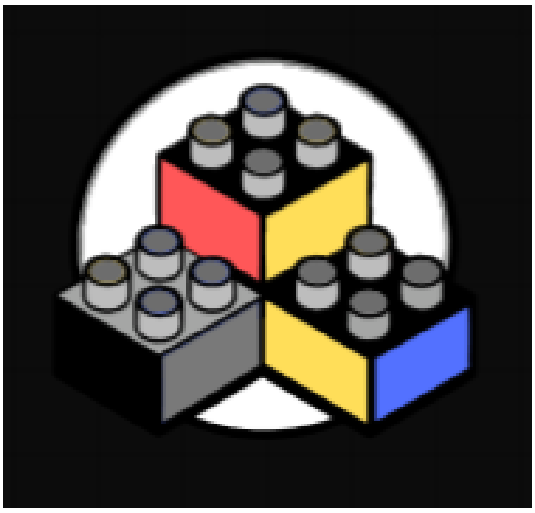


PROYECTO DE START-UP

David Pires Manzanares
Desarrollo de Interfaces 2º DAM

Índice:

Índice de imágenes

1. Introducción al proyecto.

Desde su fundación en agosto de 1932, la empresa LEGO ha experimentado dos etapas de crecimiento exponencial: la primera a partir de la Gran Depresión, y la segunda a partir de la década de los 70, en la que se estableció el modelo de juguete actual que conocemos.

A día de hoy, LEGO es una de las empresas de juguetes que más dinero factura en el mundo, debido a su gran capacidad de adaptación mercados emergentes a través de la creación de distintas líneas de productos para todo tipo de edades (infantiles como la marca DUPLO y para adultos con las líneas de IDEAS y ICONS), además de contar con licencias exclusivas de distintas franquicias muy conocidas a nivel mundial (Harry Potter, Disney, Star Wars), haciendo de cada uno de sus sets una pieza exclusiva.

Imagen 1. Oficinas de la empresa LEGO.

Aunque su plataforma es muy intuitiva y sencilla de utilizar, LEGO no dispone de ningún sistema a través del cual los coleccionistas o clientes puedan registrar los productos que ya tienen, lo que hace que los inventarios personales de los consumidores se tengan que registrar en listas de papel, o a través de documentos informáticos, que tienen una limitada capacidad de edición y accesibilidad limitada.

En este punto es donde Brick by Brick, que facilitará a los usuarios el llevar un control de sus sets de una manera muy intuitiva y sencilla, y poder acceder a ella en cualquier momento a través de su dispositivo móvil.

2. Objetivos del proyecto.

Esta aplicación tiene como objetivos principales:

- Ofrecer a los usuarios una posibilidad de crear un inventario interactivo y sencillo de utilizar.
- Permitir que los coleccionistas registren tanto los sets que poseen, como los sets que pretendan comprar a través de un sistema de "wishlist".
- Acceder de manera rápida y segura a su usuario, pudiendo tener varios perfiles en un mismo dispositivo.

3. Especificaciones técnicas de la aplicación.

En este apartado se expondrá de manera detallada cada uno de los requerimientos técnicos, herramientas, lenguajes y demás especificaciones utilizadas para la creación y correcto funcionamiento de la aplicación

3.1. Entornos de desarrollo integrados (IDE) utilizados.

En el caso de Brick by Brick se ha decidido utilizar el entorno de Android Studio. Esta elección es debido a que este IDE ha sido el marco de referencia de los desarrolladores de aplicaciones móviles desde que este editor de código se lanzó en 2013.

Entre las funciones más representativas de este IDE se encuentran:

- Editor de código avanzado con autocompletado y refactorización inteligente.
- Capacidades de depuración para todas las capas disponibles en la app.
- Funcionamiento directo con plataformas de control de versiones como Git.
- Emulador a tiempo real de la aplicación en distintos tamaños de dispositivos y versiones.

La versión de Android utilizada para el desarrollo de la app fue {METER VERSION}.

3.2. Lenguajes de programación.

Aunque la aplicación tiene cierta complejidad, sólo se han utilizado dos lenguajes de programación: uno orientado a la parte frontal de la interfaz, y otro para el backend.

- Java: Lenguaje multiplataforma con programación orientada a objetos, que sustenta aplicaciones sistemas operativos, softwares empresariales… Se aplica en toda la parte de desarrollo de código backend de esta aplicación.
- XML: Este lenguaje de programación está basado en etiquetas, y representa la información de manera muy estructurada, y es uno de los formatos más utilizados para los frontales de aplicaciones. Además, está perfectamente integrado en los proyectos de Android Studio.

Imagen 2. Lenguajes de programación Java y XML

3.3. Bases de datos.

Uno de los elementos esenciales de esta aplicación es, como no puede ser de otra manera al tratarse de un inventario, es la plataforma utilizada para registrar toda la información de los usuarios, sets…

En Brick by Brick se ha optado por la utilización de Firebase, que es una plataforma en la nube muy utilizada para el desarrollo de aplicaciones móviles. Su diseño es de tipo "no relacional", ya que sus esquemas son flexibles y la distribución de sus datos no se distribuye en filas y columnas, sino que tiene esquemas flexibles que permiten un desarrollo mucho más rápido.

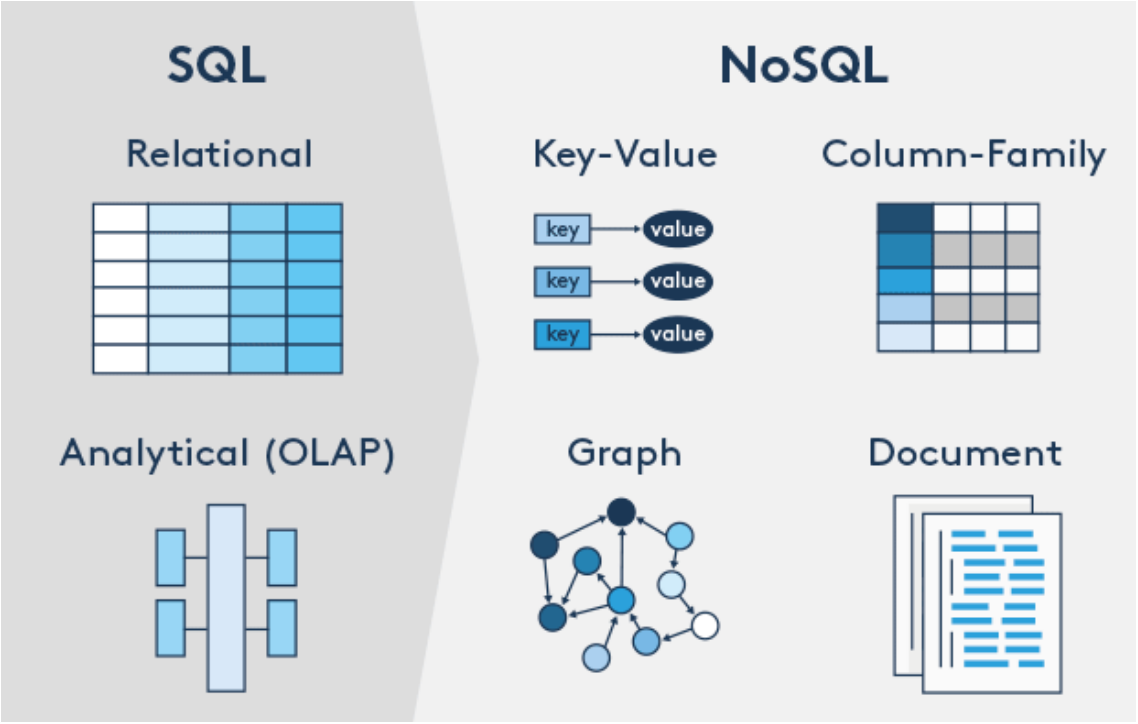


Imagen 3. Diferencias entre BBDD relacionales y no relacionales

3.4. Herramientas de diseño.

Para realizar el diseño de la interfaz de la aplicación existen muchas plataformas especializadas, las cuales ofrecen una gran variedad de herramientas de edición.

En este caso se utilizó la plataforma ***Miró,*** que es una plataforma online que permite realizar diseños muy creativos, y que pone a disposición de los usuarios una gran variedad de elementos para poder editar de manera prácticamente ilimitada las interfaces creadas.

4. Desarrollo de la aplicación.

4.1. Creación de componentes visuales.

Es evidente que, en una aplicación, crear la interfaz es una de las tareas más importantes, ya que a través de ella los usuarios interactuarán con la aplicación, y de ella dependerá que estos quieran seguir o no usándola.

Android Studio permite crear aplicaciones muy dinámicas y sencillas de utilizar a través de distintas herramientas que iremos desgranando en los siguientes subapartados.

4.1.1. Componentes y eventos.

El IDE de Android Studio permite al desarrollador generar la aplicación a su antojo, implementando todas las funcionalidades que desee a través de los distintos componentes que ofrece.

Estos componentes se organizan como una matrioska, en la que se van encontrando unos componentes dentro de otros. En orden ascendente, encontramos:

- Las **vistas**, que son los elementos que componen la interfaz, como imágenes, textos…
- Los **layouts**, que engloban todas las vistas en distintos formatos (RelativeLayout, LinearLayout...).
- Intents: Eventos para lanzar una actividad.
- **Actividades:** Es el esqueleto de la aplicación, y para cada actividad existirá un solo layout.
- **Servicios:** Procesos que se ejecutan en segundo plano cuando se ejecuta la aplicaciones y las actividades.

Estos elementos son indispensables para la creación y correcto funcionamiento de la aplicación, ya que sin ellos no se concibe ningún tipo de funcionalidad. En Brick by Brick, se han utilizado una gran cantidad de componentes distintos para hacer la aplicación atractiva e intuitiva.

Como se puede observar en la imagen inferior, se aprecian dos imágenes: una del login y otra de la colección de sets disponible, y en ellos se pueden observar distintos componentes, que se detallan a continuación. Cabe destacar que cada una de las imágenes corresponde a una actividad, dentro de la cual existe un layout con distintos componentes, como explicábamos anteriormente:

1. ImageView: Contiene la imagen del logo de la marca.
2. TextView: La aplicación está llena de ellos, ya que todo el texto que se observa en las distintas imágenes se encuentra dentro de un elemento de este tipo.
3. EditText: Estos componentes son inputs de usuario, es decir, el usuario puede hacer click en esas zonas e introducir texto.
4. Button: Componentes a través de los cuales el usuario ejecuta una acción, como hacer login, añadir productos al carrito etc.
5. CardView: Este componente lo destacamos en este apartado ya que es la base de la aplicación; todos los sets, tanto en la pantalla de inicio, como en la zona de colecciones y en el propio inventario tienen este componente, que le da un toque moderno e intuitivo.

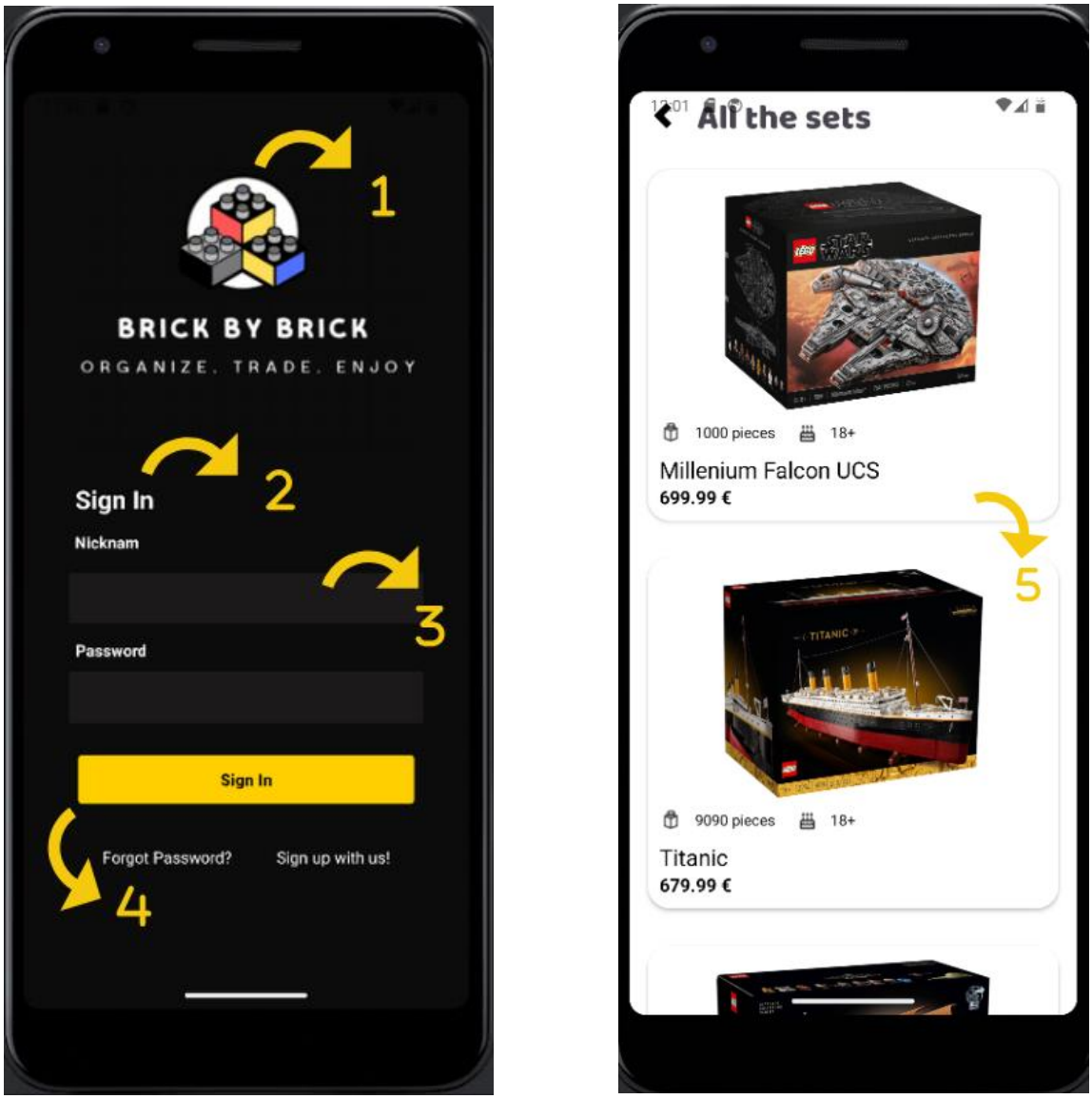


Imagen 4. Componentes de distintos layouts de la aplicación.

4.1.2. Atributos de la aplicación.

En este IDE los atributos son los encargados de habilitar funciones durante el diseño o comportamientos durante la compilación. Se encuentra en lenguaje XML, y se distribuyen mediante etiquetas.

En el apartado anterior se hablaba de que las vistas eran las que englobaban todos los componentes de la interfaz, y cada uno de esos componentes de los layouts en cuanto a su diseño, están programados en XML a través de atributos, es decir, que los atributos le dan propiedades distintas a cada uno de los componentes para así poder editarlos.

A continuación, se muestra una captura del código de la aplicación, en la que se pueden observar los distintos componentes, y cada uno de ellos tiene distintos atributos, que permiten editar su ancho, alto, orientación, contenido…

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000000"
    android:backgroundTint="#0C0C0C"
    tools:context=".LogInActivity">

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_percent="0.08" />

    <androidx.constraintlayout.widget.Guideline
        android:id="@+id/guideline14"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintGuide_percent="0.92" />

```

Imagen 5. Atributos de componentes en un layout.

4.1.3. Elementos listeners

Otro de los elementos fundamentales de las aplicaciones Android son los llamados *eventos de entrada* o *listeners,* a través de los cuales se puede identificar o definir el comportamiento del usuario respecto a una acción en la interfaz.

Estos eventos se gestionan desde código a través del lenguaje Java, en el que está programada toda la aplicación.

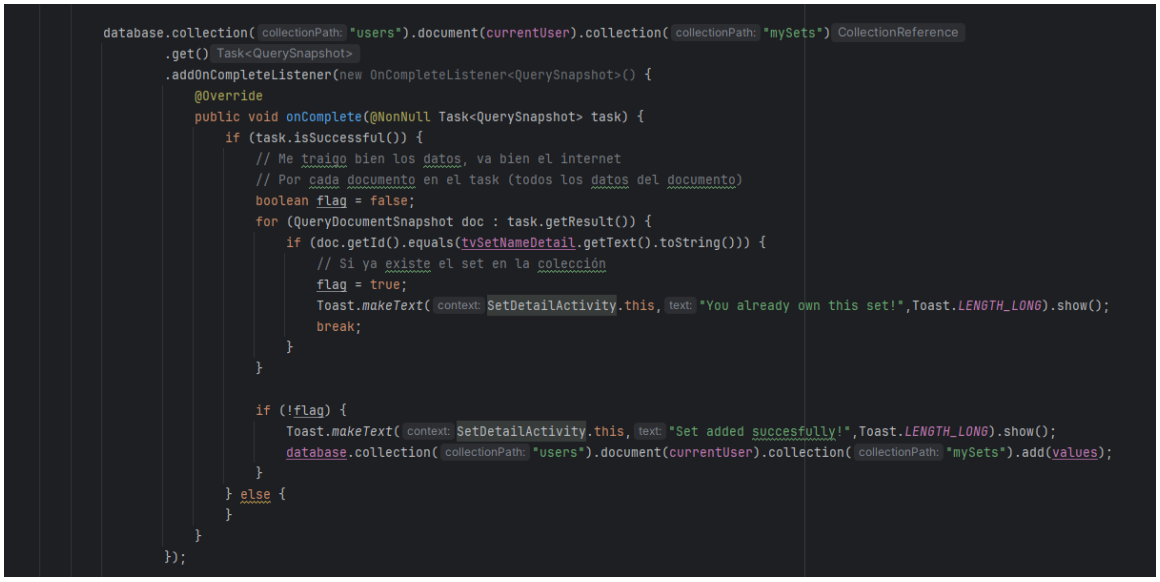


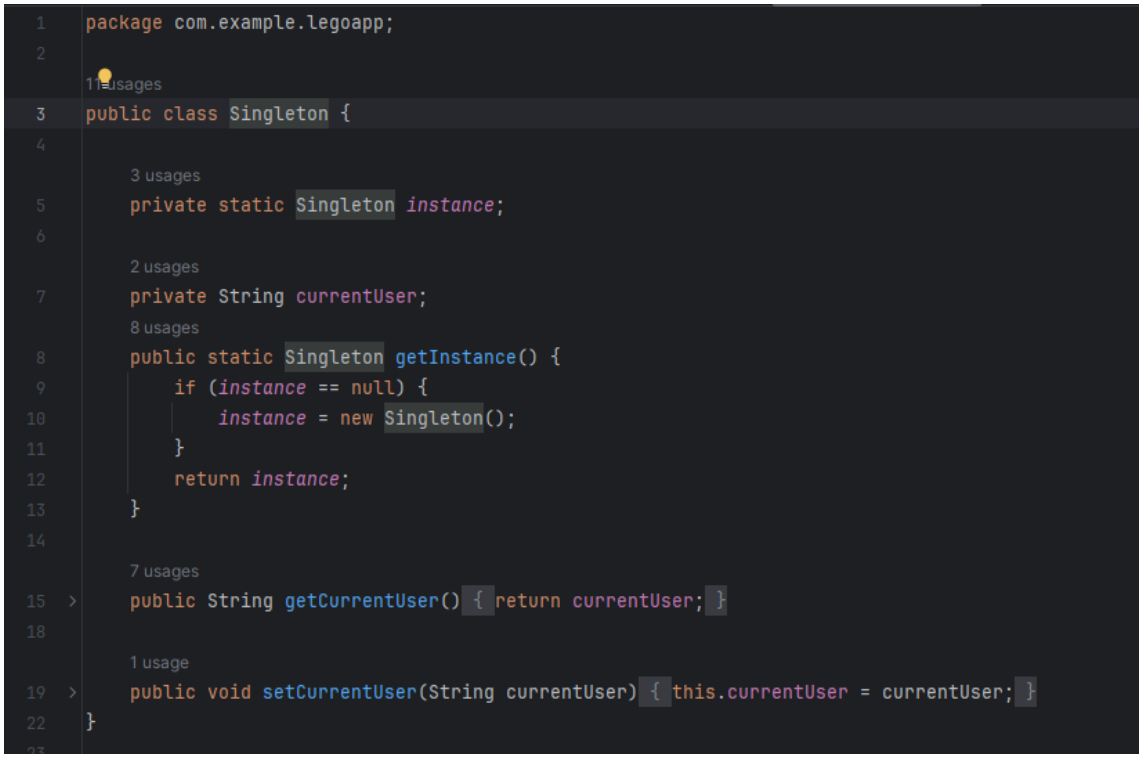
Imagen 6. Ejemplo de evento de entrada o listener

En la imagen anterior se puede apreciar cómo se utilizan los listeners. En este caso, el evento que se controla es un OnCompleteListener, que se utiliza para comprobar que una acción se haya completado de manera correcta.

4.1.4. Persistencia

La persistencia en aplicaciones es un concepto muy importante a tener en cuenta. Es común actualmente que cuando uno sale y entra de una aplicación, sus datos se quedan guardados en su dispositivo sin necesidad de volver a entrar. Además, si se está dentro de una aplicación y se pasa de una pantalla a otra, todas las aplicaciones móviles mantienen los datos de ese usuario entre pantallas, ya que sino la experiencia del usuario sería nefasta.

Esta aplicación también tiene persistencia, y esta se realiza a través de un patrón de diseño que se utiliza en la industria desde hace muchos años, y que se llama ***Singleton***.



```java
package com.example.legoapp;

11 usages
public class Singleton {

    3 usages
    private static Singleton instance;

    2 usages
    private String currentUser;
    8 usages
    public static Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }

    7 usages
    public String getCurrentUser() { return currentUser; }

    1 usage
    public void setCurrentUser(String currentUser) { this.currentUser = currentUser; }
}
```

Imagen 7. Patrón de diseño Singleton para persistencia de datos.

Con este patrón, se resuelven dos problemas importantes que pueden surgir en la aplicación:

- Garantiza una única instancia de la clase.
- Nos permite acceder a dicha clase desde cualquier parte del programa.

En el caso de los usuarios de la aplicación, se utiliza el Singleton para mantener sus datos durante todo el uso de la aplicación, y también vale para que el usuario no tenga que estar constantemente entrando en la app con sus credenciales, que se quedan guardadas en el dispositivo.

4.1.5. Empaquetado de la aplicación.

En el caso concreto de Android Studio, existe la opción de crear una APK desde el propio IDE. Una APK es un archivo ejecutable que contiene todos los datos del proyecto para poder realizar la instalación en cualquier dispositivo Android.

Para obtener la APK desde Android Studio, debemos realizar los siguientes pasos:

1. En la barra de herramientas superior, escogemos la opción de *Build.*
2. Dentro de las opciones disponibles, seleccionamos *Build Bundle(s) / APK(s).*
3. Por último, hacemos click en *Build APK(s).*

Esto realiza una exportación de todo el proyecto en formato zip, a través del cual podremos realizar la instalación.

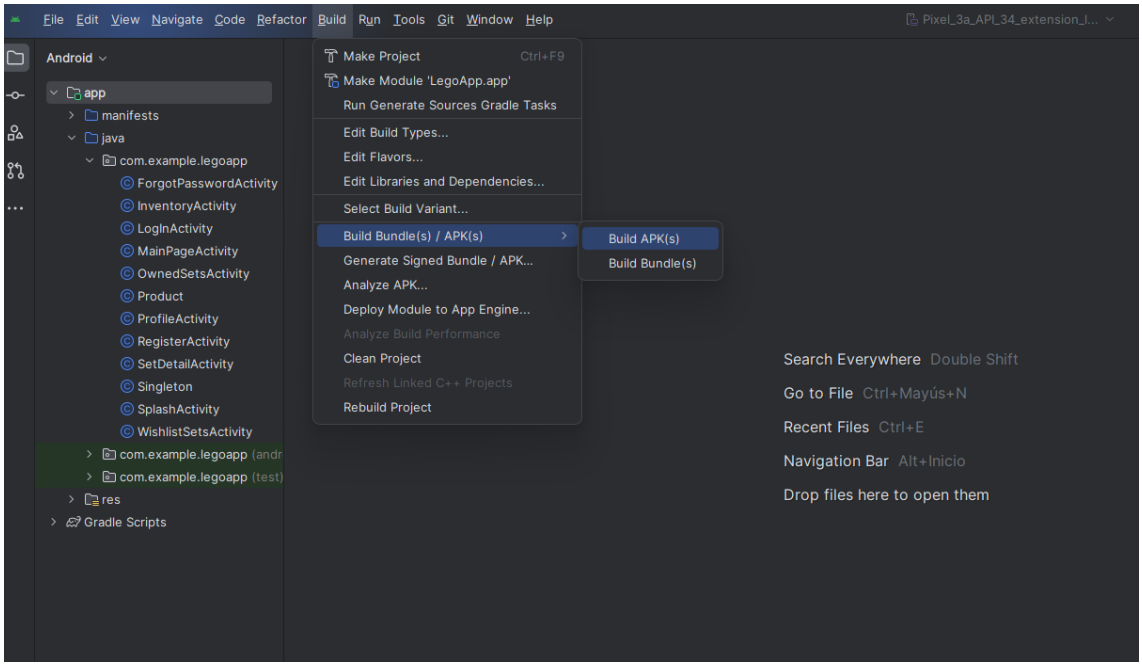


Imagen 8. Creación de una APK desde Android Studio.

4.2. Usabilidad, pautas de diseño y accesibilidad.

4.2.1. Usabilidad

En cuanto a la usabilidad de la aplicación, se han tenido en cuenta los estándares ISO para su desarrollo, destacando los siguientes:

- ISO / IEC 9126-1 (que hace referencia a la Calidad del Producto).
- ISO / IEC 9241 (referente a la Guía de Usabilidad).
- IEC TR 61997 (engloba las necesidades de una Guía de Interfaz Multimedia).

Estas normas tienen como objetico final brindar al usuario una experiencia satisfactoria, mejorando así la fidelización y aumentando el tiempo de uso de los clientes. Entre las características principales que debe cumplir la aplicación en cuanto a dichas normativas se encuentran:

- **Eficiencia:** Como es de esperar, una buena eficiencia de la aplicación mejorará de manera exponencial la experiencia de los usuarios mientras navegan por la misma.

Para el análisis de la eficiencia, Android Studio ofrece una herramienta realmente interesante, llamada ***Android Studio Profiler.***

Imagen 9. Android Profiler

Como se puede observar en la imagen inferior, esta funcionalidad se encarga de testear las aplicaciones durante su uso, para detectar principalmente:

- CPU consumida por la aplicación.
- Memoria utilizada durante su uso.

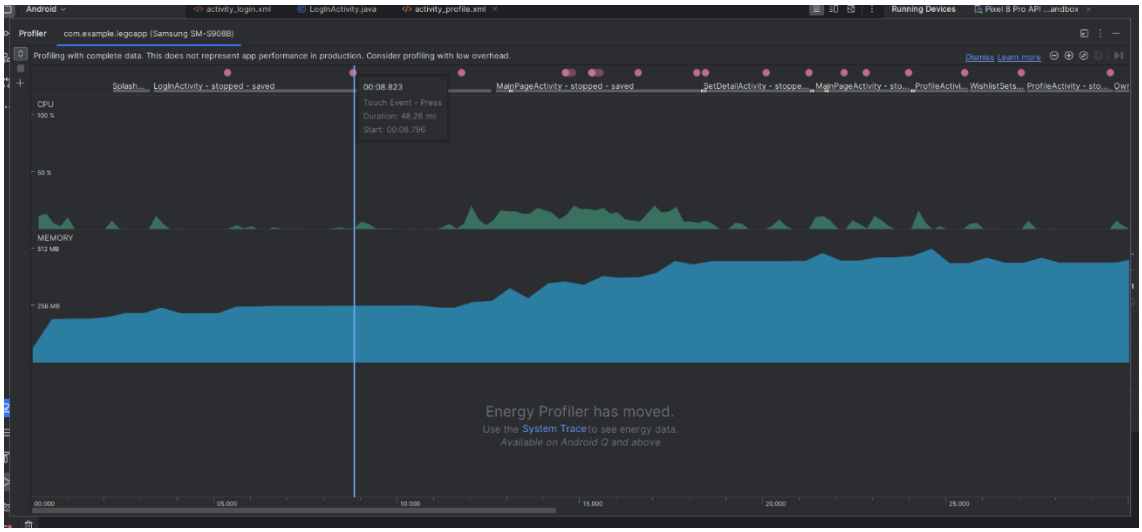


Imagen 10. Panel principal del Android Studio Profiler

Cada uno de los puntos rojos que se aprecian en la imagen corresponden a las interacciones del usuario con la aplicación, y debajo de cada punto aparece la pantalla o layout en el que se encuentra el usuario en cada momento, viendo así variaciones de la gráfica en función del contenido mostrado.

Vamos a analizar ambos parámetros en profundidad a través de las siguientes imágenes, en las que se ha entrado a cada uno de los apartados que se observan en el gráfico de la Imagen 10:

Análisis de la CPU:

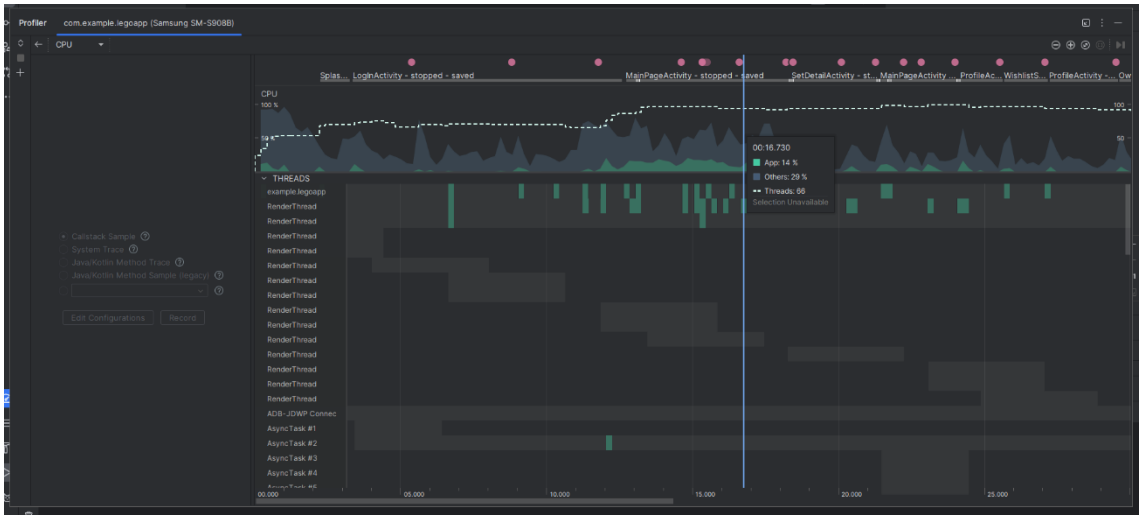


Imagen 11. Gráfico de la CPU del Android Studio Profiler.

En esta zona del profiler, podemos ver que, en la parte superior, y al igual que en el gráfico general, aparecen las interacciones del usuario con la aplicación. Si se sitúa el cursor encima de

cualquier punto, como se realizó en la imagen, se puede apreciar la distribución de la CPU entre la utilizada por la aplicación (en color verde) y por otros procesos (azul), además de los hilos que se están ejecutando en ese momento, representados a través de una línea blanca y discontinua.

Análisis de la memoria.

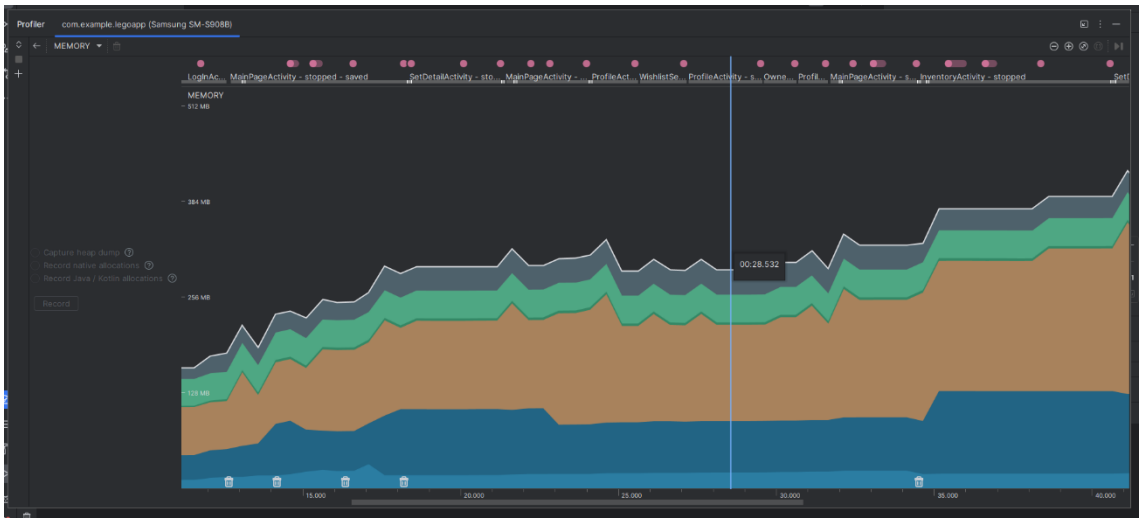


Imagen 12. Gráfico de la memoria del Android Studio Profiler.

En cuanto a la memoria consumida, también se aprecia que las cantidades no son disparatadas, ya que esta se encuentra entre los 400 y los 500 megabytes, que son cantidades muy razonables teniendo en cuenta la cantidad de pantallas y funcionalidades de la aplicación.

Tras este análisis, se puede observar que tanto en CPU y memoria, la aplicación consume pocos recursos del dispositivo, por lo que se puede afirmar **que está bien optimizada y su usabilidad es muy buena**. Esto se traduce en que la aplicación durante su uso es muy dinámica de usar, las transiciones entre pantallas son rápidas y los tiempos de carga son mínimos, **ofreciendo una experiencia de usuario óptima,** y sobre todo, **cumpliendo los estándares de las normas ISO en cuanto a eficiencia.**

4.2.2. Pautas de diseño.

Cuando se planteó la idea de la aplicación, se comenzó realizando un diseño gráfico de cómo se quería plasmar la interfaz. Se tuvieron en cuenta varios principios básicos, que son los que se suelen seguir a la hora de realizar el diseño de una aplicación:

- **Simplicidad**: Se intentó que la app fuese lo más minimalista posible, sin aplicar grandes excentricidades, y que el usuario tuviese como primera impresión una aplicación despejada, clara y atractiva a la vista.
- **Coherencia**: Además de vistosa, se quiso realizar una aplicación que fuese intuitiva, con iconos y menús descriptivos para facilitar al usuario su uso.
- **Especificidad**: Como en principio la aplicación está orientada a dispositivos móviles, el diseño debe ser consecuente con la plataforma en la que se vaya a utilizar, por lo que el planteamiento de diseño se basó en dispositivos móviles.
- **Paleta de colores**: Los colores utilizados son una parte vital en el planteamiento de cualquier interfaz gráfica, ya que estos deben ser acordes a la idea de la aplicación, y deben hacerla reconocible y enmarcarla en el contexto que se desea. Para esta

aplicación, se utilizó una gama de colores basado en la marca *LEGO,* que se puede observar en la imagen inferior.

Imagen 13. Paleta de colores RGB utilizada en la aplicación.

Con todos estos requisitos claros, se planteó el siguiente diseño inicial para la aplicación:

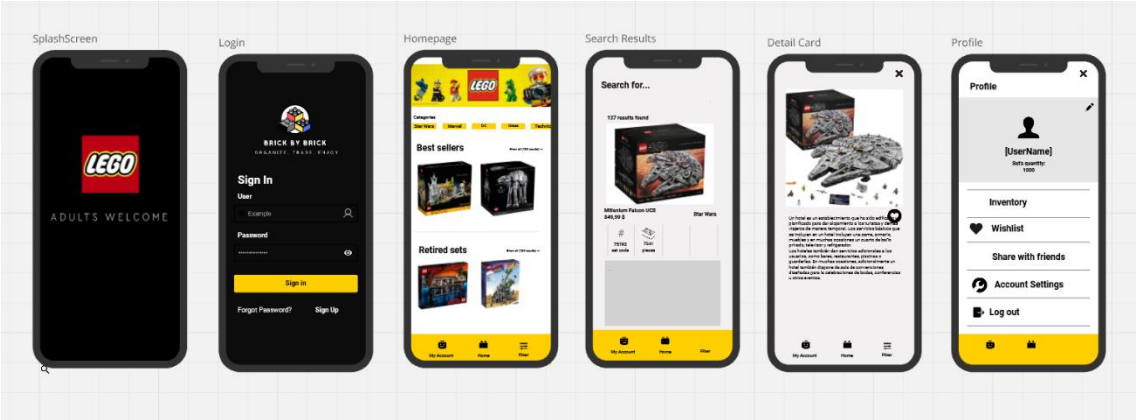


Imagen 14. Primer diseño de la aplicación.

Tras realizar un primer diseño, se decidió cambiar ciertos detalles para hacer la aplicación todavía más atractiva a la vista, suavizando los colores estridentes y cambiando el estilo de los menús.

A continuación, se desglosan las pautas de diseño relativas a las distintas partes de la aplicación final:

Para los menús.

Para este proyecto, se ha elegido realizar un estilo de menú único para el usuario, debido a su fácil comprensión y a su estilo limpio e intuitivo a través del uso de los iconos, lo que facilita la navegación y elección por parte del usuario mientras utiliza la app.

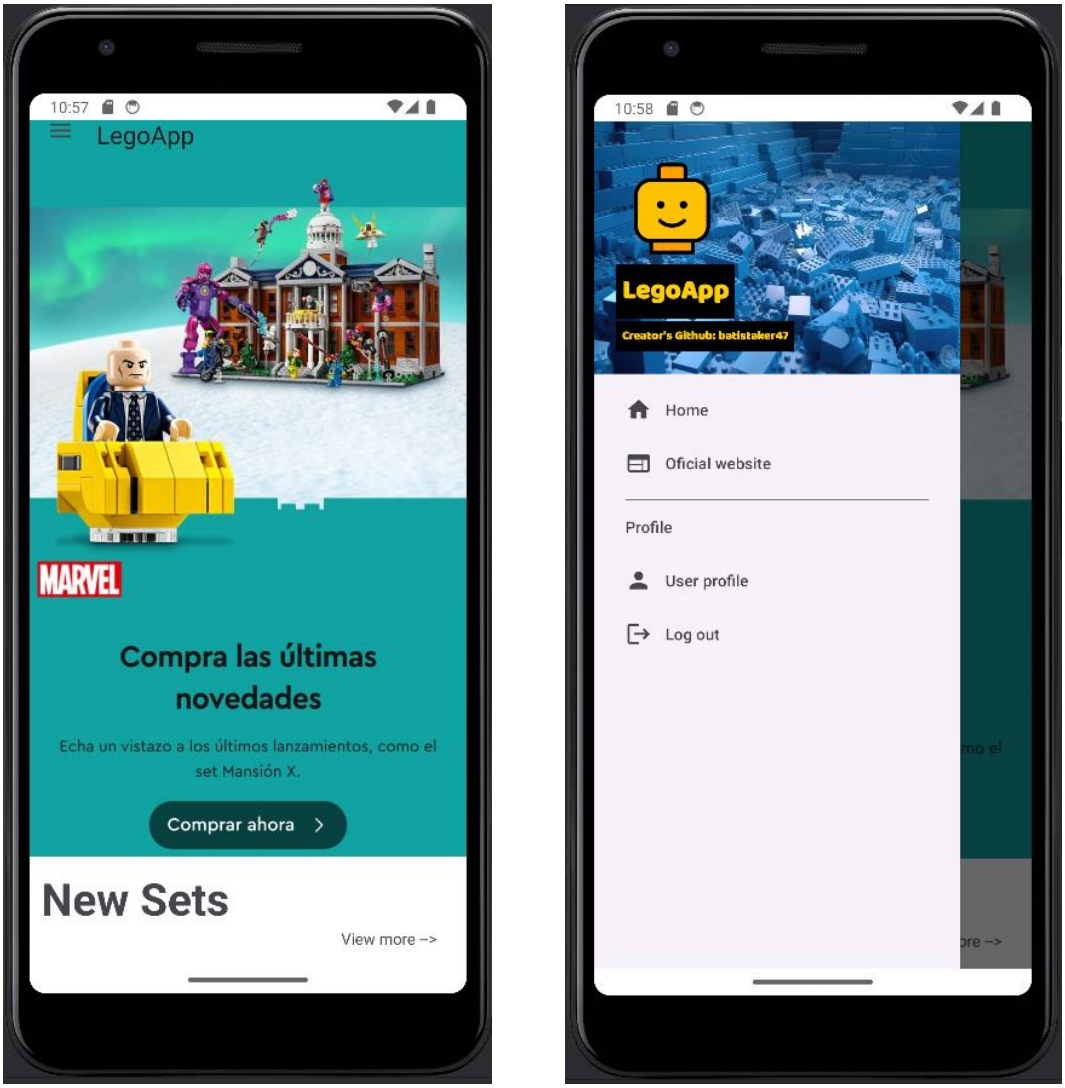


Imagen 15. Menú desplegable de la aplicación

Este menú se activa desde la parte superior izquierda de la imagen mostrada a la izquierda, a través del cual se despliega un menú lateral con distintas opciones para el usuario. Estas opciones van acompañadas de iconos atribuibles a las funciones que realizan.

Ventanas y cuadros de diálogo

Para las ventanas con las que el usuario interactúa con los productos, se ha elegido el formato de tarjeta con bordes redondeados y suaves, con una ligera sombra en la parte trasera. Esto da sensación de tarjeta flotante, lo que hace atractivo el diseño a la vez que distingue la tarjeta blanca del fondo blanco.

Al pulsar sobre la tarjeta, se abre otra actividad en la cual el usuario puede ver en detalle el producto. Esta vista también se realiza en formato tarjeta.

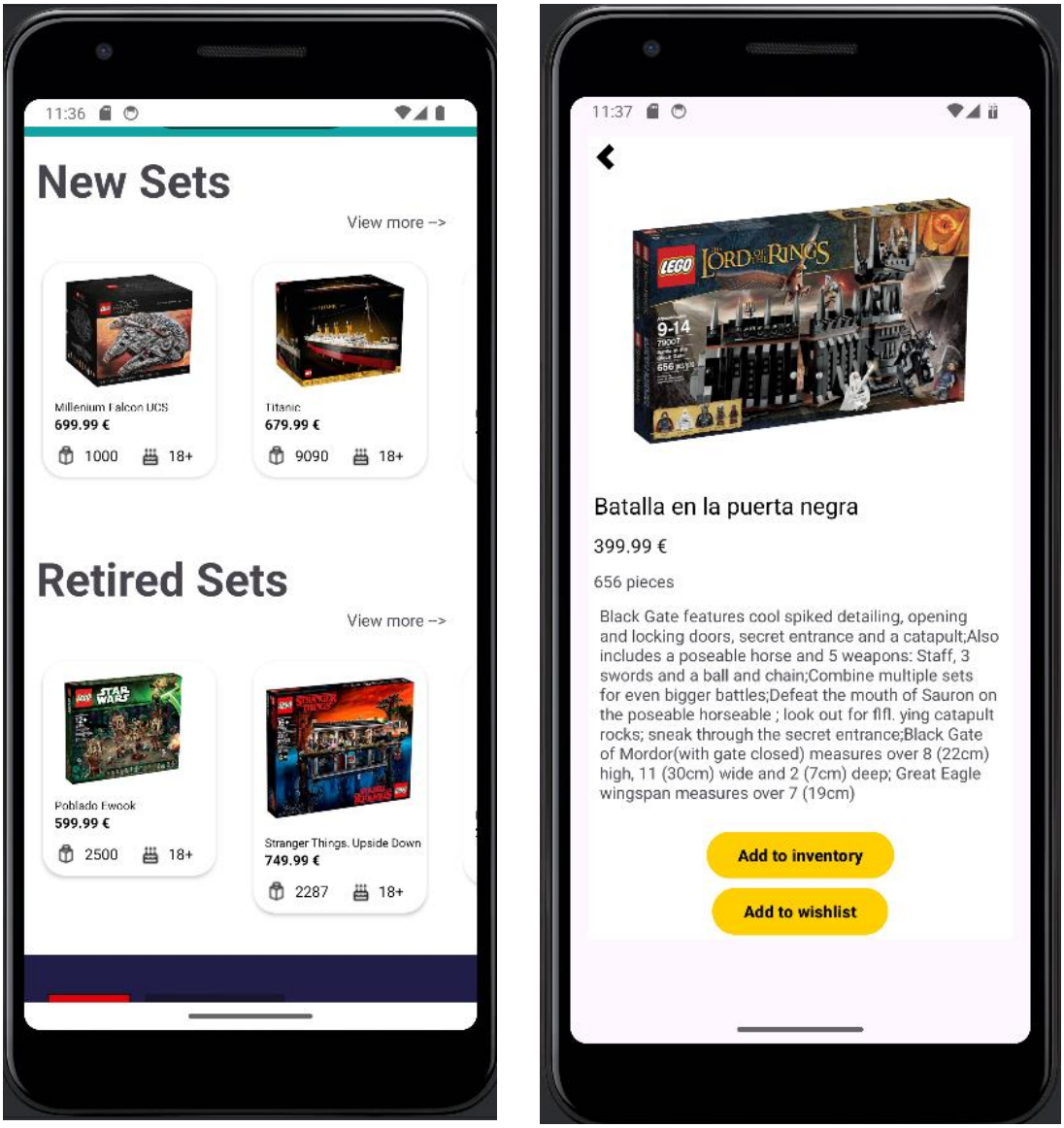


Imagen 16. Ventanas y cuadro de diálogo.

Aspecto

Aquí distinguimos entre 3 partes diferenciadas:

1) Iconos: Los iconos utilizados en la aplicación son muy representativos e intuitivos, para dar facilidades al usuario a la hora de interactuar con ellos.
2) Colores: Como se mostró en la *Imagen 13,* los colores aplicados en las distintas ventanas de la aplicación hacen referencia a la marca LEGO, pero se decidió eliminar de la paleta los colores más chillones para hacer la aplicación atractiva.
3) Fuente: La tipografía utilizada fue la fuente *Baloo*, que tiene una estética agradable a la vista gracias a sus bordes redondeados y su aspecto amigable.

En la siguiente imagen se puede comprobar un ejemplo que engloba estos tres apartados anteriores, pudiendo ver los iconos claros, los colores y la fuente citada.

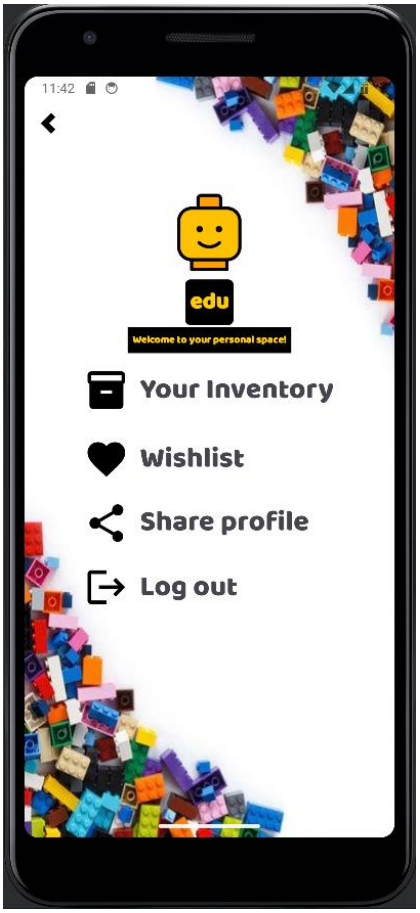


Imagen 17. Ejemplo de aspecto de la app, con colores, iconos y fuente

Elementos interactivos

La aplicación esta llena de elementos con los que poder interactuar, y todos son fácilmente distinguibles y comprensibles para los usuarios. Si nos fijamos en la *Imagen 16* y la *Imagen 17,* se pueden apreciar las distintas interacciones del usuario con la aplicación.

4.2.3. Accesibilidad.

Al encontrarse en fase beta, esta aplicación no está pensada para ser accesible para personas con discapacidad, pero en futuras versiones, y para hacer esta aplicación accesible e integradora con todo tipo de usuarios, se realizarían varias modificaciones.

Tras entrar por primera vez a la aplicación, en lugar de comenzar por la vista del registro de usuarios, la aplicación dirigiría a los usuarios a una pantalla como la que se aprecia en la *Imagen 18,* donde se preguntaría al usuario si tiene algún tipo de discapacidad, para así adaptar la aplicación a sus necesidades concretas.

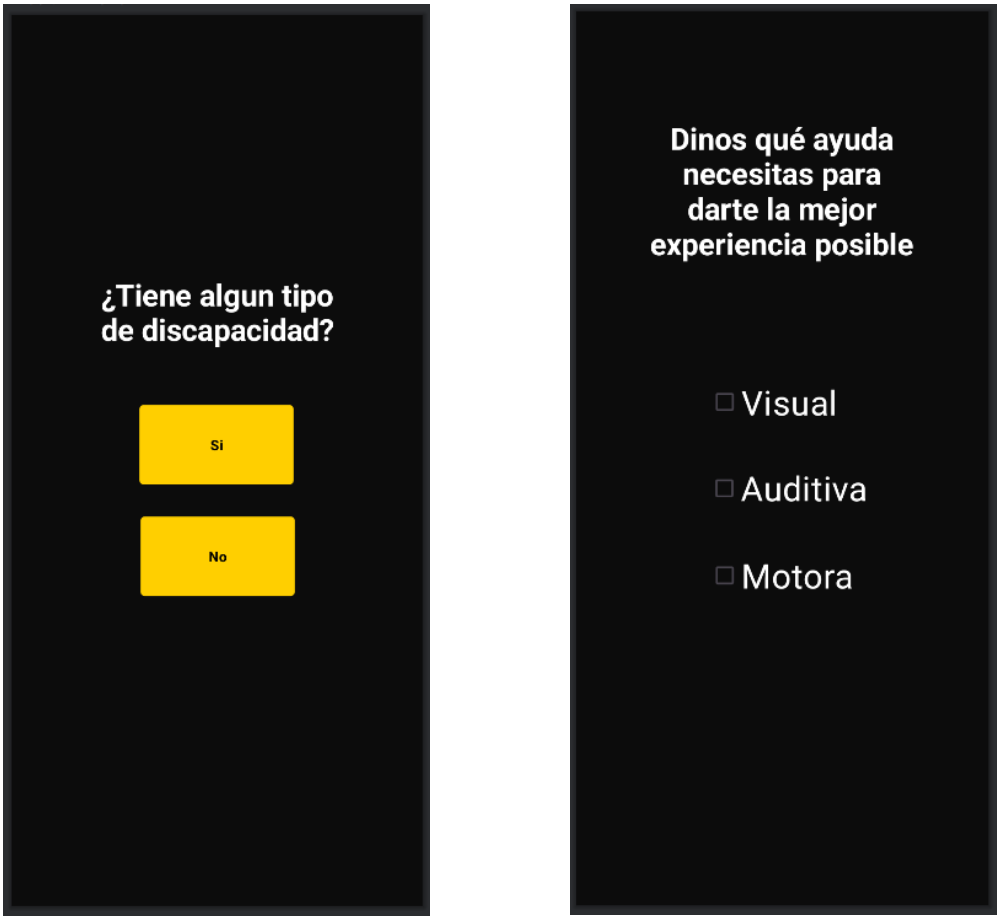


Imagen 18. Interfaz para personas con discapacidad

Una vez contestada la pregunta y marcada alguna o varias opciones, se procedería a recargar la aplicación y establecer estilos distintos a la interfaz en función de la discapacidad seleccionada:

Discapacidad visual.

Para este tipo de personas, se realizará una adaptación completa de la aplicación, aumentando el tamaño de todas las fuentes en gran medida, para que puedan tener más facilidad en la lectura de las distintas opciones que se ofrecen (*Imagen 19*).

Además, se suavizaría el tono de fondo de la aplicación, pasando de un tono blanco brillante a un blanco roto, que cansaría y forzaría menos la vista de los usuarios con estos problemas.

Además, se implementaría en código a la aplicación que si una persona selecciona dicha discapacidad, automáticamente se activaría la opción de Google llamada ***TalkBack,*** una herramienta integrada en los dispositivos Android que presta información auditiva para los elementos que se seleccionan en pantalla

Discapacidad auditiva.

Las personas con discapacidad auditiva no tendrían en principio ningún tipo de problema a la hora de utilizar la aplicación, pero si que se activaría la opción de que las notificaciones de nuevos sets o eventos aparezcan en pantalla como un cuadro de diálogo emergente para que estas personas no dependan del sonido del dispositivo para conocer dicha notificación.

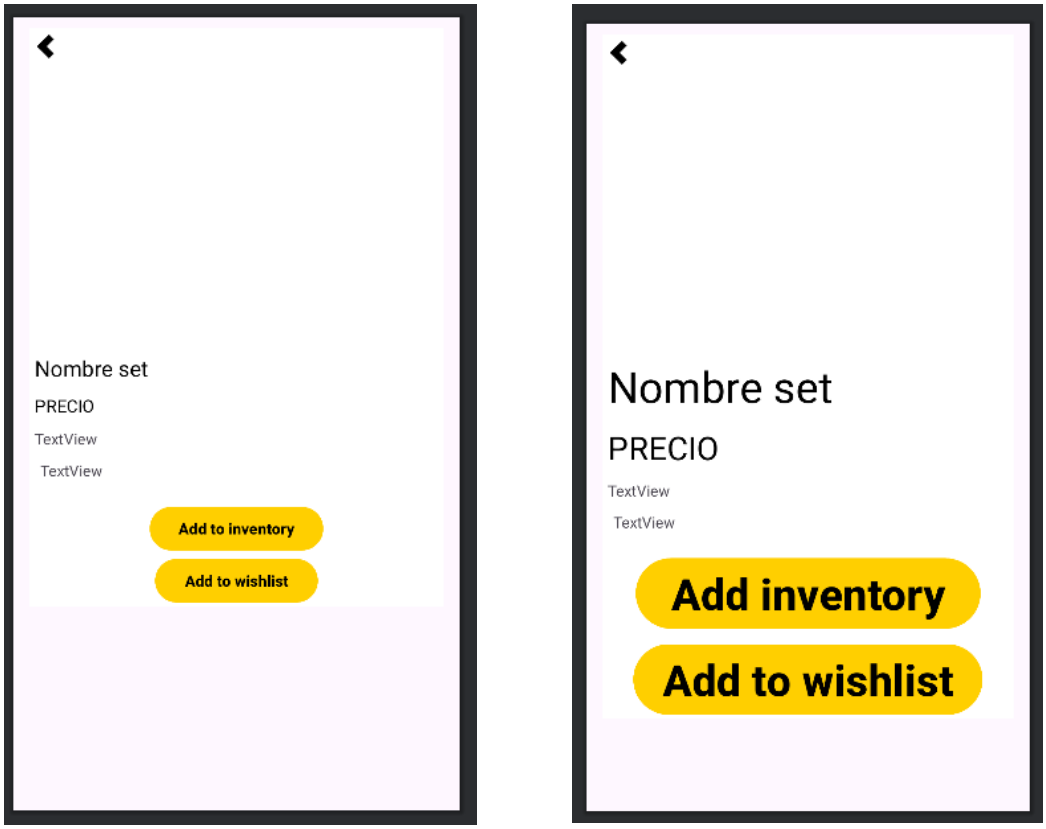


Imagen 19. Diferencias de tamaño de fuente para personas con discapacidad visual

4.3. Informes de aplicación.

Para asegurar un correcto funcionamiento de la aplicación antes de ser lanzada al mercado, se deben realizar informes acerca de la misma, en los que se recogerán las principales fallas, tanto de seguridad como de rotura o *crasheo* de la aplicación.

Estos informes obtenidos son de dos tipos:

- **Incrustados:** Estos informes se ejecutan a través del ***Logcat,*** y sólo se pueden consultar en tiempo de ejecución.

- **No incrustados:** Son informes generados con aplicaciones externas y que controlan el correcto funcionamiento del aplicativo.

4.3.1. Informes incrustados del logcat.

Durante el desarrollo de la aplicación, se fue depurando el código a través del compilador de Android Studio, para comprobar el correcto funcionamiento de las nuevas características de la aplicación. Una vez aparece un error en la consola durante la ejecución de la app, se debe localizar dónde se encuentra para poder solucionarlo.

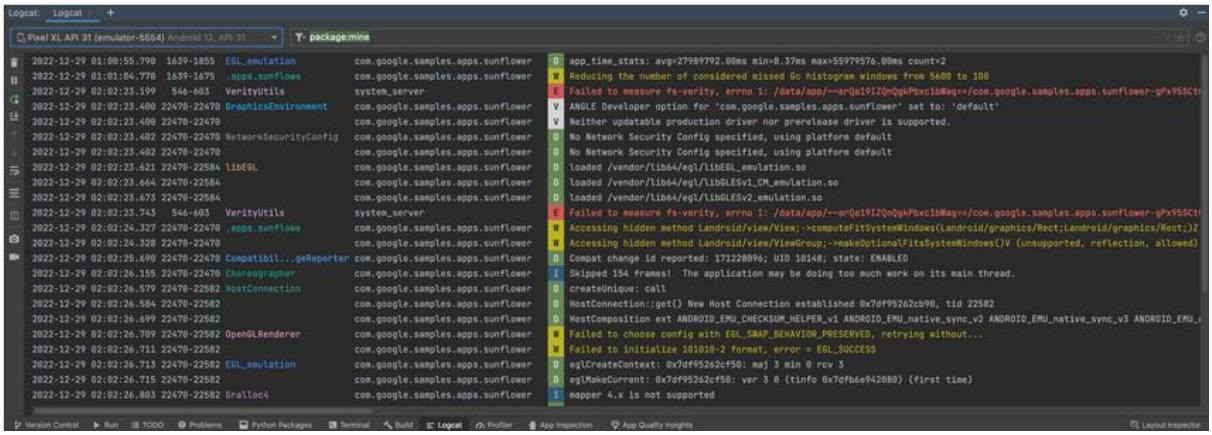


Imagen 20. Informes incrustados obtenidos del Logcat.

Entre los mensajes que pueden aparecer en estos informes encontramos muchos tipos, pero en lo que se refiere a errores, se pueden destacar dos tipos: advertencias (en color amarillo) y errores (en color rojo).

Advertencias del Logcat

Este tipo de mensajes aparecen cuando existe un error en la aplicación pero que no corta la ejecución de la misma. Por ejemplo, durante la ejecución de la aplicación en el emulador de Android Studio, la conexión con Google aparece como fallida en un mensaje de advertencia, debido a que el emulador no tiene acceso a ninguna cuenta de Google.

Este mensaje aparece en el Logcat, pero no impide que se siga utilizando el emulador, por lo que son errores menores a los que hay que prestar atención, pero no son fatales.

Errores del Logcat

Cuando aparece un mensaje de error de color rojo, la compilación se detiene y la aplicación sufre lo que se conoce comúnmente como un *crasheo*. Esto no permite seguir utilizando la aplicación, y obliga al desarrollador a solucionar el error para volver a arrancar la app.

Por ejemplo, si se tuviese una función asociada a un botón, y dicha función tiene algún error léxico en el código, al ejecutar la aplicación y pulsar dicho botón, la app crashearía, parando la compilación y avisando en la consola del error y la línea en la que se encuentra.

4.3.2. Informes de Firebase Crashlytics.

Cuando la aplicación ya tiene una versión funcional y está lista para ser lanzada al mercado o a fase de testing, los informes de herramientas externas son muy útiles para poder comprobar si la aplicación es capaz de soportar, por ejemplo, un gran volumen de usuarios simultáneos o si tiene algún error de eficiencia que empaña la experiencia del consumidor. En el caso de Brick by Brick, se ha utilizado ***Firebase Crashlytics,*** que es una excelente herramienta de control de la aplicación.

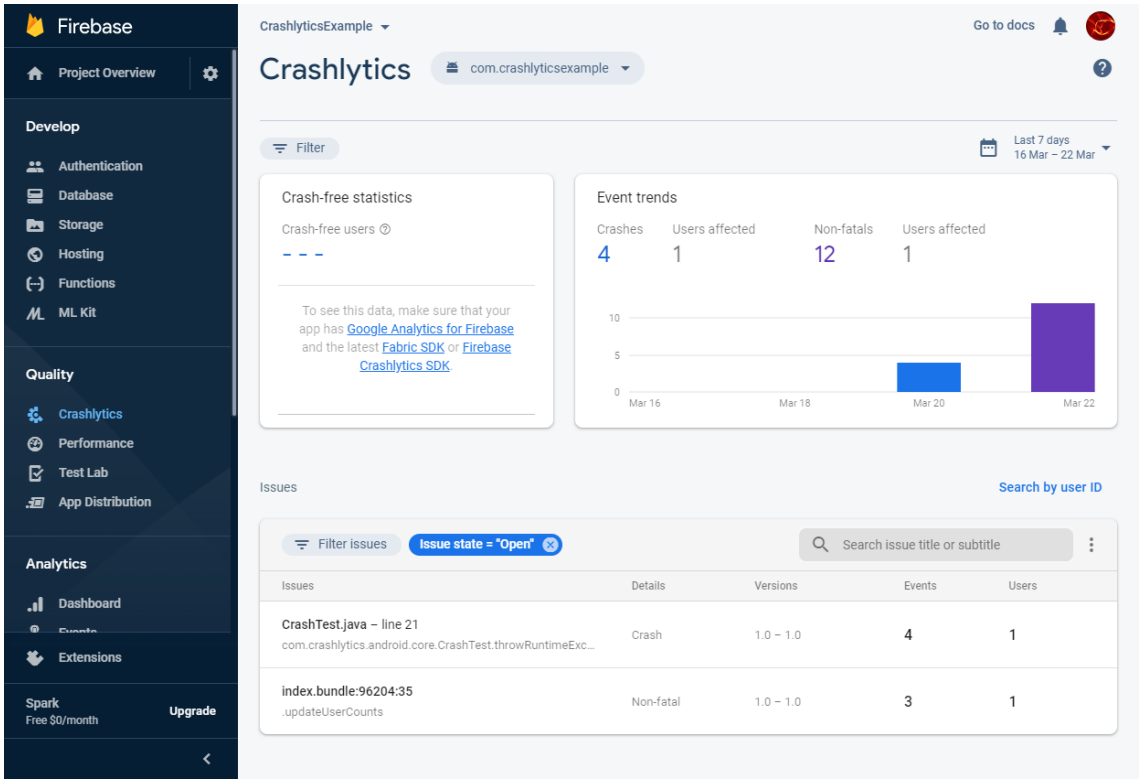


Imagen 21. Interfaz de Firebase Crashlytics

Al implementar en las dependencias de la aplicación esta característica, Firebase toma el control de la aplicación en cada registro que realiza el usuario en la misma, y estudia minuciosamente cómo es la experiencia del consumidor durante el uso de la app.

Como se observa en la *Imagen 21,* se obtiene en tiempo real y de manera gráfica cómo los usuarios interactúan con la aplicación, informando al desarrollador acerca de errores, usuarios afectados etc.

Además, la parte inferior de la interfaz ofrece una barra de búsqueda, en la que se pueden realizar filtrados para ver de manera concreta qué problemas existen durante el uso del aplicativo por parte de los usuarios, pudiendo realizar queries como la siguiente:

```
SELECT
  DISTINCT issue_id,
  COUNT(DISTINCT event_id) AS number_of_crashes,
  COUNT(DISTINCT installation_uuid) AS number_of_impacted_user,
  blame_frame.file,
  blame_frame.line
FROM
  `PROJECT_ID.firebase_crashlytics.PACKAGE_NAME_ANDROID`
WHERE
  event_timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(),INTERVAL 168 HOUR)
  AND event_timestamp < CURRENT_TIMESTAMP()
GROUP BY
  issue_id,
  blame_frame.file,
  blame_frame.line
ORDER BY
  number_of_crashes DESC
LIMIT 10;
```

Imagen 22. Ejemplo de filtro de Firebase Crashlytics para encontrar fallas en la aplicación

4.4. Documentación de la app.

Dentro de la documentación técnica de la aplicación, se distinguen dos ramas completamente distintas, pero fundamentales a la hora de entender el funcionamiento de la misma, tanto para el desarrollador como para el usuario que la va a utilizar a diario.

4.4.1. Documentación interna

Todo buen desarrollador debe implementar en su código **una buenas prácticas de desarrollo**, que comprenden tanto la legibilidad del código y la nomenclatura utilizada (camel case, snake case…), como la explicación de los métodos y clases utilizadas de manera clara y sencilla, para que así cualquier otro desarrollador que trabaje en la aplicación pueda seguir fácilmente el razonamiento que siguió el desarrollador anterior, y modificar el código o implementar nuevas líneas de manera rápida y segura.

En la aplicación de este trabajo se han comentado todas y cada una de las clases, distinguiendo entre dos tipos de comentarios: los simples, y los *JavaDoc.*

4.4.1.1. Comentarios simples

Se utilizan este tipo de comentarios con doble barra o " /*…*/ " para comentar fragmentos de código suelto que no hacen referencia a ningún método.

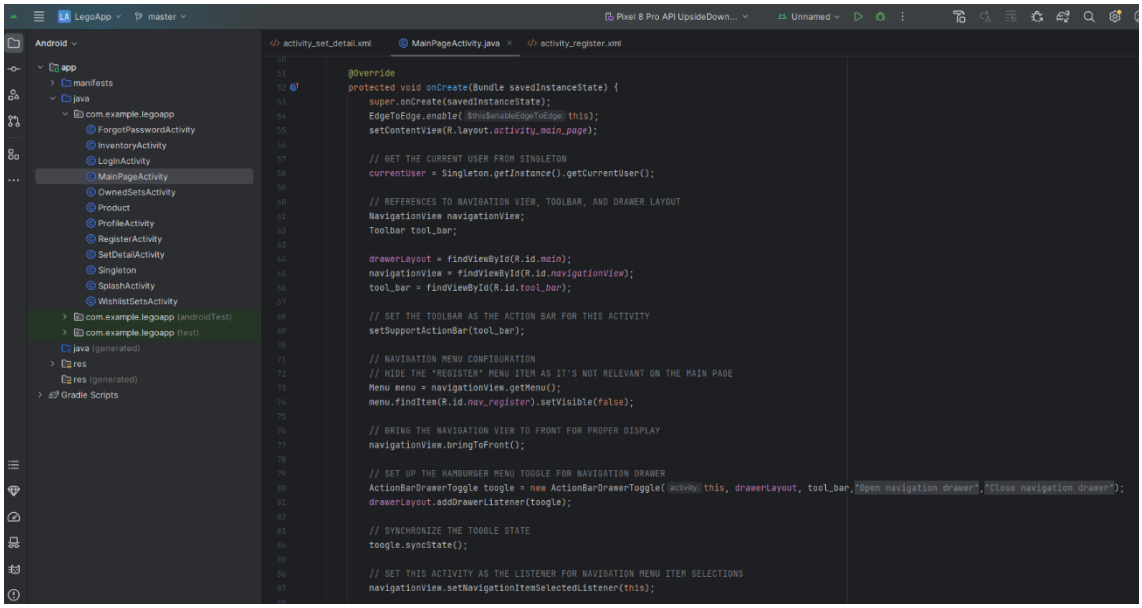


Imagen 23. Comentarios simples en una de las clases de la aplicación

Como se puede observar, dichos comentarios aparecen en un color gris apagado, y clarifican las líneas de código que se encuentran inmediatamente después de dicha aclaración.

4.4.1.2. Comentarios de JavaDoc.

A diferencia de los citados en el apartado anterior, los comentarios asociados a los métodos en Java se realizan a través de una librería interna de documentación llamada *JavaDoc.*

Esta librería se instancia a través de " /** ", y se sitúa de manera obligatoria encima de un método de la clase.

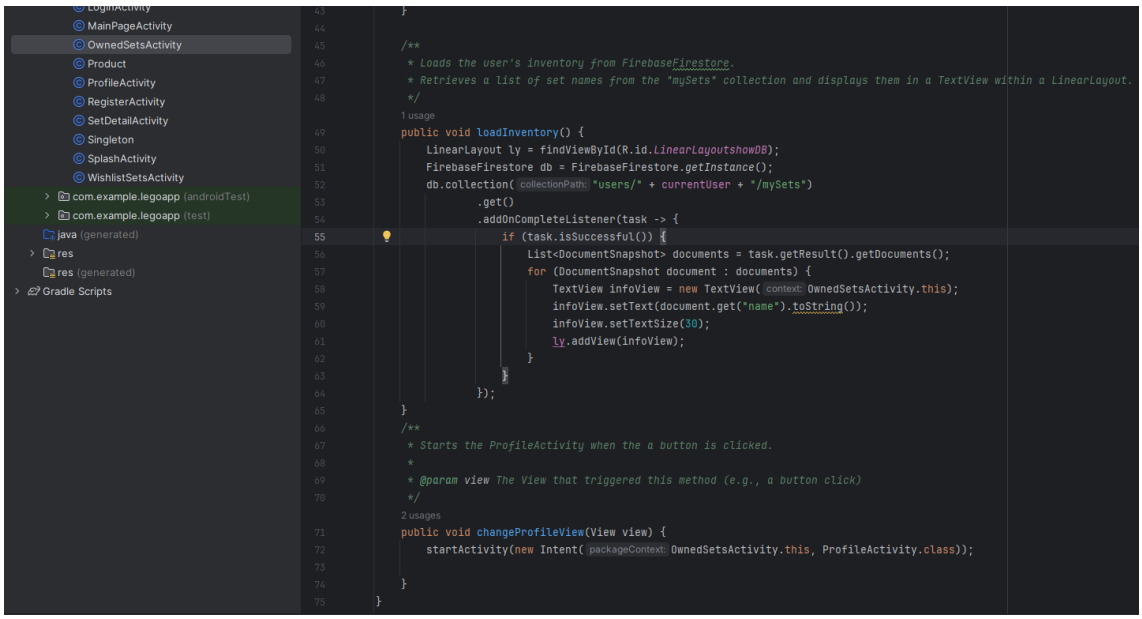


Imagen 24. Ejemplo de comentarios de JavaDoc

Esta librería detecta de manera automática si el método al que se ha aplicado recibe o no parámetros, incluyendo en su información la palabra *param,* en la que explicar qué parámetro recibe y por qué.

Es una librería extremadamente útil para cualquier desarrollador y que entra dentro de las buenas prácticas de programación comentadas al comienzo de este punto.

4.4.2. Documentación externa.

Este tipo de documentación se utiliza para dar soporte a los usuarios, y en el caso de esta aplicación se han realizado:

1) Un folleto comercial en el que se presenta la aplicación de manera simple y atractiva a potenciales clientes, con el objetivo de vender o comercializar el software. Este se adjunta como ***Anexo 1*** al final del documento.

2) Una guía de ayuda al usuario para que conozca de manera rápida y sencilla cómo utilizar correctamente este software. El manual se adjunta como documento aparte, ya que el formato no es compatible con el utilizado en este trabajo.

5. Anexos

5.1. Anexo 1. Folleto de comercialización de start-up