AutoColor

Aguilera, Virgilio Antonio;
Marín Sánchez, Juan Pablo;
Perez de Ciriza Morillo, Aritz;
Barguilla Pascual, Iván

Marzo 2024

1 Introducción

Autocolor es una aplicación móvil diseñada para simplificar la búsqueda y selección de colores de pintura de automóviles. La aplicación utiliza la cámara del teléfono para capturar imágenes de vehículos, escanea el color de la pintura y compara la información con una base de datos en un servidor. También permite la elección de colores por medio de una paleta de colores. El objetivo principal es asistir a los usuarios en encontrar la pintura que desean. Además, la aplicación proporciona información sobre talleres cercanos que pueden ofrecer servicios relacionados.

1.1 Objetivos del Proyecto:

- Facilitar a los usuarios la identificación y adquisición de colores de pintura de automóviles a través de la aplicación móvil.
- Optimizar el proceso de selección de colores, eliminando la necesidad de referencias manuales o muestras físicas.
- Proporcionar una interfaz intuitiva y amigable para mejorar la experiencia del usuario.
- Ofrecer información sobre talleres cercanos que pueden proporcionar servicios relacionados.

2 Tecnologías Utilizadas

1. **Kotlin:** La aplicación móvil está desarrollada en Kotlin, aprovechando las características modernas y expresivas para el desarrollo de aplicaciones Android.
2. **Java y Ngrok:** El servidor backend se implementa en Java, utilizando Ngrok para exponer localmente el servidor HTTP a través de Internet, facilitando la conexión con la aplicación móvil.
3. **API de Mapas de Google Cloud:** La integración de la API de mapas de Google Cloud se centra en proporcionar a los usuarios información sobre talleres cercanos que ofrecen servicios relacionados con la

aplicación. Esto mejora la experiencia del usuario al ofrecer opciones convenientes y relevantes.

4. **API Medium para Selección de Imágenes:** La aplicación utiliza la API Medium para la selección de múltiples imágenes desde la galería, brindando a los usuarios opciones adicionales al elegir entre varias imágenes almacenadas en el dispositivo.

5. **ColorPickerView de Skydoves:** La biblioteca ColorPickerView de Skydoves se incorpora para proporcionar un selector de colores eficiente y visualmente atractivo, facilitando la elección precisa del color de la pintura del automóvil.

6. **Gson:** Gson se utiliza para convertir las respuestas del servidor en archivos JSON, más eficientes a la hora de transmitir los datos, y volver a estructurarlos como objetos Kotlin, simplificando el manejo y procesamiento de los datos recibidos.

3 Estructura del Proyectos

Adapter

Esta carpeta contiene adaptadores que se utilizan para adaptar los datos de la aplicación a los Recycler Views.

- **ColorCocheAdapter.kt:** Adapta datos para los colores de los coches para la visualización en la interfaz del usuario.

- **ColorFavAdapter.kt:** Adapta datos para los colores favoritos para la visualización en la interfaz del usuario.

Database

Contiene archivos relacionados con la base de datos de la aplicación.

- **CochesRoomDatabase:** Define la base de datos de la aplicación utilizando la biblioteca de Room de Android.

- **httpClient:** Contiene la lógica de conexión y comunicación con el servidor HttpClient.

Model

Contiene clases que representan los objetos de datos de la aplicación.

DAO

- **ColorCocheDao:** Interfaz que define operaciones de acceso a datos para los colores de los coches.
- **ColorFavDao:** Interfaz que define operaciones de acceso a datos para los colores favoritos.

Entity

- **ColorCoche:** Clase de datos que representa los colores de los coches.
- **ColorFav:** Clase de datos que representa los colores favoritos.

Activities

Esta carpeta contiene las pantallas con las que interactúa el usuario.

- **ColorCocheDetail**
- **ColorMaps**
- **ColorStorage**
- **ConsultasActivity**
- **FavoritosActivity**
- **Filtrar**
- **MainActivity**
- **SplashActivity.KT**

4 Configuración del Entorno de Desarrollo

4.1 Herramientas necesarias

- **Android Studio:** Entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android.
- **Java:** Lenguaje de programación utilizado para el desarrollo de la aplicación. Se utiliza IntelliJ IDEA de JetBrains como el IDE.
- **Ngrok:** Herramienta que proporciona una URL pública para el servidor local, facilitando la exposición del servidor a través de Internet.
- **Plugin:** `com.google.android.libraries.mapsplatform.secrets-gradle-plugin`: Maneja de forma segura las claves de API de Google Maps en archivos Gradle.
- **Plugin:** `kotlin("kapt")`: Habilita la generación de código en tiempo de compilación para Kotlin.
- **Dependencia:** `com.github.skydoves:colorpickerview:2.3.0`: Proporciona un selector de colores personalizable.
- **Dependencia:** `com.google.code.gson:gson:2.10.1`: Biblioteca para la serialización y deserialización de objetos JSON.

Se deberan editar los archivos de build gradel de proyecto (Vease figura 1) y app (Vease figura 2).

La configuración del gradel de proyecto (1) debera asemejarse a la figura 3 y la configuración del gradel de app (2) debera asemejarse a la figura 4.

Ademas, en las propiedades locales (Vease figuras 5 y 6) se debera implementar la siguiente key:

MAPS_API_KEY=AIzaSyC_kDGoPt_FEL39WX0AqsojBKh8Hyte0dw

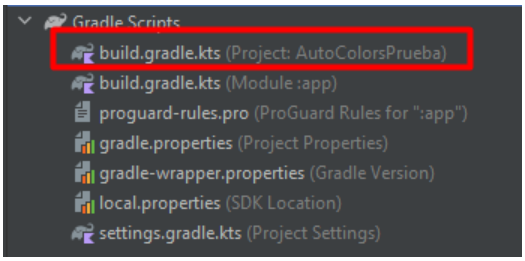

Figure 1: Gradel de proyecto.

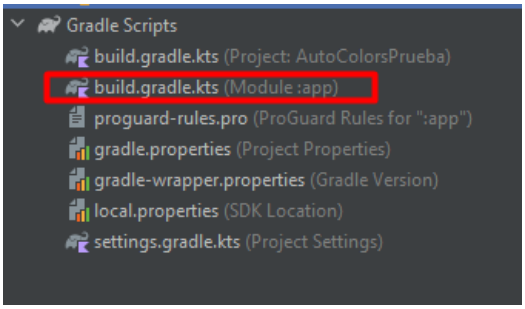

Figure 2: Gradel de la app.

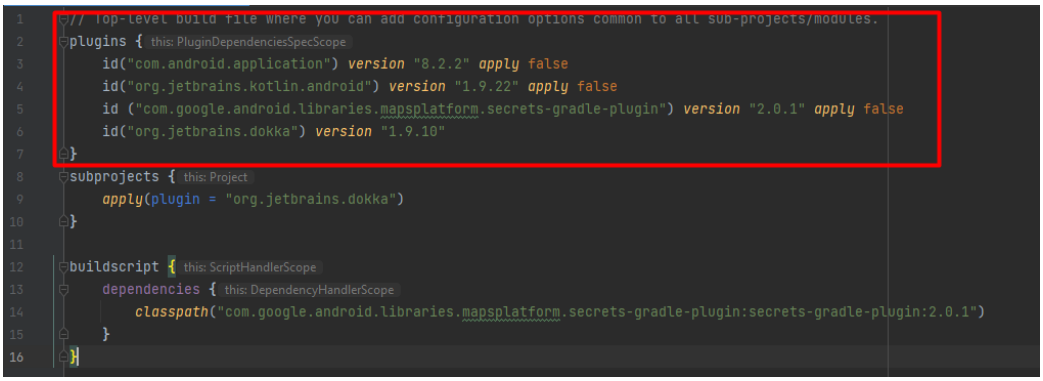

Figure 3: Codigo de gradel proyecto.

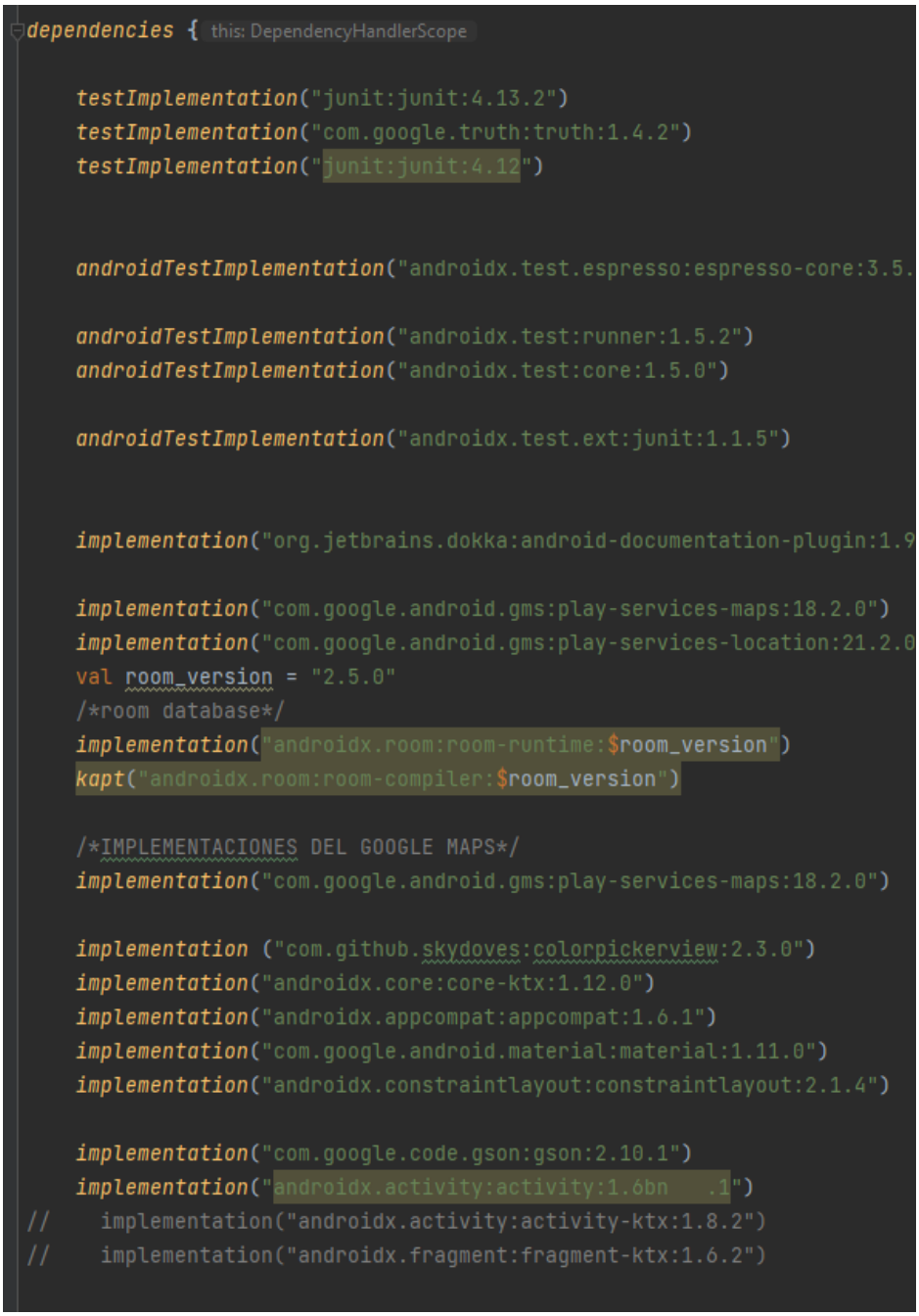

Figure 4: Codigo de gradel app.

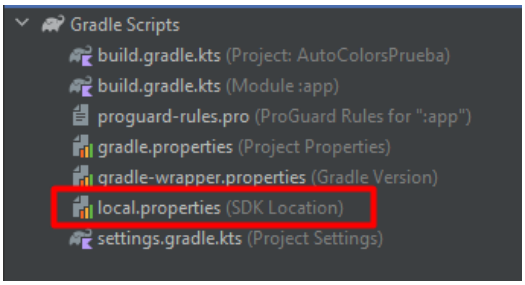

Figure 5: Propiedades locales.

```
## This file must *NOT* be checked into Version Control Systems,
# as it contains information specific to your local configuration.
#
# Location of the SDK. This is only used by Gradle.
# For customization when using a Version Control System, please read the
# header note.
#Fri Mar 15 04:23:53 CET 2024
MAPS_API_KEY=AIzaSyC_kDGoPt_FEL39WX0AqsojBKh8Hyte0dw
sdk.dir=C\:\\Users\\ivanb\\AppData\\Local\\Android\\Sdk
```

Figure 6: Propiedades locales.

5 Guía de Uso

Autocolors ofrece una serie de ventanas interactivas diseñadas para ser intuitivas y de fácil comprensión para los usuarios. A continuación, se proporciona información detallada sobre cada una de ellas, explicando su propósito y funciones.

5.1 Pantalla Principal - Selector

La pantalla principal de Autocolors presenta una interfaz atractiva y fácil de interpretar. Al iniciar la aplicación, el usuario se encuentra con una rueda de colores que le permite seleccionar entre una amplia gama de colores hexadecimales. (Figura 7) Además, cuenta con una barra deslizante que permite ajustar la intensidad de los colores, permitiendo al usuario oscurecerlos según su preferencia.

La pantalla también ofrece dos botones adicionales: uno que permite al usuario abrir su galería de fotos para seleccionar una imagen y extraer colores de ella, y otro que le permite tomar una fotografía en tiempo real para extraer colores directamente de la imagen capturada.

Las letras en hexadecimal hacen referencia al color seleccionado en la rueda de colores o al tono ajustado con la barra deslizante.

Además, hay un cuadro que cambia de color junto con la barra de herramientas y la barra de navegación de la página. Al mantenerlo presionado, se despliega un menú que ofrece tres opciones: Agregar a favoritos, Copiar Hexadecimal y Comparar. (Figura 8)

Si el usuario elige 'Agregar a favoritos', puede hacer clic en el icono de corazón en la página principal para agregar el color seleccionado a su lista de colores favoritos.

Seleccionando 'Comparar', se mostrará una lista de colores de coches de diversas marcas y modelos que tienen una gran similitud con el color elegido por el usuario. (Figura 9)

Al seleccionar 'Copiar Hexadecimal', se copiará el código hexadecimal del color seleccionado en el portapapeles. Este código puede ser pegado en el campo de filtrado para encontrar colores similares de coches de una marca específica.

Al hacer clic en un color en la lista, se abrirá una nueva pantalla con información detallada sobre ese color. (Figura 10) Desde esta pantalla, el usuario también puede agregar el color a su lista de favoritos.

5.2 Gestión de Favoritos

El usuario tiene la opción de gestionar los colores favoritos. (Figura 11) Desde la pantalla de favoritos, el usuario puede eliminar los colores que ya no desea conservar en su lista. (Figura 12)

5.3 Localización de Talleres

Finalmente, el usuario tiene acceso a una tercera ventana donde se abrirá un mapa. Al pulsar sobre el círculo gris en el mapa, será llevado directamente a su ubicación, y alrededor de él aparecerán las ubicaciones de los talleres más cercanos donde podrá pintar su coche. (Figura 13)

Figure 7: Pantalla principal.

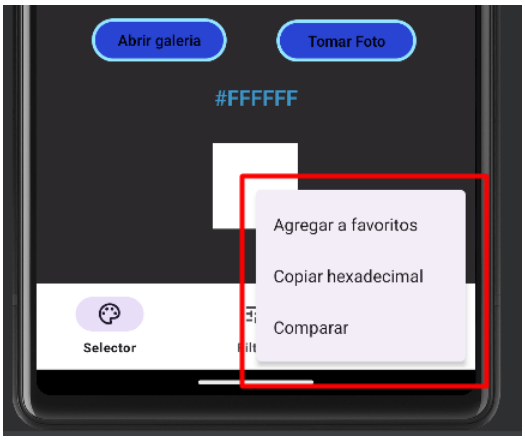

Figure 8: Menu desplegable.

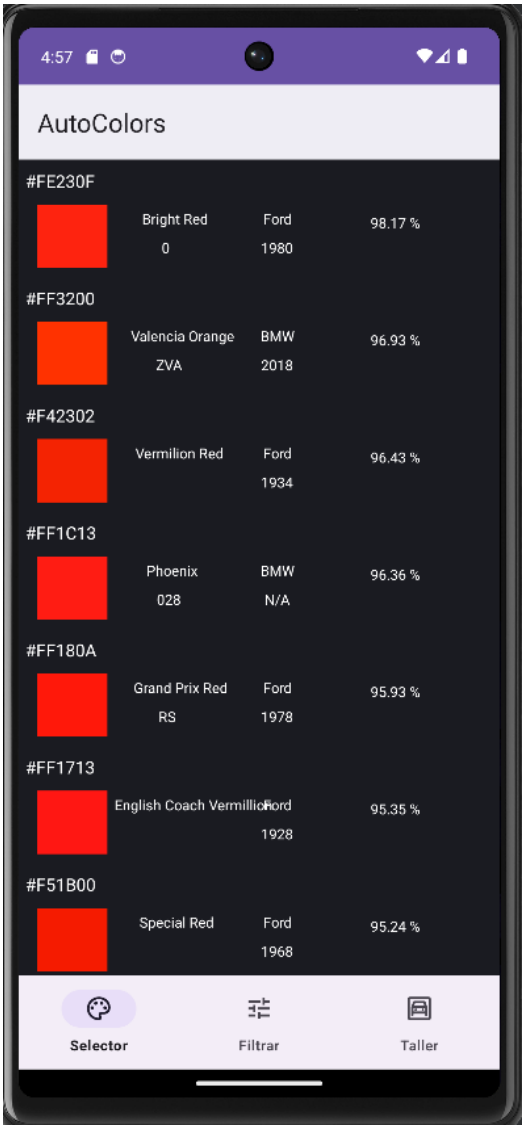

Figure 9: Lista de colores similares.

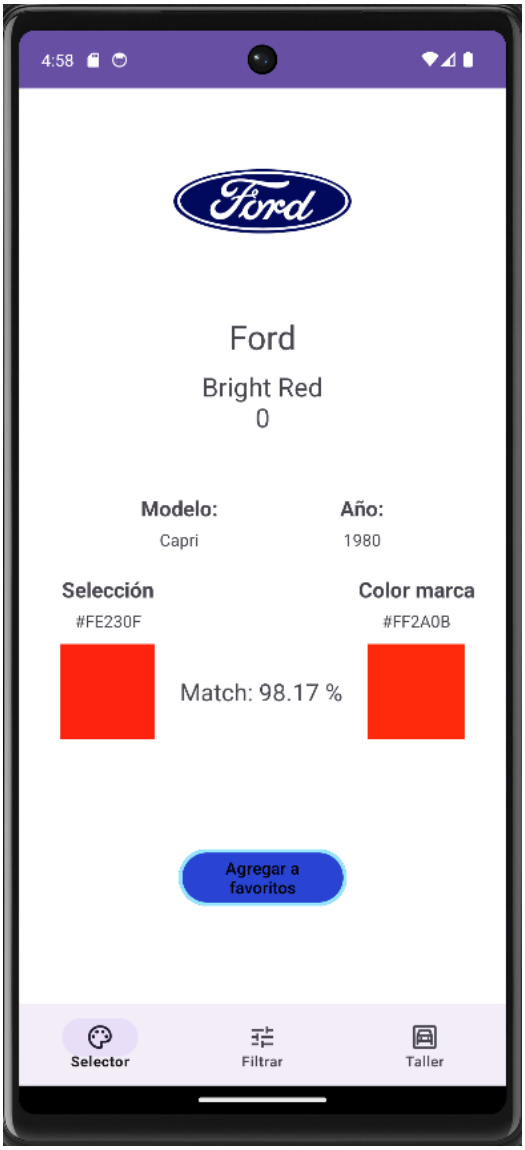

Figure 10: Información color.

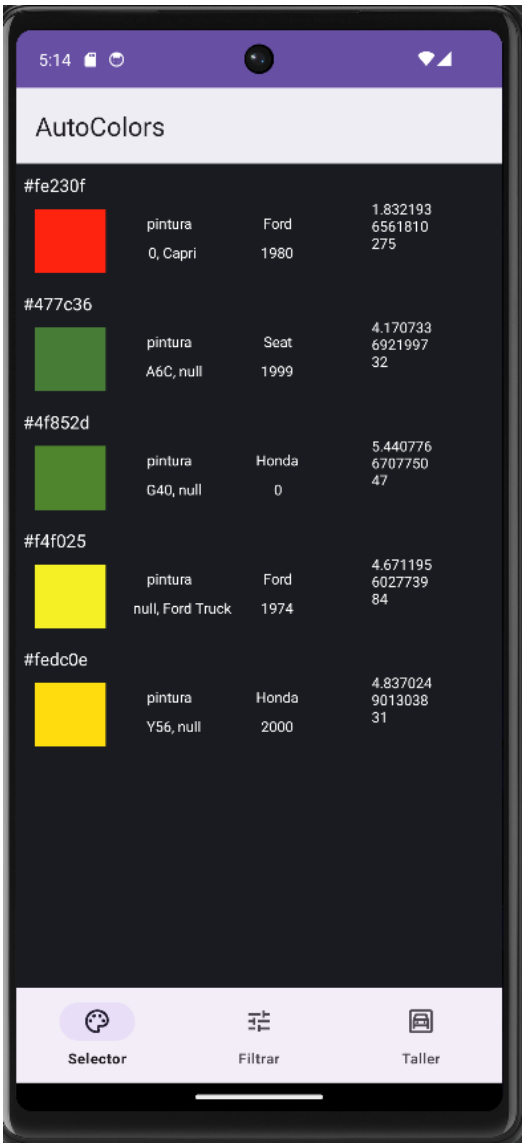

Figure 11: Lista de favoritos.

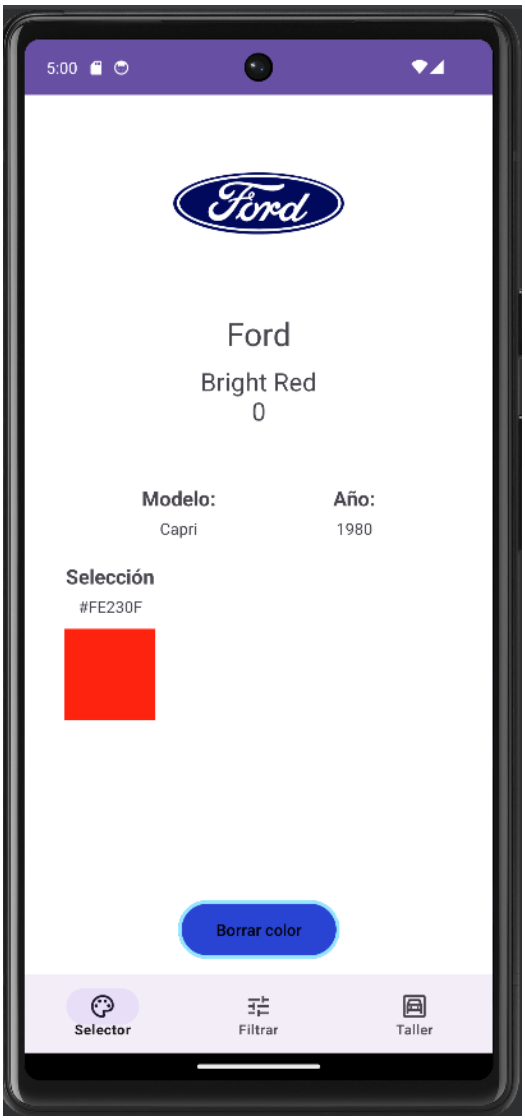

Figure 12: Información color favorito.

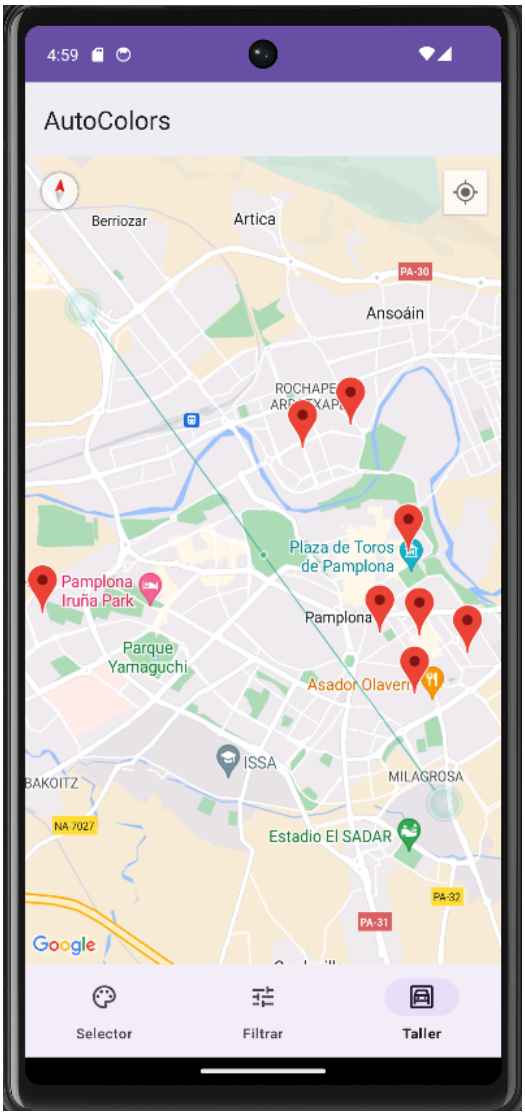

Figure 13: Mapa de talleres cercanos.

6 Arquitectura

El proyecto implementa la arquitectura MVVM (Modelo-Vista-ViewModel). En esta arquitectura, el servidor y la base de datos ROOM constituyen el Modelo, mientras que cada archivo Activity.kt forma el ViewModel. Los ViewModel preparan los datos para ser visualizados, mientras que los layouts de las pantallas forman la Vista, que muestra los resultados/datos al usuario y permite su interacción con el resto del programa.

6.1 Diagrama de Conexión con el Servidor

A continuación se muestra un diagrama que ilustra la conexión entre la aplicación móvil y el servidor:

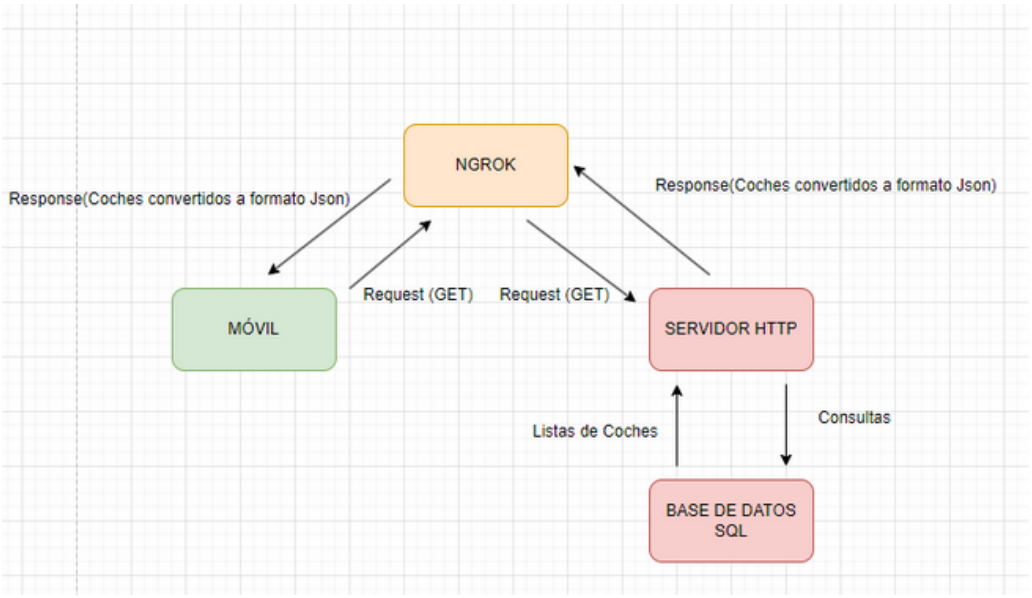

Figure 14: Diagrama de Conexión con el Servidor

7 Documentación de Código

7.1 Convenciones de Nomenclatura

En nuestro proyecto, seguimos rigurosamente las convenciones de nomenclatura para garantizar la claridad y la coherencia en todo el código. Estas convenciones facilitan la lectura, comprensión y mantenimiento del código, tanto para nosotros como para cualquier otro desarrollador que se una al proyecto.

7.1.1 CamelCase y snake_case

Utilizamos tanto CamelCase como snake_case para nombrar nuestras variables, métodos, clases y constantes. Aquí algunos ejemplos:

- Variables: hexadecimal, serverUrl, httpClient.
- Métodos: onCreate(), abrirGaleria(), copyToClipboard().
- Clases: ColorStorage, CochesRoomDatabase.
- Constantes: REQUEST_CODE_GALLERY.

7.1.2 Nombres Descriptivos

Preferimos nombres descriptivos que reflejen claramente la función o el propósito de la entidad que están representando. Esto mejora la legibilidad y comprensión del código. Algunos ejemplos incluyen:

- Variables: colorPickerView, brightnessSlideBar, alphaTileView.
- Métodos: onCreate(), abrirGaleria(), copyToClipboard().
- Clases: HttpClient, ColorStorage.

7.2 Comentarios de Código

Los comentarios de código son una parte esencial de nuestra práctica de desarrollo, ya que proporcionan explicaciones adicionales sobre el funcionamiento y el propósito de diversas partes del código. Esto es especialmente útil para comprender la lógica detrás de implementaciones específicas o para documentar decisiones importantes de diseño.

7.2.1 Ejemplo de Uso de Comentarios

A lo largo de nuestro código, utilizamos comentarios para proporcionar información adicional sobre las clases, métodos y secciones de código relevantes. Aquí hay un ejemplo de cómo se pueden utilizar los comentarios en nuestro código:

```
/**
 * Esta clase sirve para poder escoger un color de la rueda, de una foto de la galería o de nuestra cámara.
 * Se puede mover el selector por toda la rueda y de la barra para crear nuestro color.
 * Implementa la interfaz HttpClientListener
 * También compara el color escogido con los de la base de datos, haciendo una consulta en esta
 */

 Juan Pablo Marin +4
class MainActivity : AppCompatActivity(), HttpClient.HttpClientListener{
    //Navegación
    lateinit var bottomNavigationView: BottomNavigationView

    //ColorPickerView
    private lateinit var colorPickerView: ColorPickerView
    private lateinit var brightnessSlideBar: BrightnessSlideBar
    private lateinit var alphaTileView: AlphaTileView
    private lateinit var textView: TextView


    // private lateinit var appToolbar: Toolbar
    private lateinit var itemCopiar: MenuItem
    private lateinit var itemAgregar: MenuItem
    private lateinit var itemFav: MenuItem
    private lateinit var itemComparar: MenuItem


    //Toolbar
    private lateinit var toolbar: Toolbar

    private var hexadecimal: String = ""

    //Conexión con el servidor
    private lateinit var httpClient: HttpClient
    private lateinit var serverUrl: String
    private lateinit var params: Map<String,String>
```

Figure 15: Comentarios del proyecto

Ademas como en todo proyecto la documentación detallada del código se encuentra dentro del propio proyecto, incluyendo comentarios en el código fuente y cualquier documentación adicional proporcionada en archivos README o documentos de diseño.

8 Pruebas

8.1 Tipos de Pruebas Realizadas

Se llevan a cabo pruebas de diferentes tipos para garantizar la calidad y el correcto funcionamiento de la aplicación:

- Pruebas Unitarias: Se realizan para testear aspectos específicos del código a nivel de unidades individuales. En nuestro caso, las pruebas unitarias se centran en:
  - La conexión al servidor para verificar que se envíen y reciban datos correctamente.
  - El filtrado y la validación de inputs de usuarios en la clase Filtrar.
  - La correcta funcionalidad de la obtención de localizaciones desde Maps.
- Pruebas de Integración: Estas pruebas se enfocan en verificar que los distintos componentes de la aplicación funcionen correctamente juntos. Algunos ejemplos incluyen:
  - Verificar que la clase ColorCocheDetail muestra los campos correctamente.
  - Asegurarse de que el RecyclerView se inicia correctamente y admite elementos.
  - Comprobar que MainActivity carga correctamente y que su navegación entre pantallas funciona correctamente.

8.2 Cómo Ejecutar las Pruebas

Las pruebas están definidas en varias clases en los paquetes `com.example.atucolorsprueba (Test)` y `com.example.atucolorsprueba (Android Test)`. (Figura 16)
Para ejecutarlas, sigue los siguientes pasos:

1. Ir a la clase que contiene las pruebas.
2. Hacer clic sobre el triángulo verde de cada método para ejecutarlo.
3. O hacer clic en el triángulo verde de la clase para ejecutar todas las pruebas de esa clase.

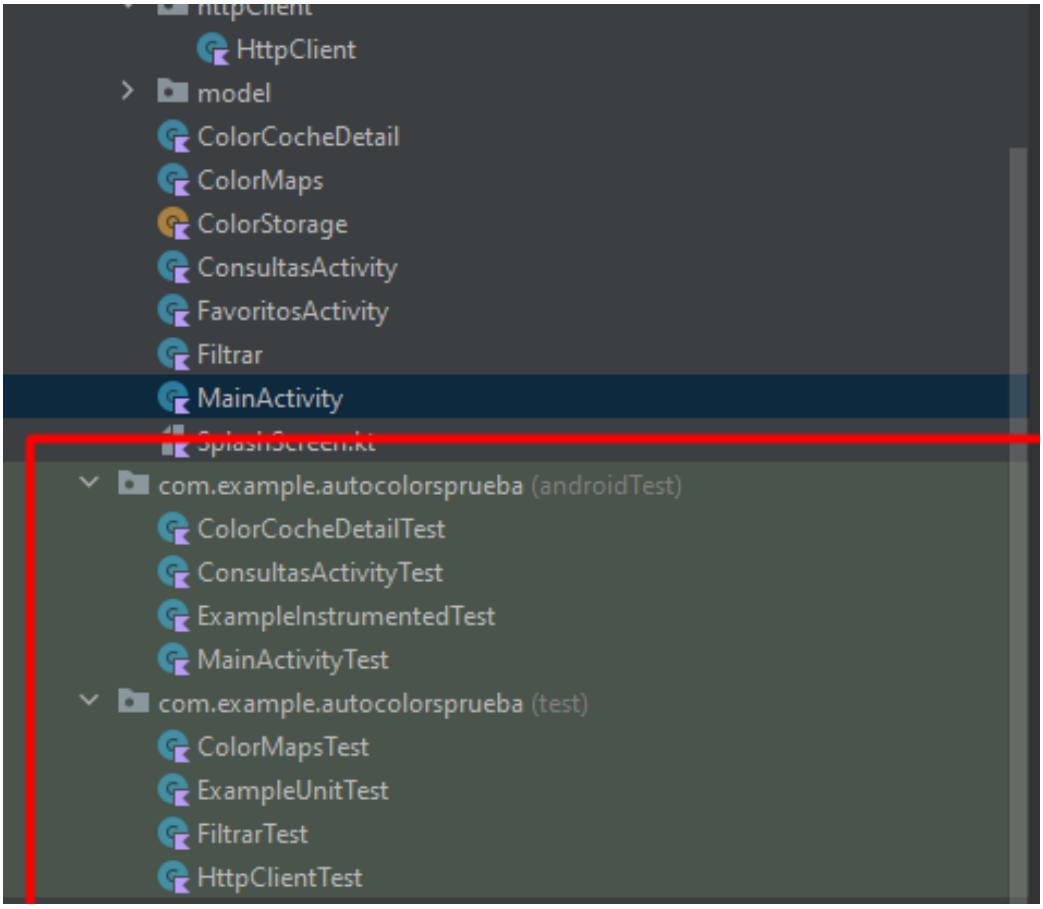

Figure 16: Direcetorio de las pruebas

9 Publicación y Distribución

9.1 Instrucciones para Compilar y Generar Versiones de la Aplicación

Para compilar y generar versiones de la aplicación, sigue estos pasos:

1. Abre el proyecto en Android Studio y asegúrate de que funciona correctamente.
2. En la pestaña superior, haz clic en **Build** y busca **Generate Signed Bundle / APK**.
3. Elige la opción **APK** y continúa.
4. Selecciona la opción para crear una nueva llave de autenticación o elige una existente.
5. Define una contraseña para la llave y elige la carpeta donde se guardará.
6. Finalmente, descarga el archivo APK generado en el dispositivo móvil donde se desea instalar la aplicación. Una vez descargado, instálalo y la aplicación estará lista para su uso.

10 Mantenimiento y Contribuciones

10.1 Políticas de Ramificación y Fusión

En nuestro proceso de desarrollo, seguimos las siguientes políticas:

- Los cambios se realizan en ramas separadas con nombres descriptivos.
- Las ramas principales están protegidas y los cambios se realizan a través de solicitudes de extracción.
- Cada solicitud de extracción es revisada por otro miembro del equipo antes de fusionarse.
- Se ejecutan pruebas automatizadas para garantizar la calidad del código.
- La fusión de ramas se realiza con rebase o merge, siguiendo las convenciones del equipo.

10.2 Proceso de Reporte y Gestión de Problemas

El proceso de reporte y gestión de problemas se lleva a cabo de la siguiente manera:

- Los problemas se reportan con detalles en el sistema del proyecto.
- Un miembro del equipo asigna y prioriza el problema.
- La solución se desarrolla en una rama separada, vinculada al problema.
- Después de las pruebas, se envía una solicitud de extracción.
- Tras la aprobación, se fusiona en la rama principal.

10.3 Cómo Contribuir al Proyecto

Los contribuyentes pueden contribuir al proyecto de la siguiente manera:

- Hacer un fork del repositorio en GitHub.
- Trabajar en ramas separadas para cada contribución.
- Implementar nuevas funcionalidades o correcciones de errores.
- Enviar solicitudes de extracción al repositorio principal.
- Participar en discusiones y ayudar en la revisión y resolución de problemas y solicitudes de extracción.

11 Recursos Adicionales

A continuación se presentan enlaces a recursos útiles relacionados con el desarrollo de la aplicación:

- Curso de Kotlin en YouTube:

  Curso de Kotlin
- Curso de programación en Kotlin para Android:

  Kotlin para Android

- Documentación oficial de Room para Android:

  Documentación Room

- Android Studio:

  Descargar Android Studio

- Documentación de Google Maps Platform:

  Documentación Maps

- Codelab de Room con una vista en Kotlin:

  Room en Kotlin

- Serie de tutoriales en YouTube sobre desarrollo de aplicaciones Android:

  Tutoriales desarrollo de aplicaciones Android

- Tutorial de Ngrok:

  Como usar Ngrok

- Otro tutorial de Ngrok:

  Como usar Ngrok II

12 Contacto

Para cualquier consulta o información adicional, puedes ponerte en contacto con el equipo de desarrollo o el responsable del proyecto a través de las siguientes direcciones de correo electrónico:

- ibarguipas@educacion.navarra.es
- vaguile@educacion.navarra.es
- jmarinsanc@educacion.navarra.es
- aperezdmor@educacion.navarra.es
- autocolorslatex@gmail.com