. En esta implementación el writer es del mismo tipo que el reader ya que se devuelve la información procesada a la base de datos (41).

- **Job Builder Factory**: Su función es construir los trabajos con los pasos que se indique.
- **Step Builder Factory**: La finalidad de este elemento es la de construir los pasos, pasándole los elementos que lo han de componer (reader, writer y processor). Al crearlo es el momento de definir el intervalo de commit, el número de pedazos que se procesaran en la misma transacción.
- **Job Launcher**: Componente encargado de lanzar los trabajos. Podemos añadirle parámetros como la fecha y hora en la que se lanza. Esto permite que el mismo trabajo se pueda ejecutar varias veces en el mismo día, ya que no te permite lanzar dos trabajos con los mismos parámetros (41).
- **Job Repository**: Este componente almacena de forma trasparente los metadatos del proceso en la base de datos. Tal y como se mencionó en el apartado 4.2.7, este elemento crea nueve tablas adicionales en la base de datos para almacenar estos metadatos (41).

Una vez implementado el proceso batch, utilizamos la etiqueta **@Scheduled** (cron = "0 5 2 * * *") para programar que se ejecute periódicamente. Con esa expresión cron el proceso se iniciará todos los días a las 2:05 AM. Se ha seleccionado la hora de ejecución durante la noche para que esta actualización no baje el rendimiento de usuarios que pudieran utilizar la aplicación.

Esta funcionalidad está disponible ya que está configurada la etiqueta @EnableScheduling en la clase principal de la aplicación, tal y como se menciona en la sección 4.2.4.1.1.

4.2.12. DISEÑO E IMPLEMENTACIÓN DEL SISTEMA DE NOTIFICACIÓN A USUARIOS

El tema estético del sistema de notificaciones se ha diseñado de forma que sea reconocible y asociable a toda la interfaz de la aplicación. Para ello se ha creado la cuenta de correo price.hunt.system@gmail.com y se ha utilizado como foto de perfil el logo de la aplicación. El estilo y los colores escogidos para los correos, como se puede ver en la sección 4.2.6.6, también complementa a los de la aplicación Android.

Para poder utilizar esta cuenta de correo de Gmail como cuenta origen y utilizar el servidor SMTP de Google para los envíos, ha sido necesario configurar una contraseña de aplicación. Las contraseñas de aplicación te permiten iniciar sesión desde otras aplicaciones menos seguras. Puedes generar esta contraseña desde la configuración de seguridad de la cuenta de Google siempre que esté activada la verificación en dos pasos.

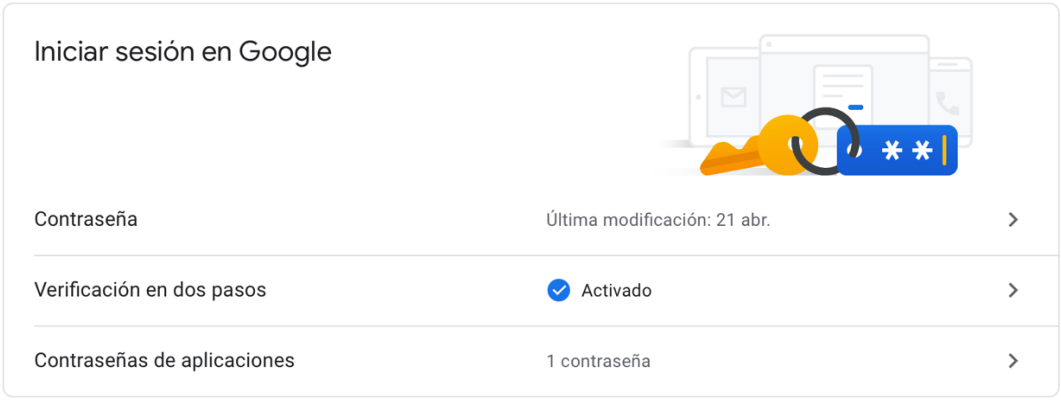

Figura 58. Configuración de Inicio de sesión en Google

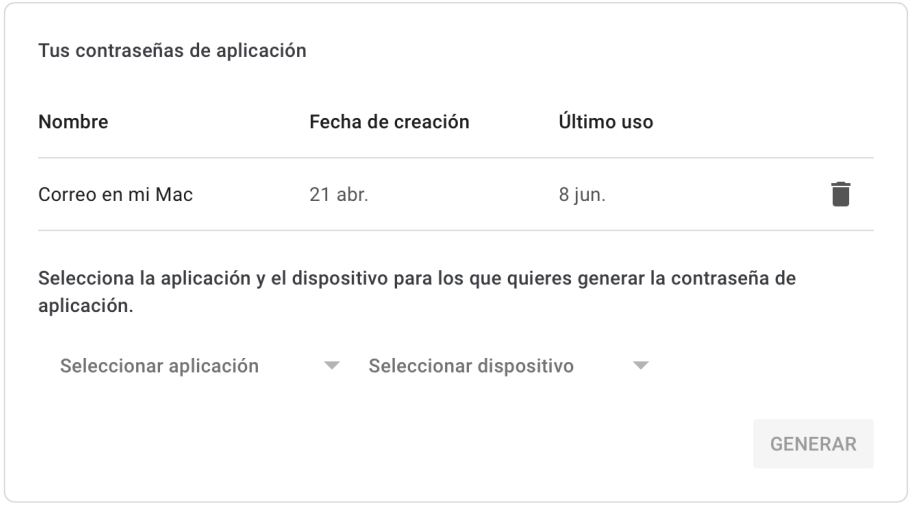

Figura 59. Configuración de contraseña de aplicación de Google

Para el envío de los correos desde el servidor se ha desarrollado el servicio **EmailService** (perteneciente al paquete del apartado 4.2.4.2.2). Este servicio utiliza Thymeleaf para crear los correos a partir de una plantilla y Spring Email para enviarlos, ambas tecnologías introducidas en el apartado 4.1.9. Existen cinco tipos de correos diferentes: Bajada de precio, subida de precio, bajada de precio por debajo de una cantidad, producto no disponible y producto disponible de nuevo.

También es necesario implementar un servicio que gestione cuándo se debe enviar una notificación y cuál se debe enviar. Este servicio es el **NotificationService**. Cada vez que se actualiza un producto es pasado a este servicio, el cual de forma asíncrona compara si el precio ha variado y, en caso positivo, introduce la configuración de todos los usuarios que siguen a ese producto en una cola. Los usuarios introducidos en la cola se van extrayendo en orden de llegada (es una cola FIFO, First In, First Out), se estudia si debe enviarse el correo según la configuración introducida y la variación ocurrida. Este proceso acaba cuando el proceso batch termina (si está activo) y la cola está vacía.

Los usuarios pueden modificar las preferencias de notificaciones desde la aplicación Android como se mostró en el apartado 4.2.6.5.

4.2.13. DISEÑO E IMPLEMENTACIÓN DE PREDICCIÓN DE PRECIOS Y EXTRACCIÓN DE CONCLUSIONES

Extrayendo la información del precio de los productos a lo largo del tiempo se puede observar una tendencia generalizada del mismo. Con el desarrollo actual del servicio no ha sido posible la obtención de datos a gran escala ni durante un periodo de tiempo suficiente como para aplicar algoritmos complejos de análisis y predicción. Por ello, se ha propuesto una regresión simple utilizando la librería Apache Commons Math, introducida en la sección 4.1.10.

Los valores de la predicción se calculan en el servidor antes de enviar la información completa de un producto a la aplicación Android. Si el producto en cuestión tiene más de 10 valores de precios, se realizan los siguientes pasos:

Figura 60. Diagrama de pasos regresión simple

I. Los precios han sido obtenidos mediante el **scraping web** y se encuentran almacenados en la base de datos.
II. Aunque el servicio está programado para actualizar los datos todos los días a la misma hora, puede darse que no todos los precios tengan la misma hora, ya que puede haberse actualizado el producto de nuevo en otro momento posterior del día por otro usuario que añada ese mismo producto a su lista. Dado que la granularidad con la que se muestran los datos es de un día, **adaptamos los datos** para eliminar granularidad menor y que la representación de los mismos quede más clara.
III. Utilizando la librería Commons Math **creamos el modelo** de regresión simple.
IV. **Entrenamos** el modelo con los datos disponibles.

V. Utilizamos el modelo entrenado para hacer la predicción de todos los días disponibles y de los 5 días siguientes.

VI. Finalmente se visualizan los datos en el histórico del producto de la aplicación Android como se ve en las siguientes figuras.

Figura 61. Histórico y predicción de precios 1

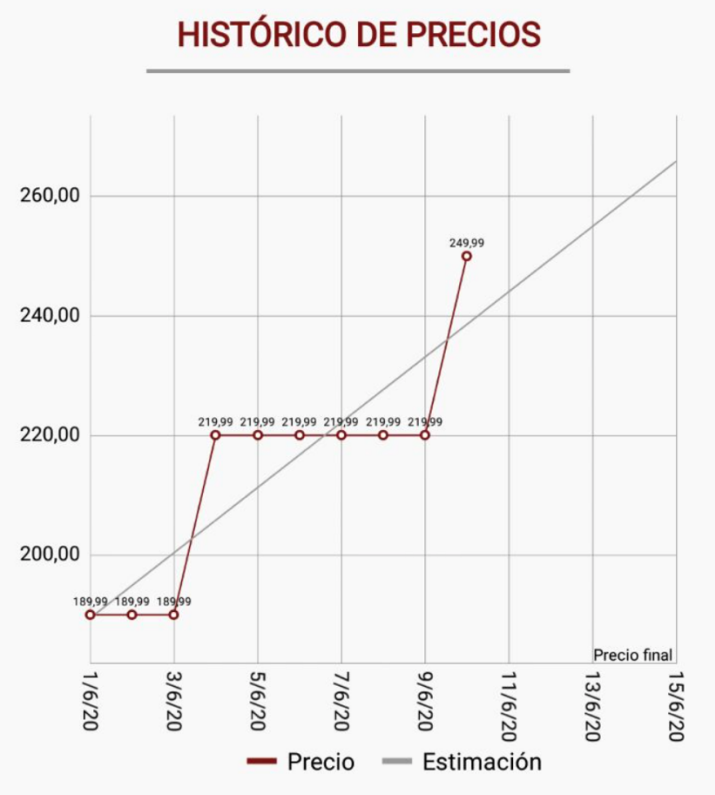

Figura 62. Histórico y predicción de precios 2

Figura 63. Histórico y predicción de precios 3

Se puede apreciar en las figuras que el modelo, al ser un modelo lineal, no se ajusta exactamente a la gráfica de precios, que varía de forma más o menos arbitraria. Este

modelo, aunque no es la opción óptima para predecir precios exactos, nos aporta una visión de la tendencia general del producto.

4.2.14. IMPLEMENTACIÓN DE LA APLICACIÓN SERVIDOR EN LA NUBE

Una vez desarrollada la aplicación servidor con Spring Boot, el objetivo final era implementarla en la nube. Para ello, se escogió la nube de Google (Google Cloud Platform) ya que, de entre las nubes más utilizadas actualmente, es la que ofrece un periodo gratuito de prueba.

Las instrucciones detalladas del despliegue en la nube se encuentran en el ANEXO I. MANUAL DE INSTALACIÓN DE LA APLICACIÓN SERVIDOR. A continuación, se muestra un diagrama con los pasos que se han seguido para realizar esta implementación y una breve explicación de los mismos.

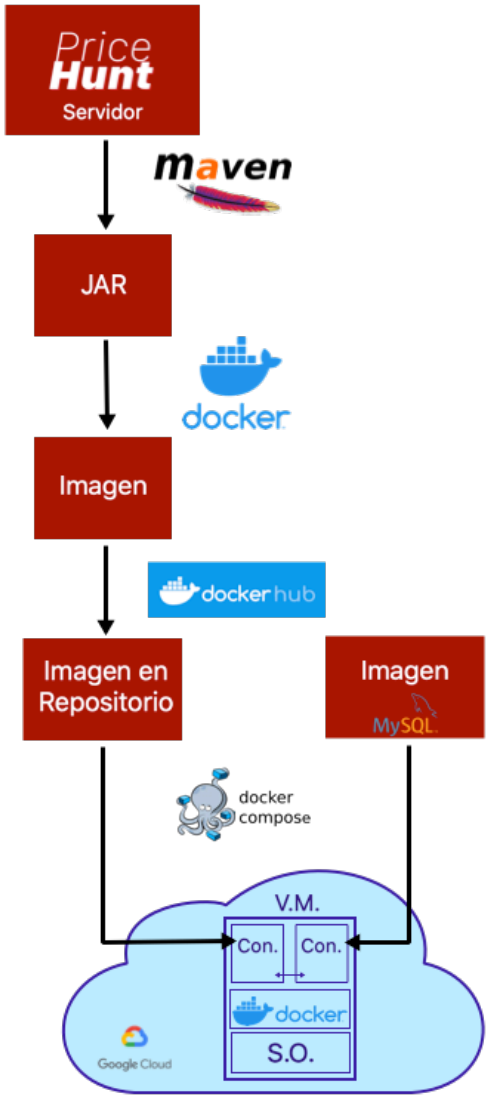

Figura 64. Diagrama de despliegue en la nube

Como inicio, se hizo uso del plug-in de Maven para Spring Boot para construir el paquete JAR de la aplicación servidor de Price Hunt. Este plug-in nos permite empaquetar nuestra aplicación de forma muy sencilla con escasa configuración en el fichero pom.xml (42).

A continuación, utilizando Docker, se construye una imagen de la aplicación. Es importante que descarguemos e instalemos PhantomJS en la imagen y que el path donde se encuentre esté correctamente configurado en el driver de nuestra aplicación. Además, debe exponerse el puerto que utilice el servidor para que sea accesible desde fuera del contenedor (43).

En este punto, la imagen de la aplicación ya podría correrse en un contenedor, pero para facilitar la instalación posterior en la nube, se subió a un repositorio público de Docker-Hub. El único requisito es una cuenta gratuita (44).

Se ha considerado oportuno no introducir la base de datos dentro del mismo contenedor, sino utilizar la imagen gratuita oficial proporcionada en Docker-Hub y desplegar una aplicación multi-contenedor (45).

Con las imágenes preparadas, el siguiente paso es preparar la infraestructura en la nube. Es necesario tener activa la facturación de la cuenta en la consola de Google Cloud Platform para poder utilizarla. En este trabajo se ha activado la prueba gratuita de 300$.

Dentro de la plataforma, se ha utilizado Compute Engine, el módulo de Infrastructure-as-a-Service de GCP. A la hora de crear la máquina virtual es importante escoger el sistema operativo optimizado para contenedores, que viene con Docker preinstalado.

Teniendo las imágenes y la infraestructura preparadas, se utiliza Docker Compose para desplegar la aplicación multi-contenedor y comunicar ambos contenedores. Partiendo de este punto, levantar la aplicación en la máquina virtual de Google se simplifica a dos comandos (46).

Una opción que se consideró fue utilizar el servicio SQL de Google Storage para la base de datos MySQL. Finalmente se escogió utilizar Docker Compose y la imagen oficial, ya que simplificaba el despliegue y la configuración de la comunicación entre las partes.

5. RESULTADO Y DISCUSIÓN

En este capítulo se exponen los resultados obtenidos con este trabajo y se hace un pequeño análisis de la consecución de objetivos respecto a las dos funcionalidades más significativas del mismo: la extracción de información y la predicción de precios.

5.1. RESULTADO DEL TFG

El resultado de este TFG ha sido un servicio plenamente funcional. El usuario puede seguir el precio de diferentes productos de tiendas online que seleccione, tal y como se muestra en la Figura 21. De cada producto, es posible visualizar gran parte de la información disponible en la web, un histórico de los precios recolectados por el servicio y una estimación de la tendencia general de los mismos, lo cual se puede observar en las Figura 24, Figura 26 y Figura 27. También dispone de la posibilidad de configurar notificaciones por correo electrónico, como las de las Figura 35 y Figura 36, mediante el formulario de configuración de la Figura 28.

Este servicio se despliega mediante una aplicación servidor y una aplicación cliente. Gracias a la estructura del mismo, queda abierta la posibilidad de desarrollar aplicaciones cliente para otras plataformas, como se muestra en el diagrama de la Figura 1.

El desarrollo de la aplicación servidor como una imagen disponible en un repositorio público permite la implementación de la misma en infraestructuras en la nube, como Google Cloud Platform. A demás, este desarrollo permitiría el despliegue de la aplicación en cualquier servidor con Docker, haciendo la aplicación servidor portable a cualquier sistema operativo.

La aplicación cliente ha sido desarrollada para la plataforma Android. El principal objetivo de la misma es que, sin perder de vista la funcionalidad, la estética tuviera un papel fundamental en el desarrollo de la misma. Como se aprecia en las figuras de la sección 4.2.6 DISEÑO DE LA INTERFAZ GRÁFICA DE LA APLICACIÓN, es una aplicación intuitiva, sencilla y con una interfaz de usuario amigable que hace la experiencia de uso agradable.

En definitiva, este tipo de servicio sufre el conocido como “efecto red”: el valor del servicio se incrementa para todos los usuarios en tanto aumente el número de usuarios. Con un número elevado de usuarios, se obtienen grandes cantidades de información sobre una amplia variedad de productos. Lo que proporciona más información ya disponible para futuros usuarios y más datos con los que trabajar en el análisis de los precios.

El servicio desarrollado en este TFG proporciona una base muy completa para empezar a trabajar con los productos y el análisis de sus precios sufriendo sin tener usuarios.

5.2. PRECISIÓN DEL SCRAPING WEB

La extracción de información de páginas web es una técnica muy utilizada por muchas empresas. Sin embargo, es una técnica generalmente compleja, y de largo desarrollo si el sitio web no tiene una estructura bien definida. Es preferible el uso de APIs, en caso de que el sitio web lo proporcione, ya que supone un desarrollo menos complejo.

Aunque algunos sitios utilizan bots para aumentar el número de visitas, otros intentan bloquear este tipo de técnicas por la sobrecarga que supone a sus servidores. Además, esta técnica puede producir el deterioro de la calidad de algunos algoritmos que utilizan las plataformas basados en datos de visitas. Por estos motivos, las páginas intentan dificultar el empleo de estas técnicas utilizando captchas, modificando las estructuras de sus páginas web cada cierto tiempo, con páginas con estructuras muy diferentes (como en el caso de foros o webs con introducción de información por usuarios).

Todo esto hace que desarrollar un crawler para un determinado sitio web sea una tarea larga e iterativa. Es imposible estudiar la estructura de todas las páginas de un sitio web, por lo que se realiza una versión inicial y se va reajustando y ampliando conforme se va usando.

Para el desarrollo de este servicio, se ha realizado un estudio exhaustivo de las páginas de Amazon y eBay. Durante la etapa de desarrollo, se han introducido en el sistema productos de diferentes categorías de ambas páginas para comprobar el correcto funcionamiento del crawler y la información obtenida. Se ha repetido el proceso diariamente durante varios meses y observado la información obtenida. Con ello se han obtenido las siguientes observaciones:

- Incluso sitios webs como los mencionados que tienen estructuras bien definidas tienen una gran variabilidad entre las diferentes páginas.
- Las estructuras de las mismas páginas han sufrido variaciones a lo largo de este tiempo, necesitando reajustar o ampliar los selectores CSS de extracción de la información.
- Obtener la información de la página no significa que la información sea correcta o esté correctamente extraída.

A raíz de estas observaciones se desarrolló el servicio de error de la aplicación. Este servicio permite a la aplicación almacenar automáticamente errores de parseo de la información que resulten críticos (como que no se pueda extraer el precio) y permite a los usuarios realizar reportes si la información que visualizan no es correcta. Esta herramienta resulta de gran utilidad para el gestor de la aplicación para detectar los errores y corregirlos, aumentando así la precisión del scraping.

5.3. ESTIMACIÓN Y PREDICCIÓN DE PRECIOS

Durante el tiempo de desarrollo del servicio se ha estado realizando un seguimiento de los precios de diversos productos y observando su variación. En el alcance de este trabajo no se encontraba la obtención de datos masivos para su análisis y las muestras obtenidas son ínfimas en relación con el total de productos existentes.

Este estudio sobre los precios de los productos ha dejado las siguientes conclusiones:

- Algunos productos no sufren variación de precios
- Con los datos obtenidos durante este periodo no se ve una relación de las variaciones con los datos disponibles (la fecha de los mismos).
- A pesar de que las variaciones no siguen un patrón definido (al menos no es observable con los datos disponibles), se observa que los productos que sufren variación en su precio siguen una tendencia general a lo largo del tiempo, ya sea a aumentar o disminuir.

Con estas observaciones y teniendo en cuenta que la muestra obtenida es muy pequeña, se consideró que una regresión lineal era la mejor forma de, con los datos disponibles, extraer una conclusión sobre los precios.

Como podemos observar en la Figura 63, algunos productos tienen precios muy estables, tal y como se extraía en la primera conclusión, por lo que se puede deducir que van a continuar en la misma línea.

Sin embargo, la Figura 61 y la Figura 62 están relacionadas con las dos últimas conclusiones. Podemos observar que los precios, aunque aparentemente no siguen ningún patrón, tienen una tendencia generalizada a aumentar o disminuir a lo largo del tiempo. Con esta información y observando el histórico, el usuario puede decidir el mejor momento para realizar la compra del artículo si está interesado.

6. CONCLUSIONES

Analizando el trabajo desde un punto de vista funcional, el servicio ha demostrado su utilidad durante el periodo de desarrollo. La autora ha podido aprovechar las funcionalidades del mismo para comparar precios de determinados productos y adquirirlos a un precio menor del que hubiera conseguido sin este servicio.

Desde un punto de vista técnico, se ha desarrollado este trabajo haciendo uso de múltiples tecnologías, intentando aprovechar las herramientas libres y las pruebas gratuitas disponibles en Internet. La mayoría de las herramientas utilizadas en este TFG son tecnologías que están siendo utilizadas actualmente por empresas reales en el desarrollo de sus productos.

Actualmente, cada vez más servicios se están mudando a la nube. Este trabajo y la prueba gratuita que ofrece Google, ha permitido conocer parte de su plataforma Cloud (Google Cloud Platform), la cual se encuentra entre las plataformas más utilizadas.

Además, el uso de Google Sign In ha permitido entrar en contacto con una de las muchas herramientas que Google pone a disposición de los desarrolladores y que facilita la integración de las aplicaciones con muchas funcionalidades que pueden suponer un factor diferenciador en la misma.

También merece una mención el uso de JPA e Hibernate en la integración del servicio con la base de datos. Estas herramientas han reducido significativamente el tiempo y el esfuerzo del desarrollo de esta parte. Han hecho posible el desarrollo completo de la base de datos sin utilizar ninguna línea de lenguaje SQL en el proceso.

La mayor parte del esfuerzo de desarrollo de este servicio ha sido destinada a realizar el scraping web, ya que suponía la base para el resto del trabajo. En conjunto con esto, y siendo conscientes de que el proceso de scraping necesita un esfuerzo continuo, se han implementado herramientas para facilitar la detección de errores (tanto automáticas como por reportes de usuarios) y proporcionar así al gestor del sistema medios para continuar con esta labor.

Cabe destacar la intención puesta en el desarrollo de la interfaz gráfica del servicio, tanto de la aplicación Android como de los correos electrónicos. Uno de los requisitos era implementar una interfaz atractiva, funcional y amigable. Para la consecución de dicho objetivo se ha utilizado una gama de colores combinables para toda la interfaz y se han diseñado logos e iconos que identifiquen el servicio. También con esa intención, se ha pretendido que la interfaz fuera lo más intuitiva posible, utilizando gestos conocidos en sistemas móviles como el arrastrar para eliminar un producto de la lista. Se ha conseguido una experiencia de usuario sencilla y estética que cumple con lo propuesto.

Finalmente, se ha perseguido que la redacción de esta Memoria sea fiel al trabajo realizado y lo más completa posible. Se han empleado diversos diagramas, imágenes y tablas para facilitar la comprensión de lo expuesto.

7. LÍNEAS FUTURAS

Aunque este TFG cumple con todos los requisitos y objetivos establecidos en un inicio, se han detectado posibles de trabajo líneas futuras tanto en relación con las funcionalidades del servicio como con la implementación del mismo:

- Implementación del inicio de sesión a través de terceros por otras compañías como Apple o Facebook.
- Desarrollar el crawler para poder hacer scraping de más sitios web de venta online (AliExpress, PcComponentes, Zara.es, etc.). Introducir páginas en otros idiomas y otras divisas, lo que implicaría la traducción de la aplicación Android.
- Implementar un buscador de productos dentro del mismo servicio para que un usuario pueda visualizar la información de un producto, aunque no lo siga.
- Implementar alguna técnica basada en análisis de cesta de la compra para sugerir a usuarios que visualicen productos que podrían interesarles a partir de los productos que están en la lista de clientes que siguen ese producto.
- Implementar alguna herramienta visual (por ejemplo, web) que permita al gestor acceder a los informes de errores y arrancar el proceso batch de actualización de productos de una forma más cómoda.
- Una vez el servicio estuviera en funcionamiento real y aumentara el número de usuarios, y por tanto los datos recopilados, se debería estudiar un modelo de machine learning que se ajustara mejor a los datos para realizar una predicción más fiel de los precios. Quizá sería interesante estudiar la relación entre la categoría de los artículos y la variación de sus precios a lo largo del tiempo.
- En la misma línea del punto anterior, con el aumento del volumen de datos podría migrarse la base de datos a un sistema de almacenamiento de datos enfocado a BigData como mongoDB.
- Siguiendo con esa casuística, si las necesidades de nuestra aplicación fueran mucho mayores en términos de tráfico, interesaría desplegar la imagen de la aplicación haciendo uso de un orquestador de contenedores más potente como puede ser Kubernetes.

8. BIBLIOGRAFÍA

1. **Comisión Nacional de los Mercados y la Competencia (CNMT).** *Nota de prensa Comercio electrónico 3T.* 2019.

2. **Mobile Operating System Market Share Worldwide.** *Statcounter globalstats.* [En línea] Mayo de 2020. [Citado el: 1 de Junio de 2020.] **https://gs.statcounter.com/os-market-share/mobile/worldwide.**

3. Mobile Operating System Market Share in Spain. *Statcounter globalstats.* [En línea] Mayo de 2020. [Citado el: 1 de Junio de 2020.] **https://gs.statcounter.com/os-market-share/mobile/spain.**

4. www.espiaprecios.com. Seguimiento de precios en Amazon. *Chrome web store.* [En línea] [Citado el: 12 de Junio de 2020.] **https://chrome.google.com/webstore/detail/seguimiento-de-precios-en/phinfcfncdeelldijhpcpoaibedhdbdd?hl=es.**

5. camelcamelcamel. *camelcamelcamel.* [En línea] [Citado el: 12 de Junio de 2020.] **https://es.camelcamelcamel.com.**

6. Technotic Solutions. Price Tracker for Amazon. *Google Play.* [En línea] [Citado el: 12 de Junio de 2020.] **https://play.google.com/store/apps/details?id=com.bomba.jcrocker.myapplication&hl=es.**

7. Kyakilika, Buneme. Fluctuate. *Google Play.* [En línea] [Citado el: 12 de Junio de 2020.] **https://play.google.com/store/apps/details?id=com.buneme.fluctuate&hl=es.**

8. idealo internet GmbH. Idealo. *Google Play.* [En línea] [Citado el: 12 de Junio de 2020.] **https://play.google.com/store/apps/details?id=de.idealo.android&hl=es.**

9. Spring Framework. *Spring.* [En línea] [Citado el: 2 de Junio de 2020.] **https://spring.io/projects/spring-framework#overview.**

10. Spring Initializr. *Spring.* [En línea] [Citado el: 2 de Junio de 2020.] **https://start.spring.io.**

11. Spring Boot. *Spring.* [En línea] [Citado el: 2 de Junio de 2020.] **https://spring.io/projects/spring-boot#overview.**

12. BBVAOPEN4U. API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos. *BBVA API_Market.* [En línea] 23 de Marzo de 2016. [Citado el: 2 de Junio de 2020.] **https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos.**

13. Lafuente, Ainhoa. Qué es el web scraping. *El Blog de Aureka.* [En línea] 10 de Enero de 2019. [Citado el: 9 de Junio de 2020.] https://aukera.es/blog/web-scraping/.

14. Scriptable Headless Browser. *Phantom JS.* [En línea] [Citado el: 2 de Junio de 2020.] **https://phantomjs.org.**
15. codeborne. ghostdriver. *GitHub.* [En línea] [Citado el: 2 de Junio de 2020.] **https://github.com/codeborne/ghostdriver.**
16. The Selenium Browser Automation Project. *Selenium.* [En línea] [Citado el: 2 de Junio de 2020.] **https://www.selenium.dev/documentation/en/.**
17. Java HTML Parser. *Jsoup.* [En línea] [Citado el: 2 de Junio de 2020.] **https://jsoup.org.**
18. Gelbmann, Matthias y Andlinger, Paul. MySQL is the DBMS of the Year 2019. *db-engines.* [En línea] 3 de Enero de 2020. [Citado el: 2 de Junio de 2020.] **https://db-engines.com/en/blog_post/83.**
19. Reference documentation. *Spring.* [En línea] [Citado el: 2 de Junio de 2020.] **https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#reference.**
20. Spring Batch. *Spring.* [En línea] [Citado el: 2 de Junio de 2020.] **https://spring.io/projects/spring-batch.**
21. Tareas con spring scheduler. *Windoctor7.* [En línea] 7 de Abril de 2017. [Citado el: 2 de Junio de 2020.] **https://windoctor7.github.io/Tareas-con-Spring-Scheduler.html.**
22. Commons Math: The Apache Commons Mathematics Library. *Apache Commons.* [En línea] [Citado el: 9 de Junio de 2020.] **http://commons.apache.org/proper/commons-math/.**
23. Integrating Google Sign-In into Your Android App. *Google Developers.* [En línea] [Citado el: 9 de Junio de 2020.] **https://developers.google.com/identity/sign-in/android/sign-in?authuser=2..**
24. PhilJay. MPAndroidChart. *GitHub.* [En línea] [Citado el: 4 de Junio de 2020.] **https://github.com/PhilJay/MPAndroidChart.**
25. danielgindi. Charts. *GitHub.* [En línea] [Citado el: 4 de Junio de 2020.] **https://github.com/danielgindi/Charts.**
26. About Glide. *Glide v4.* [En línea] [Citado el: 4 de Junio de 2020.] **https://bumptech.github.io/glide/.**
27. Garzas, Javier. ¿Qué es Docker? ¿Para qué se utiliza? Explicado de forma sencilla. *javiergarzas.com.* [En línea] 24 de Julio de 2015. [Citado el: 4 de Junio de 2020.] **https://www.javiergarzas.com/2015/07/que-es-docker-sencillo.html.**
28. Docker Hub Quickstart. *Docker Docs.* [En línea] [Citado el: 4 de Junio de 2020.] **https://docs.docker.com/docker-hub/.**
29. Overview of Docker Compose. *Docker Docs.* [En línea] [Citado el: 4 de Junio de 2020.] **https://docs.docker.com/compose/.**

30. Using the @SpringBootApplication Annotation. *Spring Docs.* [En línea] [Citado el: 7 de Junio de 2020.] **https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-using-springbootapplication-annotation.html.**
31. Enabling Server-Side Access. *Google Developers.* [En línea] [Citado el: 7 de Junio de 2020.] **https://developers.google.com/identity/sign-in/android/offline-access?authuser=2.**
32. ISO 4217. *Wikipedia.* [En línea] [Citado el: 7 de Julio de 2020.] **https://es.wikipedia.org/wiki/ISO_4217.**
33. ViewPager2. *Android Developer.* [En línea] [Citado el: 8 de Julio de 2020.] **https://developer.android.com/jetpack/androidx/releases/viewpager2.**
34. tommybuonomo. dotsindicator. *GitHub.* [En línea] [Citado el: 8 de Julio de 2020.] **https://github.com/tommybuonomo/dotsindicator.**
35. ProgressBarr. *Android Developer.* [En línea] [Citado el: 8 de Junio de 2020.] **https://developer.android.com/reference/android/widget/ProgressBar.**
36. Cómo crear una lista con RecyclerView. *Android Developer.* [En línea] [Citado el: 10 de Junio de 2020.] **https://developer.android.com/guide/topics/ui/layout/recyclerview.**
37. Tamada, Ravi. Android adding RecyclerView Swipe to Delete and Undo. *Androidhive.* [En línea] 2 de Octubre de 2017. [Citado el: 10 de Junio de 2020.] **https://www.androidhive.info/2017/09/android-recyclerview-swipe-delete-undo-using-itemtouchhelper/.**
38. Start Integrating Google Sign-In into Your Android App. *Google Developers.* [En línea] [Citado el: 8 de Junio de 2020.] **https://developers.google.com/identity/sign-in/android/start-integrating?authuser=2.**
39. Authenticate with a backend server. *Google Developers.* [En línea] [Citado el: 9 de Junio de 2020.] **https://developers.google.com/identity/sign-in/android/backend-auth?authuser=2.**
40. Signing Out Users and Disconnecting Accounts. *Google Developers.* [En línea] [Citado el: 9 de Junio de 2020.] **https://developers.google.com/identity/sign-in/android/disconnect?authuser=2.**
41. Rodríguez, Miguel Arlandy. Introducción a Spring Batch. *Adictos al trabajo.* [En línea] 17 de Julio de 2013. [Citado el: 9 de Junio de 2020.] **https://www.adictosaltrabajo.com/2013/07/17/introduccion-spring-batch/.**
42. Stephane Nicoll, Andy Wilkinson, Scott Frederick. Spring Boot Maven Plugin Documentation. ***Spring Documents.*** **[En línea] [Citado el: 10 de Junio de 2020.]**

https://docs.spring.io/spring-boot/docs/2.3.0.RELEASE/maven-plugin/reference/html/.

43. Spring Boot with Docker. *Spring.* [En línea] [Citado el: 10 de Junio de 2020.] **https://spring.io/guides/gs/spring-boot-docker/.**

44. Torres, Gisela. Publicar tu imagen en Docker Hub. *Returngis.* [En línea] 12 de Febrero de 2019. [Citado el: 10 de Junio de 2020.] **https://www.returngis.net/2019/02/publicar-tu-imagen-en-docker-hub/.**

45. MySql docker official images. *Docker Hub.* [En línea] [Citado el: 10 de Junio de 2020.] **https://hub.docker.com/_/mysql.**

46. @tswast. Running Docker Compose with Docker. *Google Cloud Community.* [En línea] 3 de Mayo de 2017. [Citado el: 10 de Junio de 2020.] **https://cloud.google.com/community/tutorials/docker-compose-on-container-optimized-os.**

47. Spring Boot Maven Plugin. *Spring Docs.* [En línea] 6 de Noviembre de 2019. [Citado el: 11 de Junio de 2020.] **https://docs.spring.io/spring-boot/docs/2.2.1.RELEASE/maven-plugin//repackage-mojo.html.**

48. Dockerfile reference. *Docker docs.* [En línea] [Citado el: 11 de Junio de 2020.] **https://docs.docker.com/engine/reference/builder/#entrypoint.**

49. Compose file version 3 reference. *Docker Docs.* [En línea] [Citado el: 12 de Junio de 2020.] **https://docs.docker.com/compose/compose-file/.**

50. Explicación del protocolo OAuth 2. *Programación y más.* [En línea] [Citado el: 12 de Junio de 2020.] **https://programacionymas.com/blog/protocolo-oauth-2.**

9. ANEXOS

En este capítulo se añaden una serie de anexos que explican los pasos a realizar para la instalación de los softwares desarrollados en este trabajo.

9.1. ANEXO I. MANUAL DE INSTALACIÓN DE LA APLICACIÓN SERVIDOR

En este anexo se detallan los diferentes pasos seguidos para la instalación de la aplicación servidor en la nube de Google. El diagrama de la Figura 64 es un claro resumen de los pasos a seguir en este manual.

También es posible desplegar la aplicación en cualquier servidor o equipo con Docker instalado realizando los mismos pasos excepto el apartado 9.1.6.

9.1.1. SOFTWARE EMPLEADO

A continuación, se indican las versiones de los softwares utilizados durante este manual:

- Sistema operativo del equipo: macOS Catalina 10.15.4
- Spring Boot: 2.2.6. RELEASE
- Java: 1.8
- Docker (para construcción de imagen): 19.03.8
- Docker (instancia VM): 17.03.2-ce
- Docker Compose: 1.26.0

9.1.2. REQUISITOS PREVIOS

1. Se recomienda revisar la configuración de la conexión a la base de datos y del path de PhantomJS, las cuales deben de ser las siguientes para seguir la instalación de este manual.
    - Para la conexión con la base de datos, en el **application.properties** que se encuentra la carpeta /src/main/resources del módulo core:

```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url = jdbc:mysql://mysqlserver:3306/PriceHunt?allowPublicKeyR
spring.datasource.username = pricehuntusername
spring.datasource.password = PriceHunt_Password
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
```

Figura 65. Configuración de la conexión a la base de datos

```
* spring.datasource.url =
jdbc:mysql://mysqlserver:3306/PriceHunt?allowPublicKeyRetrie
val=true&useSSL=false&serverTimezone=UTC&useLegacyDatetimeCo
de=false;characterEncoding=UTF-8MB4
```

- Para indicar al driver donde se encuentra el ejecutable de PhantomJS, en la clase **PhantomService** del paquete 4.2.4.3.5 Epsl.telematica.tfg.scraping.phantom :

```
private static final String PHANTOM_PATH = "/usr/local/bin/phantomjs";
```

Figura 66. Configuración del path del ejecutable de PhantomJS

2. Se considera necesario tener una cuenta en **Docker** para poder subir la imagen a un repositorio en Docker Hub.
3. Se considera necesario tener una cuenta de Google con la facturación de **Google Cloud Platform** activa, ya sea por la prueba gratuita o un método de pago real.

9.1.3. CONSTRUCCIÓN DEL EJECUTABLE DE NUESTRA APLICACIÓN

Debemos introducir el plug-in de maven en el fichero pom.xml del módulo core como se muestra en la siguiente figura:

```
<!-- ************************************************************
<!-- BUILD PROJECT -->
<build>
<finalName>PriceHunt</finalName>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <id>repackage</id>
                    <goals>
                        <goal>repackage</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Figura 67. Inserción del plug-in de maven para Spring Boot

Esta configuración va a crear un paquete con el nombre PriceHunt para poder ser ejecutado desde la línea de comandos (47).

A continuación, desde el entorno de programación Eclipse, se crea una configuración de ejecución maven como la de la siguiente imagen. Se selecciona el proyecto PriceHunt_Server (el proyecto que acoge los diferentes módulos) como directorio base y se introduce “package” en “Goals”.

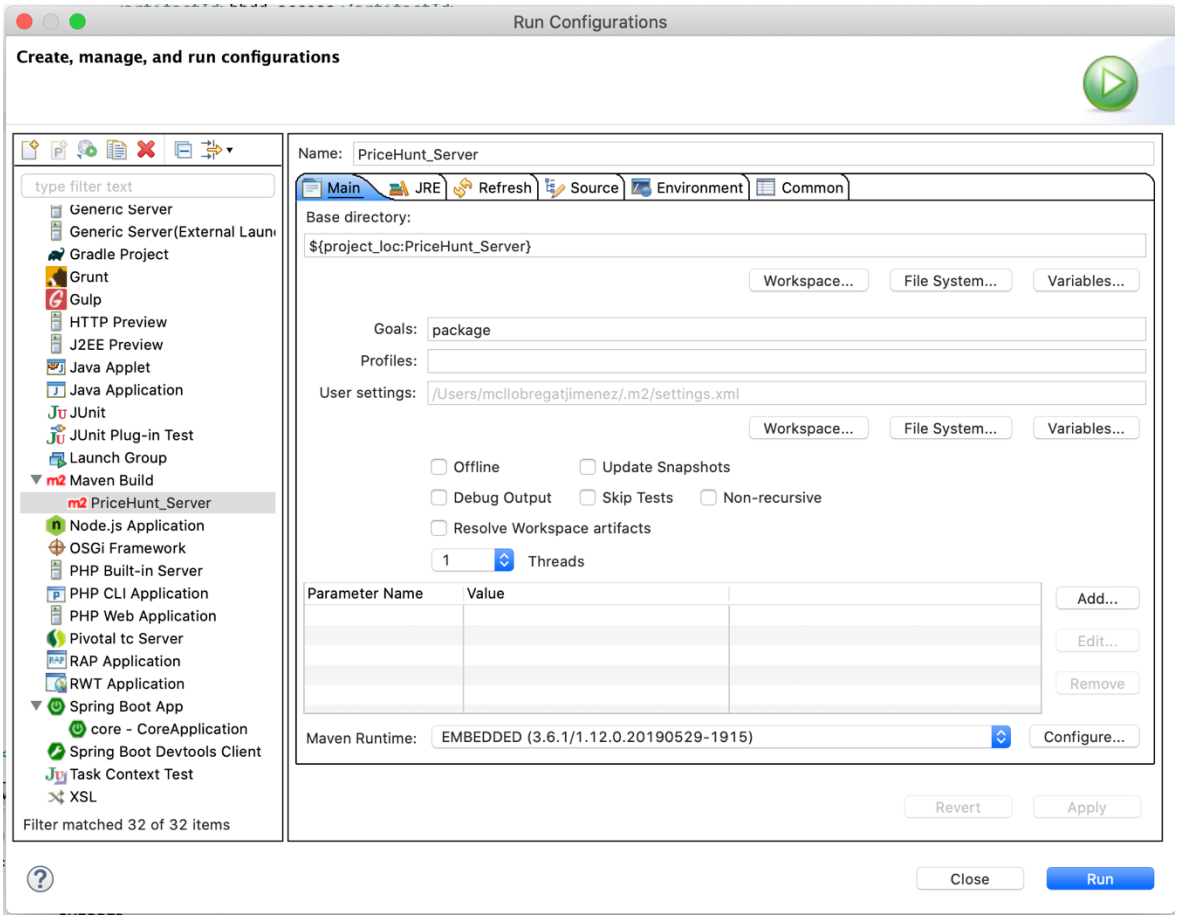

Figura 68. Creación de la configuración de ejecución maven

Finalmente, se ejecuta dicha configuración. Por consola se puede comprobar que el paquete se ha creado correctamente.

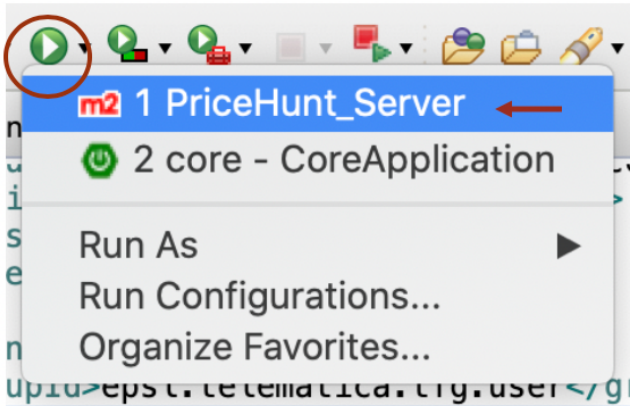

Figura 69. Ejecución de maven

```
[INFO] --- spring-boot-maven-plugin:2.1.2.RELEASE:repackage (repackage) @ core ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for PriceHunt 1.0.0:
[INFO]
[INFO] PriceHunt ........................................... SUCCESS [  0.003 s]
[INFO] bbdd_access ......................................... SUCCESS [  0.997 s]
[INFO] user ................................................ SUCCESS [  0.196 s]
[INFO] scraping ............................................ SUCCESS [  0.237 s]
[INFO] core ................................................ SUCCESS [  1.278 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.931 s
[INFO] Finished at: 2020-06-11T19:59:03+02:00
[INFO] ------------------------------------------------------------------------
```

Figura 70. Mensaje por consola tras construcción correcta del paquete JAR

9.1.4. CREACIÓN DE LA IMAGEN

Para la creación de la imagen con Docker, debemos situarnos en el directorio principal del proyecto, la carpeta PriceHunt_Server, y crear un fichero Dockerfile como el especificado en el apartado 9.2.1 del ANEXO II. DOCUMENTOS DE CONFIGURACIÓN DE DOCKER.

Desde el terminal, ejecutamos el siguiente comando para ejecutar Docker y construir la imagen:

```
docker build -t <nombre de usuario>/price_hunt:1.0 <path del fichero Dockerfile>
```

(43)

Este comando crea una imagen etiquetándola con el nombre <nombre de usuario de Docker Hub>/Price_hunt:1.0 a partir del archivo Dockerfile indicado.

9.1.5. SUBIR LA IMAGEN A UN REPOSITORIO EN DOCKER HUB

Para subir la imagen creada a un repositorio público en Docker Hub es necesario tener creada una cuenta en la plataforma. Desde el terminal, iniciamos sesión en Docker con la cuenta creada introduciendo el siguiente comando e ingresando las credenciales cuando sea requerido.

```
docker login
```

A continuación, se sube la imagen introduciendo el siguiente comando:

```
docker push <nombre de usuario>/Price_hunt:1.0
```

Tras unos minutos, la imagen queda subida al repositorio y es accesible desde cualquier dispositivo. En la siguiente figura se muestra la imagen que va a ser utilizada en adelante en este tutorial desde la web de Docker Hub.

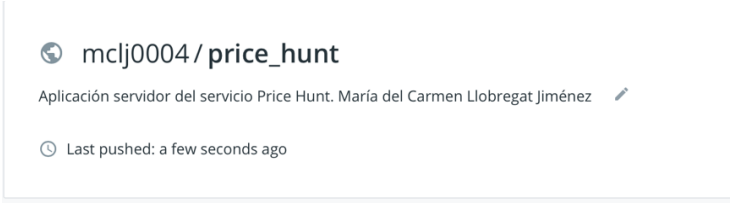

Figura 71. Imagen de Price Hunt subida a Docker Hub

9.1.6. CREAR UNA MÁQUINA VIRTUAL EN GOOGLE ENGINE

Una vez se tenga una cuenta Google con la facturación activa en GCP, entramos en la consola de la plataforma y buscamos en la barra lateral izquierda el menú Compute Engine > Instancias de VM. Seleccionamos “crear” en pantalla que se nos abre.

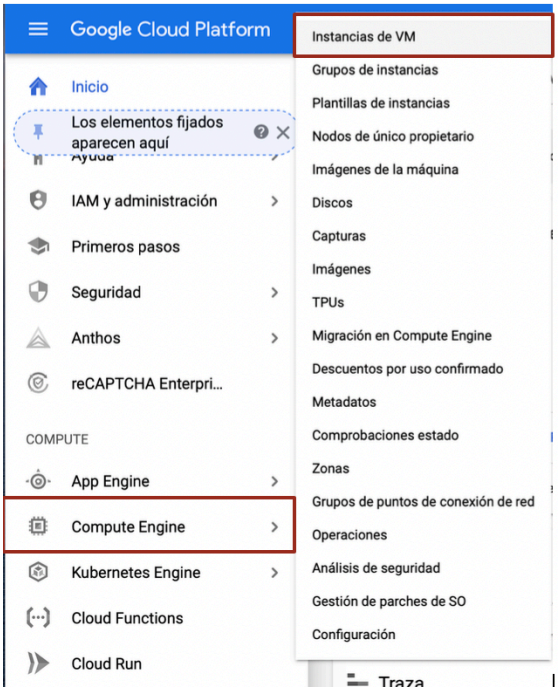

Figura 72. Compute Engine en la consola de Google Cloud Platform

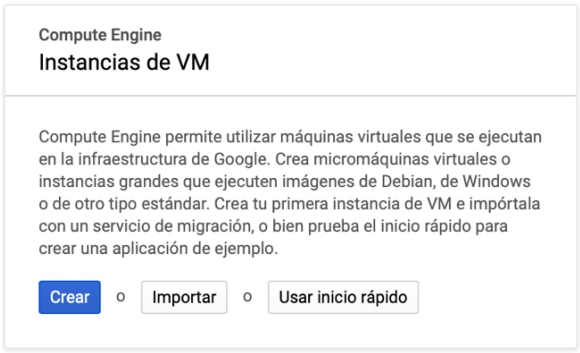

Figura 73. GCP Console, crear VM

A continuación, se debe introducir la configuración de la máquina virtual (VM). Se debe seleccionar un nombre para la instancia, la región y la zona. Estos parámetros no serán editables a posteriori.

También hay que seleccionar la configuración de la máquina, que dependerá de las necesidades del servicio. Para este trabajo se ha dejado la configuración por defecto que se muestra en la siguiente imagen.

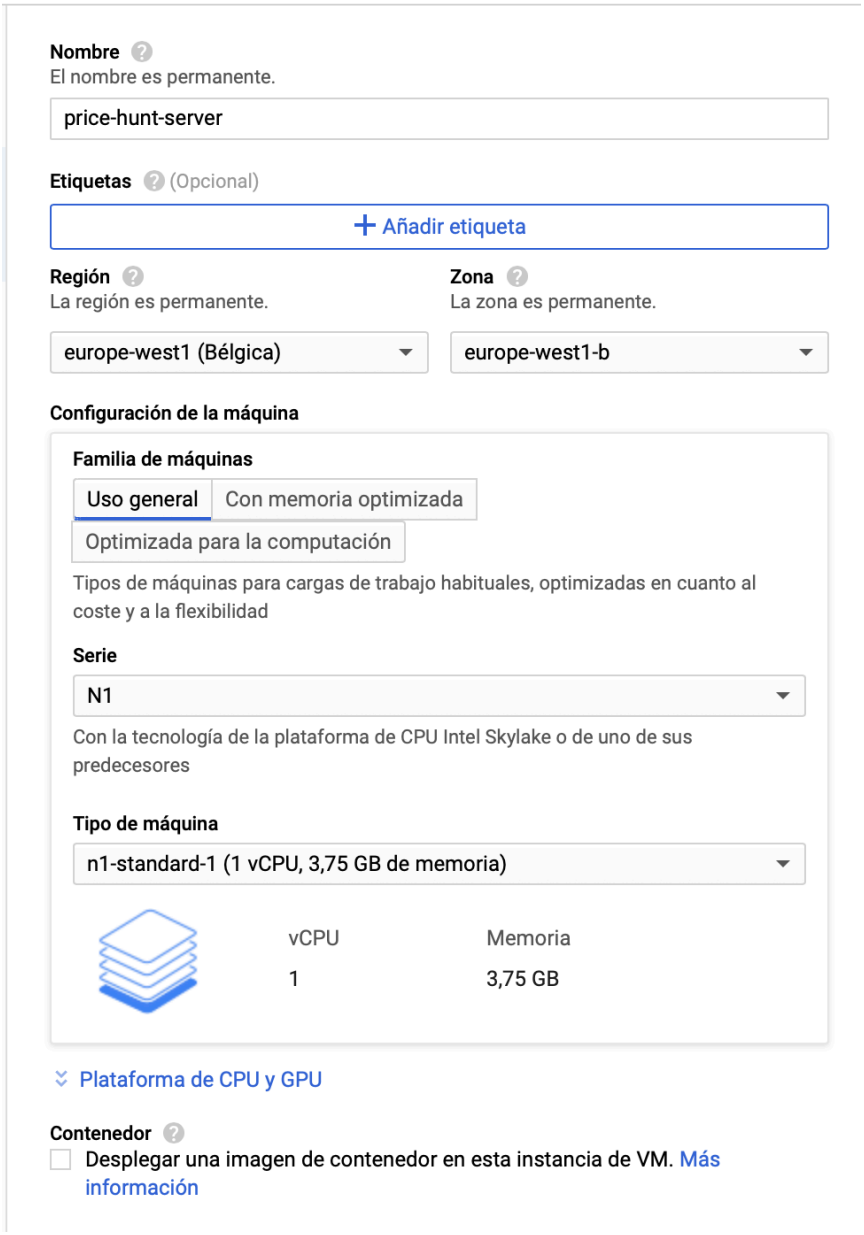

Figura 74. GCP, Configuración VM 1

Para el disco de arranque se selecciona el sistema operativo optimizado para contenedores. Este sistema operativo viene con Docker y otras herramientas preinstaladas.

Es importante seleccionar las casillas que habilitan el tráfico HTTP. Estas casillas añaden reglas al cortafuegos de la máquina para permitir dicho tráfico.

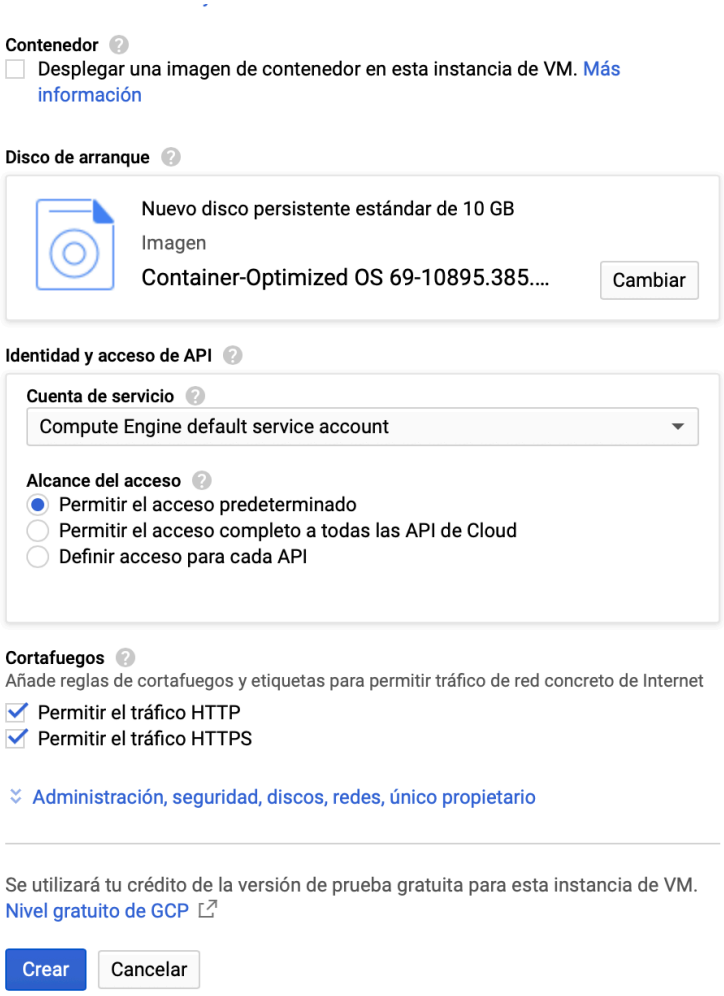

Figura 75. GCP, Configuración VM 2

En la parte derecha de la ventana, Google nos muestra una estimación del coste de la instancia con los parámetros que se han seleccionado.

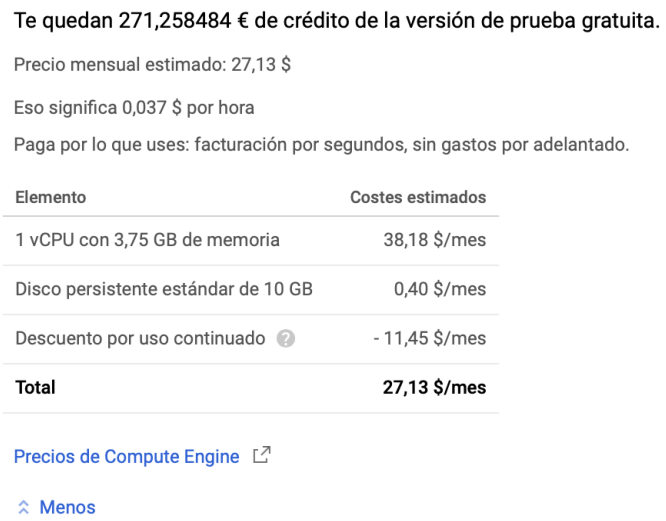

Figura 76. GCP, estimación precio VM

6.1.7. DESPLIEGE DE LA APLICACIÓN

Una vez creada la máquina virtual, nos conectamos a ella mediante SSH.

Figura 77. GCP, Instancia de VM

Desde la parte superior derecha de la ventana que se nos abre, seleccionamos el menú de configuración y subir archivo, como se muestra en la Figura 78. A continuación, debemos subir a la máquina virtual el archivo **docker-compose.yml** que se encuentra en la sección 9.2.2 del ANEXO II. DOCUMENTOS DE CONFIGURACIÓN DE DOCKER.

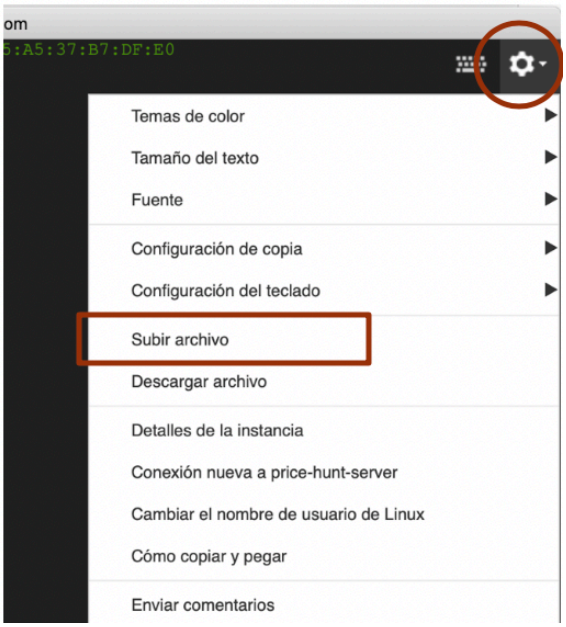

Figura 78. GCP, SSH, Subir archivo

Aunque la VM viene con Docker instalado, para el despliegue de nuestra aplicación multicontenedor necesitamos instalar Docker Compose con el siguiente comando:

```
docker run docker/compose:latest
```

Una vez instalado, se utiliza el siguiente comando para levantar la aplicación:

```
Docker run –rm \
        -v /var/run/docker.sock:/var/run/docker.sock \
        -v “$PWD:$PWD” \
        -w=”$PWD” \
        docker\compose:latest up
```

- -v /var/run/docker.sock:/var/run/docker.sock permite al contenedor de Docker Compose tener acceso al servicio Docker.
- -v "$PWD:$PWD" permite que el directorio actual sea accesible al contenedor
- -w="$PWD" cambia el directorio de trabajo al directorio actual

Ya tendríamos la aplicación corriendo en la máquina virtual.

Si queremos acceder a la base de datos MySQL desde fuera de la máquina virtual, para conectarnos con Workbench, por ejemplo, debemos añadir una nueva regla al **cortafuegos** que permita dicho tráfico.

9.2. ANEXO II. DOCUMENTOS DE CONFIGURACIÓN DE DOCKER

9.2.1. DOCKERFILE

9.2.1.1. Contenido

```
FROM openjdk:8

RUN apt-get update

ARG JAR_FILE=core/target/PriceHunt.jar
ARG PROXY_LIST_FILE=core/proxyList.txt
COPY ${JAR_FILE} app.jar
COPY ${PROXY_LIST_FILE} proxyList.txt
EXPOSE 8080

ENV PHANTOMJS_VERSION 2.1.1
RUN wget -q -O phantomjs.tar.bz2 https://bitbucket.org/ariya/phantomjs/downloads/phantomjs-$PHANTOMJS_VERSION-linux-x86_64.tar.bz2 \
    && tar -f phantomjs.tar.bz2 -xj phantomjs-$PHANTOMJS_VERSION-linux-x86_64/bin/phantomjs \
    && mv phantomjs-$PHANTOMJS_VERSION-linux-x86_64/bin/phantomjs /usr/local/bin/ \
    && rm -rf phantomjs-$PHANTOMJS_VERSION-linux-x86_64 phantomjs.tar.bz2

RUN cat /dev/null > /etc/ssl/openssl.cnf

ENTRYPOINT ["java","-jar","/app.jar"]
```

Figura 79. Configuración Dockerfile

9.2.1.2. Análisis del contenido

El fichero Dockerfile contiene las instrucciones para la creación de una imagen y el despliegue del contenedor de dicha imagen. El fichero utilizado para este trabajo contiene las siguientes instrucciones: (48)

1. Utilizar la openJDK versión 8 como imagen de partida para la construcción de esta imagen.
2. Actualiza los paquetes disponibles y sus versiones.
3. Crea los argumentos JAR_FILE y PROXY_LIST_FILE con sendos valores por defecto. Estos valores podrán ser sobrescritos desde la línea de comandos al ejecutar el comando de construcción.

4. Copia el fichero JAR como app.jar
5. Copia el fichero proxyList.txt
6. Expón el puerto 8080 (el puerto por defecto de Tomcat)
7. Crea una variable de entorno con la versión de PhantomJS.
8. Descarga, descomprime e instala PhantomJS. Al acabar elimina el fichero comprimido.
9. Vacía el fichero openssl.cnf. Dicho fichero creaba problemas con el ejecutable de PhantomJS.
10. Configura el contenedor como un ejecutable que ejecute el comando “java -jar /app.jar”.

9.2.2. DOCKER-COMPOSE.YML

9.2.2.1. Contenido

```yaml
version: "3"

networks:
  pricehunt_network:

services:

  bbdd:
    image: mysql:8.0
    environment:
      MYSQL_DATABASE: 'PriceHunt'
      MYSQL_USER: 'pricehuntusername'
      MYSQL_PASSWORD: 'PriceHunt_Password'
      MYSQL_ROOT_PASSWORD: 'rootpassword'
    ports:
      - '3306:3306'
    expose:
      - '3306'
    networks:
      pricehunt_network:
        aliases:
          - mysqlserver
    volumes:
      - my-db:/var/lib/mysql

  pricehunt_service:
    image: mclj0004/price_hunt:1.0
    restart: on-failure
    ports:
      - '80:8080'
    networks:
      - pricehunt_network

volumes:
  my-db:

```

Figura 80. Configuración Docker-compose.yml

9.2.2.2. Análisis del contenido

El fichero docker-compose.yml define las redes, servicios y volúmenes que se van a desplegar en la aplicación y los parámetros para hacerlo. El fichero utilizado en este trabajo está definido bajo la versión 3 de formatos de documentos Compose y define los siguientes elementos: (49)

- Redes:
  - **pricehunt_network**:
    - Define una red a la que nombra de esa forma.
- Servicios:
  - **bbdd**:
    - Este servicio va a desplegar la imagen MySQL versión 8.
    - Define unas variables de entorno que indican qué base de datos crear, la contraseña del usuario root y un usuario de acceso con su contraseña.
    - Expone el puerto 3306, el puerto por defecto de MySQL, para que sea accesible desde fuera del contenedor.
    - Mapea el puerto 3306 del contenedor con el mismo puerto de la máquina, de manera que accediendo al puerto 3306 de la máquina virtual estamos conectando con este contenedor.
    - El servicio está conectado a la red anteriormente creada con el alias “mysqlserver”. Esta es la dirección configurada en la aplicación servidor para el acceso a la base de datos, como se ve en el apartado 9.1.2.
    - EL servicio hace uso del volumen “my-db”.
  - **pricehunt_server**:
    - Este servicio va a desplegar la imagen mclj0004/price_hunt:1.0, que es la imagen que hemos creado en el apartado 9.1.4 y subido a Docker Hub en el apartado 9.1.5.
    - El servicio va a reiniciarse en caso de fallo. Esto se debe a que la aplicación necesita la base de datos para iniciarse correctamente y no es posible indicar en esta versión 3 el orden de despliegue de los contenedores.
    - Mapea el puerto 8080 expuesto en la imagen con el puerto 80 de la máquina virtual.
    - El servicio se encuentra conectado a la red “pricehunt_network”
- Volúmenes:

- **my-db**:
  - Se ha creado un volumen para el servicio bbdd.

9.3. ANEXO III. MANUAL DE INSTALACIÓN DE LA APLICACIÓN ANDROID

9.3.1. SOFTWARE EMPLEADO

A continuación, se detallan las versiones del software utilizado durante el desarrollo de este trabajo:

- Android Studio: 3.5
- Android: 6.0 (SDK 23) o superior.

9.3.2. REQUISITOS PREVIOS

Antes de la construcción e instalación de la aplicación debe de haberse realizado estos pasos previos:

1. Tener un certificado de producción en un almacén de claves (o key store) para firmar el APK o el Bundle de la aplicación. Android Studio te ayuda a crearlo en "Build" > "Generated Signed Bundle/APK…" > "next" > "create new…" y rellenando los datos.
2. Revisar la configuración de la dirección del servidor y el identificador OAuth en la clase **CommunicationUtils**.

9.3.3. CONSTRUCCIÓN DE LA APLICACIÓN

Para construir la aplicación, desde Android Studio seleccionamos "Build" > "Generated Signed Bundle/APK…".

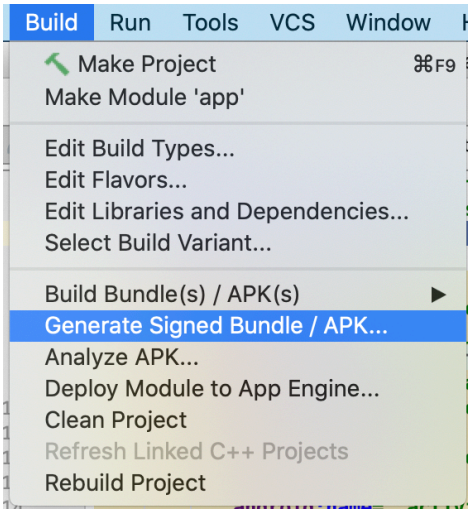

Figura 81. Android Studio. Generar APK 1

Para comenzar, tenemos que seleccionar entre App Bundle o APK. Un Android APP Bundle es archivo de la aplicación firmado para subirlo a las tiendas de aplicaciones. Un APK es un archivo firmado de la aplicación más pesado para despliegue directo en los dispositivos. En este caso se ha seleccionado **APK** ya que no se va a subir la aplicación a la tienda.

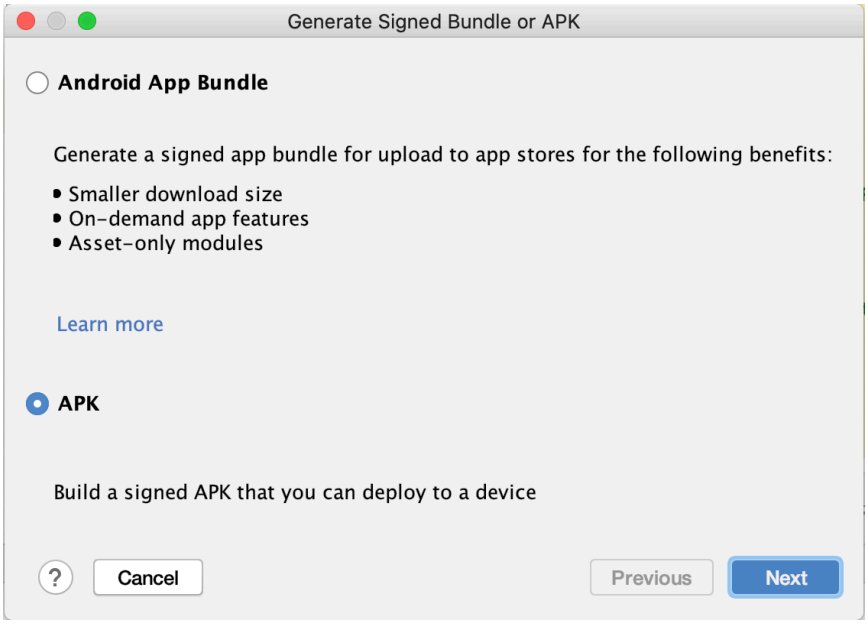

Figura 82. Android Studio, Generar APK 2

Después, tenemos que introducir los datos del almacén de claves. Si aun no se dispone de él es posible crear uno seleccionando “create new …” y rellenando los datos.

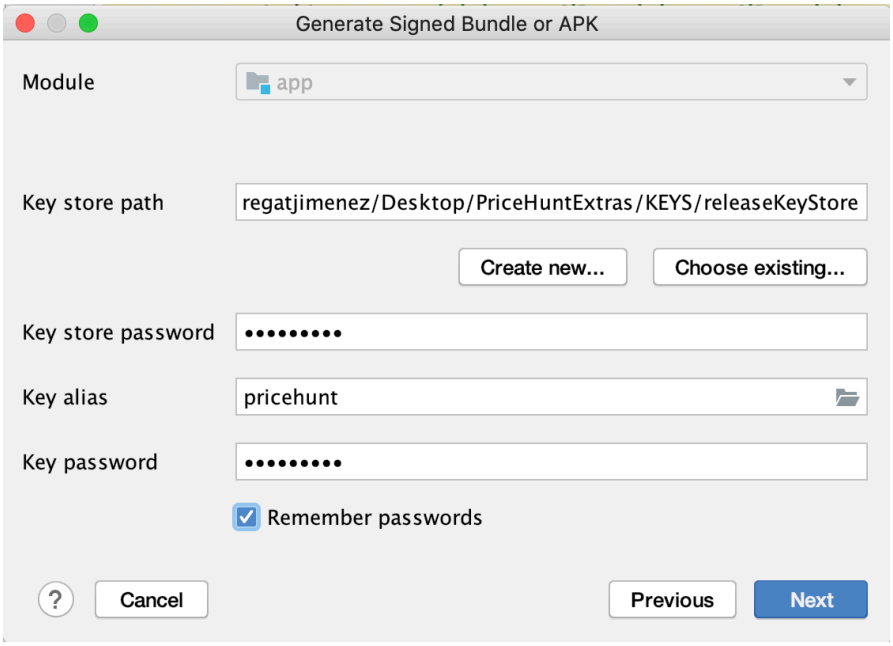

Figura 83. Android Studio, Generar APK 3

Finalmente, seleccionamos la carpeta de destino, la versión de lanzamiento o desarrollo según corresponda y la versión de la firma. Se ha seleccionado la versión 1 ya que la versión 2 es más avanzada y no funcionaba en los dispositivos más antiguos.

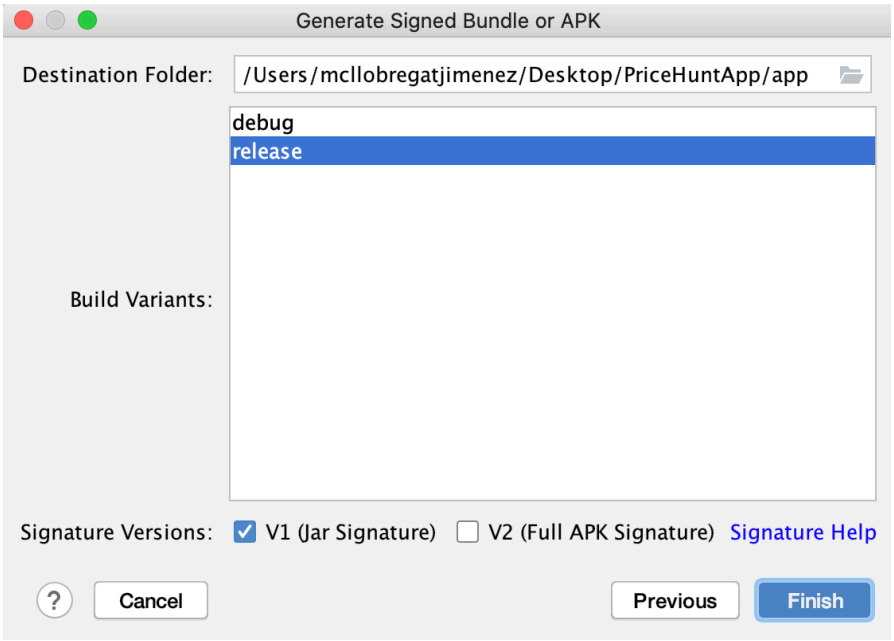

Figura 84. Android Studio, Generar APK 4

9.3.4. INSTALACIÓN DE LA APP EN EL DISPOSITIVO

Una vez construido el APK, debemos introducir el archivo en el dispositivo en cuestión. A continuación, abrimos el fichero y seguimos el flujo de instalación mostrado en las siguientes imágenes.

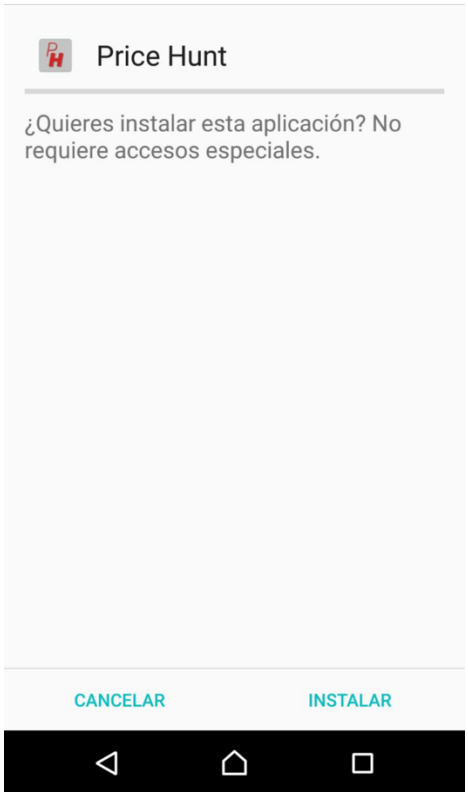

Figura 85. Confirmación de instalación del APK

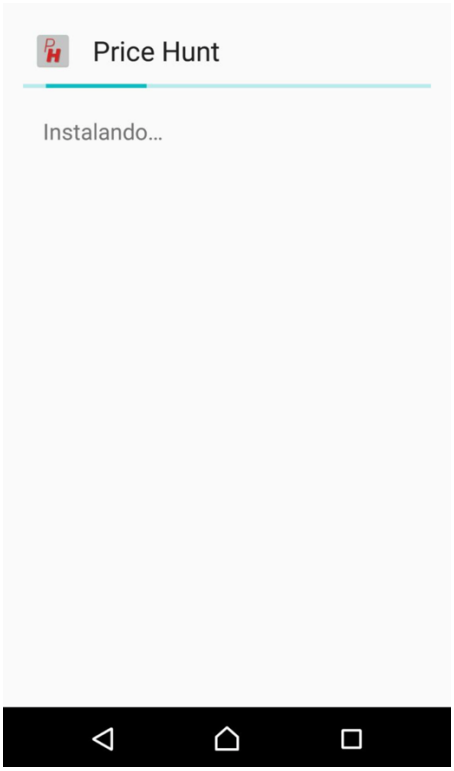

Figura 86. Instalación del APK

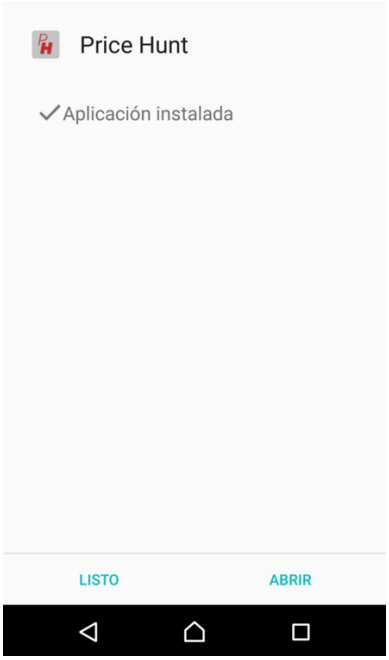

Figura 87. APK instalado

Es posible que durante el proceso o en la primera iniciación aparezca un mensaje de advertencia de que la aplicación no procede de una fuente oficial.

9.4. ANEXO IV. OBTENCIÓN DE LOS IDENTIFICADORES DE CLIENTE PARA EL PROTOCOLO OAUTH 2.0

En este anexo se expone en qué consiste el protocolo OAuth que utiliza Google para la funcionalidad de Google Sign In y cómo obtener los identificadores necesarios para integrar esta funcionalidad con nuestra aplicación.

9.4.1. ¿QUÉ ES OAUTH 2.0?

OAuth 2.0 es un framework de autorización, que permite a aplicaciones de terceros, como la de este TFG, el acceso a las cuentas de usuario de otros servicios, en este trabajo las cuentas Google (50).

En este protocolo intervienen tres partes:

- El usuario final: es el dueño de la cuenta de usuario en cuestión y decide si quiere dar permiso a la aplicación a sus datos en función de un ámbito o scope.
- El cliente: es la aplicación que quiere obtener los datos del usuario. Para ello, redirige al usuario a la API de la fuente donde este tiene que dar la autorización. En este trabajo sería la aplicación de Price Hunt, que redirige al usuario al pulsar en el botón de Google Sign In de la Figura 14.

- El servidor de recursos y autorización: Es el servidor de la fuente de información. Contiene los datos de las cuentas de usuario y, al verificar los datos del cliente por el flujo mencionado anteriormente, emite tokens de acceso a la aplicación cliente. Esta parte suele conocerse como API del servicio. En este TFG, Google representa este rol y, mediante el flujo de autenticación de la Figura 15 verifica la identidad del usuario y emite el token ID que utiliza Price Hunt para las autenticaciones.

Para poder hacer uso de este protocolo, es necesario configurar los datos de la aplicación en la Consola de Google para que nos proporcione un identificador con el que hacer uso del mismo.

En el caso de la aplicación Android, no es necesario especificar el identificador ya que Google reconoce la aplicación por el nombre del paquete y el hash SHA -1 del certificado con el que ha sido firmada. Sin embargo, es necesario especificar el identificador de cliente del servidor de back-end con el que se va a realizar la autenticación.

9.4.2. OBTENCIÓN DE LOS IDENTIFICADORES DE CLIENTE OAUTH 2.0

Las aplicaciones Android, durante su desarrollo, se firman con un certificado por defecto que tiene Android Studio, sin embargo, para construir el APK o el Bundle, se firman generalmente con otro certificado diferente. Por ello, todo este proceso debería de repetirse dos veces, con el hash SHA-1 del certificado de desarrollo y con el del certificado de producción.

Durante la creación del primer identificador para la aplicación Android se generará de forma automática el identificador para el servidor de back-end.

Obtener los identificadores del protocolo OAuth 2.0 es muy sencillo desde la documentación de Google sobre Google Sign In (38)

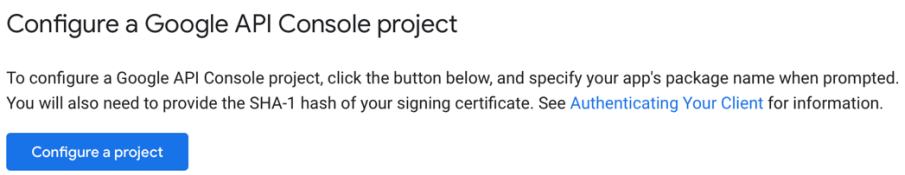

Figura 88. Configurar proyecto de Google API Console

En primer lugar, debemos seleccionar el proyecto referente a nuestra aplicación. Si no tenemos aun el proyecto creado, puede crearse desde este paso.

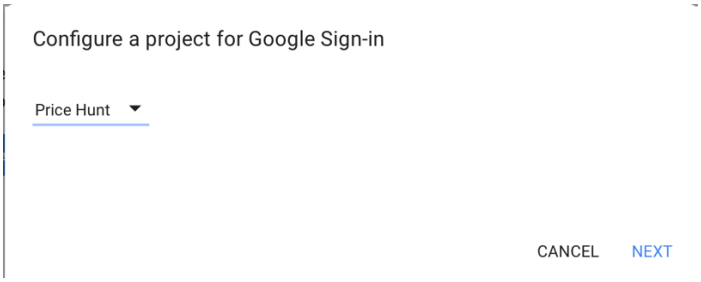

Figura 89. Configurar proyecto de Google API Console, seleccionar proyecto

A continuación, indicamos que el cliente es para acceder desde una aplicación Android e introducimos el nombre del paquete y el hash SHA-1 del certificado de producción. El nombre del paquete se encuentra en el archivo **AndroidManifest.xml**. Podemos obtener el hash del certificado a través del terminal de nuestro equipo, introduciendo el siguiente comando y la contraseña del almacén de claves:

```
keytool -keystore <path al keystore de la clave> -list -v
```

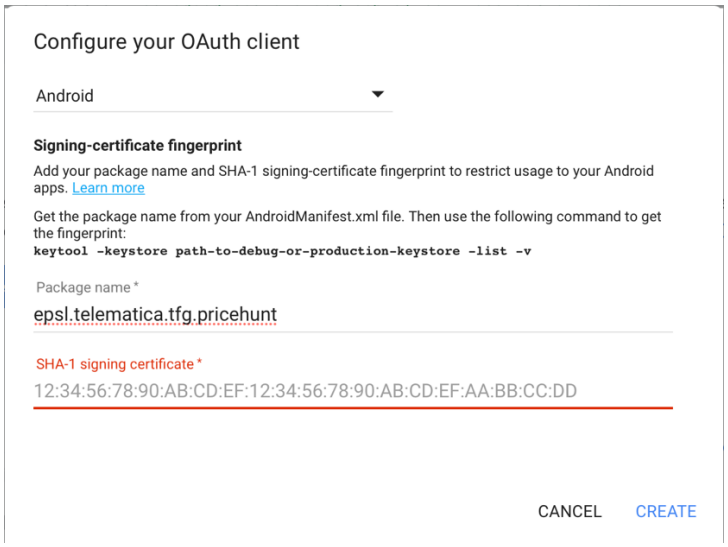

Figura 90. Configurar cliente OAuth

Al finalizar, en la consola de la API podemos ver que se ha creado el identificador.

En la siguiente imagen se muestran los 3 identificadores en la consola: el de Android de desarrollo, el de Android de producción y el autogenerado para la autenticación con el servidor.

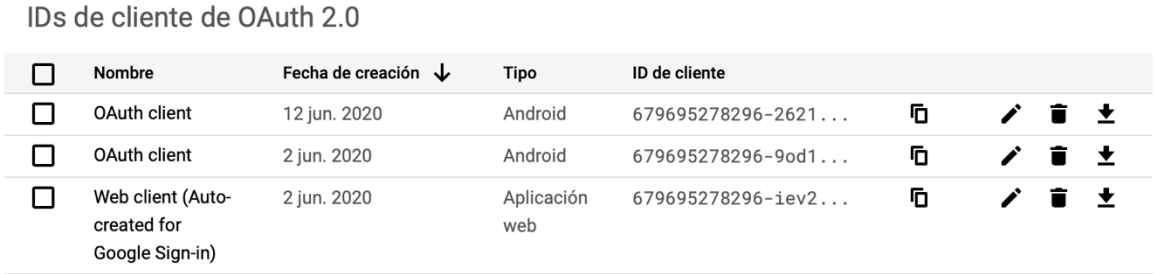

Figura 91. Identificadores de cliente de OAuth 2.0