Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática
Mención Computación

Data Warehouse para el estudio de la privacidad de aplicaciones móviles

Autor:
Alejandro Pérez de la Fuente

Tutor:
M. Mercedes Martínez González

Resumen

La privacidad juega un papel importantísimo en las aplicaciones Android ya que cada día se procesa una mayor cantidad de datos en los dispositivos móviles. Es por ello que resulta crucial que los usuarios sean conscientes en todo momento de los riesgos a los que se someten al utilizarlas. Además, en la actualidad no hay estudios a gran escala actualizados sobre el impacto que generan las distintas aplicaciones sobre la privacidad de sus usuarios. Entre las causas de esta falta, se tiene que no existen repositorios centrales que almacenen los metadatos relativos a la privacidad de las aplicaciones que permitan su estudio. Por este motivo, surge la necesidad de este proyecto, cuya finalidad es aportar un *warehouse* con los metadatos de las *apps* Android que permita el estudio de su impacto sobre la privacidad de los usuarios. Se presenta el análisis, diseño e implementación de las partes del *warehouse*. Entre ellas, una de las más importantes, el proceso ETL[1]. Donde, en primer lugar se aborda la integración de las diversas fuentes de datos de las que se nutrirá el *warehouse*. En segundo lugar, la transformación de los datos extraídos en los esquemas comunes del repositorio central. Y, por último, la carga de todos estos datos en el repositorio central.

[1]ETL es el acrónimo de *Extract Transform and Load*

Abstract

Privacy plays an extremely important role in Android applications as an increasing amount of data is processed on mobile devices every day. It is crucial for users to be aware of the risks they expose themselves to when using these applications. Furthermore, there is currently a lack of up-to-date large-scale studies on the impact that different applications have on user privacy. One of the reasons for this lack is the absence of central repositories that store metadata related to application privacy, which would allow for their study. Therefore, the need for this project arises, with the aim of providing a warehouse with metadata of Android apps, enabling the study of their impact on user privacy. It is presented the analysis, design, and implementation of various parts of the warehouse are addressed, including one of the most important aspects, the ETL (Extract, Transform, and Load) process. Firstly, the integration of various data sources that will feed the warehouse is addressed. Secondly, the transformation of extracted data into common schemas for the central repository is carried out. Finally, all this data is loaded into the central repository.

Agradecimientos

A mis padres y a mi hermana Ana Isabel, Juan José y Lydia por su apoyo y ánimo incondicional.

A mi tutora M. Mercedes Martínez y a Amador Aparicio por su tiempo, ayuda y guía durante el desarrollo del trabajo.

A todos los profesores del grado por transmitirme sus conocimientos.

A mis amigos por creer en mí.

Al ministerio de Educación y Formación Profesional por apoyarme a través de la concesión de la Beca de Colaboración para realizar tareas de investigación con el departamento de Informática de la Universidad de Valladolid bajo la dirección de la profesora M. Mercedes Martínez González publicada en el BOE del 15 de Junio de 2022.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto

En la actualidad, los teléfonos móviles son un elemento de uso cotidiano. Millones de personas tienen acceso a un teléfono móvil que les permite navegar por internet, acceder a redes sociales... Ya no se trata solo de un medio para comunicarse mediante llamadas de voz o mensajes de texto, sino que también se han convertido en una herramienta esencial para el trabajo, el entretenimiento y la educación. La importancia que tienen los teléfonos móviles en la sociedad actual es innegable y todo apunta a que lo seguirá siendo.

Las aplicaciones móviles pueden presentar **riesgos para la privacidad** de los usuarios. Muchas aplicaciones recopilan información personal, como la ubicación y la lista de contactos, sin el conocimiento o el consentimiento explícito del usuario [1]. Además, algunas aplicaciones pueden compartir esta información con terceros, como anunciantes y empresas de análisis de datos. Es importante que los usuarios sean conscientes de los riesgos de privacidad en las aplicaciones móviles y tomen medidas para proteger su información personal, como limitar el acceso a la información personal solo a las aplicaciones que sean necesarias y confiables.

Existen varios sistemas operativos de teléfonos móviles, pero los tres principales son: Android, iOS y Windows Phone. Android, desarrollado por Google, es un sistema operativo de código abierto utilizado por muchos fabricantes de dispositivos móviles. Por el contrario, iOS (desarrollado por Apple), es de código cerrado exclusivo para los dispositivos de la marca. Por último, Windows Phone (desarrollado por Microsoft), el cual se encuentra prácticamente en desuso. La cuota de mercado de cada uno de estos tres sistemas operativos es de 86,2 %, 13,7 % y 0,1 % respectivamente [2]. Como podemos apreciar, Android es el sistema operativo utilizado mayoritariamente por lo que nos centraremos en el.

El presente proyecto trata de recopilar y almacenar toda la información de aplicaciones Android necesaria para posteriormente poder aplicar métricas [3] que sean capaces de cuantificar la privacidad de las aplicaciones.

1.2. Motivación

La privacidad de los usuarios de aplicaciones móviles es de suma importancia debido a la gran cantidad de información personal que se maneja a través de estos dispositivos. Los usuarios confían en que sus datos personales, como su ubicación, historial de búsqueda y comunicaciones, se mantengan seguros y privados. Sin embargo, no siempre es así, produciéndose graves violaciones de privacidad. La privacidad de los usuarios de aplicaciones móviles es un derecho fundamental tal y como queda reflejado en la Ley Orgánica 3/2018 de Protección de Datos [4] que debe ser protegido para garantizar una experiencia segura y confiable en el uso de las aplicaciones móviles.

Por otro lado, los únicos trabajos encontrados sobre este tema se encuentran desactualizados [5] o solamente estudian un número reducido de aplicaciones [6], [7]. Lo que sí que se encuentra son repositorios de aplicaciones [8], pero estos no contienen directamente los metadatos relativos a la privacidad de las aplicaciones como sí se encuentran en el ámbito de los servicios web [9], [10].

Por último, tampoco se han encontrado líneas de trabajo actuales que se centren en almacenar los metadatos[1] de las aplicaciones en general. Y mucho menos centrándonos en los que pueden aportar información acerca de la privacidad de las aplicaciones móviles.

Estas carencias llevaron a plantear una línea de investigación en el departamento de Informática de la Universidad de Valladolid cuyo objetivo es proponer métricas de privacidad que ayuden a los usuarios a entender mejor el impacto que generan las *apps* que instalan en sus dispositivos [3]. Para poder aplicar estas métricas es necesario disponer de los datos que las nutren, que serán los metadatos de las *apps*, y aquellos otros que corresponden a los parámetros de las métricas. Por ejemplo, la utilización de permisos por el *malware*. También aquellos que permitan comparar los resultados que se obtienen con otras fuentes externas, y con los propios a medida que se presentan alternativas mejoradas de la(s) métrica(s) propuesta(s). Surgió así la necesidad de un *warehouse* que cumpla la función de repositorio central, capaz de responder a las necesidades de los diferentes servicios que implementan las métricas y las funcionalidades asociadas. Este *warehouse* cumple con la función conocida como "*Master Data Managemen*", puesto que será el responsable de garantizar la calidad de los datos que alimentan todos estos servicios.

Este *warehouse* se encastra en un sistema, cuyo diseño se resume en la figura 1.1. En este entorno se incluye el servicio *web* en desarrollo *APKFalcon* en el marco del Trabajo Fin de Grado de mi compañero y amigo Javier Crespo Guerrero [11]. El *warehouse* es accesible para este y otros servicios a través de una *API*, que abstrae los detalles de su implementación a los servicios que lo utilizan.

[1] Datos contenidos dentro de las aplicaciones que aportan información sobre estas.

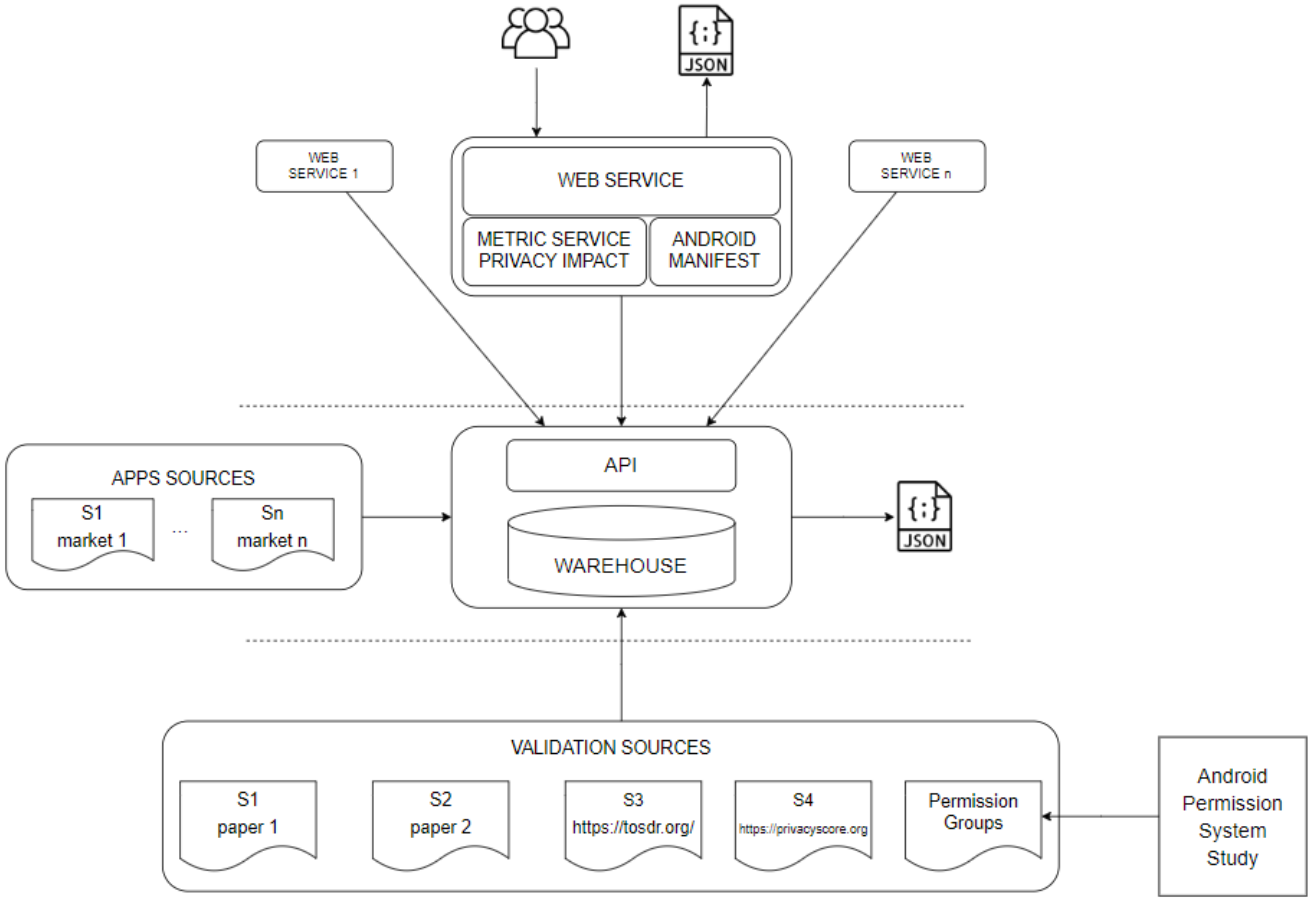

Figura 1.1: Entorno de integración del *warehouse* (gráfico facilitado por Amador Aparicio de la Fuente).

1.3. Objetivos

El objetivo principal de este proyecto es **obtener un *warehouse***[2] que almacene información relativa a la privacidad de las aplicaciones Android que permita la aplicación de métricas de privacidad en distintas versiones y comparación de estas con otras fuentes externas. Algunas de las tareas principales a llevar a cabo para conseguir el objetivo principal consisten en el análisis, diseño e implementación del *warehouse*. Además, también se plantean los siguientes subobjetivos:

- Ofrecer un método para integrar y recopilar los metadatos relativos a aplicaciones Android a partir de fuentes múltiples y heterogéneas.
- Obtención de los metadatos relativos a la privacidad de las aplicaciones Android.
- Proporcionar un mecanismo para recopilar resultados de distintas métricas de privacidad, de modo que sea posible compararlas.
- Ofrecer un mecanismo de acceso uniforme a todos estos datos.

[2]Sistema utilizado para analizar datos que funciona como un repositorio central de datos integrando una o más fuentes de información dispares.

1.4. Organización de la memoria

Este documento se estructura en siete capítulos con los siguientes contenidos:

- **Capítulo 1. Introducción.** Contextualización e introducción del proyecto que se va a realizar.
- **Capítulo 2. Planificación del proyecto.** Se realiza una planificación inicial del proyecto, incluyendo la gestión del tiempo, de los riesgos y de los costes.
- **Capítulo 3. Fundamentación teórica.** Se explica la metodología y estándares que se seguirán a lo largo del proyecto.
- **Capítulo 4. Análisis.** Análisis de requisitos que el *warehouse* ha de cumplir.
- **Capítulo 5. Diseño.** Diseño detallado del *warehouse*.
- **Capítulo 6. Implementación y pruebas.** Se explican los medios tecnológicos utilizados, detalles de la implementación y pruebas básicas del sistema.
- **Capítulo 7. Conclusiones y líneas de trabajo futuro.** Reflexión sobre los resultados obtenidos y posibles mejoras o lineas de trabajo futuro.

Capítulo 2

Planificación del proyecto

En este capítulo se realiza la planificación del proyecto siguiendo la guía de PMBOK [12]. Se lleva a cabo una elaboración progresiva de la planificación mediante un proceso iterativo en el cual se va refinando el nivel de detalle de la misma, obteniendo así estimaciones mas precisas.

En primer lugar, se realiza una planificación inicial que contiene las principales tareas del proyecto así como una estimación de la duración de las mismas. Además, se incluyen algunos hitos que facilitan el seguimiento del progreso del proyecto. Seguidamente, se realiza una gestión temporal del proyecto en forma de cronograma y una gestión de los principales riesgos que pudieran incurrir durante el desarrollo del proyecto. Por último, una estimación de los costes del proyecto.

2.1. Planificación inicial

Debido a la naturaleza del proyecto, se llevó a cabo un **desarrollo en espiral** iterativo puesto que la naturaleza del mismo y su clara división lo permite. Además, esto nos permite avanzar de forma rápida en el proyecto permitiéndonos obtener una versión inicial funcional que iría aumentando su funcionalidad y complejidad poco a poco a medida que se avance en el mismo ya sea añadiendo nuevas fuentes de datos así como otro tipo de subsistemas. En todas ellas, se lleva a cabo un **desarrollo en cascada** con las siguientes tareas:

- **Análisis:** Estudio de los requisitos del proyecto, en específico, del almacén de datos, de las fuentes de datos y de los datos a tratar [13].
- **Diseño:** Detalle de todos los elementos del sistema a confeccionar elaborado en forma de diagramas tales como modelo de datos conceptual y lógico, diagrama de arquitectura del sistema, diagrama de despliegue... Así como cualquier otro subproducto que ayude a la comprensión del sistema y su posterior implementación.
- **Implementación:** Desarrollo del sistema que engloba la revisión del diseño, el aprendizaje de las herramientas a utilizar si procede y la implementación del mismo.
- **Despliegue:** Dado que uno de los objetivos de este proyecto es la perdurabilidad del mismo, este se ha de desplegar en alguna máquina o en la nube para permitir su futuro uso y ampliación.

- **Pruebas:** Validación del funcionamiento del mismo.
- **Memoria:** De forma paralela a todas las tareas anteriormente citadas se iría confeccionando esta memoria de todo el trabajo desarrollado.

En cuanto a las distintas iteraciones, la primera de ellas se centrará en el groso del proyecto, tratando lo que vendría a ser la totalidad del núcleo del sistema mientras que las sucesivas iteraciones serán la integración de nuevas fuentes de datos así como funcionalidad de apoyo tales como clientes de acceso al sistema entre otros.

Hitos principales que guiarán el progreso del proyecto:

Hito	**Fecha**
Lista completa de requisitos del sistema	28/02/2023
Diseño general del sistema	26/03/2023
Implementación general del sistema	10/04/2023
Fin primera iteración	17/04/2023
Integración de todas las fuentes de datos	01/05/2023
Integración de todos los subsistemas	15/05/2023
Fin del proyecto	01/06/2023

Tabla 2.1: Lista de hitos del proyecto.

2.2. Gestión del tiempo

Para la realización de la gestión del tiempo se utiliza la herramienta gratuita de código abierto GanttProject en su versión 3.2 [14]. A continuación se muestra un desglose detallado de las tareas a realizar con fechas de inicio y fin estimadas de cada una de estas, así como los hitos anteriormente mencionados.

Cabe destacar que las tareas *"Incorporación de todas las fuentes de datos"* e *"Integración de todos los subsistemas"*, son agrupaciones de iteraciones de desarrollo en cascada para cada posible fuente de datos a integrar y subsistema respectivamente. Esta simplificación se hace debido a que no todas las fuentes de datos (resp. subsistemas) son igual de complejos por lo que estimar su duración concreta sería poco preciso. En su lugar, se ha preferido estimar la duración total de cada una de las agrupaciones guiada por los requisitos temporales del proyecto.

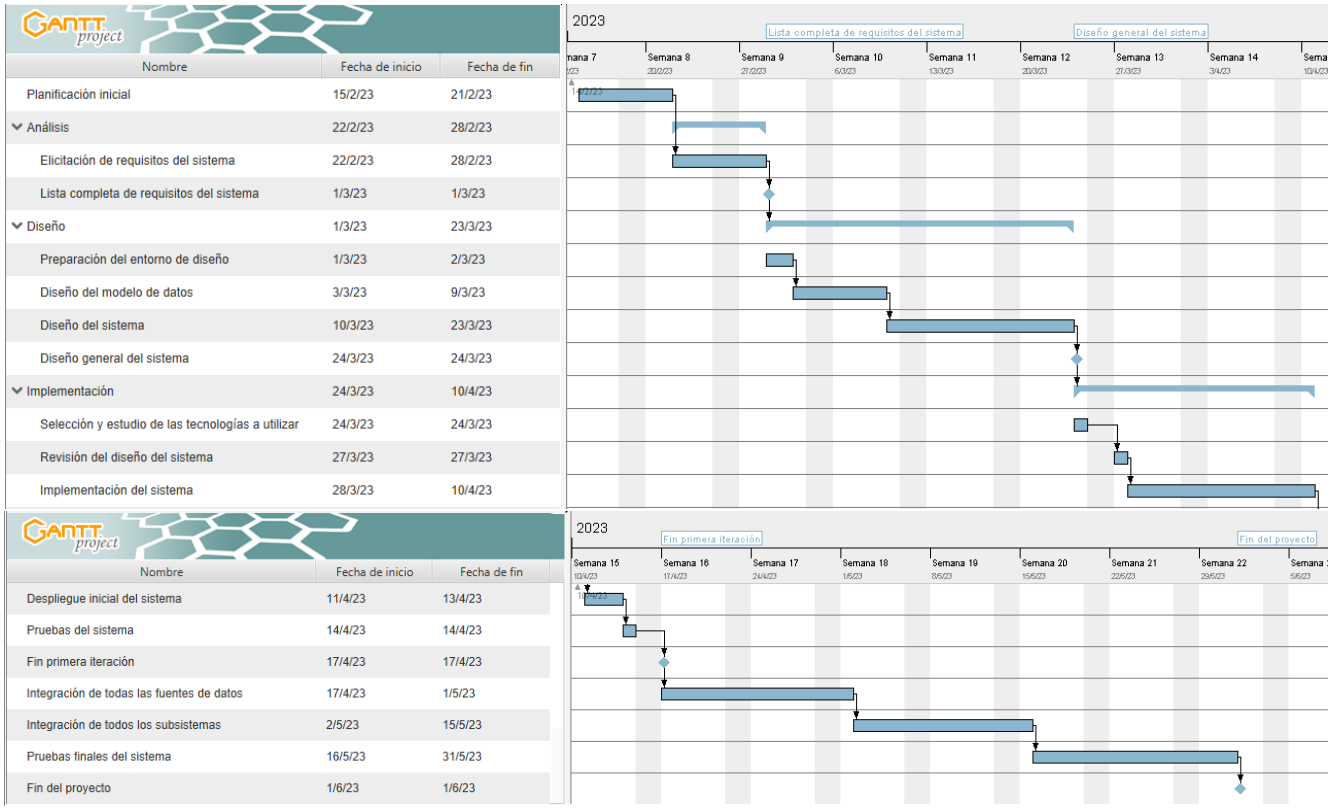

Figura 2.1: Cronograma del proyecto.

2.3. Gestión de riesgos

En este apartado, se realiza una identificación de los principales riesgos asociados a este proyecto así como un análisis cuantitativo y un plan de respuesta de los mismos.

Se identifican riesgos comunes al desarrollo de proyectos software así como propios de este proyecto en concreto enumerados en la tabla 2.2.

Nº	**Riesgo**
1	Errores en la planificación
2	Uso de tecnologías desconocidas
3	Encontrar un fallo de diseño tardío
4	Recursos necesarios inaccesibles
5	Fecha de finalización posterior a la prevista
6	Menor disponibilidad de la necesaria
7	Fuentes de datos cambiantes

Tabla 2.2: Lista de riesgos del proyecto.

Cabe destacar la importancia de alguno de los riesgos anteriormente citados por su alta probabilidad de ocurrencia así como el impacto que tendrían.

En primer lugar, *"Encontrar un fallo de diseño tardío"*, en específico en el modelo de datos, es algo con una alta probabilidad de ocurrencia puesto que al ser un proyecto de investigación es muy cambiante.

En segundo lugar, *"Recursos necesarios inaccesibles"*. Esto es particularmente cierto en el caso de busqueda de un alojamiento (*host*) para el sistema, dado que las limitaciones económicas son muchas y la Universidad de Valladolid no dispone de un servicio de *hosting* propio para este tipo de recursos vinculados a la investigación.

Por último, *"Fuentes de datos cambiantes"*, como ya sabemos a día de hoy todas las páginas web van cambiando de diseño y herramientas migrando de unas a otras para obtener el mayor número de clientes posible. Además, día a día van surgiendo nuevas fuentes de datos y dejando de estar disponibles otras. Todo ello, representa un grave problema ya que puede obligar a rediseñar partes del sistema.

Para ver qué riesgos debemos evitar en mayor medida, se realiza un análisis cuantitativo de estos clasificándolos en riesgos bajos, moderados y altos según su probabilidad de ocurrencia e impacto que tienen [12]:

Probabilidad	**Amenazas**				
0,90	0,05	0,09	0,18	0,36	0,72
0,70	0,04	0,07	0,14	0,28	0,56
0,50	0,03	0,05	0,10	0,20	0,40
0,30	0,02	0,03	0,06	0,12	0,24
0,10	0,01	0,01	0,02	0,04	0,08
	0,05/ Muy Bajo	0,10/ Bajo	0,20/ Moderado	0,40/ Alto	0,80/ Muy Alto

Tabla 2.3: Matriz de probabilidad e Impacto.

Calificación de los riesgos del proyecto:

Riesgo (Nº)	**Probabilidad**	**Impacto**	**Calificación**
1	0,70	0,40	0,28 (Alto)
2	0,90	0,10	0,09 (Moderado)
3	0,30	0,40	0,12 (Moderado)
4	0,70	0,40	0,28 (Alto)
5	0,10	0,80	0,08 (Moderado)
6	0,10	0,80	0,08 (Moderado)
7	0,90	0,20	0,18 (Alto)

Tabla 2.4: Calificación de los riesgos.

Como podemos apreciar, los riesgos más importantes a tener en cuenta son: *"Errores en la planificación"*, *"Recursos necesarios inaccesibles"* y *"Fuentes de datos cambiantes"*.

Realizaremos un plan de acciones en caso de que cada uno de los riesgos se materialice:

1. Llevar a cabo una replanificación al vuelo del proyecto teniendo en cuenta los errores de la primera.
2. Llevar a cabo una formación rápida de la tecnología a utilizar.
3. Rediseño del sistema teniendo en cuenta el diseño actual y las nuevas necesidades encontradas.
4. Llevar a cabo un despliegue en local.
5. Replanificar las fechas de entrega a unas nuevas.
6. Redistribuir el tiempo en las tareas más importantes.
7. Usar fuentes de datos más estables de apoyo para evitar la nula disponibilidad.

2.4. Gestión de los costes

Los principales costes del proyecto se podrían englobar en tres grandes categorías:

- **Costes tecnológicos:** Entre estos se encuentran los costes de las herramientas a utilizar tales como *frameworks*, editores... En este caso en concreto, debido a las limitaciones económicas se utilizan herramientas libres. Además, se utiliza un ordenador para realizar todo el proyecto con un coste aproximado de 600 €.
- **Costes de despliegue:** Entre estos se encuentran los costes de *host*. Se busca la solución más económica que permita el cumplimiento de los objetivos del proyecto. Y, se opta por una solución gratuita.
- **Costes personales:** En cuanto al gasto del personal, teniendo en cuenta que el salario medio anual de un desarrollador *junior* en España [15] ronda los 22.500 € y que se utilizan un total de 300 h para la realización del proyecto, esto nos daría una estimación de 3.516 €.

2.5. Modificaciones de la planificación inicial

Las principales modificaciones que ha sufrido la planificación inicial del proyecto durante su desarrollo se deben principalmente a la materialización del riesgo número 1 que aparece en la tabla 2.2. En específico, se ha subestimado el tiempo necesario para llevar a cabo la memoria del trabajo realizado lo que ha producido los siguientes cambios:

- Expansión del tiempo de duración de todas y cada una de las tareas del proyecto contenidas en el cronograma de la figura 2.1 debido al incremento del tiempo utilizado en la realización de la parte correspondiente de la memoria.
- Retraso de 15 días de la fecha de finalización del proyecto hasta el día 15 de Junio de 2023.

Capítulo 3

Fundamentación teórica

En este capítulo se detallan algunos de los estándares y procedimientos habituales en el diseño y desarrollo de sistemas tipo *warehouse*.

3.1. Almacén de datos (*Data warehouse*)

Hace referencia a un repositorio central de datos (*Data warehouse* o *Warehouse*) orientado a un determinado ámbito concreto ya sea empresarial, educativo, de investigación, etc. Las principales características de estos almacenes según Bill Inmon [16] son:

- **Orientado a temas**: Los datos estarán organizados en torno a los distintos ámbitos a los que pertenezcan estando entre si relacionados.

- **Variante en el tiempo**: Los cambios que se produzcan en los datos quedarán reflejados para poder realizar análisis e informes.

- **No volátil**: La información no se puede modificar ni eliminar. Una vez almacenado un dato, este solo puede ser consultado.

- **Integrado**: La base de datos ha de contener todos los datos referentes al ámbito del que trate y han de ser consistentes entre sí.

Estos almacenes son utilizados habitualmente por las organizaciones como sistemas de soporte para la toma de decisiones o sistemas de información ejecutiva. Además, permiten hacer consultas y/o informes de interés para la organización.

Otra de las principales ventajas de los almacenes de datos es la transferencia de conocimiento. Para ello, los *warehouses* habitualmente implementan algún sistema que permita acceder a su información. Hoy en día el sistema de acceso más utilizado son *API REST* [17] por sus ventajas ya que permiten manejar los objetos a través de enlaces, aportan una interfaz de acceso uniforme y son independientes del lenguaje y plataforma utilizados para su despliegue.

3.2. Proceso *Extract Transform Load (ETL)*

Es un proceso que se popularizó en los años 70 [18] que consiste en la integración de diversas fuentes de datos en un gran repositorio central. En la actualidad, este proceso es fundamental en el ámbito empresarial debido a que cada vez se generan y recopilan más datos provenientes de todo tipo de fuentes.

Es un proceso en tres fases (figura 3.1) donde los datos en primer lugar se extraen, posteriormente se transforman y, por último, se cargan en un repositorio central (*warehouse*).

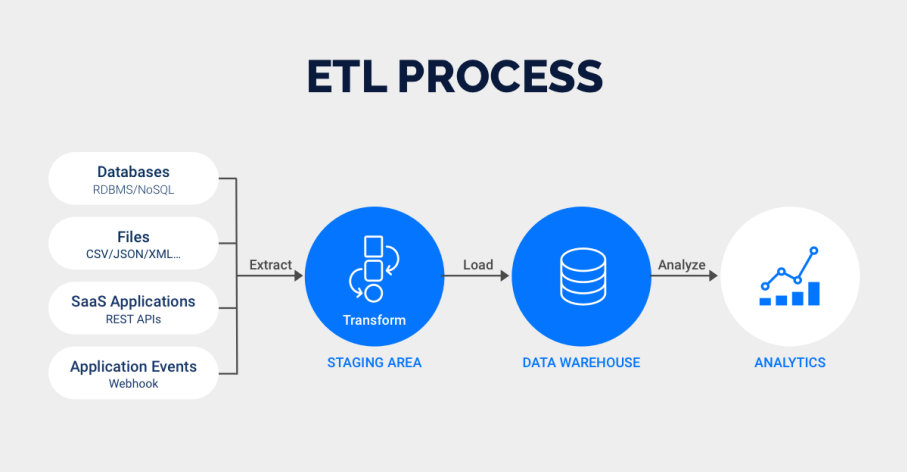

Figura 3.1: Proceso *Extract Transform Load (ETL)* [19].

Fases:

1. ***Extract***

   La primera parte del proceso trata de solucionar el problema de la extracción de la información de las fuentes origen. Estas fuentes seguramente sean heterogéneas en cuanto a acceso. Por ello, esta parte del proceso suele implicar distintos métodos para acceder a los datos. Por ejemplo: *web scraping*, uso de *APIs*, consultas a bases de datos estructuradas (*SQL*)...

2. ***Transform***

   La segunda parte del proceso trata de solucionar el problema de la heterogeneidad en la estructura de los datos provenientes de las distintas fuentes. Es decir, integrar toda la información por medio de transformaciones de esta en los esquemas del *warehouse*. Además, esta parte del proceso también involucra otras transformaciones de los datos como normalizaciones y filtrado de datos no válidos entre otras.

3. ***Load***

   La tercera y última parte del proceso trata de solucionar el problema de la carga de todos los datos transformados en el *warehouse*. Existen múltiples aproximaciones para solucionar el problema, entre ellas, nos encontramos con el patrón de diseño *software Data Mapper*. Este patrón define un mapeado entre los datos transformados y el esquema de la base de datos relacional que los contendrá. Para cada tipo de dato se crea un objeto que contiene dicho mapeado siendo este el encargado la carga de datos a la base de datos.

3.3. Protocolos y estándares

3.3.1. Protocolo *HTTP* y *SSL/TLS*

HTTP (Hypertext Transfer Protocol) es un protocolo de comunicación *web* basado en mensajes de texto. La comunicación se realiza por medio de mensajes de texto plano con un formato definido según los RFCs 2616 [20], 7230 [21] y 7231 [22]. En la figura 3.2 se encuentra la estructura básica de los mensajes.

```
Client request:

  GET /hello.txt HTTP/1.1
  User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
  Host: www.example.com
  Accept-Language: en, mi


Server response:

  HTTP/1.1 200 OK
  Date: Mon, 27 Jul 2009 12:28:53 GMT
  Server: Apache
  Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
  ETag: "34aa387-d-1568eb00"
  Accept-Ranges: bytes
  Content-Length: 51
  Vary: Accept-Encoding
  Content-Type: text/plain

  Hello World! My payload includes a trailing CRLF.
```

Figura 3.2: Formato de los mensajes *HTTP* [21].

Los elementos más importantes de los mensajes son:

- Recurso solicitado o *URI*. Es la dirección del recurso al que se esta accediendo (*/hello.txt* en el caso de la figura 3.2).

- Método de la petición del cliente (figura 3.3).

```
Method         = "OPTIONS"
               | "GET"
               | "HEAD"
               | "POST"
               | "PUT"
               | "DELETE"
               | "TRACE"
               | "CONNECT"
               | extension-method
extension-method = token
```

Figura 3.3: Métodos *HTTP* [20].

  Este parámetro nos indica el método que se ha ejecutar en el recurso solicitado. Entre ellos, *GET* sirve para la obtención de información sobre el recurso solicitado [20] y *POST* para la actualización de la información del recurso solicitado [20]. Por ejemplo, por medio de la actualización de una base de datos.

- Cabecera *Authorization*, sirve para la autenticación de acceso a un recurso concreto. Existen múltiples estándares que definen el formato de las autenticaciones, por ejemplo *OAuth2.0* [23] define como contenido de la cabecera: “Bearer PASSWORD” para autenticar a un usuario.

- Código respuesta. Define el estado de una respuesta a una petición en forma de un código numérico [22]. Alguno de los más comunes se puede ver en la tabla 3.1.

Code	Reason-Phrase
200	OK
400	Bad Request
401	Unauthorized
404	Not Found
422	Unprocessable Content
500	Internal Server Error

Tabla 3.1: Códigos respuesta *HTTP* [22].

Este protocolo de comunicación no es seguro por si solo puesto que los mensajes se transfieren por texto plano. Por ello, surgió la capa de *sockets* seguros (*SSL*) que posteriormente se convertiría en la capa de transporte seguro (*TLS*). Gracias a esta, los mensajes ya no se transmiten por medio de texto plano sino que se transmiten encriptados con algoritmos asimétricos. En la actualidad uno de los más utilizados es *RSA* que utiliza una clave pública para la encriptación de los datos y una privada para desencriptarlos. En la figura 3.4 se ve el nuevo flujo de mensajes Cliente-Servidor para la comunicación segura: en primer lugar, se establecen las claves de encriptación que utilizarán en la transmisión de los mensajes, seguidamente, se lleva a cabo la transmisión de los datos encriptados.

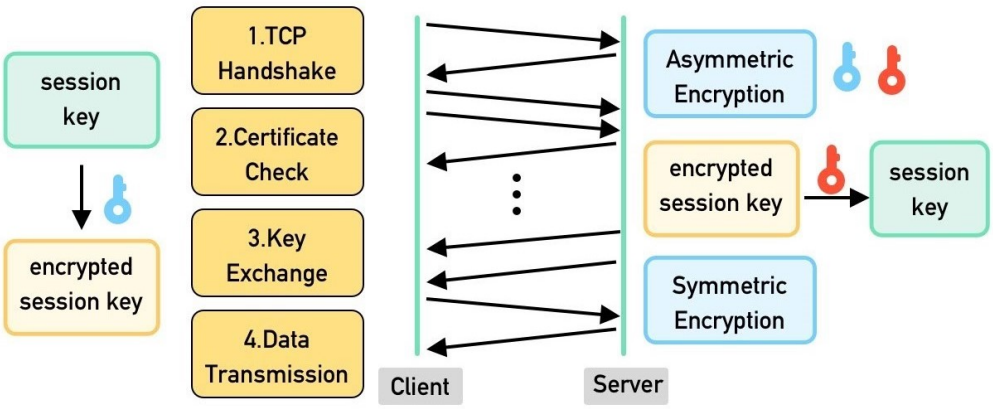

Figura 3.4: Funcionamiento de *HTTPS* [24].

3.3.2. Estándar *OpenAPI*

OpenAPI es una especificación que surgió en el 2010 para el diseño y documentación de interfaces de programación (*APIs*) [25] como sucesor al anterior estándar *Swagger*. Su principal objetivo era convertir a *Swagger* en un estándar abierto y neutral. Es un estándar que a día de hoy se sigue manteniendo y evolucionando por la *Linux Fundation* para adaptarse a las necesidades cambiantes del desarrollo de las *APIs*. Dentro de las múltiples especificaciones que realiza, alguna de las más importantes son:

- Especificación de la estructura de la documentación de la *API* para cada recurso:

  Método *HTTP* y URI del recurso.
  Descripción de la petición.
  Parámetros de la petición: {Nombre / Descripción, ...}.
  Respuestas a la petición: {Código *HTTP* / Descripción, ...}.

- Especificaciones sobre interoperabilidad.

  Para fomentar la interoperabilidad de las *APIs*, el estándar especifica que se utilicen en sus respuestas a peticiones estructuras ya definidas por otros protocolos y/o estándares. Entre ellos:

  - Los códigos de estado *HTTP* definidos en el RFC 7231 [22].
  - Los tipos de medios respuesta definidos por el RFC 6838 [26] (por ejemplo, *application/json*).
  - Estándares para la autenticación de los usuarios como *OAuth2.0* [23].

Capítulo 4

Análisis

En este capítulo, se presenta el análisis de los requisitos del *warehouse*. En él, se enuncian las restricciones y funcionalidades que ha de cumplir el sistema. Es importante ya que gracias a él, se evitan posibles malentendidos acerca de que ha o no de hacer el sistema así como su alcance. Además, establece una base sólida que sirve como comienzo para el diseño y desarrollo del sistema.

4.1. Roles de usuarios del sistema

Se detecta la necesidad de separar la funcionalidad del *warehouse* por roles de usuario protegiendo así la funcionalidad que pudiera ser más crítica de los roles más básicos y limitados:

- **Usuario general**: Este tipo de usuario es el más básico y limitado, puede realizar consultas sobre el *warehouse* así como solicitar la carga de nuevas aplicaciones.

- **Administrador**: Este tipo de usuario tiene los máximos privilegios y por tanto puede actualizar directamente el *warehouse* con acciones como la inserción de nuevas métricas de privacidad, nuevas puntuaciones de aplicaciones y la carga masiva de nuevas aplicaciones.

4.2. Requisitos

En esta sección se especifican los requisitos que el sistema ha de cumplir para considerarse exitoso.

4.2.1. Requisitos de información

Los requisitos de información se centran en la información que el *warehouse* ha de almacenar para su funcionamiento.

Requisitos de información del *warehouse*:

- El sistema ha de ser capaz de almacenar persistentemente los metadatos de las aplicaciones que se obtienen a partir de su paquete (*.apk*).

- El sistema ha de ser capaz de almacenar persistentemente el nombre de la fuente de la que se extraen las *apps* y la marca temporal del momento de la extracción.
- El sistema ha de ser capaz de almacenar persistentemente los resultados de aplicar métricas que puntúan el impacto de privacidad de las *apps*.
- El sistema ha de ser capaz de almacenar persistentemente puntuaciones que se asignan a los permisos Android en diversos *rankings* que recogen datos sobre su utilización por *malware* [27].
- El sistema ha de ser capaz de almacenar persistentemente metainformación sobre la asociación por defecto entre permisos y grupos de permisos en el sistema operativo Android.

4.2.2. Requisitos funcionales

Los requisitos funcionales son una descripción de los servicios que el sistema ha de ofrecer [28]. En general, engloba las entradas que el sistema permitirá, como se comportará y qué salidas proporcionará.

Requisitos funcionales del *warehouse*:

- El sistema ha de ser capaz de autenticar dos tipos distintos de usuario: los usuarios generales y los administradores del *warehouse*. Estos tipos de usuario tendrán distinta funcionalidad disponible.
- El sistema ha de autenticar a cada usuario a través de su contraseña.
- El sistema ha de permitir a los usuarios generales la descarga de los metadatos de una aplicación a través de su hash o de su nombre de paquete.
- El sistema ha de permitir que los usuarios generales hagan peticiones de carga de aplicaciones concretas al *warehouse* por su nombre de paquete.
- El sistema ha de permitir a los usuarios administradores la carga de puntuaciones de aplicaciones, nuevas métricas de privacidad y actualizaciones de la metainformación acerca de los permisos y grupos de permisos del sistema operativo Android.
- El sistema ha de permitir a los usuarios administradores aplicar métricas de privacidad a las aplicaciones.
- El sistema ha de permitir a los usuarios administradores la carga de una cantidad arbitraria de aplicaciones aleatorias para nutrir el *warehouse*.
- El sistema ha de ser capaz de integrar varias fuentes de aplicaciones.
- El sistema ha de ser capaz de extraer la categoría de una aplicación según Google Play.
- El sistema ha de ser capaz de extraer los metadatos correspondientes a la privacidad de una aplicación contenidos en su paquete.
- El sistema ha de ser capaz de obtener metainformación acerca de los permisos y grupos de permisos declarados en el sistema Android.

- El sistema ha de ser capaz de integrar varias fuentes que puntúan el impacto sobre la privacidad de las aplicaciones.

4.2.3. Requisitos no funcionales

Los requisitos no funcionales son aquellos que especifican características y/o restricciones acerca del funcionamiento del sistema [29].

Requisitos no funcionales del *warehouse*:

- El sistema deberá manejar conexiones concurrentes.
- El sistema deberá utilizar bases de datos relacionales.
  En la actualidad un buen gestor de bases de datos relacionales es *MySQL*.
- El sistema deberá proporcionar acceso web.
  En la actualidad los protocolos de acceso web más utilizados son *HTTP* y *HTTPS*.
- El sistema deberá utilizar un formato estándar para comunicarse con sus usuarios.
  En la actualidad uno de los formatos más utilizados en la comunicación de sistemas informáticos es el formato *JSON*.

Requisitos de seguridad

Cabe mencionar por separado los requisitos no funcionales referentes a la seguridad que ha de cumplir nuestro sistema:

- El sistema deberá estar protegido contra accesos no autorizados.
- El sistema deberá utilizar un estándar seguro para la autenticación de usuarios.
  En la actualidad el estándar *OAuth 2.0* proporciona un marco seguro de acceso ampliamente utilizado por aplicaciones *web* y móviles.
- El sistema deberá estar protegido contra entradas de datos malintencionadas y/o malformadas.
  Por ejemplo, ante ataques de tipo SQLi (Inyección de SQL).
- El sistema deberá utilizar una función para evitar colisiones a la hora de identificar elementos.
  En la actualidad la función *hash SHA256* es una de las más utilizadas en este respecto.

4.3. Casos de uso

En esta sección se identifican en la figura 4.1 cuales son los casos de uso que el sistema tendrá para ser capaz de satisfacer los requisitos funcionales anteriormente mencionados. Se diferenciarán según que tipo de usuario sea: un usuario general o administrador.

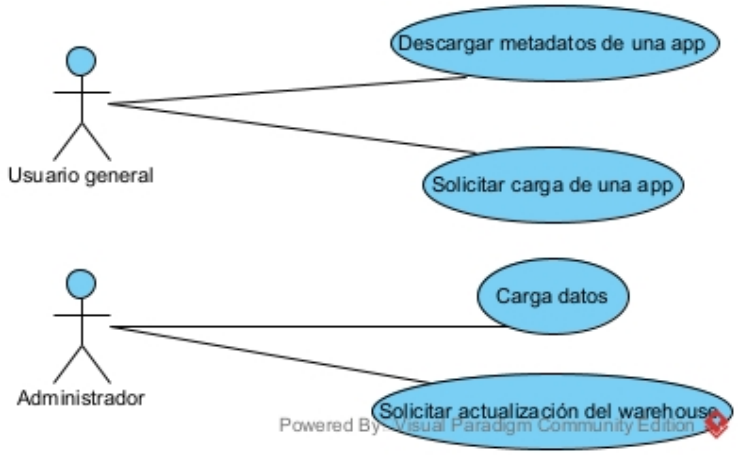

Figura 4.1: Diagrama de casos de uso.

4.3.1. UC-1: Descargar metadatos de una *app*

En la tabla 4.1 se encuentra la especificación del caso de uso de descargar los metadatos de una aplicación y en la figura 4.2 se encuentra el diagrama de actividades asociado a este caso de uso.

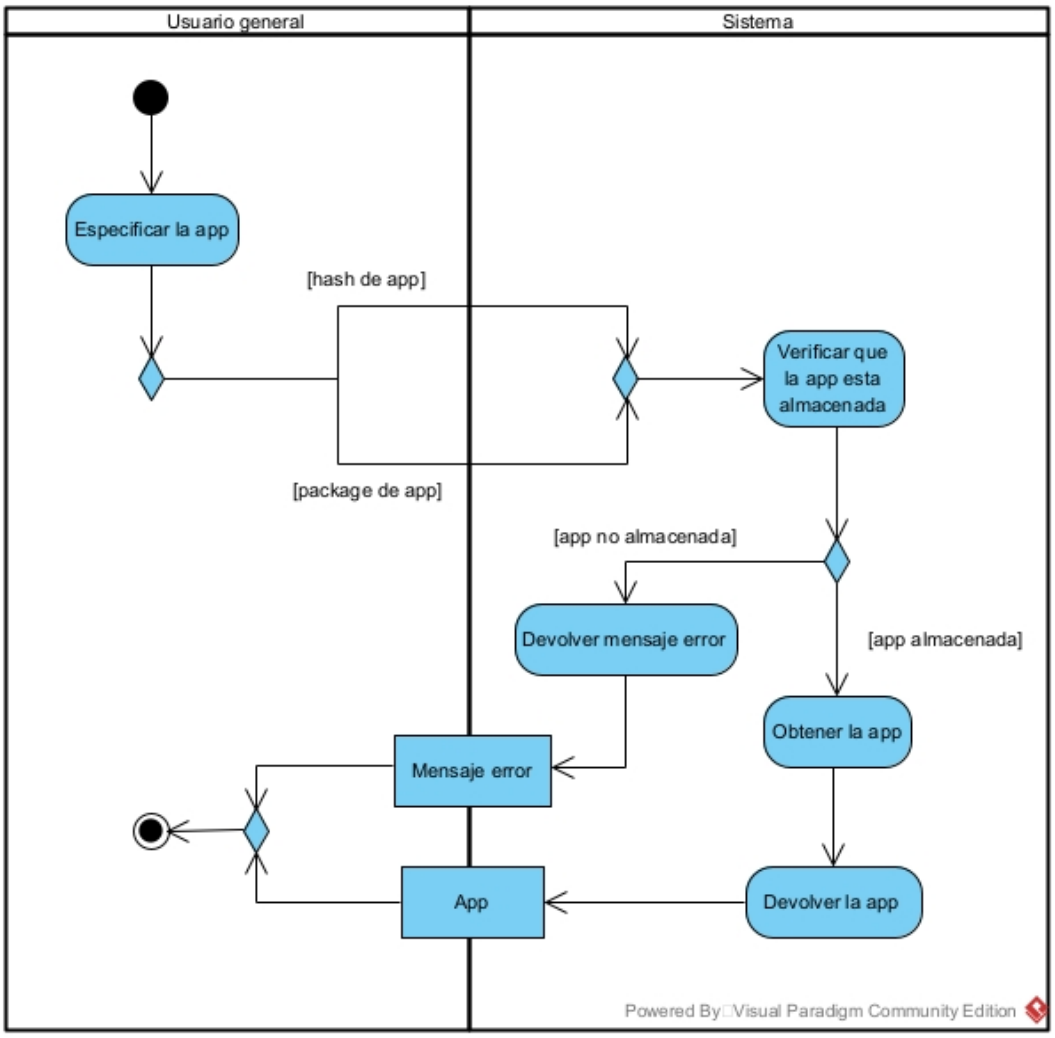

Figura 4.2: Diagrama de actividades del UC-1.

UC-1	Descargar metadatos de una *app*
Descripción	Descarga los metadatos de una *app* del *warehouse*.
Precondición	El usuario general ha de estar autenticado.
Secuencia normal	1. El usuario general introduce el *hash* o *package* de la *app* que desea. 2. El sistema comprueba que la *app* se encuentra almacenada. 3. El sistema obtiene la *app* almacenada. 4. El sistema devuelve los metadatos de la *app*.
Flujos alternativos	2a-1. La *app* no se encuentra en el sistema. 2a-2. El sistema devuelve un error indicando que la *app* solicitada no se encuentra en el sistema. 2a-3. El caso de uso queda sin efecto.
Postcondición	El usuario general ha obtenido los metadatos de la *app*.

Tabla 4.1: Caso de uso: Descargar metadatos de una *app*.

4.3.2. UC-2: Solicitar carga de una *app*

En la tabla 4.2 se encuentra la especificación del caso de uso de descargar los metadatos de una aplicación y en la figura 4.3 se encuentra el diagrama de actividades asociado a este caso de uso.

UC-2	Solicitar carga de una *app*
Descripción	Solicita la carga de una *app* al *warehouse*.
Precondición	El usuario general ha de estar autenticado.
Secuencia normal	1. El usuario general introduce el *package* de la *app* que desea cargar. 2. El sistema comprueba que hay hilos disponibles para subir *apps*. 3. El sistema reserva un hilo para subir la *app*. 4. El sistema indica al usuario general que la petición se esta procesando. 5. El sistema descarga la *app* solicitada. 6. El sistema extrae la información de la *app* descargada. 7. El sistema almacena la información extraída. 9. El sistema libera el hilo reservado.
Flujos alternativos	2a-1. Todos los hilos de subida se encuentran ocupados. 2a-2. El sistema informa al usuario general de que el sistema se encuentra ocupado. 2a-3. El caso de uso queda sin efecto. 5a-1. No se encuentra la *app* solicitada en las fuentes de datos. 5a-2. El sistema libera el hilo reservado. 5a-3. El caso de uso queda sin efecto.
Postcondición	La *app* solicitada queda almacenada en el *warehouse*.

Tabla 4.2: Caso de uso: Solicitar carga de *app*.

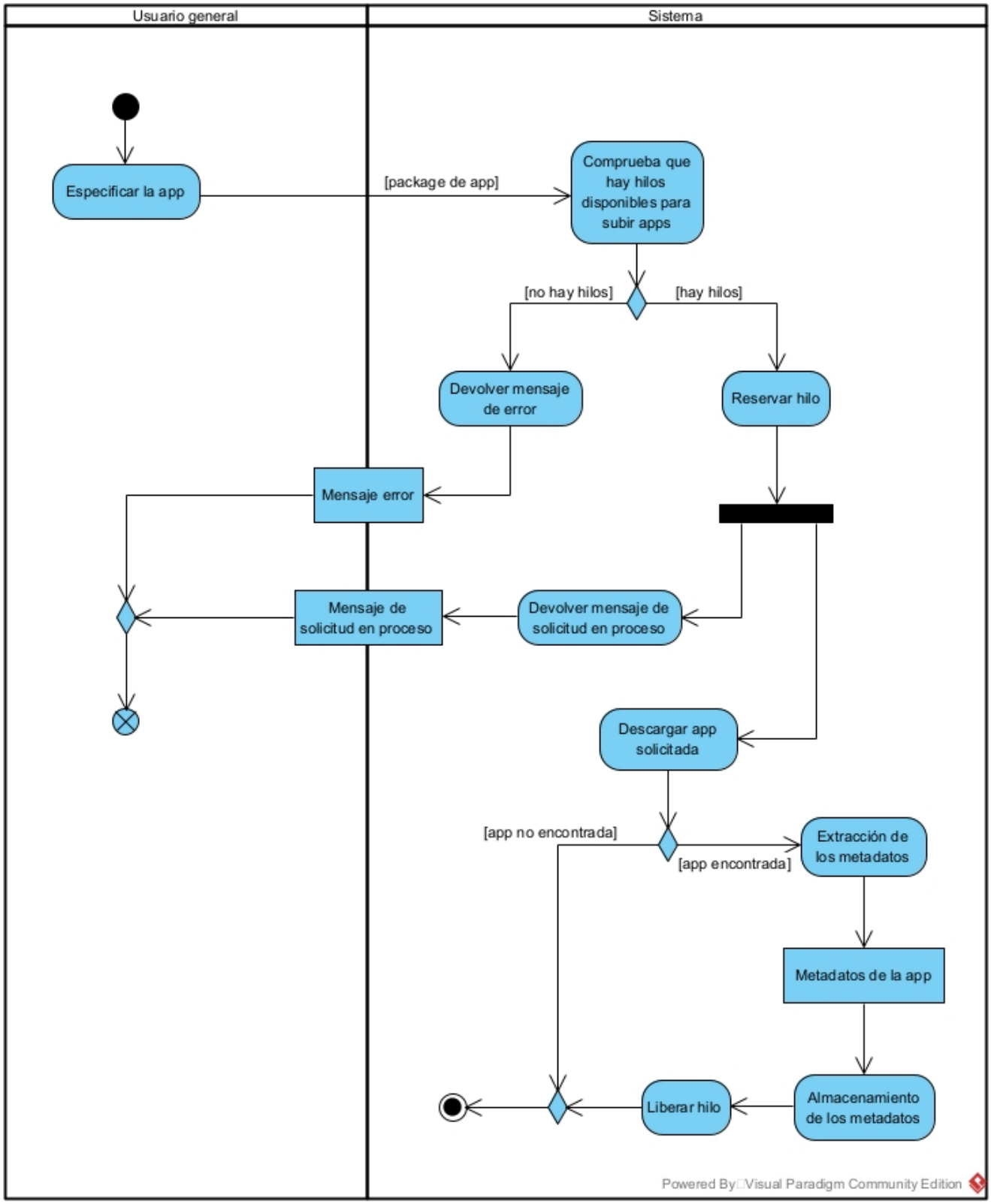

Figura 4.3: Diagrama de actividades del UC-2.

4.3.3. UC-3: Carga datos

En la tabla 4.3 se encuentra la especificación del caso de uso de actualizar el *warehouse* y en la figura 4.4 se encuentra el diagrama de actividades asociado a este caso de uso. En este caso de uso se

entenderán como datos todos aquellos enumerados en la especificación de requisitos de información de la sección 4.2.1.

UC-3	**Carga manual de datos**
Descripción	El sistema carga manualmente los datos aportados por el administrador.
Precondición	El administrador ha de estar autenticado.
Secuencia normal	1. El administrador introduce los datos que desea cargar. 2. El sistema comprueba que los datos son válidos. 3. El sistema almacena los datos aportados. 4. El sistema informa al administrador de que los datos se han almacenado.
Flujos alternativos	2a-1. El sistema comprueba que los datos no son válidos. 2a-2. El sistema informa de que los datos no son válidos al administrador. 2a-3. El caso de uso queda sin efecto.
Postcondición	Los datos quedan almacenados en el *warehouse*.

Tabla 4.3: Caso de uso: Carga datos.

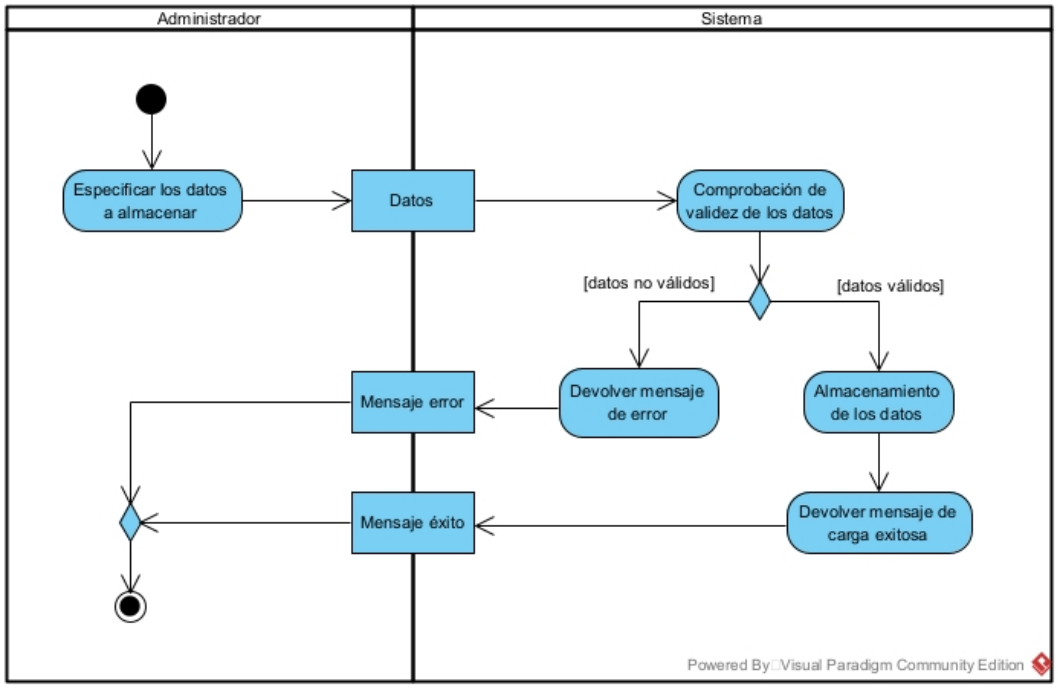

Figura 4.4: Diagrama de actividades del UC-3.

4.3.4. UC-4: Solicitar actualización del *warehouse*

En la tabla 4.4 se encuentra la especificación del caso de uso de actualizar el *warehouse* y en la figura 4.5 se encuentra el diagrama de actividades asociado a este caso de uso.

UC-4	**Solicitar actualización del *warehouse***
Descripción	El sistema actualiza el *warehouse* en función de la opción escogida por el administrador.
Precondición	El administrador ha de estar autenticado.
Secuencia normal	1. El administrador escoge solicitar la actualización de los datos AOSP, solicitar la aplicación de las métricas de privacidad o solicitar la carga masiva de n aplicaciones. 2. El sistema comprueba que no se esta cursando ya la orden. 3. El sistema informa al administrador de que la petición se esta procesando. 4. El sistema procesa la orden solicitada por el administrador. 5. El sistema almacena la información descargada.
Flujos alternativos	2a-1. El sistema comprueba que ya se esta cursando la orden. 2a-2. El sistema informa al administrador de que el sistema se encuentra ocupado. 2a-3. El caso de uso queda sin efecto.
Postcondición	El *warehouse* queda actualizado en función de la opción escogida por el administrador.

Tabla 4.4: Caso de uso: Solicitar actualización del *warehouse*.

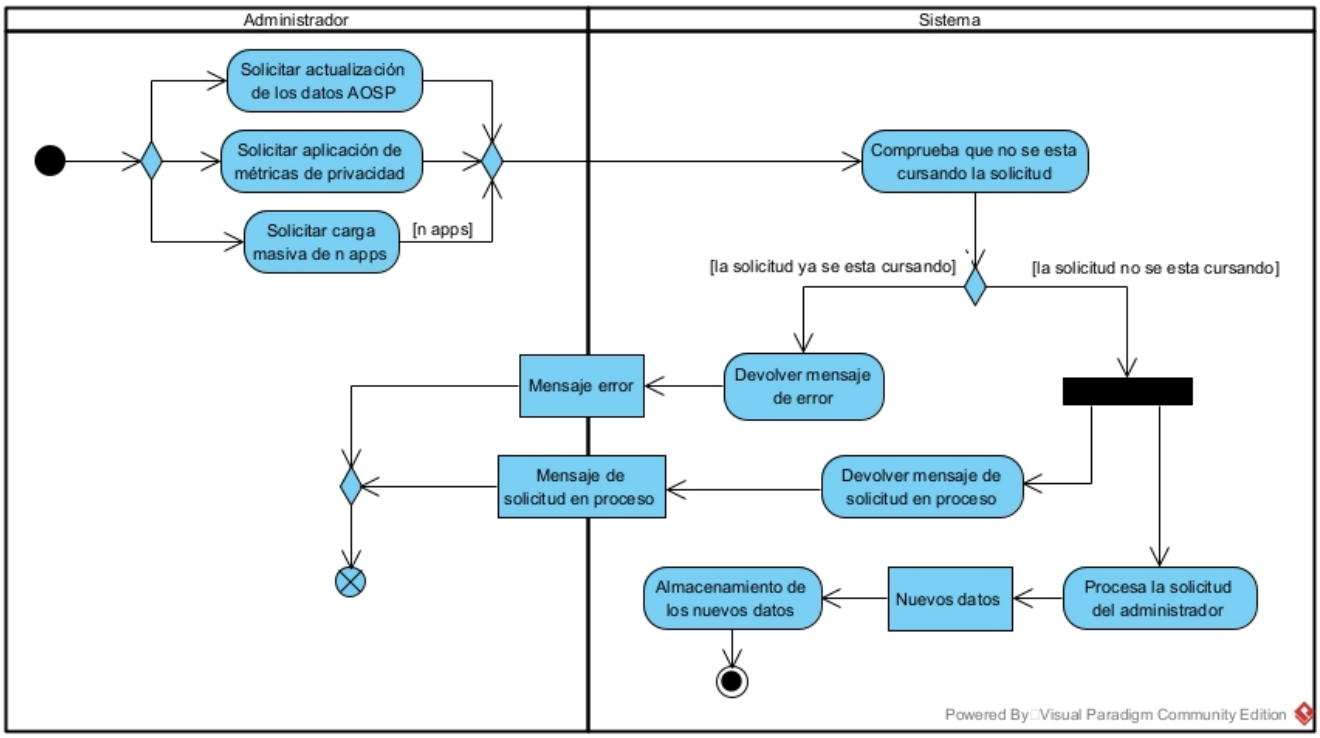

Figura 4.5: Diagrama de actividades del UC-4.

Capítulo 5

Diseño

En este capítulo, se realiza el diseño de todos los componentes necesarios para cumplir con los requisitos *software* elicitados en el capítulo 4. En esencia, se documenta todo el diseño *software* del *warehouse* desarrollado y las justificaciones de las decisiones de diseño tomadas.

Todo el diseño se documenta con diagramas UML (*Unified Modeling Language*) utilizando el programa *Visual Paradigm 17.0 Community Edition*. Estos diagramas, se realizan en inglés para facilitar el cumplimiento de las guías de estilo [30], [31], su internacionalización y reutilización, entre otros.

5.1. Principales componentes del sistema

Partiendo de los objetivos enunciados en la sección 1.3, se puede realizar una descomposición del *warehouse* en componentes más sencillos que ayudan diseño del sistema.

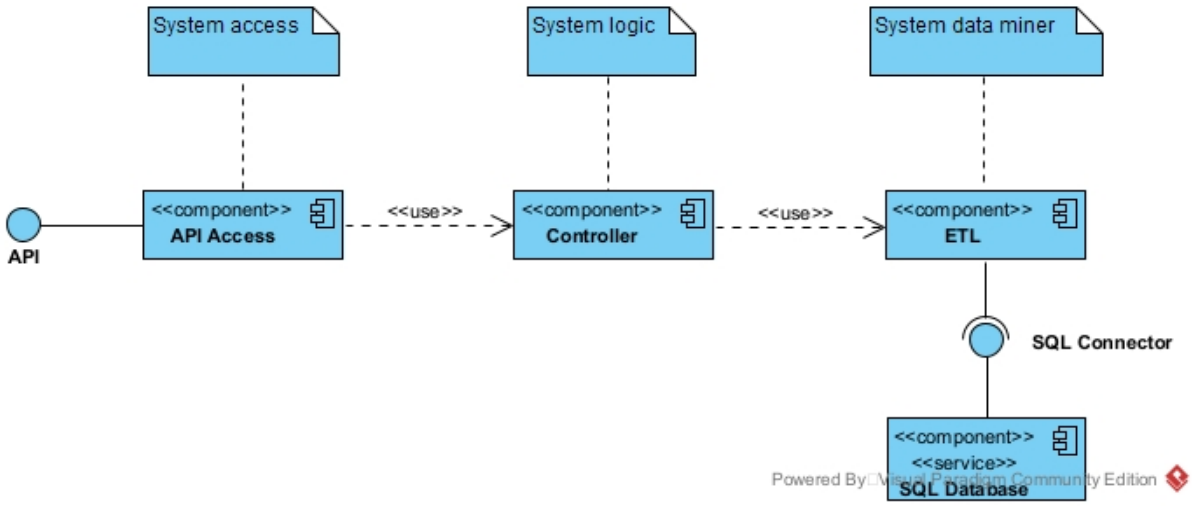

Figura 5.1: Diagrama de componentes.

Los componentes en los que se descompone el *warehouse* que aparecen en la figura 5.1 son:

- Un componente que de acceso al *warehouse* (*API Access*). Este componente aporta un acceso uniforme al contenido del *warehouse* permitiendo a otras aplicaciones y/o servicios de terceros lo

utilicen además de las personas[1].

- Un componente que controle el *warehouse* (*Controller*). Este componente se encarga de llevar a cabo las acciones necesarias para satisfacer las peticiones que realizan los usuarios al *warehouse* así como las acciones propias de su administración como por ejemplo, nutrirlo de aplicaciones.

- Un componente que lleve a cabo la obtención de datos de los que se nutrirá el *warehouse* (*ETL*[2]). Este componente se encarga de integrar las diversas fuentes de datos en un único repositorio central, el *warehouse*.

- Un componente que gestiona el almacenamiento persistente de todos los datos (*SQL Database*). Este componente se encarga de almacenar todos los datos que se requieran para el funcionamiento del *warehouse* así como para la satisfacción de los requisitos del mismo.

5.2. Diseño arquitectónico del *warehouse*

El diseño de la arquitectura software del *warehouse* se centra en el cumplimiento de una serie de características que aportan calidad al sistema [32], [33]:

- **Modularidad.** El sistema ha de dividirse en componentes o módulos independientes para facilitar su desarrollo, mantenimiento y reutilización.

- **Escalabilidad.** El diseño arquitectónico debe permitir que el sistema sea escalable, es decir, el crecimiento futuro. En este contexto, se tendrá en especial interés la capacidad de añadir o eliminar componentes del sistema como nuevas fuentes de datos.

- **Flexibilidad.** El diseño arquitectónico ha de permitir la incorporación de nueva funcionalidad o la modificación de la ya existente sin afectar a otras partes del sistema. Por ejemplo, la modificación de la obtención de datos de una fuente concreta para adaptarse a los cambios de la misma.

- **Desacoplamiento.** Los componentes deben depender lo mínimo posible entre si para facilitar la reutilización y mantenimiento de cada uno de ellos. Esto es de interés ya que la funcionalidad que aporta el *warehouse* es interesante para otro tipo de sistemas como por ejemplo, un sistema extractor de metadatos de las aplicaciones.

Teniendo en cuenta esta serie de características y partiendo del diagrama de los principales componentes definido en la sección 5.1, el diseño arquitectónico al que se ha llegado se encuentra en la figura 5.2.

[1] Usuarios del *warehouse*: persona, aplicación o servicio de terceros que hace uso del sistema.

[2] ETL es el acrónimo de Extract Transform y Load, el proceso habitual para combinar datos de múltiples fuentes en un solo repositorio central (el *warehouse*).

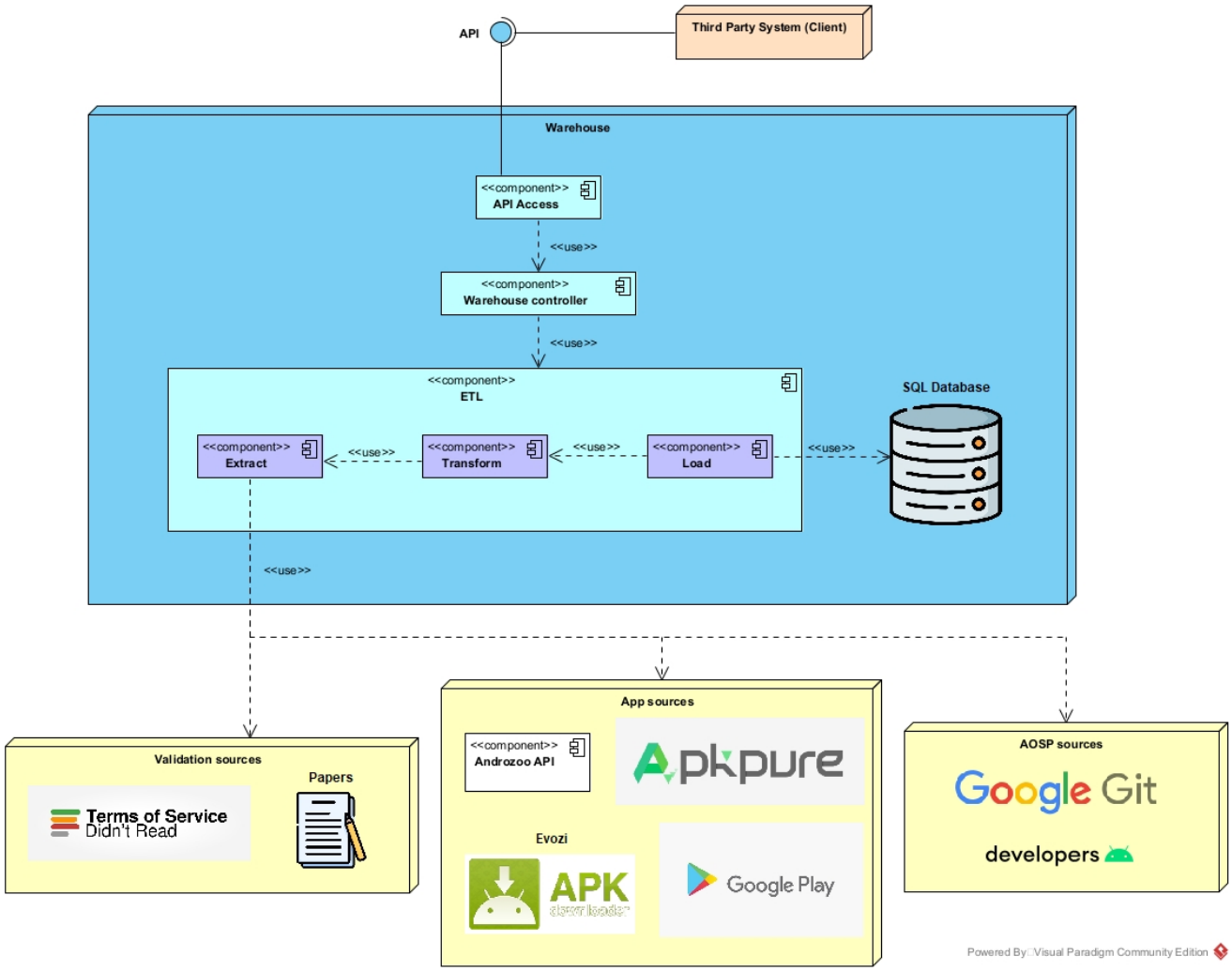

Figura 5.2: Diagrama de arquitectura del *warehouse*.

En el diseño arquitectónico del *warehouse* realizado se aplica principalmente el patrón arquitectónico de capas abiertas ya que permite alcanzar las características de modularidad, escalabilidad y desacoplamiento deseadas.

Se aplica el patrón en los principales componentes del *warehouse* definidos en la sección 5.1: *API Access*, *Warehouse controller* y *ETL*. De este modo, se desacoplan los 3 componentes añadiendo modularidad al sistema. Esto permitirá al sistema poder ampliarse con otros componentes extra, como por ejemplo otro componente de acceso al *warehouse* que podría ser una interfaz gráfica.

Además, se aplica el patrón *data pipeline* en los componentes del componente encargado de la obtención de los datos de los que se nutre el *warehouse* (*ETL*). Esta solución de diseño es habitualmente utilizada para el diseño de sistemas de procesamiento de datos [34]. El primero de los componentes, el de extracción (*Extract*), se dedica a la extracción de los datos de las diversas fuentes como por ejemplo, páginas web por medio de *scraping*. El segundo de los componentes, el de transformación (*Transform*), se dedica a aplicar las transformaciones necesarias como la normalización de los datos o enriquecimiento de los mismos, entre otras. Por último, el componente de carga (*Load*), carga los datos en el destino final. En nuestro caso, en una base de datos estructurada.

5.3. Modelo de datos

El modelado de datos es importante en el diseño y la construcción de un *warehouse* ya que al definir y estructurar los datos que se van a almacenar, prevenimos errores futuros. Esto es gracias a que permite la identificación y resolución temprana de problemas en la estructura de los datos que, previsiblemente, causarían grandes atrasos en el proyecto al momento de llevar a cabo la implementación.

En este caso en concreto puesto que el objetivo clave del trabajo es el estudio de la privacidad de aplicaciones móviles, se ha de almacenar información acerca de las aplicaciones en si que se estudiaran así como de los permisos que estas utilizan. De este modo, tendremos la información necesaria para poder aplicar la métrica [3].

Es por esto que, en esta sección, se realiza el modelado de datos del *warehouse* llevando a cabo los diagramas: conceptual, conceptual detallado y físico de los datos [35].

5.3.1. Modelo de datos conceptual

En primer lugar, se realiza el diagrama entidad relación conceptual para representar los elementos relevantes a almacenar así como sus relaciones. Este diagrama se lleva a cabo utilizando la notación de Chen [36] publicada en el 1976 puesto que con esta, se ve claramente el tipo de entidad fuerte o débil (rectangulo simple o doble respectivamente) y relación identificativa o no (diamante doble o simple respectivamente) que estamos representando.

Como se aprecia en el diagrama de la figura 5.3, y como habíamos avanzado anteriormente, apreciamos una entidad central fuerte *app* que representa una aplicación concreta.

Ya que otro de los objetivos del *warehouse* es la integración de datos de diversas fuentes heterogéneas, es necesario la inclusión de la entidad débil *extraction_metadata* respecto a *app* donde se almacenará información acerca de la fuente y método de extracción de la aplicación. Esta relación, *has_extraction_metadata*, será identificativa ya que la entidad débil *extraction_metadata* no puede existir sin *app* y, además, su identificación depende de la *app* concreta.

De modo similar surge la entidad débil *score* y su respectiva relación *has_score* con *app*. En este caso, la información a almacenar será la puntuación otorgada por una métrica concreta a la *app*. Dicha métrica queda definida por la entidad fuerte *privacy_rank*, asociada a través de la relación identificativa *source_of_score*.

Nos encontramos también, con la entidad fuerte *permission_group* que representa grupos de permisos [37]. Estos, son declarados por aplicaciones por medio de la relación no identificativa *app_defines_group* ya que tanto *app* como *permission_group* son entidades fuertes totalmente independientes que no han de estar relacionadas para existir.

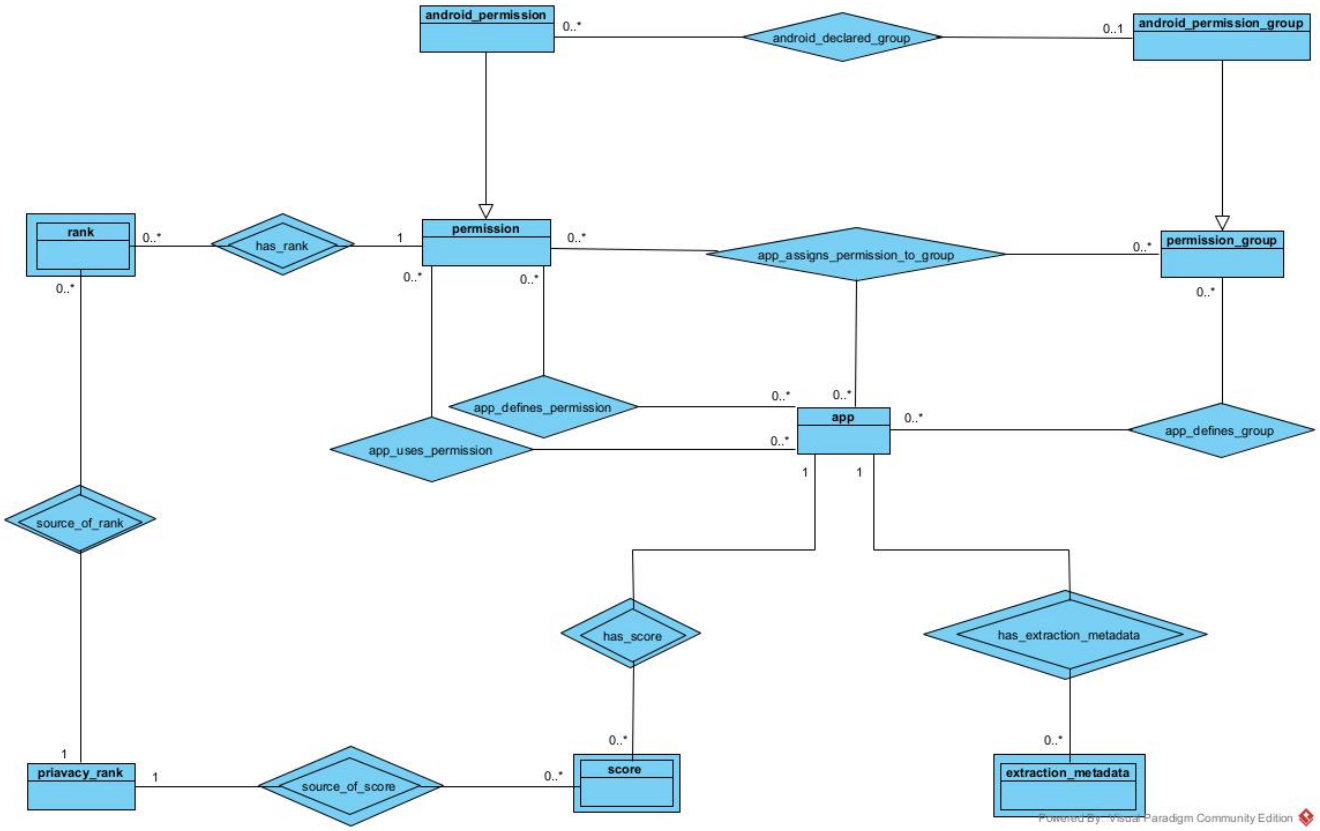

Figura 5.3: Diagrama conceptual (notación Chen).

Además, una especialización de *permission_group*, *android_permission_group* para los grupos de permisos declarados por el sistema Android [38]. Estos, agrupan permisos declarados por el sistema a través de la relación no identificativa *android_declared_group* ya que no todos los permisos declarados por el sistema han de pertenecer a un grupo.

Por otro lado, nos encontramos con la entidad fuerte *permission* la cual representa un permiso en Android [13]. Estos, pueden ser definidos por una applicación o usados por la misma a través de sus relaciones no identificativas *app_defines_permission* y *app_uses_permission* respectivamente ya que ambas entidades son independientes.

Los permisos tienen asignada una clasificación que mide el nivel de intrusión del permiso. Dicha clasificación proviene de una métrica concreta por lo que se trata como una entidad débil *rank* ya que sin la clasificación y el permiso no puede existir. Se asocia de manera identificativa con permiso y la clasificación a través de la relación *has_rank* y *source_of_rank* respectivamente.

Una especialización de permiso, *android_permission* la cual representa un permiso en Android definido por el sistema [39]. Estos permisos son los más utilizados por las aplicaciones puesto que son los que otorgan acceso a otras partes del sistema como la cámara, la localización o incluso, el almacenamiento interno entre otros lo que justifica su especialización.

Las aplicaciones asignan los permisos a grupos de permisos por medio de la relación no identificativa *app_bind_permission_to_group* ya que como ya hemos visto, tanto *app* como *permission* como *permission_group* pueden existir de manera independiente sin estar necesariamente relacionados.

5.3.2. Modelo de datos conceptual detallado

En segundo lugar, refinaremos el diagrama de la figura 5.3 añadiendo los atributos de cada entidad. Para ello, mantendremos la misma notación utilizada [36] en el nuevo diagrama contenido en la figura 5.4.

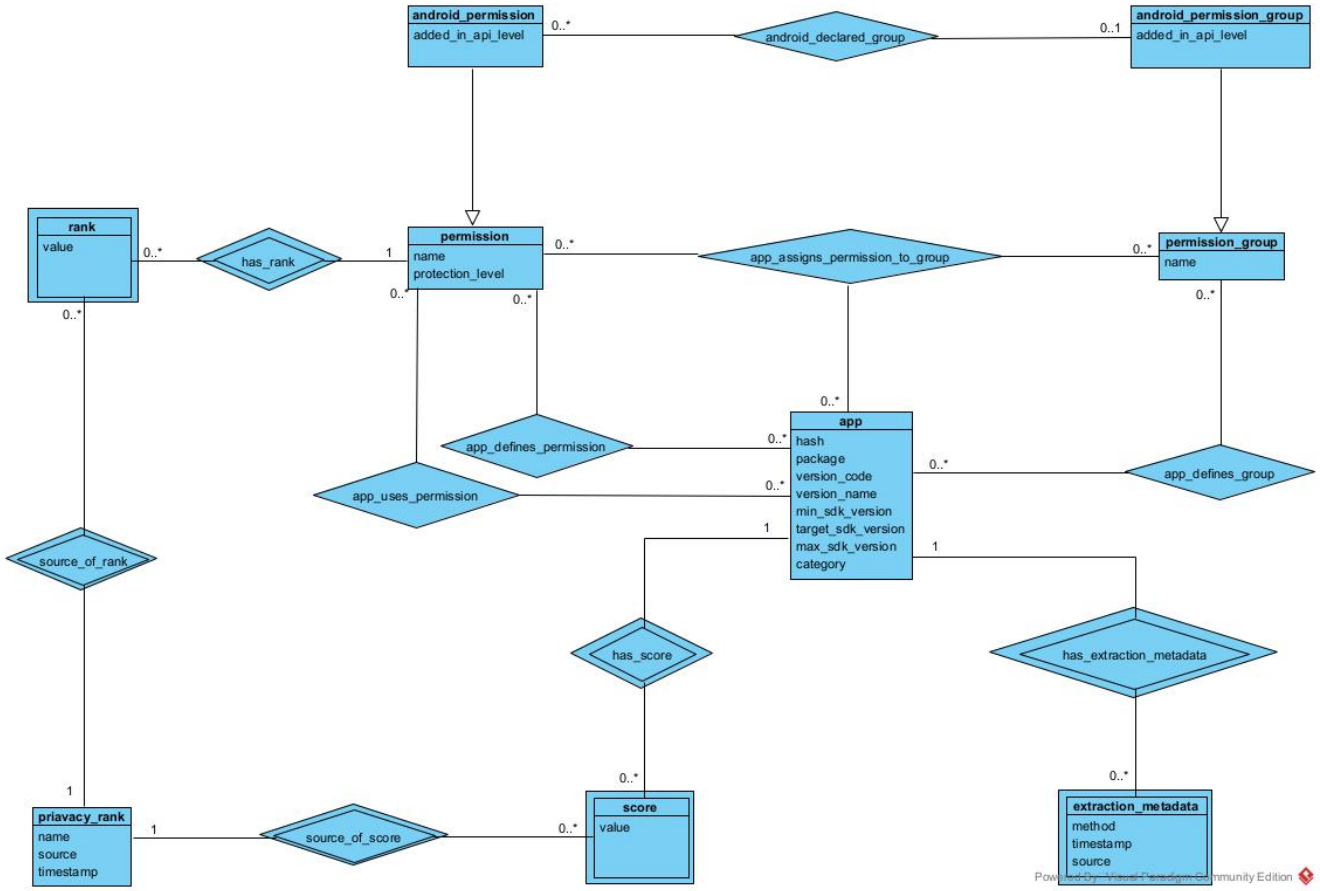

Figura 5.4: Diagrama conceptual detallado (notación Chen).

En vistas de la figura 5.4, se desglosa en tablas los atributos de cada entidad así como una descripción de lo que significan.

En cuanto a la aplicación, almacenaremos información útil para su identificación (tabla 5.1).

app	
hash	SHA256 hash de la apk que nos servirá para identificarla inequívocamente
package	Nombre del paquete
version_code	Versión numérica de la aplicación
version_name	Versión en formato cadena de caracteres de la aplicación
min_sdk_version	Versión mínima de sdk en el que puede ejecutarse la aplicación
target_sdk_version	Versión esperada de sdk para ejecutar la aplicación
max_sdk_version	Versión máxima de sdk en el que pude ejecutarse la aplicación
category	Categoría de la aplicación

Tabla 5.1: Atributos de *app*.

En cuanto a los datos de extracción (tabla 5.2).

extraction_metadata	
method	Método de extracción de la aplicación
timestamp	Marca temporal de la extracción de la aplicación
source	Fuente de extracción de la aplicación

Tabla 5.2: Atributos de *extraction_metadata*.

Atributos de la puntuación de la aplicación (tabla 5.3).

score	
value	Valor de puntuación obtenido para la aplicación normalizado entre 0 y 10

Tabla 5.3: Atributos de *score*.

Atributos de los grupos de permisos (tabla 5.4).

permission_group	
name	Nombre del grupo de permisos

Tabla 5.4: Atributos de *permission_group*.

Atributos de los grupos de permisos definidos en el sistema Android (tabla 5.5).

android_permission_group	
added_in_api_level	Nivel de api en que se añadió el grupo de permisos

Tabla 5.5: Atributos de *android_permission_group*.

Atributos de los permisos (tabla 5.6).

permission	
name	Nombre del permiso
protection_level	Nivel de protección del permiso

Tabla 5.6: Atributos de *permission*.

Atributos de los permisos definidos en el sistema Android (tabla 5.7).

android_permission	
added_in_api_level	Nivel de api en que se añadió el permiso

Tabla 5.7: Atributos de *android_permission*.

Atributos de la clasificación otorgada a un permiso (tabla 5.8).

rank	
value	Valor de clasificación del permiso definido por la clasificación

Tabla 5.8: Atributos de *rank*.

Atributos de una clasificación de privacidad (tabla 5.9).

privacy_rank	
name	Nombre identificativo de la clasificación
source	Fuente de la clasificación
timestamp	Marca temporal de cuando se creó la clasificación

Tabla 5.9: Atributos de *privacy_rank*.

5.3.3. Modelo de datos físico

Por último, se lleva a cabo el modelo físico que se implementa. En este caso, se utiliza la notación pie de gallo [40] ya que al no utilizar diamantes en las relaciones es más compacta y, permite la inclusión del tipo de dato que se utiliza para cada atributo sin incrementar el tamaño del diagrama. Es necesario tener en cuenta la tecnología que se utiliza en concreto en la implementación para poder realizar un diagrama fiel a la verdadera implementación. En este caso, se utiliza el sistema gestor de bases de datos *MySQL*. Además, se tiene en cuenta la guía de estilo publicada por Simon Holywell [31] sobre las convenciones de nombres en lenguajes *SQL*.

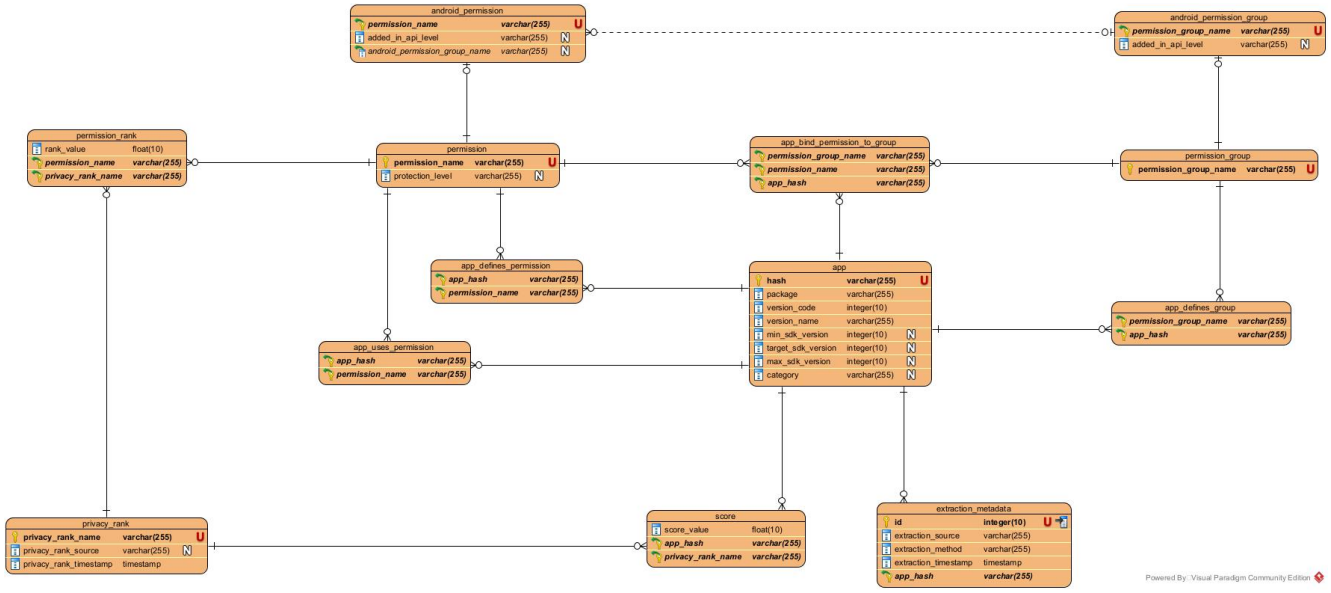

Figura 5.5: Diagrama físico (notación pie de gallo).

Como se aprecia en el diagrama de la figura 5.5, en este se hace una traducción 1:1 del modelo conceptual detallado de la figura 5.4 a excepción de las relaciones entre entidades fuertes las cuales se realizan mediante la inclusión de una tabla intermedia que referencia las dos que contendrán las entidades fuertes. En cuanto al tipo de relación, en este caso ahora todas son identificativas a excepción de la de *android_permission* con *android_permission_group* (con línea punteada) ya que las entradas en las tablas intermedias no pueden existir sin que existan las entidades correspondientes que esta relacionando.

Se ha decidido utilizar como clave primaria (representada con una llave al lado) de cada tabla su atributo identificativo a excepción de la entidad débil *extraction_metadata* donde se ha tenido que incluir un identificador numérico para poder identificar cada entrada. Las claves foráneas se encuentran representadas con una flecha verde que indica que apunta fuera de la tabla que la contiene.

En cuanto a los tipos de datos utilizados, se utilizan enteros para los identificadores, números de tipo *float* para las clasificaciones y valoraciones y cadenas de 255 caracteres para las cadenas de texto. También se utiliza el tipo de dato *MySQL timestamp* para las marcas temporales, estas se representan siguiendo el formato “1990-01-01 01:01:01.00000”. Además, se representa con una “U” en rojo los atributos que han de ser únicos, con una “N” en blanco los atributos que pueden tomar el valor nulo y con un icono de una hoja en blanco señalada los atributos que indican el índice en cada tabla.

Finalmente, se aprecia que hay nombres de tablas y atributos que no concuerdan con los del diagrama conceptual detallado contenido en la figura 5.4 como: *rank*, *value*, *source*, *timestamp*, *name* y *method*. Estos, han tenido que ser modificados añadiendo como prefijo el nombre de su tabla o en el caso de *rank* el de la entidad fuerte con la que se enlaza (*permission*) ya que son palabras reservadas en *SQL* y, es buena práctica no utilizarlas para facilitar la portabilidad del código generado entre los distintos sistemas gestores de bases de datos *SQL*.

5.4. Diseño del proceso *ETL*

En esta sección se detalla el diseño del flujo general de datos del *warehouse* así como los distintos procesos *ETL* que gestionan el flujo de datos de cada una de las fuentes que se ven en el diagrama de la figura 5.2 al repositorio central. Las fuentes concretas que se utilizan en función del tipo de dato que se extrae de ella son:

- **Aplicaciones:** *Androzoo* [8] como única fuente *API* y *Apkpure* (`https://m.apkpure.com/es/`), *Evozi* (`https://apps.evozi.com/apk-downloader/`), *Apkmonk* (`https://www.apkmonk.com/`) y *Apkfollow* (`https://www.apkfollow.com/es/`) como fuentes *web*.
- **Metainformación sobre permisos y grupos de permisos Android:** se utilizan las páginas *web* de documentación de los desarrolladores de Android (`https://developer.android.com/`) y de control de versiones del código fuente de Android (`https://android.googlesource.com/`).
- **Datos de validación:** se utiliza el servicio *API* de *Tosdr* [10] y distintos *papers* [6], [7].

Para realizar los diagramas de flujo de datos se utiliza la herramienta *draw.io* de manera excepcional en esta sección.

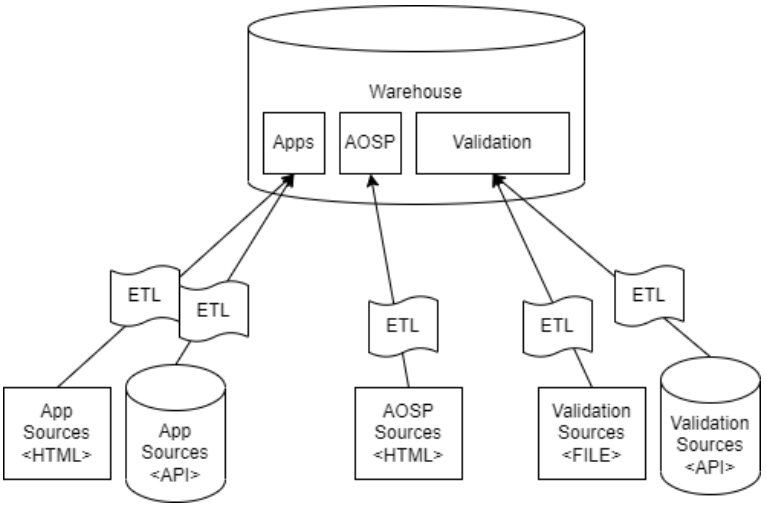

Figura 5.6: Visión global del *warehouse* y sus fuentes.

En el diagrama de la figura 5.6 se ve pueden ver los distintos tipos de datos que almacena el *warehouse* vistos anteriormente en la sección 5.3 del modelado de datos. Estos tipos de datos son: los metadatos de las aplicaciones, la metainformación sobre los permisos y grupos de permisos del sistema operativo Android (*AOSP*[3]) y los datos que sirven de validación para las métricas de privacidad. Además, se ve como cada tipo de fuente de datos tiene un proceso *ETL* asociado.

En todos los casos de *scraping web*, nos encontramos con un grave problema y es que las páginas *web* son dinámicas, es decir, no mantienen la misma estructura de clases en distintas sesiones. Es por ello que se replantean los *scraping* como búsquedas de los enlaces que queremos aplicando expresiones

[3] AOSP es el acrónimo de *Android Open Source Project*

regulares sobre todos los enlaces ya que los enlaces a recursos suelen estar estructurados pues son *APIs* las que los sirven.

5.4.1. Diseño detallado del flujo de datos de *Apps*

Para las fuentes de aplicaciones, podemos apreciar 2 tipos diferentes de fuentes de datos: a través de *API* y a través de páginas *web*. En ambos casos, una vez extraída la aplicación empaquetada (archivo *.apk*) el flujo de transformación y carga es el mismo.

Una vez se tiene el archivo *.apk* se obtiene la categoría de la aplicación de *Play Store* como se ve en la figura 5.7. En primer lugar, se carga la página *web* de la aplicación de *Play Store*. Seguidamente, se extraen todos los enlaces presentes en la página *web* filtrandose estos por los que se encuentran visibles en esta y por la expresión regular ".*category.*". Finalmente, obtenemos la categoría de la *app* filtrando en enlace que nos queda con la expresión regular ".*/CATEGORY$". Donde "CATEGORY" es la categoría buscada.

Figura 5.7: Flujo de datos de obtención de la categoría de una aplicación.

Una vez se tiene la categoría y el archivo *.apk*, ya se pueden juntar los datos y subirlos al *warehouse* tal y como se ve en la figura 5.8.

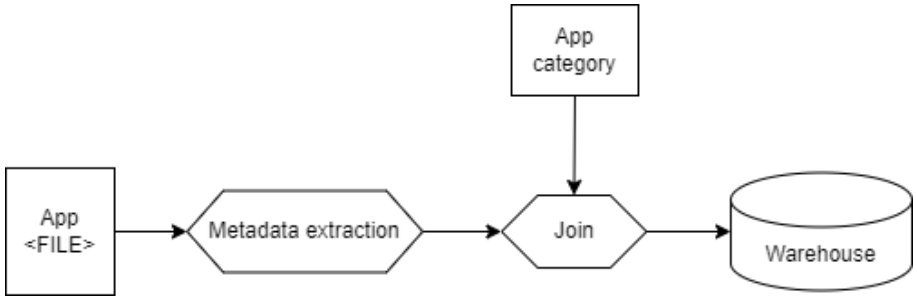

Figura 5.8: Flujo de datos de subida de una *app* al *warehouse*.

Diseño detallado de la extracción de la aplicación para las fuentes *API*

En específico cuando la fuente del archivo de la aplicación es una *API*, únicamente se necesita realizar la petición a dicha *API* de la aplicación que queremos, obteniéndose así el fichero directamente.

Diseño detallado de la extracción de la aplicación para las fuentes *web*

Ahora, se diferenciará el flujo de datos de la obtención del archivo de la aplicación dependiendo de la fuente de datos *web* concreta.

Apkpure, Apkfollow* y *Apkmonk

En este caso, la estructura base del flujo de datos es el mismo para todas y cada una de las fuentes solamente diferenciándose por las expresiones regulares que se utilizan para el filtrado de los enlaces.

Como se ve en la figura 5.9, el proceso se inicia con el nombre del paquete de la aplicación que queremos descargar. Con el, se carga la página de consulta de cada *web*:

- *Apkpure*: `https://m.apkpure.com/es/search?q=app.package`
- *Apkfollow*: `https://www.apkfollow.com/search?q=app.package`
- *Apkmonk*: `https://www.apkmonk.com/ssearch?q=app.package`

Seguidamente, se extrae la página *web* de la aplicación que buscamos aplicando las siguientes expresiones regulares a todos los enlaces:

- *Apkpure*: ".*/download" y ".*app.package.*".
- *Apkfollow* y *Apkmonk*: ".*/app/.*" (el primer enlace de la página).

Con el enlace de la aplicación, se carga y se extrae el enlace de descarga con las siguientes expresiones regulares:

- *Apkpure*: ".*?version=latest.*".
- *Apkfollow*: ".*/download/.*" (el primer enlace de la página).
- *Apkmonk*: ".*/download-app/.*" (el primer enlace de la página).

Por último, se carga el enlace de descarga obteniendo asi el archivo de la aplicación que queremos.

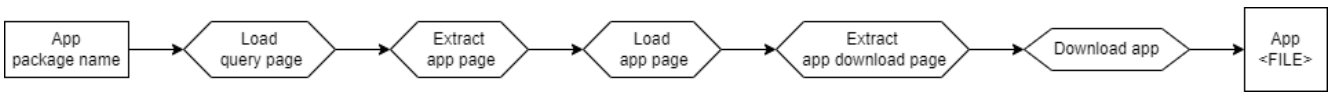

Figura 5.9: Flujo de datos de extracción de una *app*.

Evozi

En este caso, como se aprecia en la figura 5.10, simplemente se carga la página de consulta, seguidamente se hace *click* en el botón que genera el enlace de descarga encontrándolo por aquel que tiene el texto "Generate DownIoad Link", se extrae el enlace de descarga por aquel que contiene el patrón ".*.apk.*" y, por último, se carga dicho enlace para descargar la aplicación.

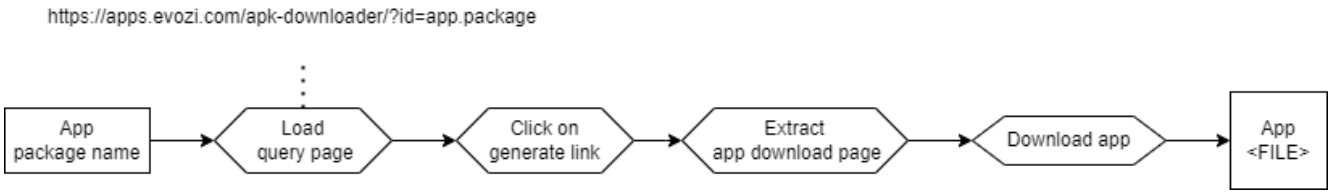

Figura 5.10: Flujo de datos de extracción de una *app* de *Evozi*.

5.4.2. Diseño detallado del flujo de datos de *AOSP*

Para la extracción de datos de los permisos y grupos de permisos del sistema operativo Android se aplica el mismo flujo de datos con dos diferencias dependiendo de si nos encontramos ante permisos o grupos de permisos.

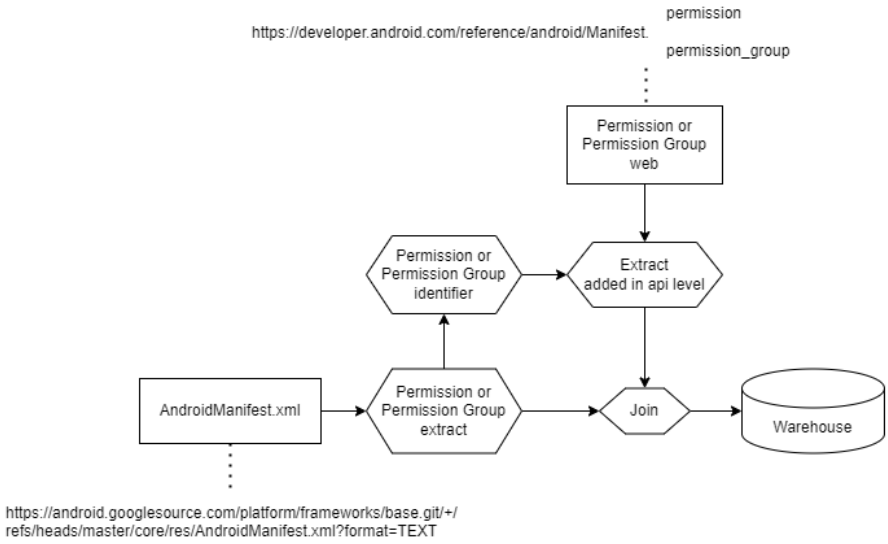

Figura 5.11: Flujo de datos AOSP.

En la figura 5.11 se ve dicho flujo de datos. En primer lugar, se parte del archivo *AndroidManifest.xml* descargado de la página *web* mencionada en la figura 5.11. Seguidamente se extraen los metadatos sobre los permisos o grupos de permisos en función de la etiqueta *xml* por el que se busque (*permission* o *permission-group*). Con el identificador único del permiso o grupo de permisos que se obtiene con el atributo *xml name*, extraemos de la página *web* de los desarrolladores de Android el nivel de *API* en que se añadió dicho permiso o grupo de permisos extrayendo el texto del elemento *web* cuyo identificador (*id*) es el identificador único que tenemos. Juntamos la información obtenida del *xml* con el nivel de *API* y ya podemos cargar la información al *warehouse*.

5.4.3. Diseño detallado del flujo de datos de validación

El flujo de datos es trivial tal y como se aprecia en la figura 5.12. Se basa en buscar en la *API* (*Tosdr* [10]) mediante una consulta o en el archivo [6], [7], si dicha fuente aporta una puntuación para nuestra aplicación, se normaliza entre 0 y 10 y, se carga al *warehouse*.

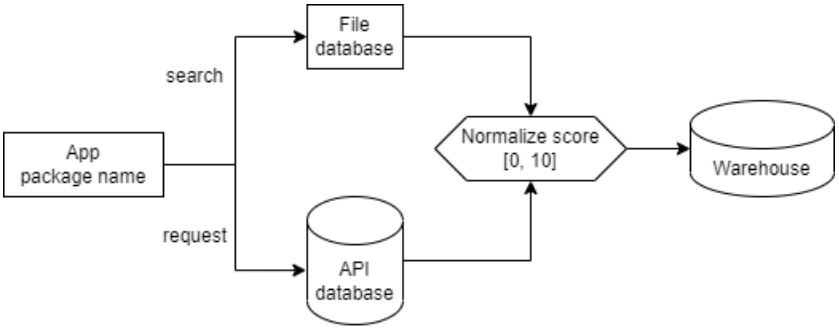

Figura 5.12: Flujo de datos de validación.

5.5. Diseño detallado del *warehouse*

En esta sección se desglosa el diseño de todo el *warehouse*. Para ello, se lleva a cabo un diseño descendente (*top-down*) ya que esta aproximación facilita la comprensión del diseño al centrarse primero en la visión general del sistema e ir descomponiéndose poco a poco [33].

El diseño detallado general del sistema partiendo del diseño arquitectónico anteriormente enunciado se encuentra en la figura 5.13. En esta respecto al diseño arquitectónico, todos los componentes se han traducido en paquetes. Además, se ha añadido un paquete *common* el cual contendrá el modelo de dominio (*domain*) que será una traducción 1:1 del modelo de datos ya explicado en la sección 5.3 y otro paquete *util* el cual contendrá funcionalidad miscelanea de utilidad. Además, se han juntado los componentes *cotroller*, *etl* y el paquete *common*.

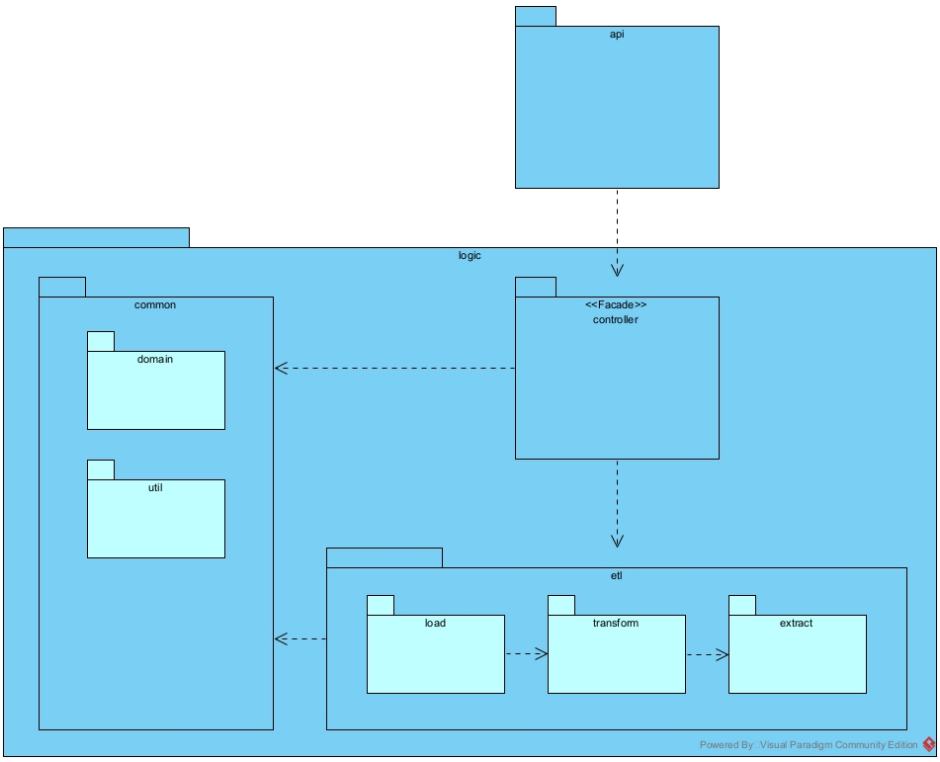

Figura 5.13: Diagrama de paquetes general del *warehouse*.

Este diseño se centra en desacoplar la *api* de acceso al *warehouse* de la lógica de negocio del sistema la cual se encuentra en el paquete *controller* y del paquete *etl* que contiene toda la funcionalidad sobre el proceso de obtención, transformación y carga de los datos. Para ello, se mantiene el patrón capas abiertas entre la *api* y *logic*. Y, en el interior de esta segunda capa, se utiliza un patrón a capas abiertas en la que se añade a mayores una capa transversal *common* con los elementos del dominio (conocido a veces como patrón a capas con orientación al dominio [41]). Además, se aplica el patrón fachada en el paquete *controller* que sirve como único punto de acceso a la funcionalidad del *warehouse* para la *api*. Dentro del paquete *etl* se utiliza el patrón *data pipeline* entre *extract*, *transform* y *load*.

A lo largo del diseño, se utilizan los patrones de diseño *Singleton* y *DataMapper* así como la creación de interfaces y generalizaciones. De este modo, se consigue un diseño que cumple con los requisitos de calidad del diseño enumerados en la sección 5.2.

5.5.1. Diseño detallado del paquete *common*

Diseño detallado del paquete *domain*

Este paquete contiene todos los elementos del modelo de datos. Es necesario para que nuestro sistema tenga una representación *software* de los mismos para que los pueda tratar. Para representar toda esta información, se realizan dos diagramas: uno de clases detallado contenido en la figura 5.15 y otro de herencia contenido en la figura 5.14 ya que toda la información en un único diagrama dificultaría su comprensión. En estos diagramas se han omitido los constructores puesto que para todas las clases estos serán con todos sus atributos y estarán sobrecargados para que se puedan inicializar también a través de un diccionario con sus atributos en forma clave/valor.

Como se aprecia en la figura 5.14, el paquete contiene una interfaz *Element*. Esta interfaz define la funcionalidad que ha de tener un elemento del dominio del sistema ayudando así a la escalabilidad y desacoplamiento del sistema gracias a hacer uso de la interfaz en vez de los objetos concretos. Esta interfaz es implementada por todos los elementos del dominio. Además, los permisos y grupos de permisos definidos por el sistema operativo Android heredan de permiso y grupo general donde también se incluyen los permisos y grupos que definen otras aplicaciones del mismo modo que pasaba en el modelo de datos.

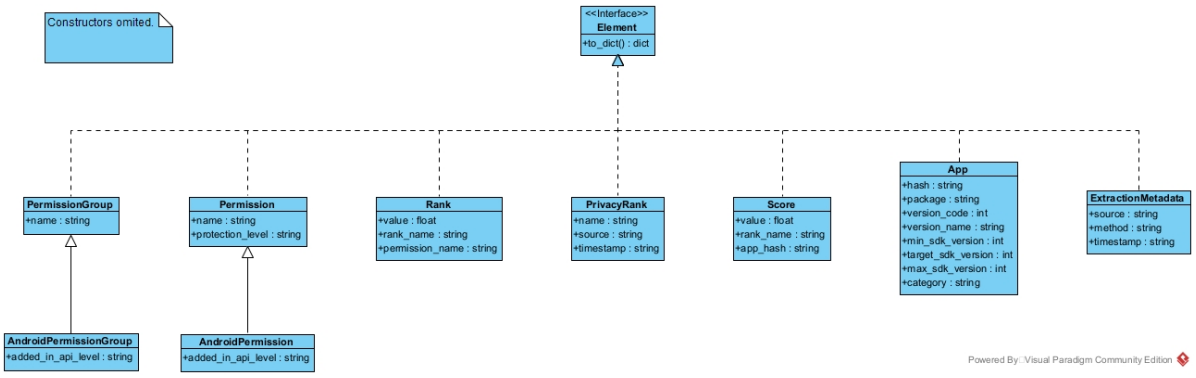

Figura 5.14: Diagrama de herencia de *domain*.

En cuanto al diagrama de la figura 5.15, se aprecian todos los elementos del dominio junto con sus atributos, su tipo y el sentido de navegabilidad entre ellos. La navegabilidad escogida se centra en un tomar como punto de partida los objetos que contienen a los otros definiéndose así el sentido. Teniendo esto en cuenta, se definen las mismas relaciones que ya se enumeraron en el modelo de datos.

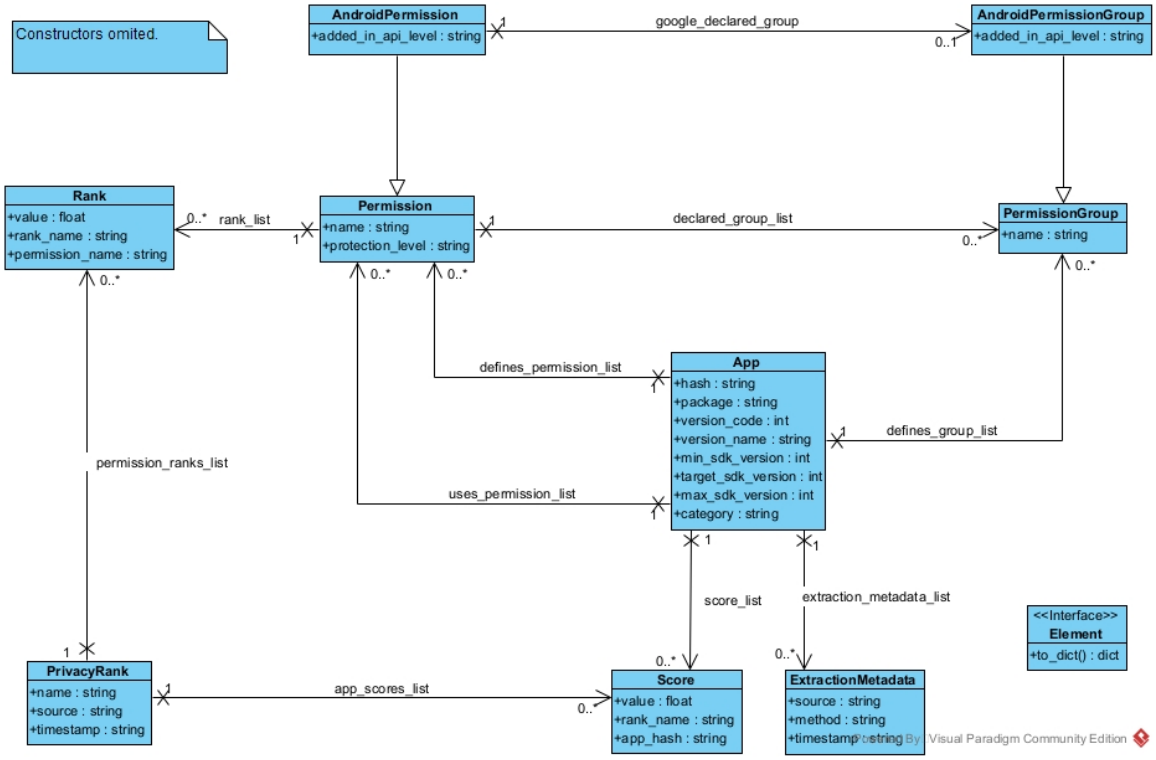

Figura 5.15: Diagrama de paquetes de *domain*.

Por último, la funcionalidad que ha de implementar todos los elementos del dominio al implementar la interfaz *Element* es una traducción de si mismo al tipo de datos diccionario (*to_dict*). Se utiliza para la traducción de los elementos del dominio a *JSON* que facilita la interopelabilidad del *warehouse* con el resto de sistemas de terceros. La definición del *JSON* de cada objeto se hace de la siguiente manera:

1. Cada atributo simple se almacena como una entrada clave/valor del *JSON* usando como nombre de clave el nombre del atributo que aparece en los diagramas y como valor el valor del atributo.

2. Cada relación con otro elemento del dominio, se almacenará como una entrada clave/valor usando como nombre de clave el nombre de la relación que aparece en los diagramas y como valor el objeto *JSON* de dicho elemento.

3. Los atributos que sean una lista de elementos ya sean sencillos o complejos, se trata como una lista de estos siguiendo las indicaciones de los dos puntos anteriores.

4. Por último, se encapsula este objeto *JSON* obtenido de aplicar los pasos anteriores en otro *JSON* con clave el nombre del elemento del dominio.

Un ejemplo práctico de estas reglas aplicadas para la traducción de una aplicación (*App*) tomando como ejemplo "*Whatsapp*" se encuentra en el apéndice A.

Diseño detallado del paquete *util*

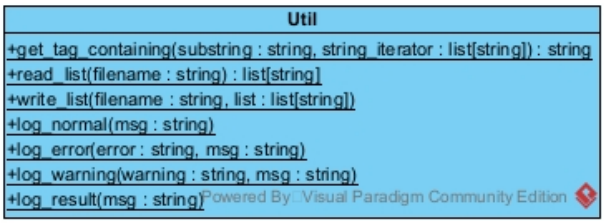

Figura 5.16: Diagrama de paquetes de *util*.

En este caso, el paquete será una única clase *Util*. Esta clase contiene funcionalidad para leer y escribir listas en ficheros de texto, encontrar el *string* de una lista que contiene otro y por último, un sistema de *logs* para poder monitorizar el uso del *warehouse*.

5.5.2. Diseño detallado del paquete *etl*

Diseño detallado del paquete *extract*

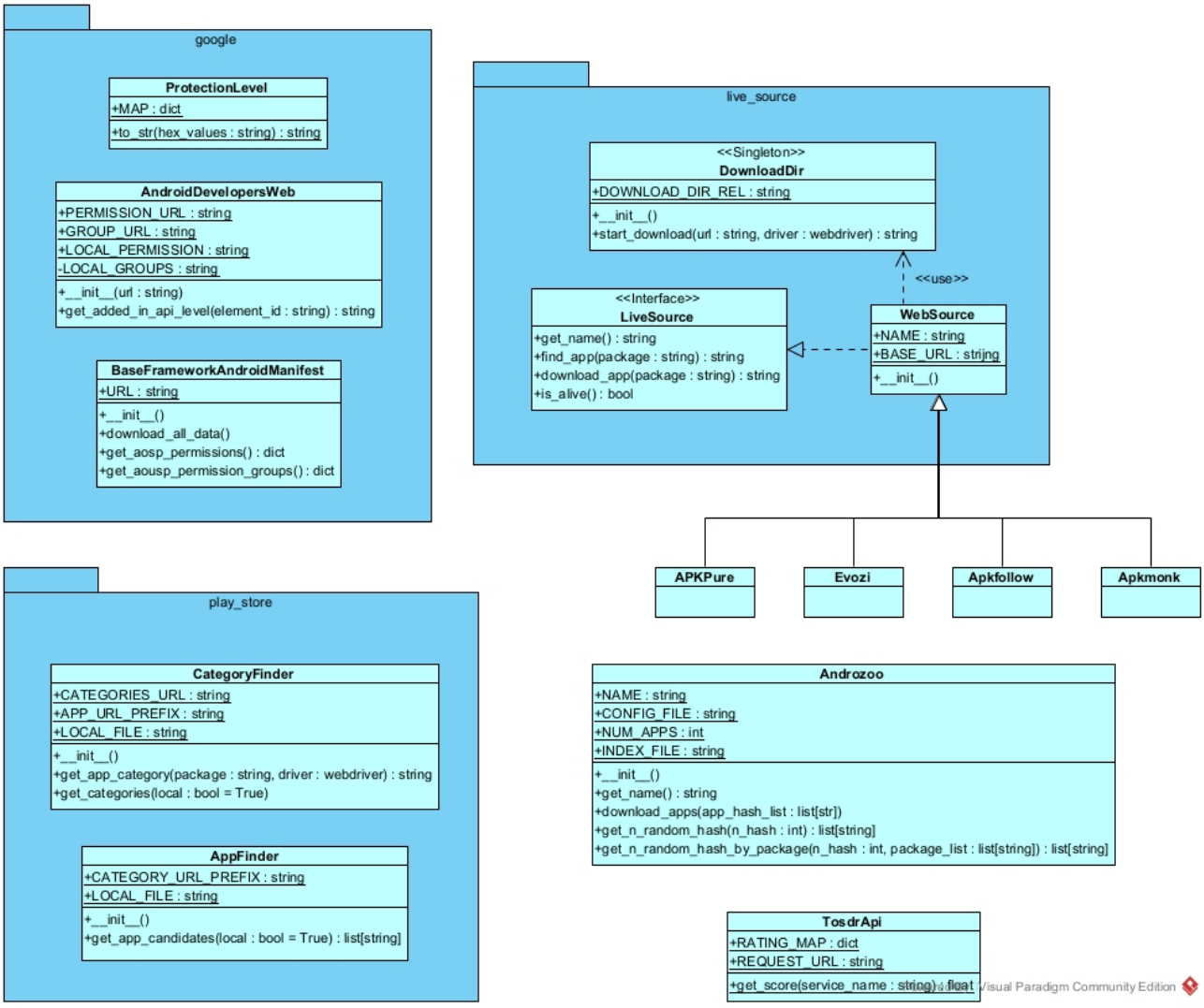

Figura 5.17: Diagrama de paquetes de *extract*.

En este paquete, se encuentra toda la funcionalidad que permite la obtención de datos de distintas fuentes:

Paquete *google*

En el paquete *google* se encuentra la obtención del nivel de *api* en que se añadió un permiso o grupo de permisos (*get_added_in_api_level*) al sistema Android de la web de desarrolladores de Android (`https://developer.android.com/`) y la obtención de los permisos y grupos de permisos del sistema Android (en la clase *BaseFrameworkAndroidManifest*) del *git* del código Android (`https://android.googlesource.com/`). Además, ya que el *protection_level* de los permisos se encuentra almacenado en hexadecimal, también se encuentra el mapeado de estos datos a cadena de carácteres en la clase *ProtectionLevel*.

Paquete *play_store*

En este paquete se encuentra la funcionalidad encargada de la obtención de la categoría de una aplicación (*get_app_caregory*) así como la obtención de las aplicaciones más populares de cada categoría (*get_app_candidates*).

Paquete *live_source*

En este paquete, se encuentra la funcionalidad para descargar *apps* de fuentes síncronas, es decir, en vivo. Puesto que hay múltiples fuentes de aplicaciones que han de funcionar concurrentemente, se plantea un problema para la gestión de las descargas provenientes de diversas fuentes. Por ello, el directorio donde se descargan las *apps* es manejado por la clase *DownloadDir* que sigue el patrón de diseño *Singletone*. Es decir, un único objeto es el que inicia las descargas (*start_download*) evitando así que las distintas fuentes de datos accedan al directorio pudiendo modificar archivos que no son suyos.

A mayores, se tiene la interfaz *LiveSource* la cual han de implementar todas las fuentes de datos síncronas. Las únicas operaciones que una fuente de datos síncrona ha de tener para poder integrarse en el sistema son: la obtención del nombre de la fuente (*get_name*), la búsqueda de *apps* en la fuente por el nombre de paquete (*find_app*), la descarga de las *apps* (*download_app*) y una función que compruebe si la fuente es utilizable (*is_alive*). Esta interfaz facilita la excalabilidad del sistema permitiendo añadir futuras fuentes de datos síncronas.

Por último, se tiene la clase *WebSource* que representa una fuente *web* de *apps*. Esta, implementa la interfaz *LiveSource* puesto que es una fuente de datos síncrona y además, utiliza la clase *DownloadDir* ya que utilizará el directorio de descargas para descargar las *apps*.

Paquete *extract*

A mayores, se tienen las clases sueltas: *APKPure*, *Evozi*, *Apkfollow* y *Apkmonk* que son especializaciones de *WebSource* donde se encuentran los *scraping* concretos de sus *webs* para la obtención de las aplicaciones.

También la clase *Androzoo* en la cual se encuentra funcionalidad para buscar aplicaciones en su repositorio [8] así como descargarlas. Esta fuente de datos no utiliza el directorio de descarga de fuentes en vivo ya que no lo es, se utilizará únicamente como fuente masiva asíncrona de aplicaciones.

Por último, la clase *TosdrApi* utiliza la *API* de acceso al servicio de `https://tosdr.org/` de donde se obtienen calificaciones de la intrusividad de las aplicaciones. Una de las fuentes de validación de las calificaciones de las aplicaciones.

Diseño detallado del paquete *transform*

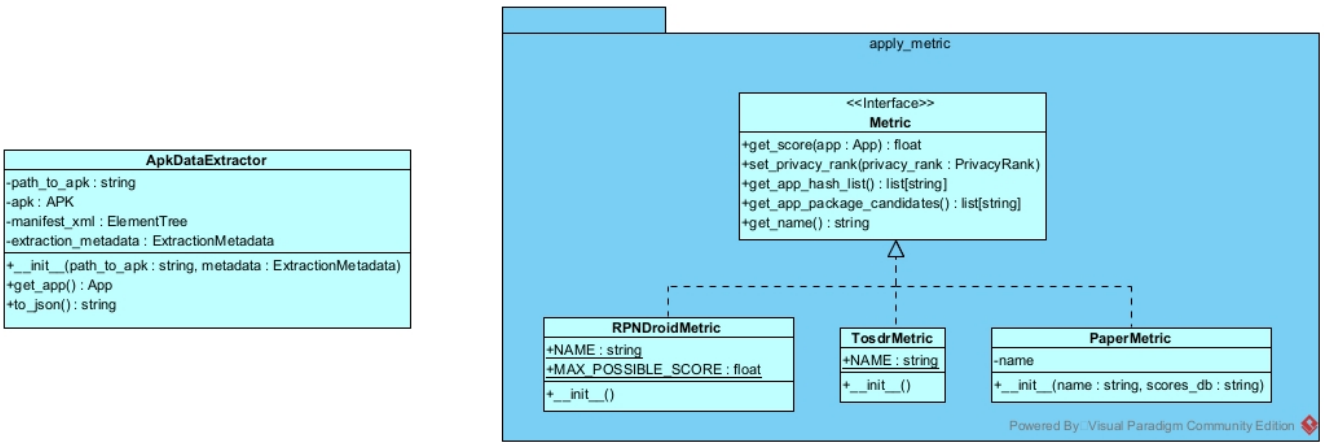

Figura 5.18: Diagrama de paquetes de *transform*.

En este paquete, se encuentra la clase *ApkDataExtractor* que contiene la funcionalidad necesaria para la extracción de los metadatos relativos a la privacidad de una aplicación (*get_app*) mencionadas en la sección 5.3 del modelo de datos.

También se encuentra el paquete *apply_metric* en el que se encuentran todas las métricas de privacidad definidas. Es decir, las métricas que se quieren probar [3] como las de validación de esta [6], [7], [10]:

- En primer lugar, una interfaz (*Metric*) que define la funcionalidad que ha de implementar una métrica. Esta interfaz facilita la escalabilidad del sistema ya que para añadir futuras métricas solo será necesario que estas la implementen. La funcionalidad que han de aportar las métricas es: una función que puntué las aplicaciones (*get_score*), otra que asocie al objeto métrica la métrica almacenada (*set_privacy_rank*), otra que devuelva los *hash* de las aplicaciones que ya tienen la métrica aplicada (*get_app_hash_list*), otra que devuelva los nombres de los paquetes de las *apps* que pueden utilizar la métrica para ser puntuada (*get_app_package_candidates*) y otra que devuelva el nombre de la métrica (*get_name*).

- En segundo lugar, la clase *RPNDroidMetric* que implementa la métrica [3], la clase *Tosdr* que utiliza la métrica [10] y *PaperMetric* que utiliza datos de *papers* como [6], [7] para puntuar las aplicaciones.

Diseño detallado del paquete *load*

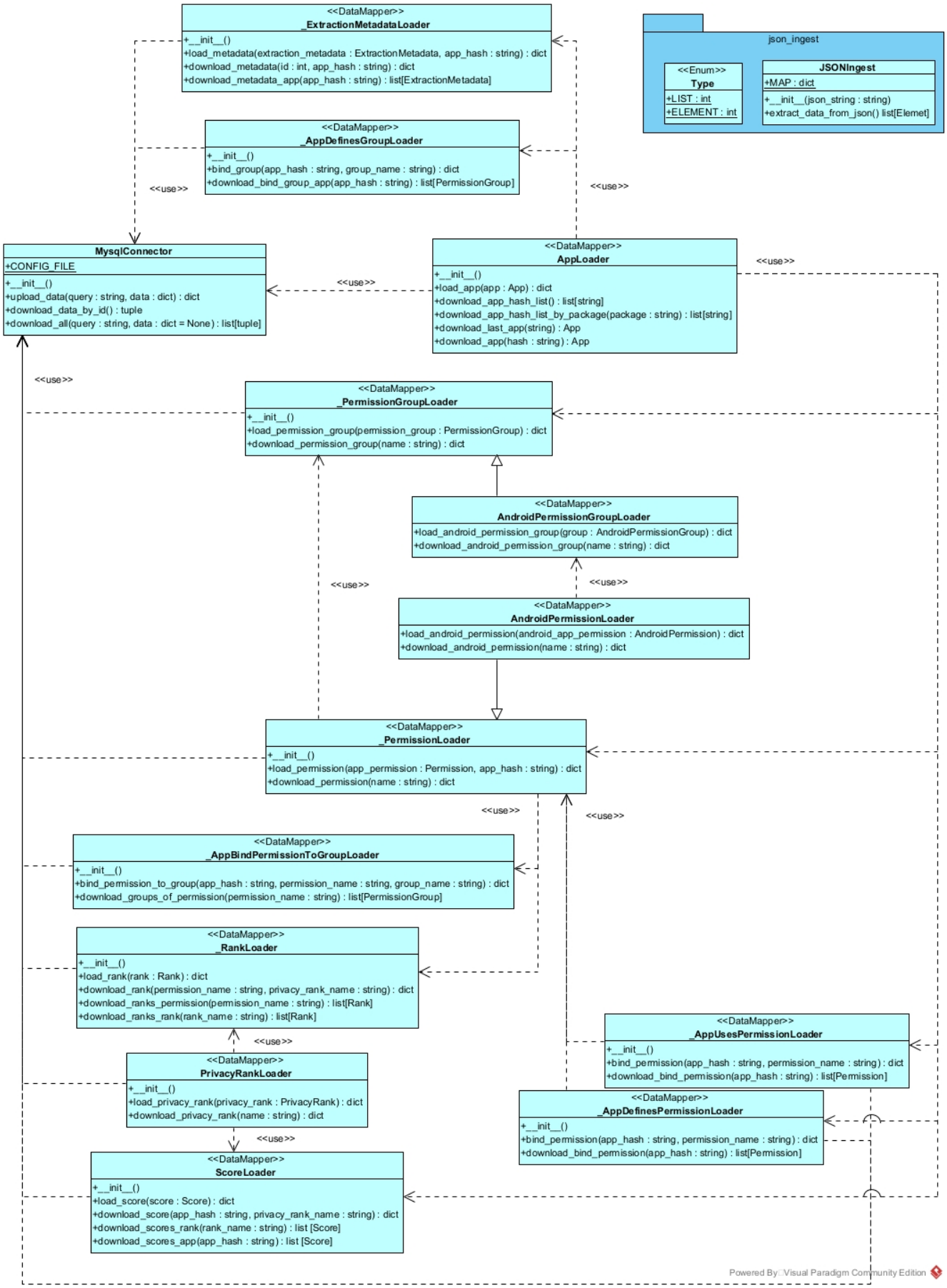

Figura 5.19: Diagrama de paquetes de *load*.

En este caso, en la figura 5.19 hay dos zonas diferenciadas, el paquete *json_ingest* y el resto de clases que se encuentran en el paquete *load*.

Paquete *json_ingest*

Es el encargado de cargar datos contenidos en formato *JSON* a elementos del dominio (*domain*). Para ello, la funcionalidad que aporta es la extracción de todos los elementos del dominio mencionados en la sección 5.3 de modelo de datos contenidos en el *JSON* con la función *extract_data_from_json*. El formato de cada uno de los elementos es el ya mencionado en la sección 5.5.1 del diseño del dominio (*domain*).

Paquete *load*

Del resto de clases que nos encontramos en el paquete *load* también tenemos dos agrupaciones lógicas de las mismas:

- *MysqlConnector*. Es el conector a la base de datos estructurada donde se almacena la información de manera persistente. Aporta funcionalidad para ejecutar consultas de manera segura evitando fallos comunes de seguridad como *SQL Injection* (SQLi) [42].
- El resto de clases son una aplicación del patrón de diseño *Data Mapper*. Son una asociación 1:1 de todos y cada uno de los elementos del dominio así como sus relaciones contenidas en la sección 5.3 del modelo de datos. El cometido de cada una de estas clases es abstraer los elementos del dominio de la base de datos. De este modo, se mejora la flexibilidad y mantenibilidad del sistema ya que se permite cambiar la estructura de la base de datos sin afectar a los objetos del dominio así como una mejor cohesión ya que los elementos del dominio únicamente se centran en la lógica de negocio sin estar acoplados a la lógica de persistencia.

5.5.3. Diseño detallado del paquete *controller*

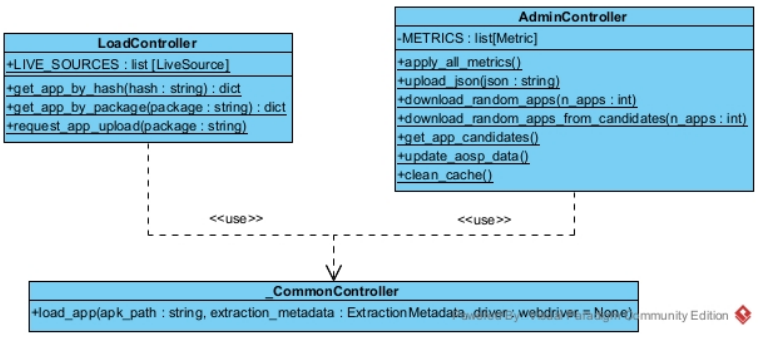

Figura 5.20: Diagrama de paquetes de *controller*.

Este paquete como ya se mencionó anteriormente contiene la lógica de negocio necesaria para llevar a cabo la funcionalidad citada en el capítulo 4 de análisis. Para ello, se desglosa en dos clases: la funcionalidad de los usuarios generales (*LoadController*) y la de los usuarios administradores (*AdminController*). Como ambos suben aplicaciones, se abstrae dicha lógica en una clase (*_CommonController*) que facilita la reutilización del código.

Dentro de la funcionalidad de los usuarios generales, nos encontramos la definición de todas las fuentes síncronas que queremos utilizar (*LIVE_SOURCES*), una función que busca en el *warehouse* aplicaciones según su *hash* (*get_app_by_hash*), una función que busca en el *warehouse* aplicaciones según su nombre de paquete (*get_app_by_package*) y una función que solicita la subida de una nueva aplicación al *warehouse* por su nombre de paquete (*request_app_upload*).

Por otro lado, dentro de la funcionalidad de los administradores nos encontramos esencialmente: la definición de todas las métricas que queremos utilizar (*METRICS*), una función que permite aplicar todas las métricas que queremos utilizar (*apply_all_metrics*), una función que permite subir datos al *warehouse* a través de su representación *JSON* (*upload_json*), dos funciones que permiten la descarga de aplicaciones de forma masiva de *Androzoo* [8] (*download_random_apps* y *download_random_apps_from_candidates*), una función que permite la obtención de los nombres de los paquetes más populares de cada categoría de *Google Play* (*get_app_candidates*), una función que permite actualizar la información acerca de los permisos y grupos de permisos del sistema Android (*update_aosp_data*) y una función que permite la limpieza de los directorios de datos temporales como el de descarga síncrona de aplicaciones (*clean_cache*).

5.5.4. Diseño detallado de la *api*

Se diseña la *api* enfocándola en la funcionalidad que aporta. Para ello, se utiliza el estándar *OpenAPI* [25] para documentar el diseño de la misma. Gracias al uso de este estándar, facilitamos la interoperabilidad del *warehouse* con otros sistemas de terceros ya que este estándar es ampliamente adoptado en la industria para documentar la estructura y comportamiento de las *APIs*.

Se muestran todas las posibles peticiones a la *api* categorizadas según tipo de usuario y tipo de funcionalidad que aporta. Para cada petición a la *api* se muestra el tipo de petición (*GET* o *POST*), la dirección (*endpoint*) de esta, los parámetros que tiene y las posibles respuestas. Todas las peticiones cuyo mensaje respuesta sea el estado de la solicitud tendrán el mismo formato de mensaje respuesta:

```
{
    "status": "possible.status"
}
```

Su estado (*status*) al realizarlas pudiendo ser este: "*busy*", "*requested*", "*inactive*" o "*successful*" dependiendo de si la petición no se puede llevar a cabo porque ya se han alcanzado los límites de uso del *warehouse*, se va a llevar a cabo, no se esta llevando a cabo o se ha realizado con éxito.

Funcionalidad de usuario general

Para poder usar todas estas peticiones, se ha de autenticar el usuario por medio de su *API_KEY* la cual se aportará en una cabecera de la petición *"Authorization"* con valor *"Bearer API_KEY"* tal y como indica el estándar de seguridad utilizado (*OAuth2PasswordBearer*) [43].

Funcionalidad de descarga de aplicaciones del *warehouse*:

- ***GET /get/app/hash***

  Descripción: petición síncrona que solicita la descarga de una aplicación del *warehouse* especificado su *SHA256 hash*.

  Parámetros de la petición:

Nombre	**Descripción**
hash	*SHA256 hash* de la aplicación que se quiere descargar

Tabla 5.10: Parámetros de la petición *GET /get/app/hash*.

  Respuesta de la petición:

Código	**Descripción**
200	aplicación en formato *JSON* (sección 5.5.1)
404	la aplicación no se encuentra en el *warehouse*
422	la petición formulada no es válida
401	autenticación no válida

Tabla 5.11: Respuestas de la petición *GET /get/app/hash*.

- ***GET /get/app/package***

  Descripción: petición síncrona que solicita la descarga de una aplicación del *warehouse* especificado su nombre de paquete.

  Parámetros de la petición:

Nombre	**Descripción**
package	*package_name* de la aplicación que se quiere descargar

Tabla 5.12: Parámetros de la petición *GET /get/app/package*.

  Respuesta de la petición:

Código	**Descripción**
200	aplicación en formato *JSON* (sección 5.5.1)
404	la aplicación no se encuentra en el *warehouse*
422	la petición formulada no es válida
401	autenticación no válida

Tabla 5.13: Respuestas de la petición *GET /get/app/package*.

Funcionalidad de subida de aplicaciones al *warehouse*:

- ***POST /post/app/package***

  Descripción: petición asíncrona que solicita la subida de una aplicación al *warehouse* especificado su nombre de paquete.

  Cuerpo de la petición:

```
{
    "package": "package.name.of.application"
}
```

  Respuesta de la petición:

Código	Descripción
200	estado de la petición en formato *JSON*
400	el formato del cuerpo de la petición no es válido
401	autenticación no válida

Tabla 5.14: Respuestas de la petición *POST /post/app/package*.

Funcionalidad de usuario administrador

Para poder usar todas estas peticiones, se ha de autenticar el administrador por medio de su *API_KEY* la cual aportará en una cabecera de la petición *"Authorization"* con valor *"Bearer API_KEY"* tal y como indica el estándar de seguridad utilizado (*OAuth2PasswordBearer*) [43].

Funcionalidad de actualización del *warehouse*:

- ***GET /admin/update/aosp***

  Descripción: petición asíncrona que solicita la actualización de los datos de permisos y grupos de permisos del sistema Android del *warehouse*.

- ***GET /admin/update/aosp/status***

  Descripción: petición síncrona que solicita el estado de la actualización de los datos de permisos y grupos de permisos del sistema Android del *warehouse*.

- ***GET /admin/update/app_candidates***

  Descripción: petición asíncrona que solicita la actualización de la lista de aplicaciones más populares por categoría de *Play Store* del *warehouse*.

- ***GET /admin/update/app_candidates/status***

  Descripción: petición síncrona que solicita el estado de la actualización de la lista de aplicaciones más populares por categoría de *Play Store* del *warehouse*.

- ***GET /admin/update/apply_metrics***

  Descripción: petición asíncrona que solicita la aplicación de todas las métricas disponibles a las aplicaciones que no las tengan aplicadas.

- ***GET /admin/update/apply_metrics/status***

  Descripción: petición síncrona que solicita el estado de la aplicación de todas las métricas disponibles a las aplicaciones que no las tengan aplicadas.

- ***GET /admin/get/n_random_apps/status***

  Descripción: petición síncrona que solicita el estado de la descarga de *n* aplicaciones de forma aleatoria.

- ***GET /admin/get/n_random_app_candidates/status***

  Descripción: petición síncrona que solicita el estado de la descarga de *n* aplicaciones entre las más populares por categoría de *Play Store*.

  Todas las peticiones anteriores al no tener parámetros tienen las mismas respuestas posibles:

Código	**Descripción**
200	estado de la petición en formato *JSON*
401	autenticación no válida

Tabla 5.15: Respuestas de las peticiones de actualización del *warehouse*.

- ***GET /admin/get/n_random_apps***

  Descripción: petición asíncrona que solicita la carga de *n* aplicaciones de forma aleatoria al *warehouse*.

  Parámetros de la petición:

Nombre	**Descripción**
n_apps	número de aplicaciones que se quieren descargar

Tabla 5.16: Parámetros de la petición *GET /admin/get/n_random_apps*.

Respuesta de la petición:

Código	**Descripción**
200	estado de la petición en formato *JSON*
422	la petición formulada no es válida
401	autenticación no válida

Tabla 5.17: Respuestas de la petición *GET /admin/get/n_random_apps*.

- ***GET /admin/get/n_random_app_candidates***

  Descripción: petición asíncrona que solicita la carga de *n* aplicaciones entre las más populares por categoría de *Play Store* al *warehouse*.

Parámetros de la petición:

Nombre	Descripción
n_apps	número de aplicaciones que se quieren descargar de cada candidato

Tabla 5.18: Parámetros de la petición *GET /admin/get/n_random_app_candidates*.

Respuesta de la petición:

Código	Descripción
200	estado de la petición en formato *JSON*
422	la petición formulada no es válida
401	autenticación no válida

Tabla 5.19: Respuestas de la petición *GET /admin/get/n_random_app_candidates*.

- ***POST /admin/post/json***

  Descripción: petición síncrona que solicita la subida de los datos contenidos en el cuerpo de la solicitud al *warehouse*.

  Cuerpo de la petición: elemento o lista de elementos del dominio bien formados según su definición *JSON* realizada en la sección 5.5.1.

  Respuesta de la petición:

Código	Descripción
200	estado de la petición en formato *JSON*
400	el formato del cuerpo de la petición no es válido
401	autenticación no válida

Tabla 5.20: Respuestas de la petición *POST /admin/post/json*.

Funcionalidad miscelánea de mantenimiento del *warehouse*:

- ***GET /admin/clean_cache***

  Descripción: petición síncrona que solicita la limpieza de los datos temporales del *warehouse*.

  Respuesta de la petición:

Código	Descripción
200	número de ficheros eliminados en formato *JSON*
401	autenticación no válida

Tabla 5.21: Respuestas de la petición *GET /admin/clean_cache*.

5.6. Diagrama de secuencia del caso de uso UC-1

Se realiza en la figura 5.21 el diagrama de secuencia de uno de los casos de uso más importantes del *warehouse*, la obtención de los metadatos de una aplicación. En este caso, por medio de su *package*.

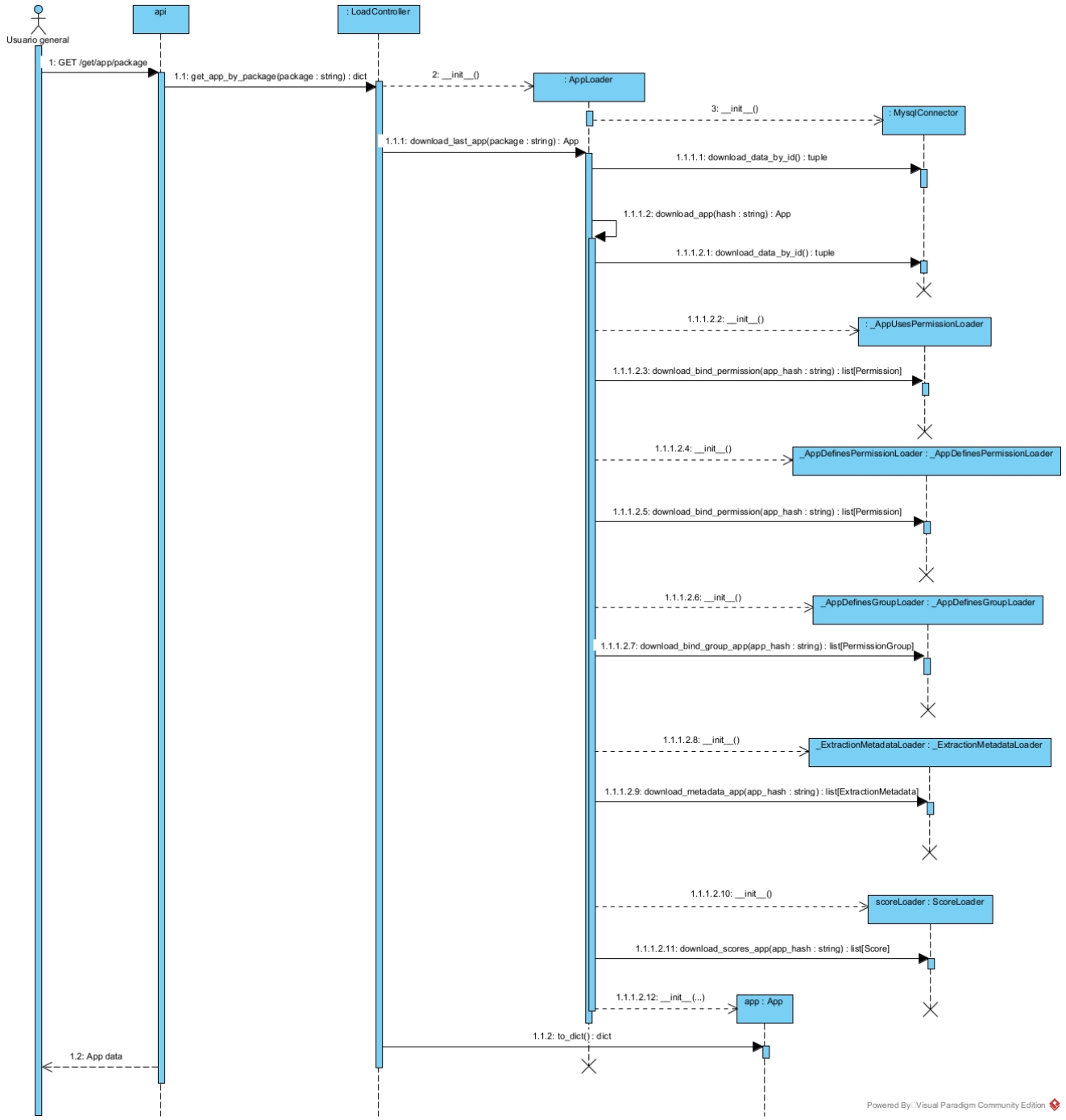

Figura 5.21: Diagrama de secuencia del caso de uso 1.

En la figura 5.21 se ve como la petición le llega a la *api*, esta delega en *LoadController* quien conoce la lógica de negocio necesaria para llevar a cabo la petición. A partir de este punto, la secuencia consiste en la creación de cada uno de los *Data Mappers* necesarios para la obtención de todos los metadatos de una aplicación. Con estos, se crea el objeto aplicación del cual se obtiene su representación en diccionario para traducirse directamente a *JSON* y devolvérselo al usuario.

Capítulo 6

Implementación y pruebas

En este capítulo se detalla el entorno de desarrollo utilizado y detalles sobre la implementación de alguno de los paquetes del *warehouse* del capítulo anterior. Por ejemplo: las librerías[1] que se han utilizado y para que, los patrones de *web scraping* utilizados para la extracción de datos, la estructuración del directorio de datos temporales y configuración del *warehouse*... Las versiones concretas de los programas y librerías utilizados se encuentran en la guía de instalación del *warehouse* contenido en el apéndice B. Además, se llevan a cabo una serie de pruebas sobre la funcionalidad esencial del sistema.

Todo el código desarrollado se encuentra en el repositorio público *Github*: `https://github.com/peres317/app_warehouse.git`. Su puesta en marcha se encuentra documentada en la guía de instalación contenida en el apéndice B.

6.1. Entorno de desarrollo

El entorno de desarrollo utilizado se ha escogido en función de las herramientas y facilidades que aporta así como el conocimiento previo del mismo.

En primer lugar, para el desarrollo del código *Python* y *SQL* del *warehouse*, se ha utilizado el editor avanzado de código abierto *Visual Studio Code*. Entre las virtudes que tiene este editor, cabe mencionar las que se han utilizado principalmente:

- El paquete de extensiones *Python Extension Pack* que aporta herramientas para manejar entornos virtuales *Python* y la depuración de código *Python*.
- La extensión *autoDocstring* que permite la creación de documentación del código *Python* de manera rápida siguiendo el estándar de documentación de código *Python* PEP 257 [44].
- La extensión *autopep8* que ayuda a mantener el estándar de estilo de código *Python* PEP 8 [30].
- Las extensiones *Community Server Conectors* y *MySQL* que permiten la conexión del editor con el servidor *MySQL*. Esto ayuda a visualizar rápidamente los cambios que genera el código sobre

[1] Todas las librerías utilizadas se encuentran subidas en el sistema de administración de paquetes de *Python* (PIP)

la base de datos.

En segundo lugar, se ha utilizado entornos virtuales *Python* (*venv*) para un mayor control de las versiones de las librerías que se utilizan así como una mayor portabilidad del *warehouse*.

Para el despliegue del *warehouse*, tanto el código del servidor *web* de la *api* como su sistema gestor de bases de datos *MySQL* se ha utilizado una máquina virtual *Linux Ubuntu 22.04.2 LTS* desplegada con el gestor de máquinas virtuales *Virtual Box 7.0*.

Por último, para realizar pruebas del funcionamiento del *warehouse* mientras se lleva a cabo el desarrollo, se utiliza la herramienta *Postman* que permite el uso de *APIs* de forma sencilla. Se basa en realizar las peticiones concretas a través de un navegador *web Google Chrome*. Además, permite exportar las colecciones de peticiones y ejecuciones de estas a modo de ejemplo en formato *JSON* facilitando la portabilidad de las pruebas y sirviendo a modo de documentación interactiva para los desarrolladores de los sistemas cliente que accedan al *warehouse*.

6.2. Implementación del paquete *api*

En primer lugar, se necesita disponer de un certificado *SSL* para que el servidor pueda trabajar bajo el protocolo de hipertexto seguro *HTTPS*. Para ello, se genera desde Linux directamente con la siguiente orden:

```
openssl req -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -sha256 -days 365
```

Esta orden genera un certificado autofirmado *SSL* con caducidad de 365 días desde su creación. El par de claves *RSA* es de 4096 bits y utiliza el algoritmo *SHA256* para firmar el certificado. Se generan dos archivos: el certificado *cert.pem* y su clave *key.pem*.

En segundo lugar, se utilizan las siguientes librerías *Python* en la implementación de la *api* de acceso al *warehouse*:

- ***uvicorn:*** una implementación de un servidor *HTTP/1.1* y *WebSockets* para *Python*. Se usa como servidor de las peticiones a la *api*. Este servidor se configura para que se ejecute con el certificado anteriormente generado especificándo la ruta a dichos archivos. Además, también se configura para que se ejecute en el puerto 8000. Esto se hace ya que no es un puerto conocido evitando así gran cantidad de ataques y/o usos malintencionados.

- ***fastapi:*** un *framework* que permite la construcción de *APIs*[2] en *Python*. Se utiliza por su sencillez de uso y conocimiento previo en el. Entre las configuraciones que permite, se hace uso de la securización de las peticiones por medio del estándar *OAuth2.0* [23].

[2] API es el acrónimo de Application Programming Interface

- ***threading:*** un módulo que aporta una interfaz para la creación y administración de hilos en *Python*. Se utiliza para llevar a cabo concurrentemente las peticiones de los usuarios. Además, dispone de mecanismos de bloqueos (*Lock*) que permiten marcar secciones de código sensibles para evitar que estas se puedan ejecutar simultáneamente.

6.3. Implementación del paquete *util*

En la implementación de este paquete se hace uso de la librería *Python termcolor* la cual permite escribir mensajes de texto en pantalla a color. Esto facilita el sistema de monitorización (*logs*) básico implementado. Se crean distintos tipos de mensajes de monitorización haciendo uso de la metáfora del semáforo y sus colores. Los distintos tipos de *logs* que se implementan ordenados de menor a mayor impacto en el sistema son:

1. *log_result*: Se imprime un mensaje informativo sobre el resultado de un proceso del sistema. Se imprime a color verde ya que es algo deseado.

2. *log_normal*: Se imprime un mensaje informativo sobre un proceso del sistema. Se imprime a color blanco/negro ya que es un mensaje meramente informativo.

3. *log_warning*: Se imprime un mensaje de advertencia sobre un proceso del sistema que no ha seguido su flujo normal de trabajo. Se imprime a color amarillo ya que es un mensaje que podría estar indicando un posible error.

4. *log_error*: Se imprime un mensaje de error sobre un proceso del sistema que no ha sido capaz de finalizar. Se imprime a color rojo ya que es un mensaje de error y ha de ser revisado.

6.4. Implementación del paquete *extract*

En esta sección se detallan las librerías *Python* y patrones *scraping* utilizados para la extracción de datos de las diversas fuentes que se utilizan.

6.4.1. *Androzoo*

Para la extracción de aplicaciones de *Androzoo* se hace uso de la librería *Python pyandrozoo* [45]. Esta librería implementa funcionalidad de acceso a la *API* de *Androzoo*. Además, permite la descarga en paralelo de hasta 20 *apps*.

6.4.2. *Scraping* a páginas *web*

Para realizar los *scrapings* se utiliza la librería *Python selenium* ya que es una de las más utilizadas para este cometido. Además, también se utiliza la librería *requests* para descargar páginas *web* completas y los bloqueos de *threading* para proteger las secciones de código que no deben ejecutarse concurrentemente.

Apkfollow y *Apkmonk*

Para la extracción de datos de estas dos fuentes *web* se aplican los mismos patrones de *scraping* ya que ambas páginas *web* comparten su estructura:

1. En primer lugar se busca la aplicación que se quiere descargar modificando el enlace de consulta con la aplicación que se desea buscar (figura 6.1). Enlaces de consulta: `https://www.apkfollow.com/search?q=app.a.buscar` o `https://www.apkmonk.com/ssearch?q=app.a.buscar`.

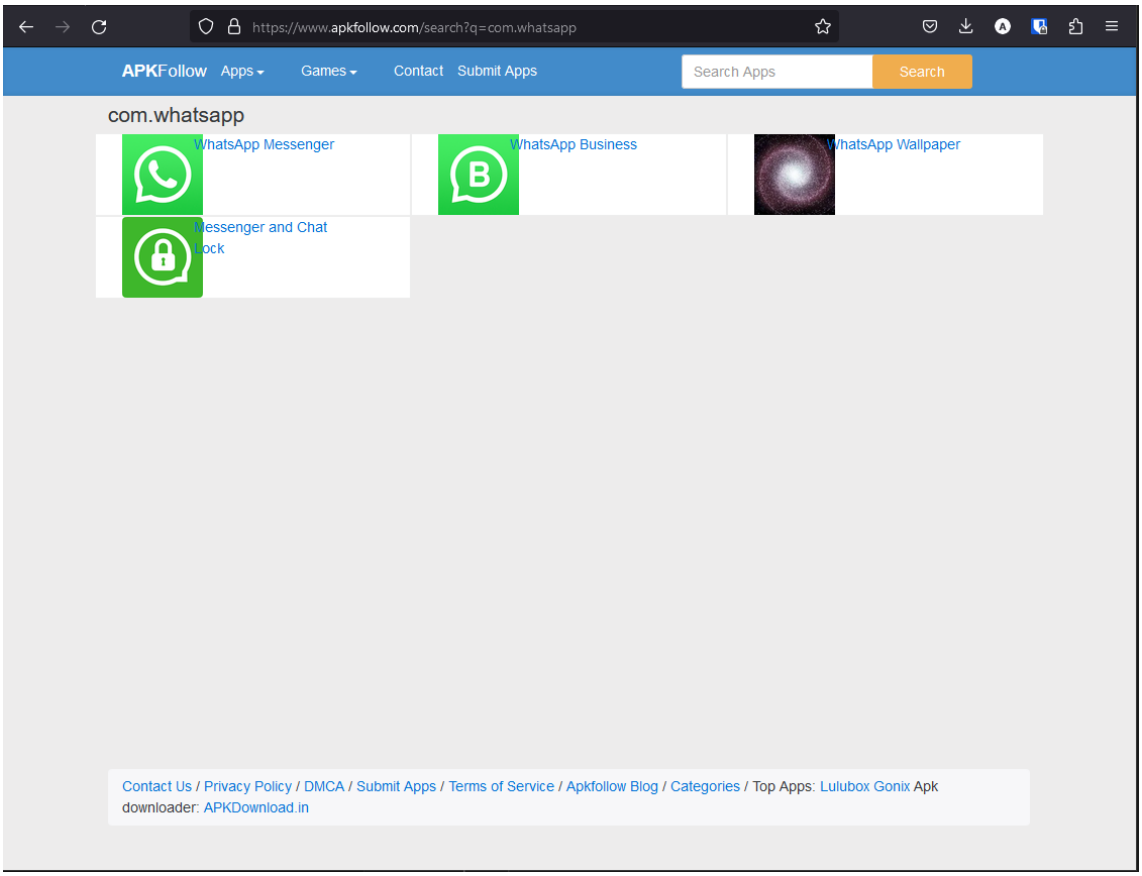

Figura 6.1: Búsqueda de la aplicación *Whatsapp* en *Apkfollow*.

Como resultado, obtenemos una lista con las aplicaciones encontradas en la página *web*. Para extraer el enlace de la aplicación que buscamos, extraemos todos los *links* (*html tag* <*a*>) presentes en la *web*, los filtramos por aquellos que contengan el patrón ".*/app/.*" y obtenemos el primer enlace que aparece en la página (en el ejemplo de la figura 6.1 `https://www.apkfollow.com/app/whatsapp-messenger/com.whatsapp/`).

2. Una vez tenemos el enlace de la página de la *app* que queremos descargar, lo cargamos (figura 6.2). De esta página, obtenemos el enlace directo de descarga de la *app* filtrando todos los enlaces de la página por los que contienen el patrón ".*/download/.*". De estos, nos quedamos con el primer enlace que aparece en la página ya que corresponde a la última versión de la aplicación (en el ejemplo de la figura 6.2 `https://www.apkfollow.com/download/apks_new_com.whatsapp_2023-06-05.apk/`).

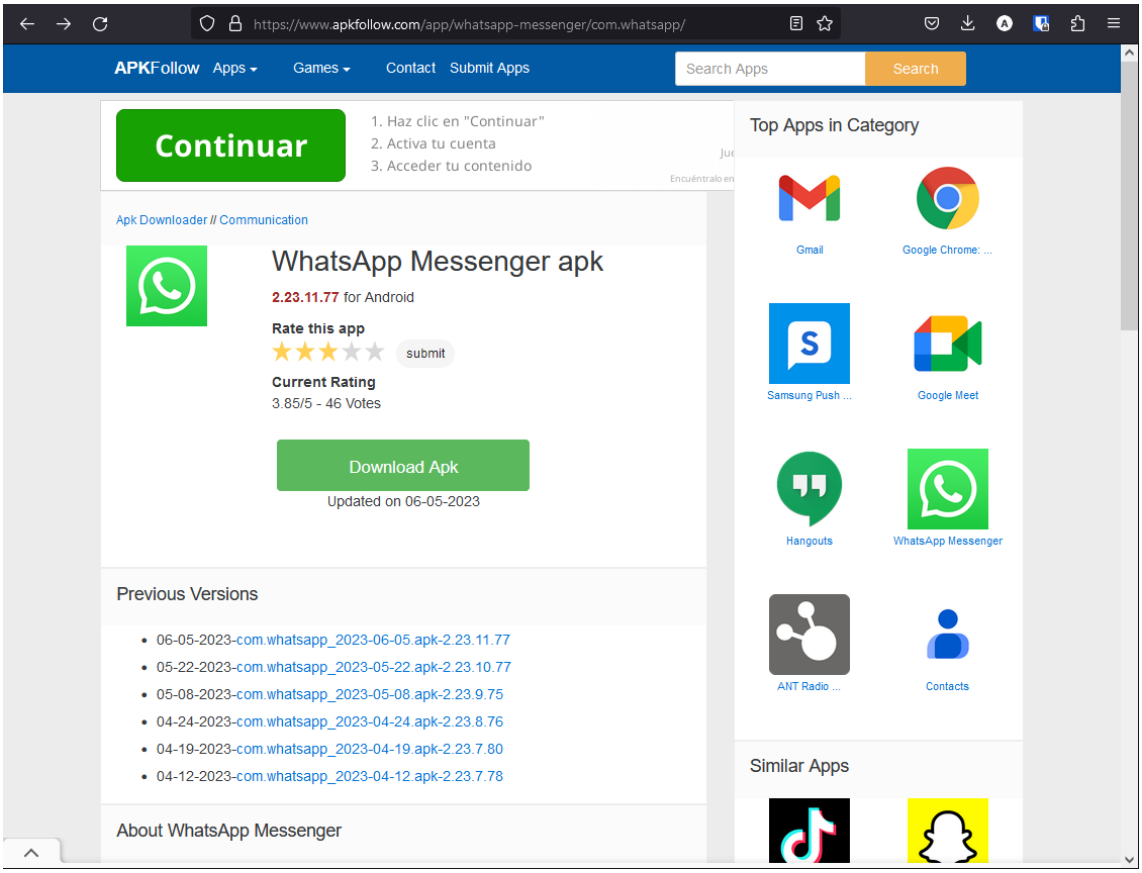

Figura 6.2: Descarga de la aplicación *Whatsapp* de *Apkfollow*.

3. Por último, cargamos el enlace de descarga directo y comienza a descargarse la aplicación.

Apkpure

1. En primer lugar se busca la aplicación que se quiere descargar modificando el enlace de consulta con la aplicación que se desea buscar (`https://m.apkpure.com/es/search?q=app.a.buscar`) (figura 6.3).

   Como resultado (figura 6.3), obtenemos una página *web* que contiene una lista con las aplicaciones encontradas. Para extraer el enlace de la aplicación que buscamos, extraemos todos los *links* presentes en la *web*, los filtramos por aquellos que no contengan el patrón ".*?.*" y de estos, los que contengan el nombre de la *app*. De la lista de enlaces que cumplen el filtrado, nos quedamos con el primer enlace que aparece en la página (en el ejemplo de la figura 6.3 `https://m.apkpure.com/es/whatsapp-android/com.whatsapp`).

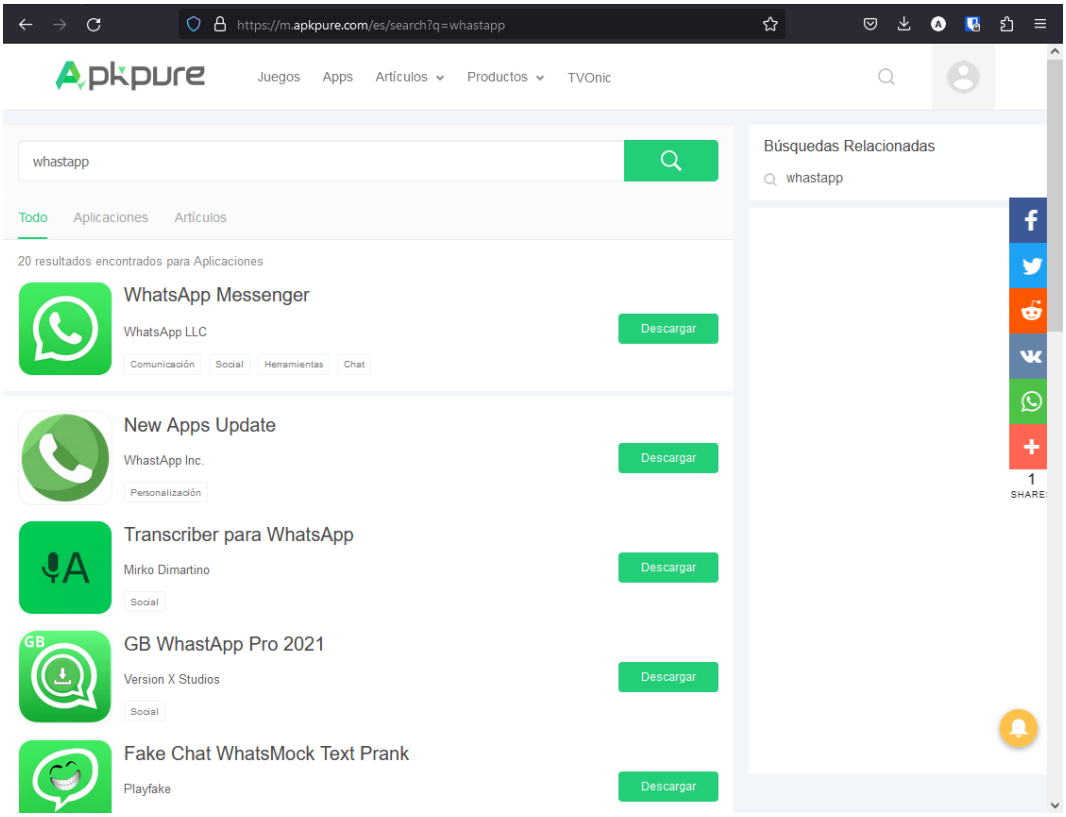

Figura 6.3: Búsqueda de la aplicación *Whatsapp* en *Apkpure*.

2. Una vez tenemos el enlace de la página de la *app* que queremos descargar, lo cargamos (figura 6.4).

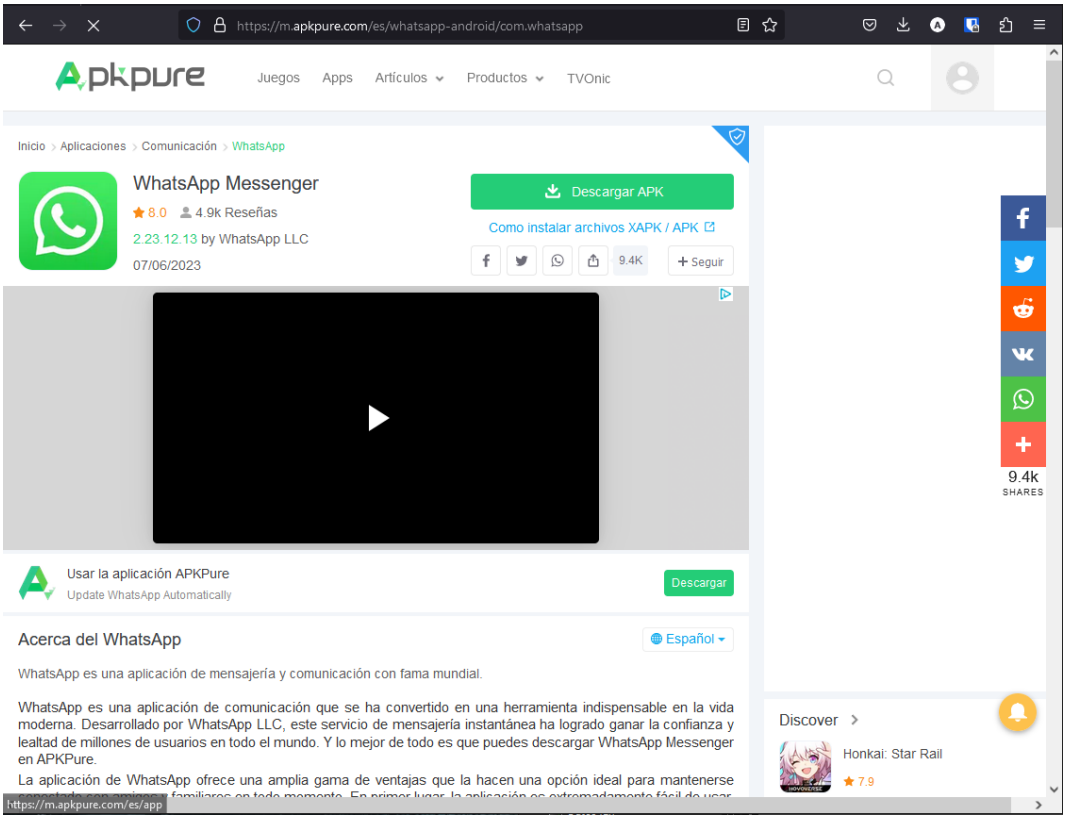

Figura 6.4: Descarga de la aplicación *Whatsapp* de *Apkpure*.

De esta página, obtenemos el enlace directo de descarga de la *app* filtrando todos los enlaces de la página por los que contienen el patrón ".*?version=latest". De estos, nos quedamos con el primer enlace que aparece en la página (en el ejemplo de la figura 6.4 `https://d.apkpure.com/b/APK/com.whatsapp?version=latest`).

3. Cargamos el enlace de descarga directo y comienza a descargarse la aplicación.

4. En este caso dado que *Apkpure* almacena aplicaciones en formato *APK* y *XAPK*. Si la aplicación descargada se encuentra en formato *XAPK* ha de descomprimirse para estar en formato *APK*.

Evozi

1. En primer lugar se busca la aplicación que se quiere descargar modificando el enlace de consulta con la aplicación que se desea buscar (`https://apps.evozi.com/apk-downloader/?id=app.a.buscar`) (figura 6.5).

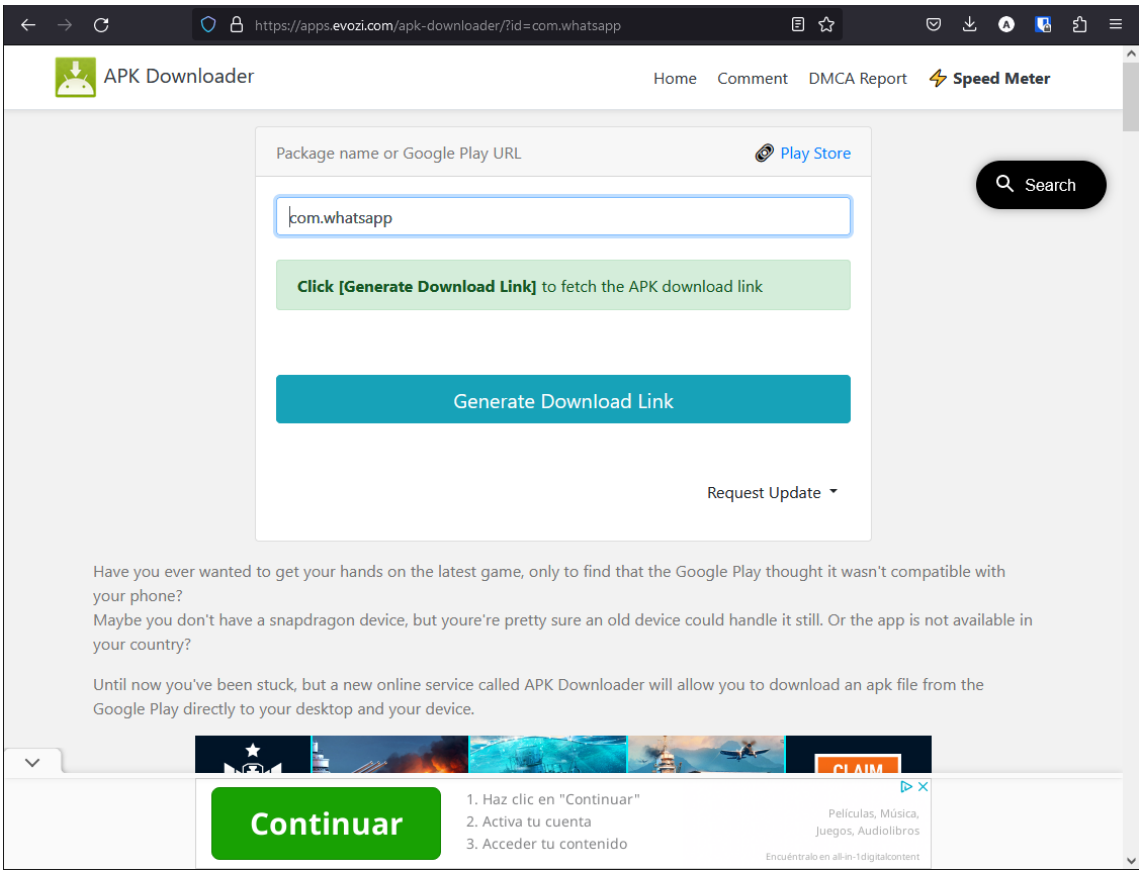

Figura 6.5: Búsqueda de la aplicación *Whatsapp* en *Evozi*.

Para que la búsqueda tenga efecto se ha de hacer *click* en el botón "*Generate Download Link*". Para ello, se busca el elemento botón de la página *web* por dicho texto y se emula *click* en el.

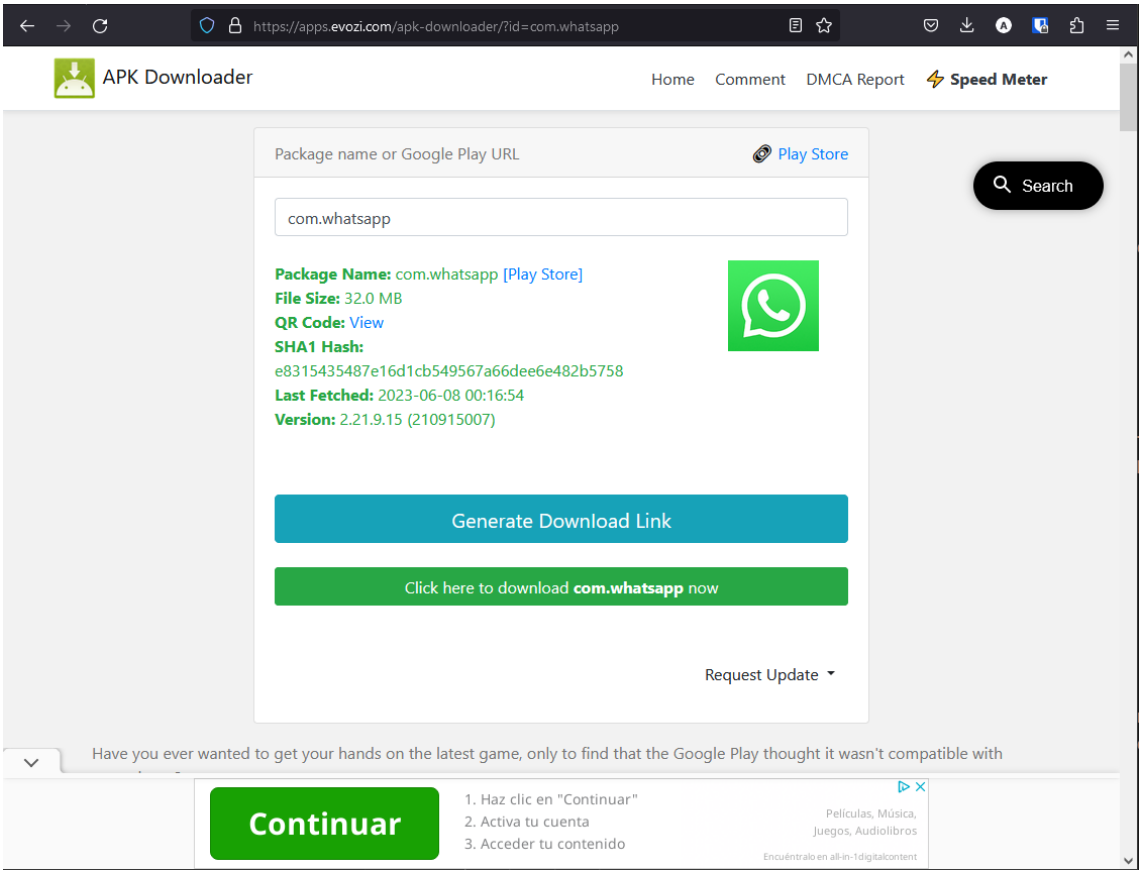

Figura 6.6: Búsqueda de la aplicación *Whatsapp* en *Evozi*.

Como resultado (figura 6.6), obtenemos un resumen de la aplicación encontrada. Para extraer el enlace de descarga, filtramos todos los enlaces de la página *web* por los que acaban en "*.apk*" (en el ejemplo de la figura 6.6 `https://storage.evozi.com/apk/dl/16/09/05/com.whatsapp_210915007.apk`).

2. Por último, cargamos el enlace de descarga directo y comienza a descargarse la aplicación.

Play Store

- **Obtención de todas las categorías de *apps* de *Play Store*:**

1. En primer lugar, cargamos la página *web* `https://support.google.com/googleplay/android-developer/answer/9859673?hl=en-419#zippy=%2Capps` donde se encuentra un listado de todas las categorías de aplicaciones (figura 6.7).

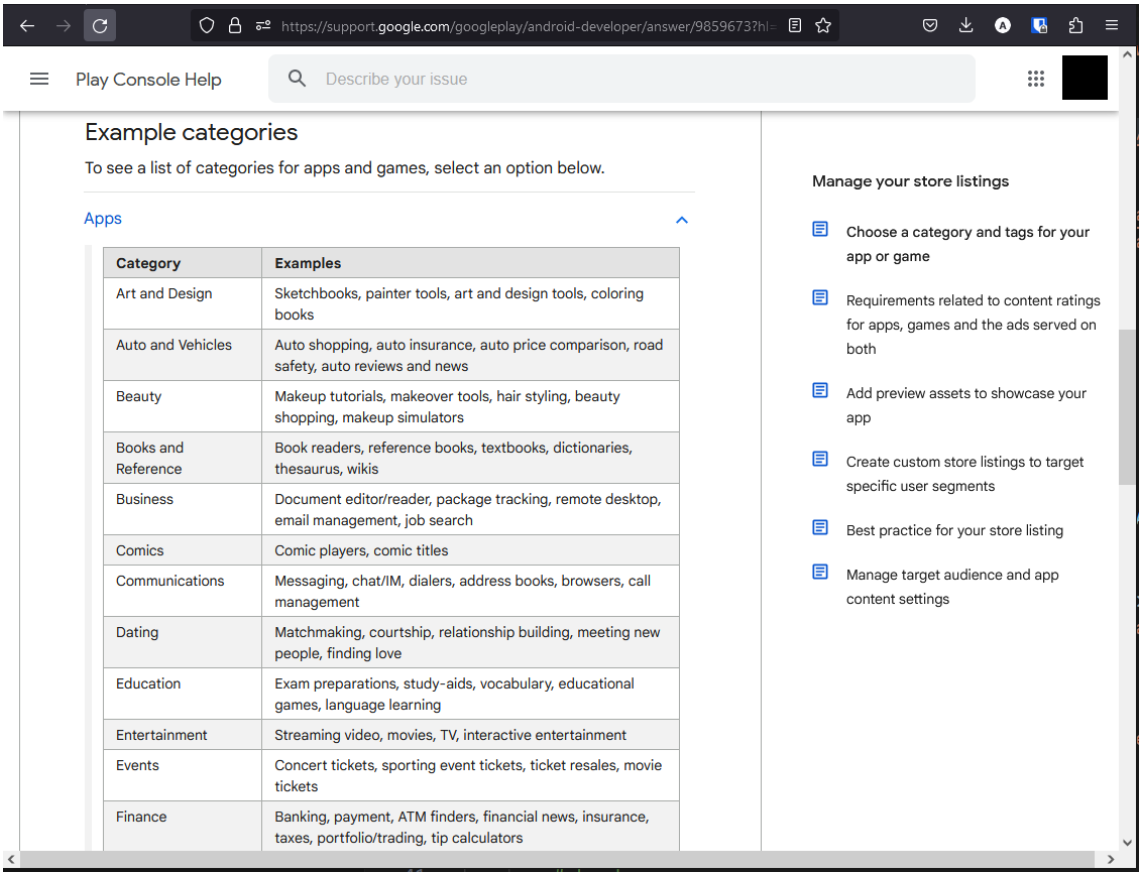

Figura 6.7: Categorías de *apps* según *Play Store*.

2. De esta página, se extraen los contenidos de todas las celdas de la primera columna de la tabla, para ello, se filtran todas las celdas (*html tag* <*td*>) que tienen la misma posición "x" en la página que la que contiene el texto "*Category*".

3. Por último, se formatean las categorías obtenidas a mayúsculas y con barras bajas en vez de espacios.

- **Obtención de los nombres de paquete de las *apps* más populares por categoría de *Play Store*:**

1. En primer lugar, creamos una lista con las páginas portada de cada categoría concatenando el enlace base de la página *web* portada de una categoría con cada una de las categorías extraídas anteriormente (`https://play.google.com/store/apps/category/CATEGORIA_EXTRAIDA`).

2. Se cargan secuencialmente las páginas (figura 6.8) de la lista generada en el paso anterior extrayendo de cada una de ellas los nombres de paquete de las aplicaciones que aparecen.

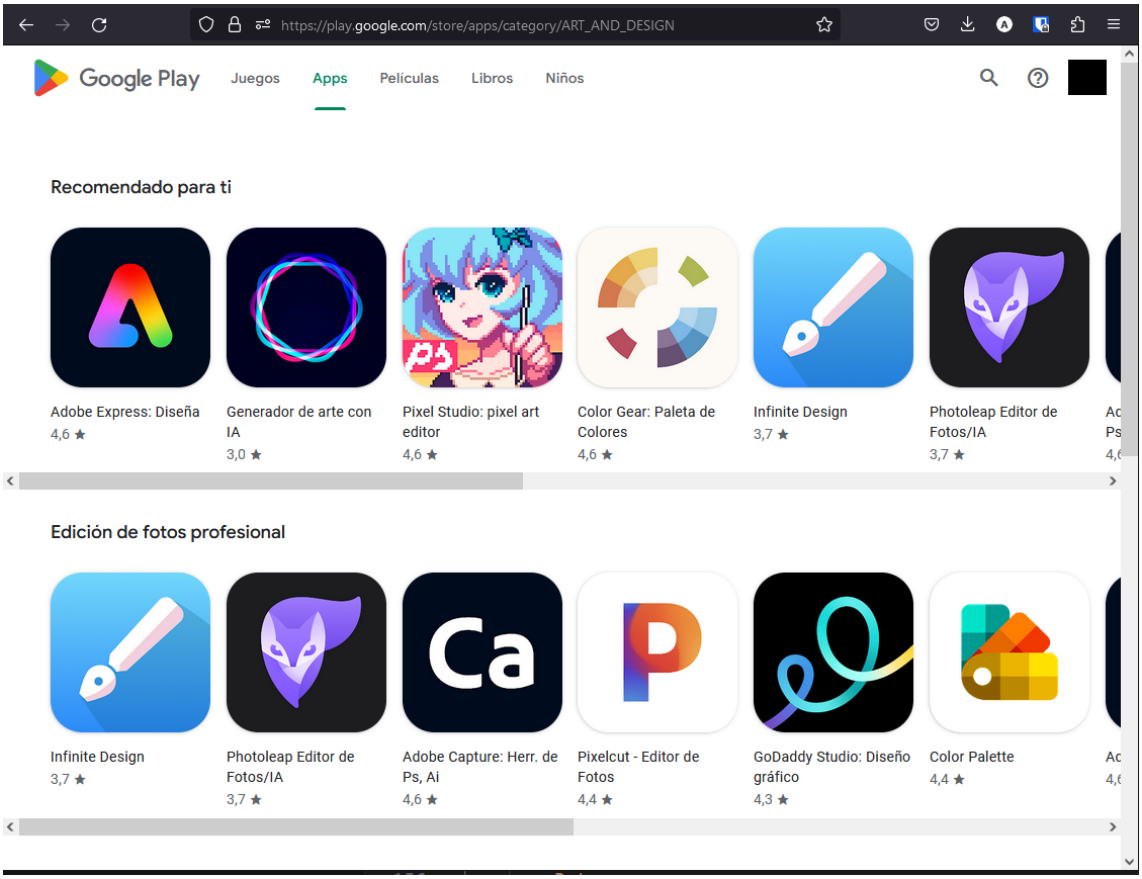

Figura 6.8: *Apps* mas populares de la categoría *Art and Design*.

3. Para extraer los nombres de paquetes de las aplicaciones que se encuentran en una página portada de una categoría concreta, se obtienen todos los enlaces de la página que contienen el patrón ".*details?id=.*". De estos, el texto después de "details?id=" es el nombre de paquete de la aplicación. Por ejemplo, la primera aplicación de la figura 6.8 tiene el enlace: `https://play.google.com/store/apps/details?id=com.adobe.spark.post`. Del que se extraría el nombre de paquete "com.adobe.spark.post".

6.4.3. *Google*

- **Extracción de la metainformación sobre los permisos y grupos de permisos del sistema operativo Android:**

  1. En primer lugar se descarga el *AndroidManifest.xml* del sistema operativo Android de la rama *master* del sistema de control de versiones (*git*) que se utiliza en su desarrollo (`https://android.googlesource.com/platform/frameworks/base.git/+/refs/heads/master/core/res/AndroidManifest.xml?format=TEXT`).
  2. Con el archivo descargado puesto que tiene un formato *XML*, se utiliza la librería *Python lxml* para buscar en sus contenidos por *tag*. Los permisos se obtienen del *tag permission*, mientras que los grupos de permisos del *tag permission-group* tal y como esta definido en su documentación [13], [37].
  3. Una vez se tienen los elementos, se extraen sus atributos:

- De los permisos: *name*, *protectionLevel* y *permissionGroup*.
- De los grupos de permisos solamente *name*.

- **Extracción del nivel de *api* en que se añadió un permiso o grupo de permisos al sistema operativo Android:**

  1. En primer lugar, se carga la página *web* oficial de los desarrolladores de Android `https://developer.android.com/reference/android/Manifest.permission` o `https://developer.android.com/reference/android/Manifest.permission_group` dependiendo de si queremos extraer el nivel de *API* de un permiso o de un grupo de permisos respectivamente (figura 6.9).

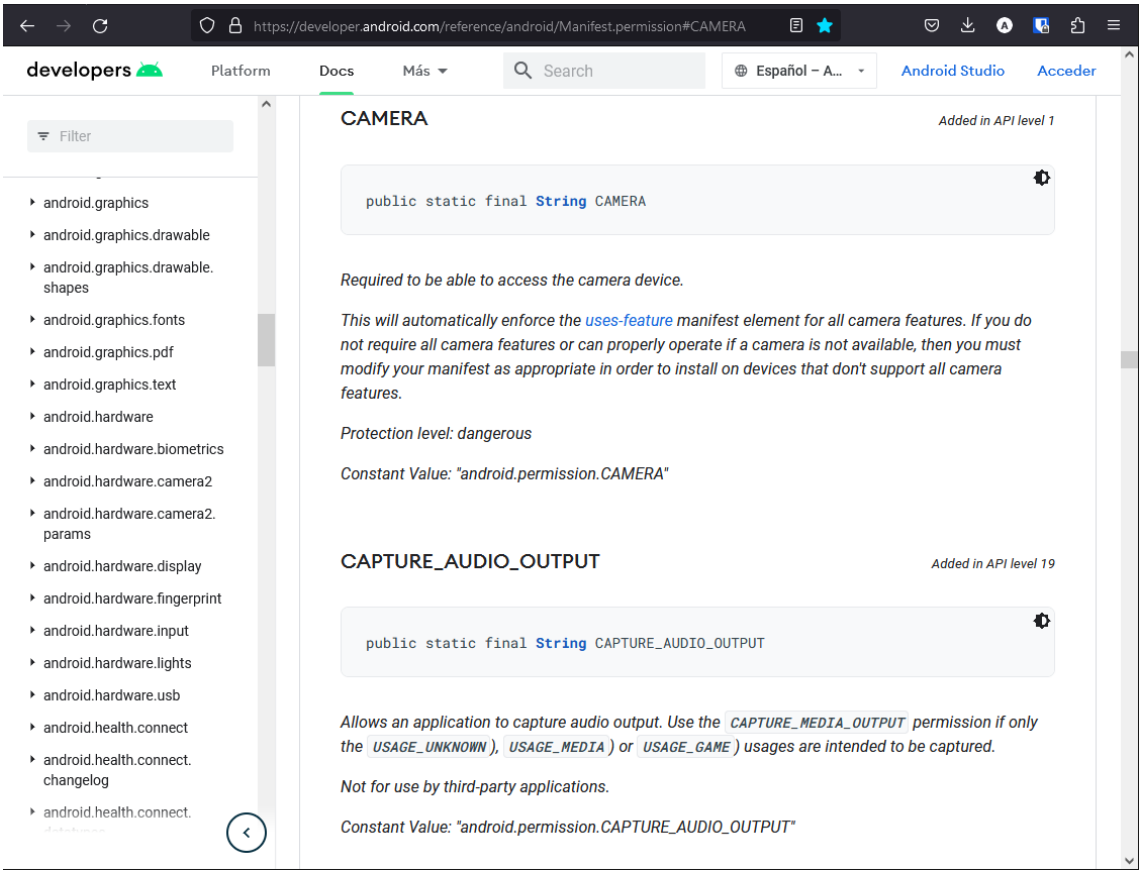

Figura 6.9: *API level* en que se añadió el permiso cámara.

  2. En ambos casos, se extrae el *API level* por medio del identificador único (*id*) del permiso o grupo de permisos (en el ejemplo de la figura 6.9 “android.permission.CAMERA”). Se obtiene el elemento de la página *web* con dicho identificador, y de el, se extrae el atributo *data-version-added* en el que se encuentra la versión de *API* en que se añadió el permiso o grupo de permisos al sistema operativo Android.

6.5. Implementación del paquete *transform*

En esta sección se detallan las librerías utilizadas para la transformación de los *.apk* de las aplicaciones en los metadatos que almacena el *warehouse*:

- ***hashlib*:** Se utiliza esta librería para la obtención del *hash SHA256* de un *.apk*.
- ***androguard*:** Se utiliza esta librería para la extracción de los siguientes metadatos de una aplicación: el nombre de paquete, la versión de la aplicación, el mínimo *SDK*[3] requerido, el máximo *SDK* permitido, el *SDK* esperado y la lista de permisos que solicita.
- ***lxml*:** Gracias a esta librería tenemos una interfaz para tratar los objetos *XML* de forma sencilla. Se utiliza para obtener los metadatos del *AndroidManifest.xml* de una aplicación cuya obtención no esta implementada en *androguard*. En específico, los siguientes metadatos son extraídos haciendo uso de esta librería: los grupos de permisos que declara la aplicación (*tag permission-group*) [37] y los permisos que declara la aplicación (*tag permission*).
- ***json*:** Se utiliza para la transformación de textos en formato *JSON* a elementos del modelo de dominio.

6.6. Implementación del paquete *load*

La librería *Python* utilizada en la implementación del paquete *load* es *mysql-connector*. Esta librería aporta una interfaz de conexión a bases de datos *MySQL*. Además, tiene un método de ejecución de consultas seguras ante ataques *SQLi* gracias a la validación de los parámetros de las consultas.

6.7. Base de datos del *warehouse*

La base de datos del warehouse se ha implementado con el gestor de bases de datos *MySQL 8.0.33*.

6.8. Estructura del directorio de datos del *warehouse*

Partiendo del diseño llevado a cabo y de todas las secciones de implementación anteriores, se estructura un directorio que contendrá toda la metainformación y configuración que el *warehouse* ha de almacenar para su correcto funcionamiento:

```
data/
├── androzoo_downloads/
├── app_downloads/
├── google_downloads/
├── play_store_downloads/
├── config.json
├── cert.pem
└── key.pem
```

Figura 6.10: Estructura del directorio *data/*.

[3]SDK es el acrónimo de Software Development Kit

En la figura 6.10 se encuentran los siguientes directorios y archivos:

- ***androzoo_downloads/***: Un directorio que almacene temporalmente las aplicaciones descargadas de Androzoo [8] hasta su carga en el *warehouse*.
- ***app_downloads/***: Un directorio que almacene temporalmente las aplicaciones descargadas por fuentes síncronas hasta su carga en el *warehouse*.
- ***google_downloads/***: Un directorio que almacene temporalmente la metainformación sobre permisos y grupos de permisos del sistema operativo Android extraída hasta su carga en el *warehouse*.
- ***play_store_downloads/***: Un directorio que almacene temporalmente la metainformación sobre categorías de aplicaciones y aplicaciones más populares por categoría extraídas de *Play Store* hasta su carga en el *warehouse*.
- ***cert.pem***: El certificado seguro *SSL* definido según el RFC[4] 7468 [46].
- ***key.pem***: La clave del certificado *SSL* definida según el RFC 7468 [46].
- ***config.json***: Un archivo con formato *JSON* que almacena toda la configuración del *warehouse*. La estructura de este fichero es la contenida en la figura 6.11.

```
{
    "MYSQL": {
        "host": "...",
        "database": "...",
        "user": "...",
        "password": "..."
    },
    "ANDROZOO": {
        "API_KEY": "...",
        "INDEX_FILE": "...csv.gz"
    },
    "API_ACCESS_KEYS": [
        "...",
        ...
    ],
    "API_ADMIN_ACCESS_KEYS": [
        "...",
        ...
    ]
}
```

Figura 6.11: Estructura del fichero *config.json*.

[4]RFC es el acrónimo de *Request for Comments.* Los documentos que contienen especificaciones técnicas, estándares y protocolos relacionados con internet y las tecnologías de la información.

En este fichero contenido en la figura 6.11, se almacena bajo la clave *MYSQL* la configuración de acceso a la base de datos del *warehouse*. Bajo la clave *ANDROZOO* la clave de acceso a la *API* de Androzoo y el fichero índice de las aplicaciones del repositorio. Y, por último, las claves de acceso que otorgamos al *warehouse*. Estas se separan en dos listas *API_ACCESS_KEYS* para usuarios generales y *API_ADMIN_ACCESS_KEYS* para usuarios administradores. Este fichero esta securizado frente a accesos maliciosos teniendo solo acceso de lectura y escritura por su usuario Linux propietario (el *warehouse*).

6.9. Pruebas del *warehouse*

En esta sección se llevan a cabo unas pruebas básicas de funcionamiento y rendimiento del *warehouse*. Para ello, se realizan las peticiones correspondientes por medio de la aplicación *Postman* (ver manual de uso en el apéndice C).

6.9.1. Prueba de carga masiva de datos

Puesto que el *warehouse* ha de nutrirse de un gran número de aplicaciones para poder posteriormente satisfacer las necesidades de sus usuarios, una de las funcionalidades más importantes del *warehouse* es la carga masiva de datos. En este contexto, se realiza una prueba de carga de 100 aplicaciones por medio de la consulta asíncrona "https://localhost:8000/admin/get/n_random_apps?n_apps=100" en un tiempo total de 12 minutos y 36 segundos. Que corresponde a un tiempo promedio de 7,5 segundos por aplicación cargada en el *warehouse*. El uso de tiempo desglosado según las tareas que se realiza es:

- Respuesta a la petición: 28 milisegundos.
- Selección aleatoria de las aplicaciones a descargar: 51 segundos.
- Descarga de las aplicaciones: 9 minutos 26 segundos.
- Procesado y carga de las aplicaciones: 2 minutos 19 segundos.

6.9.2. Pruebas de consulta

Otra de las funcionalidades más importantes del *warehouse* es la consulta de las aplicaciones. En este contexto, se realizan pruebas de consulta al *warehouse* de aplicaciones que se encuentran y no almacenadas. Además, también se realizan pruebas conjuntas con el servicio *APKFalcon* desarrollado en el marco del TFG de Javier Crespo Guerrero [11].

Consulta de una aplicación almacenada

En este caso, la aplicación que se solicita se encuentra almacenada por lo que la respuesta son los metadatos de la misma. Se realiza la consulta de los datos de la aplicación *Netflix* por medio de la siguiente petición: "https://localhost:8000/get/app/package?package=com.netflix.mediaclient" con un tiempo respuesta de 381 milisegundos.

Consulta de una aplicación no almacenada

Ahora, la aplicación que se solicita no se encuentra almacenada por lo que la respuesta es un error 404 indicando esta situación. La petición realizada es: “https://localhost:8000/get/app/package?package=not.found” obteniéndose el error 404 en 42 milisegundos.

6.9.3. Prueba de rendimiento

Se realizan pruebas de acceso al *warehouse* desde distintas redes externas a su despliegue para comprobar los tiempos de respuesta que podrían esperar los servicios que se conectan como *APKFalcon* de Javier Crespo Guerrero [11]. En este aspecto, por motivos logísticos, todas las pruebas se realizan desde redes en el mismo municipio en que se encuentra el despliegue del *warehouse* (Valladolid). La petición que se realiza en todos los casos para medir el tiempo medio de respuesta es la petición de los metadatos de la aplicación *Netflix* vista anteriormente. Se obtiene un tiempo medio de respuesta de 754 milisegundos promediando un total de 10 pruebas realizadas en distintas redes.

6.9.4. Prueba de actualización conflictiva del contenido del *warehouse*

Se plantea el caso de que se intente insertar una aplicación que ya se encuentre almacenada en el *warehouse*. En este contexto, una petición que puede llevar a esta situación es la petición asíncrona mostrada en la sección C.2.3 del apéndice C. En caso de descargarse la misma versión que ya se encuentra almacenada, los datos a insertar ya estarían contenidos por lo que esta petición no realizaría ninguna actualización sobre el *warehouse*. El tiempo empleado en la misma es de 1 minuto y 25 segundos desglosado en las siguientes tareas:

- Respuesta a la petición: 8 milisegundos.
- Búsqueda de la aplicación en las fuentes de datos: 20 segundos.
- Descarga de la aplicación: 45 segundos.
- Procesado de la aplicación: 20 segundos.

Una vez procesada la aplicación, a la hora de la carga de la misma al *warehouse* puesto que ya se encuentra almacenada, no se realiza la carga.

Capítulo 7

Conclusiones y lineas de trabajo futuro

En este capítulo se desarrollan las conclusiones del trabajo realizado incluyendo los objetivos conseguidos así como el estado actual de los contenidos extraídos de las distintas fuentes de datos. Por último, se enumeran las posibles lineas de trabajo futuro.

7.1. Conclusiones

Haciendo referencia a los objetivos fijados en la sección 1.3, se han encontrado múltiples problemas a la hora de conseguir cada uno de estos. A continuación, se enumeran los principales problemas encontrados, así como su solución.

En primer lugar, la integración de datos de distintas fuentes siempre plantea un problema debido a que es necesario unificar distintos esquemas de datos en uno solo. Para solventar este problema se han utilizado los ficheros (*.apk*) de las aplicaciones, en específico, el archivo *AndroidManifest.xml* que contiene los metadatos de las aplicaciones. De este modo, gracias a la extracción de los datos del mismo esquema de fichero este problema queda solucionado. No obstante, surgió otro problema a la hora de la extracción de dichos archivos de las fuentes *web*, en este caso, las páginas *web* de hoy en día utilizan estructuras dinámicas puesto que las interfaces de usuario se cambian con mucha frecuencia. Para solventarlo, se hace uso de la premisa de que la información que se ofrece en las páginas *web* se almacena en repositorios y se distribuye por medio de *APIs*. De este modo, dicha información se puede extraer de las páginas por medio de la búsqueda de los enlaces de peticiones a dichas *APIs*. Obteniéndose así una solución mucho más duradera a lo largo del tiempo, ya que las *APIs* cambian con menos frecuencia que las interfaces de usuario.

En segundo lugar, la extracción de los metadatos de las aplicaciones no presenta ningún problema más allá de conocer la estructura que usa. Esto es gracias a que se extraen de un archivo con formato XML. Además, esta estructura se encuentra detallada en la documentación de Android. No obstante, la obtención de su categoría sí que plantea un problema puesto que es información dependiente de la tienda de aplicaciones concreta en la que se encuentra la aplicación. Para mantener la uniformidad, se extrae la categoría de las aplicaciones de la página de *Play Store* haciendo uso de la solución al *scraping web* mencionada en el párrafo anterior.

Por último, en cuanto a la carga de los datos extraídos en el almacén central, se ha utilizado una solución ya implementada gracias a que dichos datos son estructurados. En este caso, el sistema gestor de bases de datos *MySQL*.

Se ha poblado el *warehouse* con información sobre un conjunto de aplicaciones amplio, sin que ello suponga requerimientos de almacenamiento que puedan causar problemas de almacenamiento y/o rendimiento. En la fecha de finalización de este TFG[1] el *warehouse* aloja información sobre 9015 aplicaciones y sobre 5 métricas. El espacio que requieren estos metadatos es inferior a 43 MB, lo cual se mantiene por debajo de la capacidad de almacenamiento de la máquina virtual donde se aloja (25 GB). La figura D.1 del apéndice D muestra una captura de pantalla de la herramienta donde se visualizan estos datos.

7.2. Líneas de trabajo futuro

Debido al gran tamaño del sistema desarrollado y de todas las posibles ampliaciones que puede tener, surgen múltiples futuras líneas de continuación de este trabajo. Entre ellas, todas las que tienen relación con ampliar y/o complementar la funcionalidad del *warehouse*:

- Permitir la carga de aplicaciones al *warehouse* por medio de su archivo de paquete (*APK*). Esto da la posibilidad de subir aplicaciones que no se encuentren en repositorios. Por ejemplo, las aplicaciones en desarrollo para estudiar el impacto en la privacidad que pueden tener antes de su publicación.
- Permitir el borrado de información del *warehouse* para el caso en que se decida que se quiere dejar de almacenar alguna información. Por ejemplo, porque algunas versiones de *apps* dejen de utilizarse, o cualquier otra razón similar.
- Un sistema de monitorización del uso del *warehouse* para ayudar a estimar los recursos necesarios por este para su correcto funcionamiento.
- Un sistema de registros en ficheros que permita la depuración de los fallos que pudieran surgir en el *warehouse*. Por ejemplo, si ha quedado expuesta a internet una clave de acceso y está siendo utilizada de manera abusiva.
- Un sistema con interfaz gráfica que facilite administrar el *warehouse* de forma similar a los que ofrecen los sistemas gestores de bases de datos, de redes, etc.

[1] 11 de junio de 2023

Bibliografía

[1] N. Kesswani, H. Lyu y Z. Zhang, «Analyzing Android App Privacy With GP-PP Model,» *IEEE Access*, vol. 6, págs. 39 541-39 546, 2018. DOI: 10.1109/ACCESS.2018.2850060.

[2] Rina, «A Comparative Analysis of mobile Operating Systems,» dirección: https://www.ijcseonline.org/pub_paper/11-IJCSE-05378.pdf.

[3] A. Aparicio, M. Martínez González y V. Cardeñoso, «Métrica basada en grupos de permisos para entender el impacto de las aplicaciones Android sobre la privacidad,» en *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, 2022, págs. 1-5. DOI: 10.23919/CISTI54924.2022.9820147.

[4] España. «Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales.» (2018), dirección: https://www.boe.es/buscar/pdf/2018/BOE-A-2018-16673-consolidado.pdf. (último acceso: 16/05/2023).

[5] H. Jin, M. Liu, K. Dodhia et al., «Why Are They Collecting My Data?: Inferring the Purposes of Network Traffic in Mobile Apps,» *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 2, n.º 4, 173:1-173:27, 2018. DOI: 10.1145/3287051. dirección: https://doi.org/10.1145/3287051.

[6] M. Hatamian, N. Momen, L. Fritsch y K. Rannenberg, «A Multilateral Privacy Impact Analysis Method for Android Apps,» en *Privacy Technologies and Policy*, M. Naldi, G. F. Italiano, K. Rannenberg, M. Medina y A. Bourka, eds., Cham: Springer International Publishing, 2019, págs. 87-106, ISBN: 978-3-030-21752-5.

[7] S. Barth, M. de Jong, M. Junger, P. H. Hartel y J. C. Roppelt, «Putting the privacy paradox to the test: Online privacy and security behaviors among users with technical knowledge, privacy awareness, and financial resources,» *Telematics Informatics*, vol. 41, págs. 55-69, 2019.

[8] K. Allix, T. F. Bissyandé, J. Klein e Y. Le Traon, «AndroZoo: Collecting Millions of Android Apps for the Research Community,» en *Proceedings of the 13th International Conference on Mining Software Repositories*, ép. MSR '16, Austin, Texas: ACM, 2016, págs. 468-471, ISBN: 978-1-4503-4186-8. DOI: 10.1145/2901739.2903508. dirección: http://doi.acm.org/10.1145/2901739.2903508.

[9] M. contributors. «PRIVACYSCORE.» (2017), dirección: https://privacyscore.org/. (último acceso: 16/05/2023).

[10] H. Roy. «Terms of Service Didn't Read.» (2012), dirección: https://tosdr.org/. (último acceso: 16/05/2023).

[11] J. Crespo Guerrero, «APKFalcon: Servicio de usuarios para la evaluación y comprensión del impacto sobre la privacidad de aplicaciones móviles,» Universidad de Valladolid, 2023.

[12] PMI, ed., *Guía de los fundamentos para la dirección de proyectos (Guía del PMBOK)*, 5.ª ed. Newtown Square, PA: Project Management Institute, 2013, ISBN: 978-1-62825-009-1.

[13] Android Developers. «Permission tag.» (2022), dirección: https://developer.android.com/guide/topics/manifest/permission-element. (último acceso: 08/03/2023).

[14] «GanttProject - Free Project Management Application.» (2022), dirección: https://www.ganttproject.biz/. (último acceso: 09/03/2023).

[15] Indeed. «¿Cuánto se gana como uno Programador/a junior en España?» (2023), dirección: https://es.indeed.com/career/programador-junior/salaries. (último acceso: 12/03/2023).

[16] Wikipedia, *Almacén de datos — Wikipedia, La enciclopedia libre*, 2023. dirección: https://es.wikipedia.org/w/index.php?title=Almac%C3%A9n_de_datos&oldid=151173257, (último acceso: 10/06/2023).

[17] D. Truong. «Accessing your Data Warehouse with REST API's.» (2020), dirección: https://www.bizone.co.th/blogs/business-intelligence/rest-api-and-data-warehouse. (último acceso: 10/06/2023).

[18] R. Kimball y J. Caserta, *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Indianapolis, IN: Wiley, 2004.

[19] K. Bartley. «What is the difference between ETL and ELT?» (2023), dirección: https://rivery.io/blog/etl-vs-elt/. (último acceso: 10/06/2023).

[20] R. Fielding, et al., *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, Internet Engineering Task Force, 1999. dirección: https://tools.ietf.org/html/rfc2616.

[21] R. Fielding, et al., *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, RFC 7230, Internet Engineering Task Force, 2014. dirección: https://tools.ietf.org/html/rfc7230.

[22] R. Fielding, et al., *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, RFC 7231, Internet Engineering Task Force, 2014. dirección: https://tools.ietf.org/html/rfc7231.

[23] D. Hardt, *OAuth 2.0*, Internet Engineering Task Force (IETF), RFC 6749, 2012. dirección: https://tools.ietf.org/html/rfc6749.

[24] ByteByteGo. «SSL, TLS, HTTPS Explained.» (2022), dirección: https://www.youtube.com/watch?v=j9QmMEWmcfo. (último acceso: 11/06/2023).

[25] O. I. (OAI). «OpenAPI.» (2022), dirección: https://www.openapis.org/. (último acceso: 07/06/2023).

[26] R. Fielding, et al., *Media Type Specifications and Registration Procedures*, RFC 6838, Internet Engineering Task Force, 2013. dirección: https://tools.ietf.org/html/rfc6838.

[27] M. Upadhayay, A. Sharma, G. Garg y A. Arora, «RPNDroid: Android Malware Detection using Ranked Permissions and Network Traffic,» en *2021 Fifth World Conference on Smart Trends in Systems Security and Sustainability (WorldS4)*, 2021, págs. 19-24. DOI: 10.1109/WorldS451998.2021.9513992.

[28] M. Martin. «What is a Functional Requirement in Software Engineering?» (2023), dirección: https://www.guru99.com/functional-requirement-specification-example.html. (último acceso: 25/05/2023).

[29] K. Wiegers y J. Beatty, *Software Requirements*, 3rd. Microsoft Press, 2013, ISBN: 978-0735679665.

[30] G. van Rossum, F. L. Drake Jr. y N. Gough, *PEP 8 – Style Guide for Python Code*, Python Enhancement Proposal, RFC 822, 2001. dirección: https://pep.python.org/pep-0008/.

[31] S. Holywell. «SQL Style Guide.» (2022), dirección: https://www.sqlstyle.guide/. (último acceso: 29/04/2023).

[32] L. Bass, P. Clements y R. Kazman, *Software Architecture in Practice*. Addison-Wesley Professional, 2012.

[33] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[34] R. Kimball y M. Ross, *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Wiley, 2013.

[35] A. Silberschatz, H. F. Korth y S. Sudarshan, *Database System Concepts*, 7th. McGraw-Hill, 2019, ISBN: 978-1-259-34606-4.

[36] P. Dybka. «Chen Notation.» (2014), dirección: https://vertabelo.com/blog/chen-erd-notation/. (último acceso: 27/04/2023).

[37] Android Developers. «Permission-group tag.» (2023), dirección: https://developer.android.com/guide/topics/manifest/permission-group-element. (último acceso: 29/04/2023).

[38] Android Developers. «Archivo de manifiesto de la app: permission group.» (2023), dirección: https://developer.android.com/reference/android/Manifest.permission_group. (último acceso: 29/04/2023).

[39] Android Developers. «Archivo de manifiesto de la app: permission.» (2023), dirección: https://developer.android.com/reference/android/Manifest.permission. (último acceso: 29/04/2023).

[40] P. Dybka. «Crow's Foot Notation.» (2016), dirección: https://vertabelo.com/blog/crow-s-foot-notation/. (último acceso: 29/04/2023).

[41] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[42] H. Benita. «Preventing SQL Injection Attacks With Python.» (2019), dirección: https://realpython.com/prevent-python-sql-injection/. (último acceso: 07/06/2023).

[43] D. Hardt, *The OAuth 2.0 Authorization Framework*, RFC 6749, 2012. dirección: https://tools.ietf.org/html/rfc6749, (último acceso: 07/06/2023).

[44] G. Van Rossum, *PEP 257 – Docstring Conventions*, Python Enhancement Proposal, RFC 822, 2001. dirección: https://pep.python.org/pep-0257/.

[45] Vasileios Kemerlis. «pyAndroZoo.» (2017), dirección: https://pypi.org/project/pyAndroZoo/. (último acceso: 07/06/2023).

[46] J. Salowey, S. Turner, R. Housley, T. Polk y D. Cooper, «Textual Encodings of PKIX, PKCS, and CMS Structures,» inf. téc., 2015, RFC 7468. dirección: https://tools.ietf.org/html/rfc7468.

[47] J. Pomeyrol. «Guía de instalación de Ubuntu 20.04 LTS.» (2020), dirección: https://www.muylinux.com/2020/05/21/guia-instalacion-ubuntu-20-04-lts/. (último acceso: 08/06/2023).

Apéndice A

Ejemplo de app representada en *JSON*

```
{
    "App": {
        "hash": "...",
        "package": "com.whatsapp",
        "version_code": 230712004,
        "version_name": "2.23.7.12",
        "min_sdk_version": 16,
        "target_sdk_version": 33,
        "max_sdk_version": null,
        "category": "COMMUNICATION",
        "uses_permission_list": [
            {
                "Permission": {
                    "name": "android.permission.CAMERA",
                    "protection_level": "dangerous|instant",
                    "declared_group_list": null,
                    "rank_list": [
                        {
                            "Rank": {
                                "value": 0.0243902,
                                "rank_name": "RPNDroid",
                                "permission_name": "android.
                                   permission.CAMERA"
                            }
                        }
                    ]
                }
            },
            ...
        ],
```

```
        "defines_group_list": null,
        "extraction_metadata_list": [
            {
                "ExtractionMetadata": {
                    "source": "Androzoo",
                    "method": "Handmade",
                    "timestamp": "2023-05-08T17:38:40"
                }
            }
        ],
        "score_list": [
            {
                "Score": {
                    "value": 6.29756,
                    "rank_name": "RPNDroid",
                    "app_hash": "..."
                }
            },
            {
                "Score": {
                    "value": 5.0,
                    "rank_name": "Tosdr",
                    "app_hash": "..."
                }
            }
        ]
    }
}
```

Apéndice B

Guía de instalación

En este apéndice se redacta una guía de instalación de todos los componentes del *warehouse* así como su puesta en marcha. Por motivos de seguridad, las contraseñas utilizadas se encuentran censuradas con la palabra "*password*" en las figuras y/o códigos que se muestren.

Como precondición al proceso de instalación del *warehouse*, se ha de tener un sistema Linux instalado y en funcionamiento. En el caso concreto de esta instalación, se utiliza un sistema Linux (Ubuntu 22.04.2 LTS) ya que tiene soporte de actualizaciones de seguridad y mantenimiento hasta Abril de 2027. El sistema se ejecuta sobre una máquina virtual con 4 núcleos de CPU, 4 GB de memoria RAM y 25 GB de disco duro de almacenamiento. Existen múltiples guías de instalación de este sistema operativo, entre ellas [47].

B.1. Descarga del código fuente del *warehouse*

En primer lugar, es necesario tener instalado el sistema de control de versiones *git* en nuestro sistema. Para ello, en el caso de los sistemas Linux basados en Ubuntu, basta con ejecutar en una terminal la siguiente línea de código:

```
sudo apt install git
```

Una vez tenemos *git* en nuestro sistema, nos dirigimos con el comando *cd* hasta el directorio en que queremos instalar el *warehouse*. Una vez en el, ejecutamos el comando:

```
git clone https://github.com/peres317/app_warehouse.git
```

Este comando, descargara todo el código necesario para la instalación del *warehouse*. Por último, una vez descargado el código, tendremos que movernos al directorio del *warehouse*: *cd ./app_warehouse/*.

B.2. Instalación del sistema gestor de base de datos

Primero tenemos que instalar el sistema gestor de bases de datos *MySQL*, para ello, en el caso de los sistemas Linux basados en Ubuntu, ejecutamos la siguiente línea de código:

```
sudo apt install mysql-server
```

Ahora, pasamos a configurar *MySQL* siguiendo los siguientes pasos:

1. Modificamos *"password"* en el archivo de configuración *#install_scripts/mysql-config.sql* por una contraseña segura que utilizará el *warehouse* para acceder a su base de datos en *MySQL*.

2. Entramos en *MySQL* con el usuario *root* con el comando:

   ```
   sudo mysql
   ```

   Y modificamos la contraseña de acceso *root* del *MySQL*:

   ```
   ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password';
   EXIT
   ```

3. Entramos ahora a *MySQL* con el usuario *root* y ejecutamos el archivo de creación del usuario que usará el *warehouse* para acceder a la base de datos con el siguiente comando:

   ```
   mysql -u root -p
   source #install_scripts/mysql-config.sql
   ```

4. Por último, ejecutamos el *script* que creará las tablas y las entradas del último estado guardado del *warehouse*:

   ```
   source #install_scripts/warehouse-backup.sql
   EXIT
   ```

B.3. Instalación del entorno *Python* del *warehouse*

Ubuntu 20.04.2 LTS viene con *Python 3.10* instalado por defecto luego no es necesario instalarlo. Si lo fuera, se instala con la orden: *sudo apt install python3*. Una vez se tiene *Python* instalado, hay que instalar el paquete *python3-venv* para poder crear entornos virtuales y ejecutar el archivo de creación del entorno virtual:

```
sudo apt install python3-venv
./#install_scripts/python-config.sh
```

Por último, hay que modificar el código de la librería *pyandrozoo* para que descargue en el directorio correcto y con un *timeout* de seguridad para que no se quede bloqueado en descargas muy largas:

```
nano venv/lib/python3.10/site-packages/pyandrozoo/api.py
```

```
  GNU nano 6.2   venv/lib/python3.10/site-packages/pyandrozoo/api.py *
20      def __write_file(self, r, **kwargs):
21          sha256 = parse_qs(urlparse(r.url).query)['sha256'][0]
22          if r.status_code == 200:
23               with open(sha256+'.apk', "wb") as apk_file:
24                   apk_file.write(r.content)
25          else:
26              print('Request failed for: {}'.format(sha256))
27          return r
28
29      def get(self, sha256_list):
30          reqs = []
31          parts = urlparse(self.root_url)
32          for sha256 in sha256_list:
33              payload = {'sha256': sha256}
34              payload.update(self.payload)
35              url = urlunparse(parts[:4] + (urlencode(payload),) + parts[5:])
36              reqs.append(grequests.get(url,
37                                        hooks={'response': [self.__write_file]}))
38          grequests.map(reqs, size=30, exception_handler=exception_handler)
```

```
  GNU nano 6.2   venv/lib/python3.10/site-packages/pyandrozoo/api.py *
20      def __write_file(self, r, **kwargs):
21          sha256 = parse_qs(urlparse(r.url).query)['sha256'][0]
22          if r.status_code == 200:
23               with open('data/androzoo_downloads/'+sha256+'.apk', "wb") as ap>
24                   apk_file.write(r.content)
25          else:
26              print('Request failed for: {}'.format(sha256))
27          return r
28
29      def get(self, sha256_list):
30          reqs = []
31          parts = urlparse(self.root_url)
32          for sha256 in sha256_list:
33              payload = {'sha256': sha256}
34              payload.update(self.payload)
35              url = urlunparse(parts[:4] + (urlencode(payload),) + parts[5:])
36              reqs.append(grequests.get(url,
37                                        hooks={'response': [self.__write_file]},
38                                        timeout=300))
39          grequests.map(reqs, size=30, exception_handler=exception_handler)
```

Figura B.1: Modificaciones del paquete *pyandrozoo*.

Las líneas modificadas son la 23 y la 37 introduciendo las modificaciones que se aprecian en la figura B.1. Para almacenar el fichero si se ha utilizado el editor propuesto (*nano*), se usa la combinación de teclas Ctrl + X.

B.4. Configuración del *warehouse*

Estamos ya entrando en la recta final de la instalación del *warehouse*. Ahora hay que modificar el fichero de configuración del *warehouse data/config.json* con las contraseñas que hemos puesto en *MySQL*, la clave de *API* facilitada por Androzoo, las claves de usuarios que deseamos habilitar, el directorio absoluto del *warehouse* y el del fichero índice comprimido de aplicaciones de Androzoo [8].

Además, tenemos que generar el certificado *SSL* con el comando:

```
openssl req -x509 -newkey rsa:4096 -keyout data/key.pem -out data/cert.pem -sha256
    -days 365
```

Se solicitará una contraseña para encriptar la clave primaria. Es muy importante recordarla puesto que se utilizará cada vez que se arranque el *warehouse*.

B.5. Ejecución del *warehouse*

Para iniciar el *warehouse* partiendo de una terminal en el directorio base del código *app_warehouse/*, se han de ejecutar los siguientes comandos:

```
source venv/bin/activate
python api/main.py
```

Apéndice C

Manual de uso

En este apéndice se muestran ejemplos de uso del *warehouse* a través de su *API*. Por motivos de seguridad, las contraseñas utilizadas se encuentran censuradas con la palabra "password" en las figuras y/o códigos que se muestran. Para hacer uso de la *API* de acceso del *warehouse* se utiliza la herramienta *Postman* en su versión gratuita. Para la realización de este manual las peticiones al *warehouse* se realizan desde la misma máquina en que se encuentra desplegado, por ello, la dirección de acceso a este es "localhost". Esta dirección de acceso junto con la con la contraseña de acceso han de ser modificadas en las peticiones acorde al despliegue concreto del *warehouse* al que se quiere acceder para su funcionamiento.

C.1. Guía básica de uso de *Postman*

En primer lugar, es necesario registrarse en su página *web* (`https://www.postman.com/`) para hacer uso de su aplicación. Una vez tenemos una cuenta, se ha de instalar el programa en nuestro sistema y ejecutarlo. La primera vez que se ejecuta, se pedirá que inicies sesión en la cuenta que te has creado y, una vez iniciada sesión, se obtiene una ventana similar a la de la figura C.1.

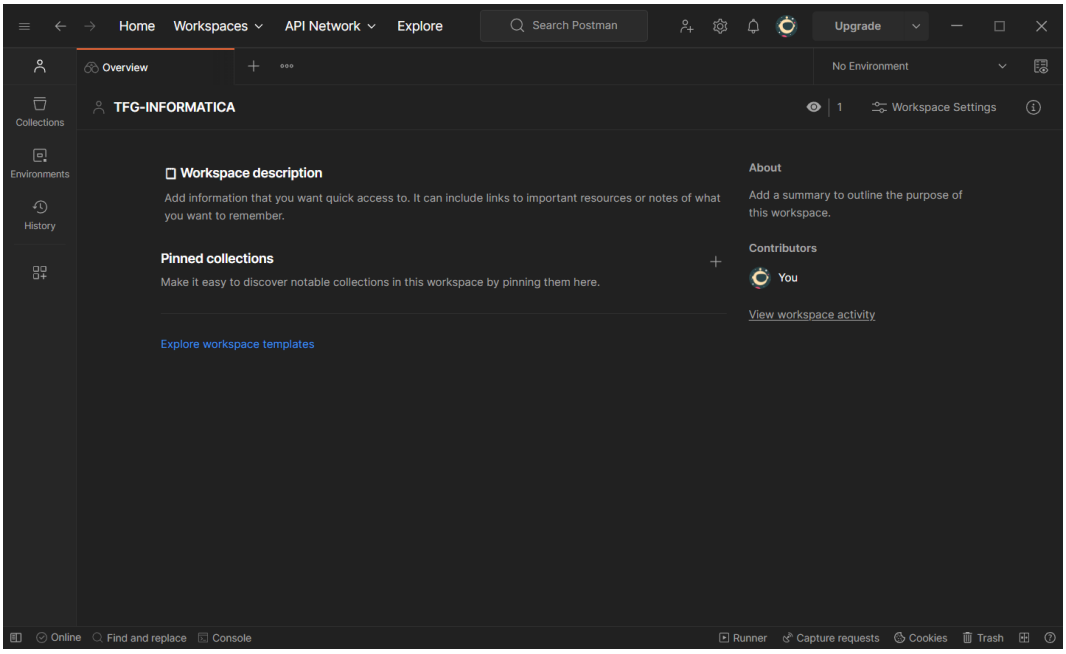

Figura C.1: Ventana de inicio de *Postman*.

C.1.1. Creación de una petición

Una vez se tiene el programa en funcionamiento, se crean nuevas peticiones por medio del botón "+" que aparece a la derecha del panel "Overview" (figura C.2).

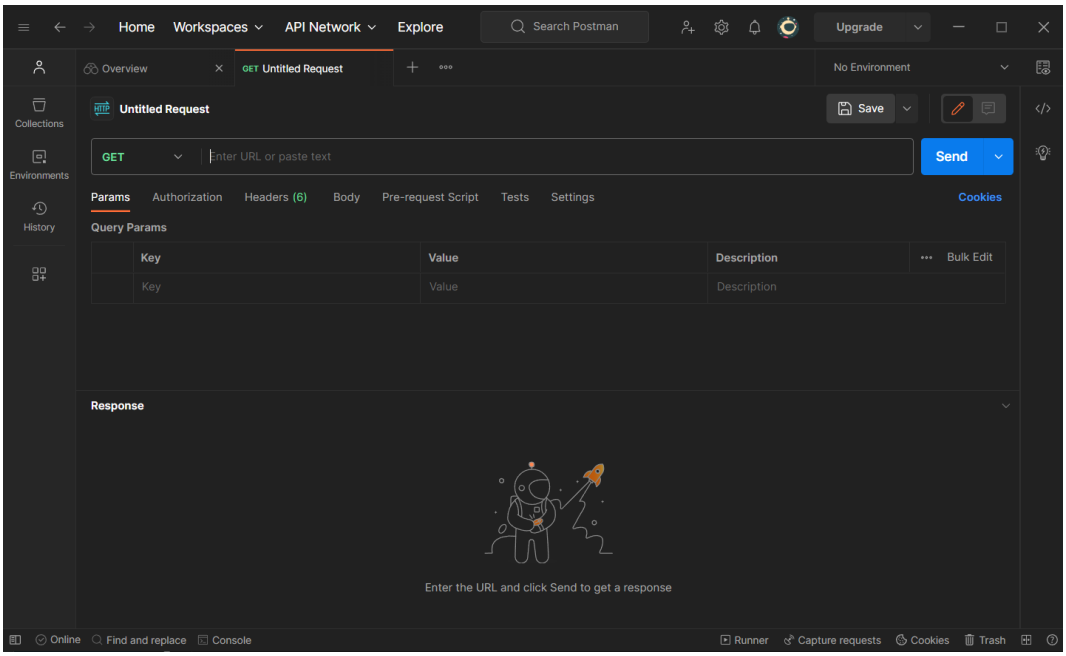

Figura C.2: Ventana de petición de *Postman*.

En esta (figura C.2), se ha de especificar el tipo de petición que es: *GET* o *POST* por medio del desplegable que por defecto tiene la opción *GET*. La dirección *URL* de la petición en el cuadro de texto situado a la derecha. Los parámetros de la petición en forma clave valor que aparecen como "Query Params". Las cabeceras de la petición en forma clave valor en la pestaña *Headers*. Y, en caso de ser necesario, el cuerpo de la petición por medio de la pestaña "Body". Una vez se tiene la petición preparada se puede lanzar por medio del botón azul "Send".

Para facilitar la creación de todas las peticiones que aparecen en este manual, se facilita un archivo *JSON* que se puede importar directamente en *Postman* con todas ellas así como respuestas de ejemplo.

En todas y cada una de las peticiones, se indica la contraseña de acceso a través de una cabecera "Authorization" con valor "Bearer password" (figura C.3).

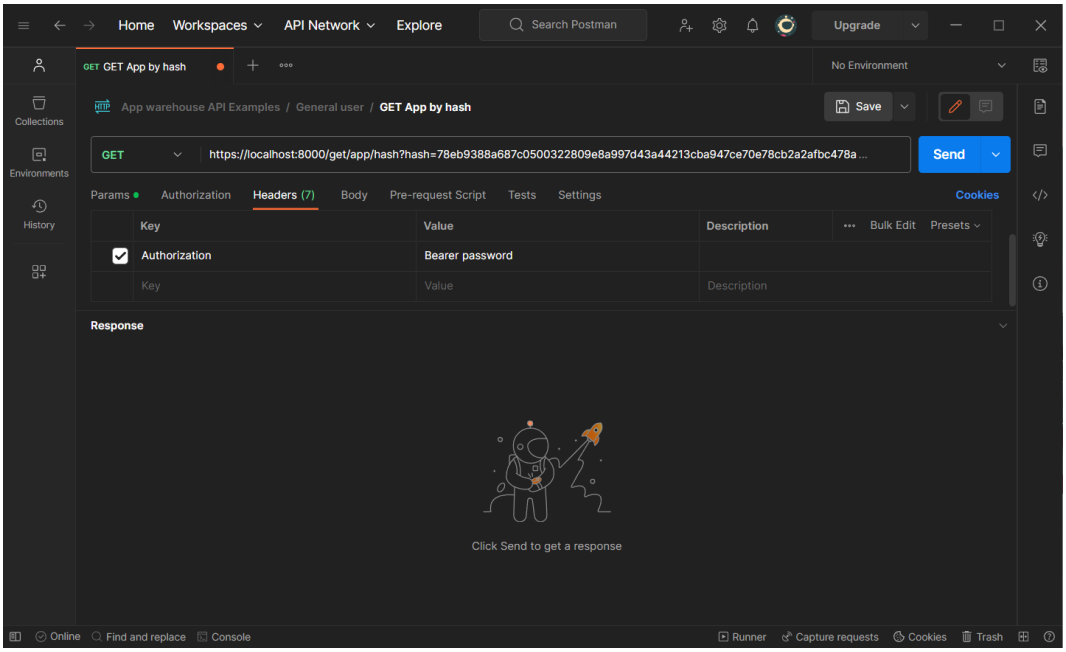

Figura C.3: Cabecera de contraseña.

C.2. Funcionalidad de los usuarios generales

En esta sección se muestran ejemplos de las peticiones que pueden realizar los usuarios generales.

C.2.1. Descargar metadatos de *app* por *hash*

A mayores de lo ya indicado en la sección C.1.1, para llevar a cabo esta petición se ha de modificar la *URL* (*https://ip:8000/get/app/hash*) e introducir el parámetro *hash* con valor el *hash* de la aplicación que queremos descargar. En la figura C.4 se muestra un ejemplo de la petición realizada y su respuesta.

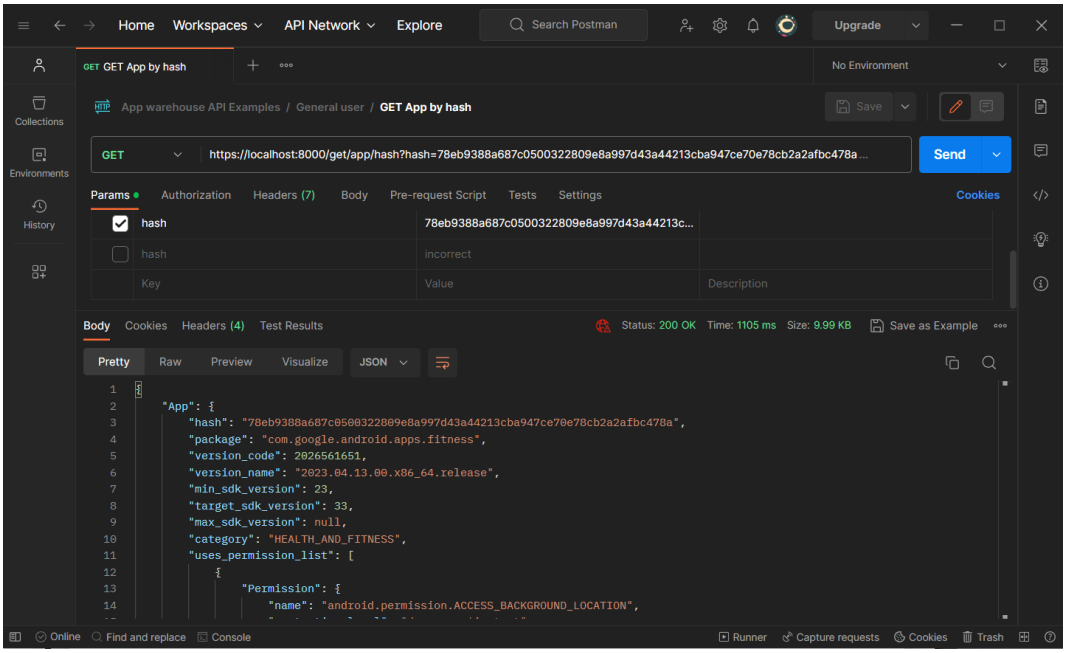

Figura C.4: Descargar metadatos de *app* por *hash*.

C.2.2. Descargar metadatos de *app* por *package*

A mayores de lo ya indicado en la sección C.1.1, para llevar a cabo esta petición se ha de modificar la *URL* (*https://ip:8000/get/app/package*) e introducir el parámetro *package* con valor el *package* de la aplicación que queremos descargar. En la figura C.5 se muestra un ejemplo de la petición realizada y su respuesta.

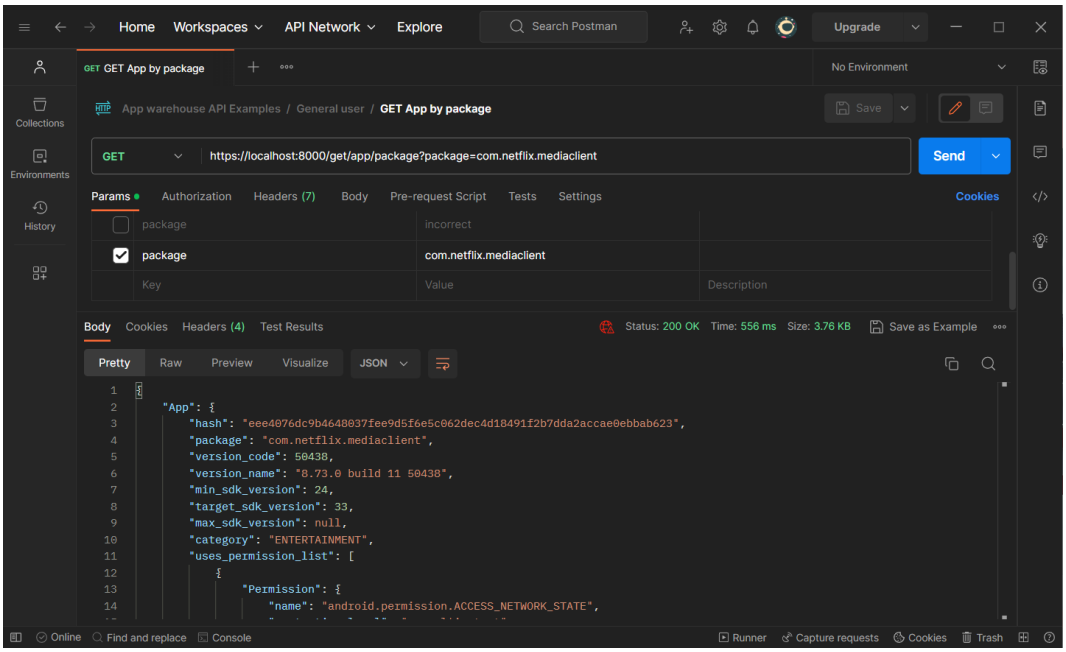

Figura C.5: Descargar metadatos de *app* por *package*.

C.2.3. Solicitar carga de *app* por *package*

A mayores de lo ya indicado en la sección C.1.1, para llevar a cabo esta petición se ha de modificar el tipo de petición a *POST*, la *URL* (*https://ip:8000/post/app/package*) e introducir como cuerpo de la petición:

```
{
    "package": "package.name.of.application"
}
```

En la figura C.6 se muestra un ejemplo de la petición realizada y su respuesta.

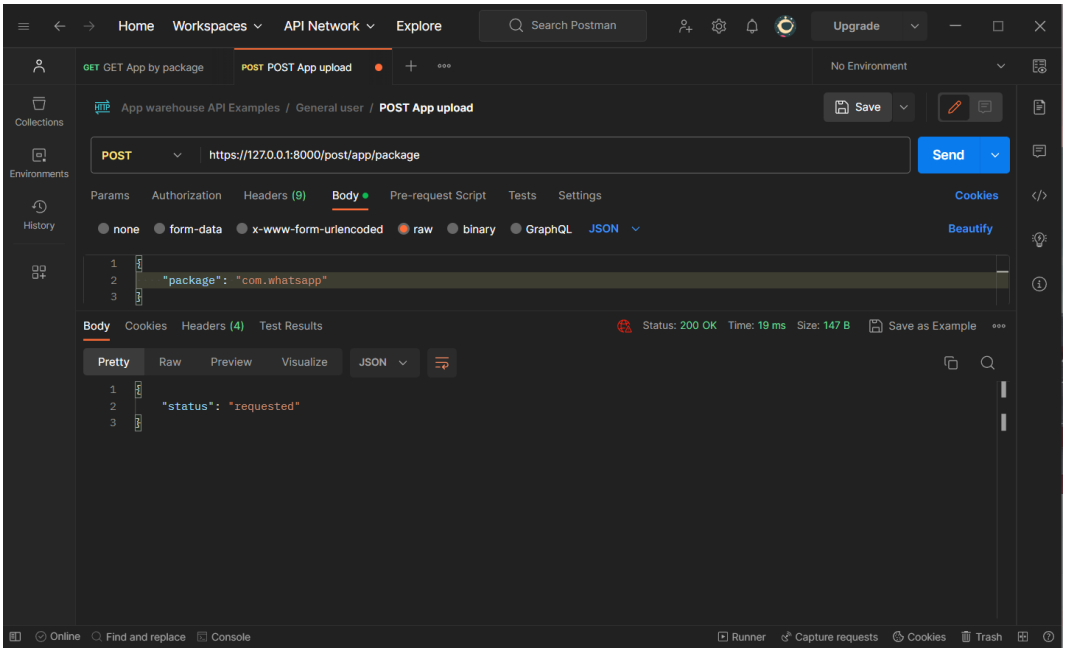

Figura C.6: Solicitar carga de *app* por *package*.

C.3. Funcionalidad de los administradores

En esta sección se muestran ejemplos de las peticiones que pueden realizar los administradores.

C.3.1. Aplicar métricas de privacidad

En este caso, solamente se ha de modificar la *URL* de la petición (*https://ip:8000/admin/update/apply_metrics*). En la figura C.7 se muestra un ejemplo de la petición realizada y su respuesta.

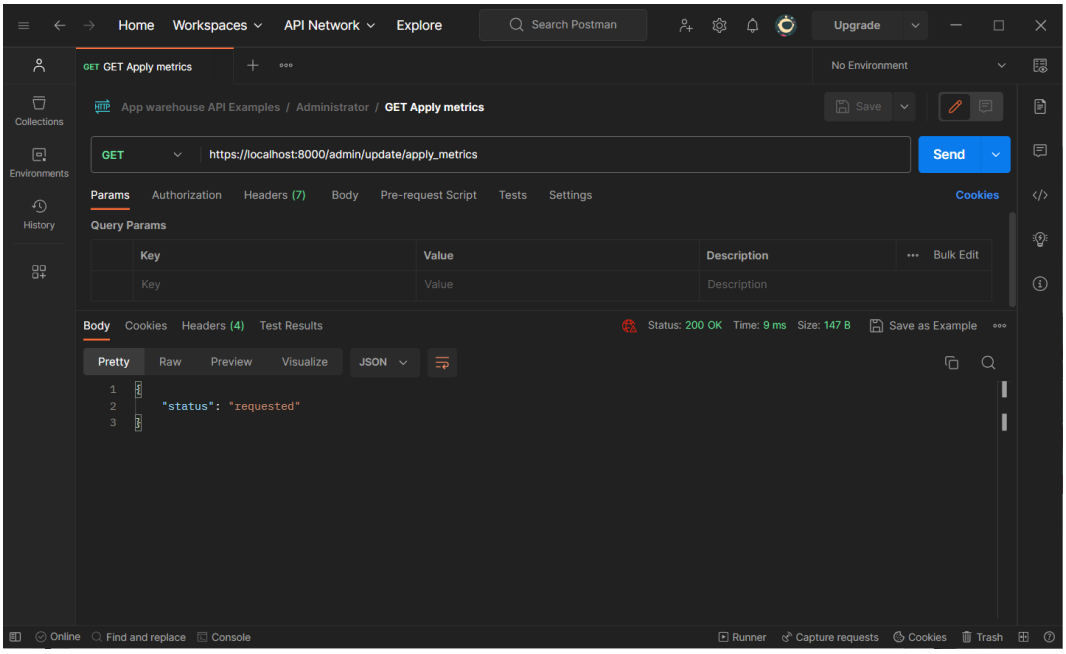

Figura C.7: Aplicar métricas de privacidad.

C.3.2. Solicitar carga de n *apps* aleatorias

Ahora, se ha de modificar la *URL* de la petición (*https://ip:8000/admin/get/n_random_apps*) e introducir el parámetro *n_apps* indicando el número de aplicaciones a cargar en el *warehouse*. En la figura C.8 se muestra un ejemplo de la petición realizada y su respuesta.

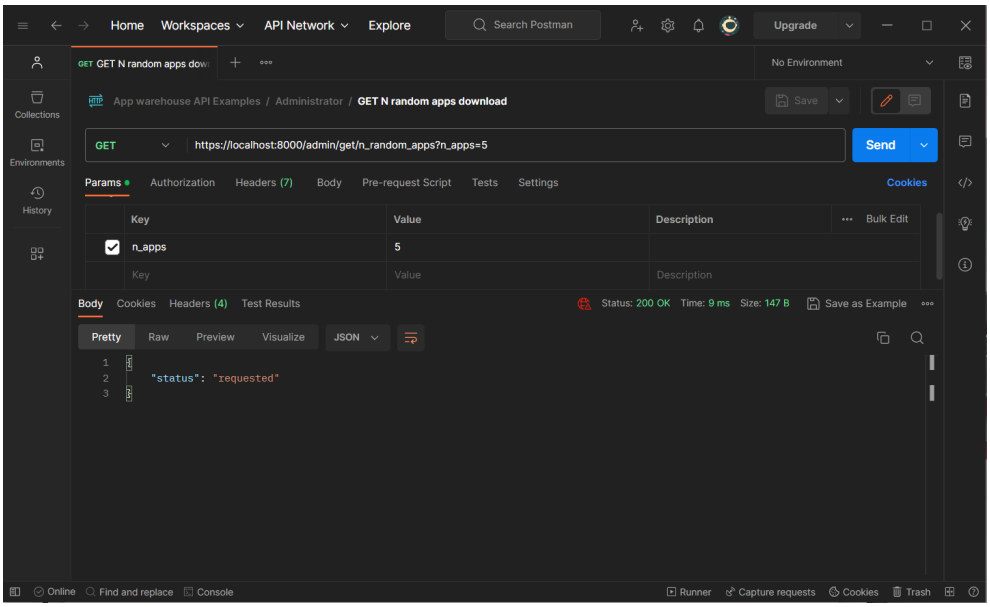

Figura C.8: Solicitar carga de n *apps* aleatorias.

C.3.3. Carga datos

Ahora, se ha de modificar el tipo de petición a *POST*, la *URL* de la petición (*https://ip:8000/admin/post/json*) y el cuerpo indicando el dato a cargar en el *warehouse*. En la figura C.9 se muestra un ejemplo de la petición realizada de subida de una nueva métrica y su respuesta.

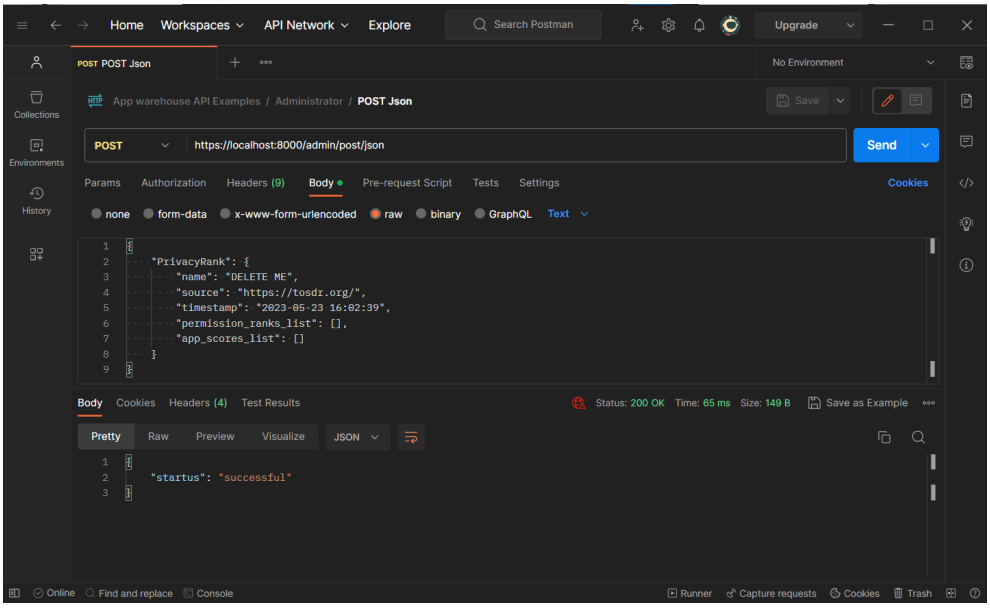

Figura C.9: Carga datos.

C.3.4. Solicitar actualización de los datos AOSP

Ahora, se ha de modificar la *URL* de la petición (*https://ip:8000/admin/update/aosp*). En la figura C.10 se muestra un ejemplo de la petición realizada y su respuesta.

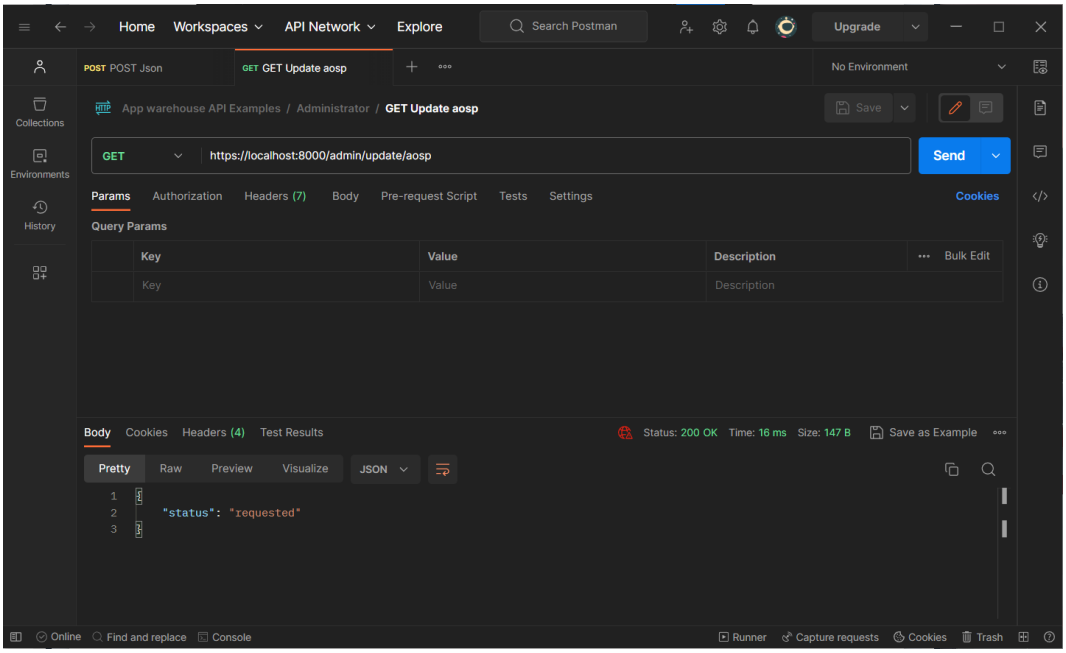

Figura C.10: Solicitar actualización de los datos AOSP.

Apéndice D

Contenidos del *warehouse*

En este apéndice se detalla el contenido del almacén conseguido a fecha 11 de Junio de 2023. Para ello, se muestran 2 visualizaciones interactivas de tipo *dashboard* realizadas con la herramienta *PowerBI* de *Microsoft*. Estas visualizaciones se actualizan gracias al filtrado de datos que se puede realizar por medio de sus objetos visuales.

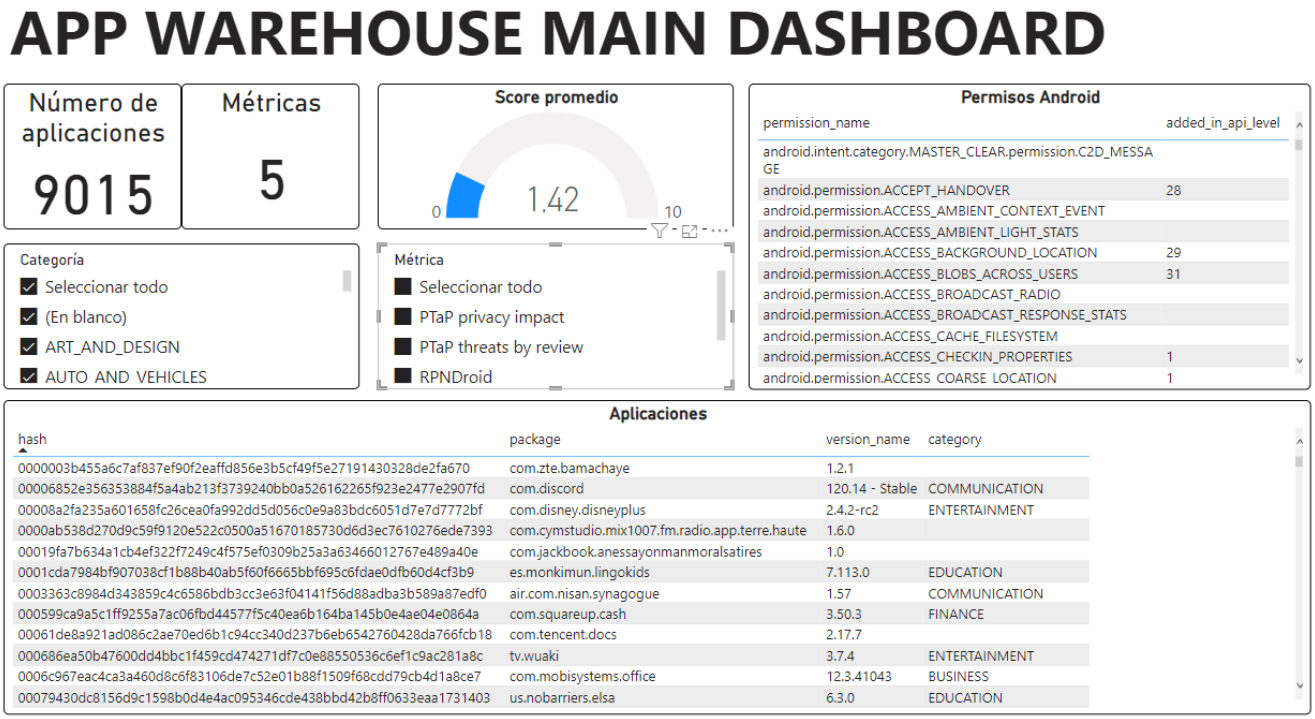

Figura D.1: *Dashboard* principal.

En la figura D.1 se puede ver la primera visualización. Esta se centra en mostrar un resumen de todos los contenidos del *warehouse*. En primer lugar se ve la cantidad total de aplicaciones y métricas que hay almacenadas en el *warehouse*: 9015 y 5 respectivamente. Seguidamente, se muestra un gráfico resumen de la puntuación promedio que obtienen las aplicaciones filtradas con las métricas indicadas. Estos filtros se encuentran en forma de listas de elementos seleccionables permitiendo filtrar los datos por medio de la elección de la categoría de la aplicación y la métrica de privacidad que se aplica. Por último, en la zona inferior se muestra una tabla con información sobre las aplicaciones seleccionadas y en la zona derecha

la metainformación sobre los permisos Android que tiene el *warehouse*.

En la figura D.2 se puede ver la segunda visualización. Esta se centra en toda la información que contiene el *warehouse* sobre una aplicación concreta. En el caso concreto de la captura que se muestra se muestra la información sobre la aplicación *Netflix*.

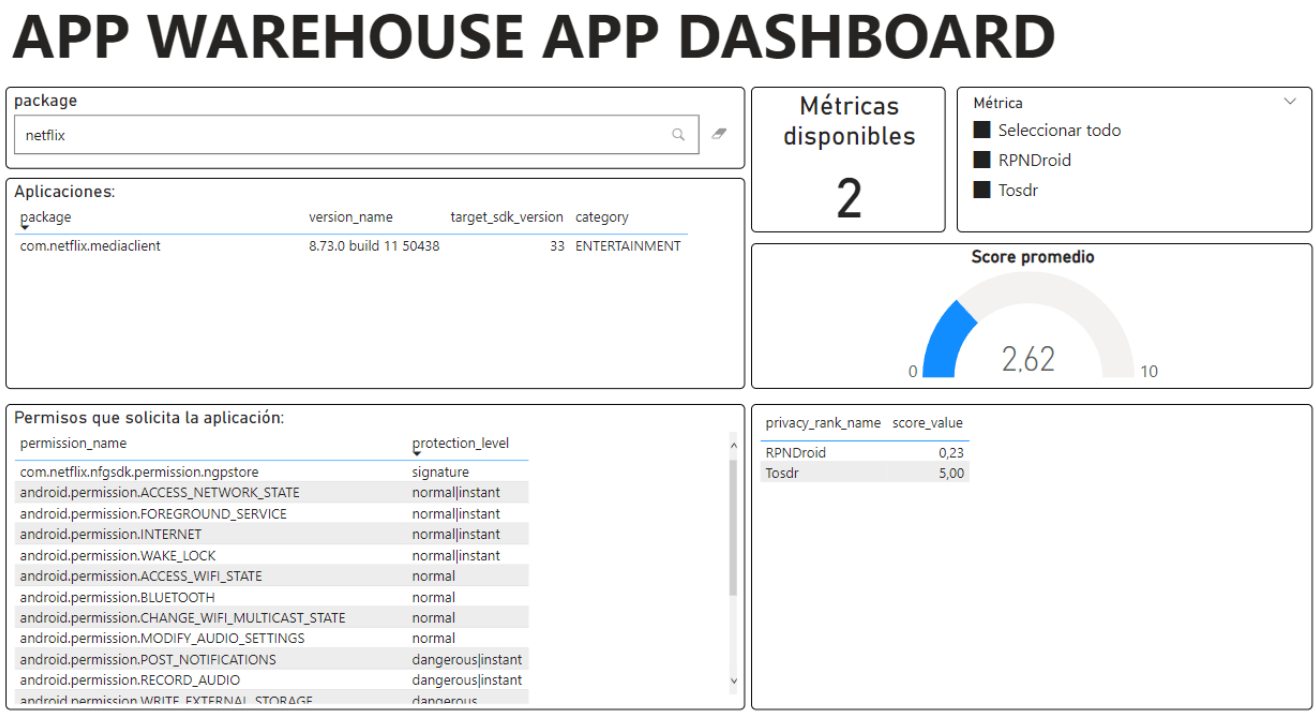

Figura D.2: *Dashboard* de aplicaciones concretas.

En primer lugar, se permite la búsqueda de aplicaciones por medio de un buscador que filtra las aplicaciones por aquellas que contienen en su nombre de paquete la cadena por la que se esta filtrando. Seguidamente, se muestra una lista con las aplicaciones que cumplen el filtro para poder seleccionar de entre estas la que queremos ya que cabe la posibilidad de que haya distintas versiones de una misma aplicación. A la derecha, se muestra las métricas disponibles para la aplicación, una lista que permite seleccionar cada una de estas y un gráfico resumen de la puntuación promedio que recibe la aplicación por las métricas. Finalmente, en el área inferior se muestran los permisos que la aplicación solicita para su funcionamiento y las puntuaciones que tiene.