



MANUAL DE MANTENIMIENTO

Documentación de Cash Scan

UNPSJB - Desarrollo de Software

Cátedra
Alumnos

Mayorga Ernesto, Pacheco Cristian
Rivero Agustín, Casteglione Matias

Contenido

1. Objetivos	4
1.1. Objetivos Específicos	4
2. Alcance	5
3. Requerimientos Técnicos	5
3.1. Requerimientos Mínimos de Hardware	6
3.2. Requerimientos Mínimos de Software	6
4. Herramientas Utilizadas para el Desarrollo	6
5. Instalación	7
5.1. Android Studio	7
5.2. Entorno de Desarrollo Python	13
6. Configuración	20
7. Diseño de la Arquitectura	26
7.1. Base de Datos	26
7.2. Billeto	28
7.3. Item de cada Billeto	30
7.4. Adaptador de interfaz para los Billetes	31
7.5. Adaptador Gráfico del Tutorial	34
7.6. Adaptador para los Modelos de Clasificación	36
7.7. Configuración	39
7.8. Historial de Billetes	42
7.9. Introducción de la Aplicación	45
7.10. Menú Principal	47
7.11. Escaneo de Billetes	49
7.12. Tutorial	52
8. Desarrollo	55
8.1. SQLiteHelper.kt	55
8.2. BillData.kt	58

8.3. ScreenItem.kt	58
8.4. BillsAdapter.kt	59
8.5. IntroViewPagerAdapter.kt.....	62
8.6. ModelListAdapter.kt.....	64
8.7. ConfigActivity.kt.....	66
8.8. CountBillActivity.kt.....	69
8.9. IntroActivity.kt	73
8.10. MainActivity.kt	75
8.11. ScanBillActivity.kt	78
8.12. TutorialActivity.kt	84
8.13. MCBIA7.py	88
9. Privilegios	91

1. Objetivos

Este documento ha sido elaborado con el objetivo principal de describir el diseño de la aplicación, proporcionando una guía clara y detallada para facilitar la interacción la misma. Además, sirve como referencia para realizar actualizaciones o llevar a cabo tareas de mantenimiento en caso de fallos.

De manera general, el diseño del documento está orientado a ofrecer al programador encargado una visión integral de cómo fue desarrollado el sistema, incluyendo aspectos que se detallaran en breve.

1.1. Objetivos Específicos

El manual incluye los siguientes apartados:

- **Guía de instalación de la aplicación**

Una descripción paso a paso de cómo instalar la aplicación, incluyendo los comandos necesarios, las configuraciones específicas y los posibles problemas que podrían surgir durante la instalación, con sus respectivas soluciones.

- **Código fuente para futuras actualizaciones**

Una sección dedicada al código fuente, organizada y comentada adecuadamente, para facilitar la comprensión y permitir actualizaciones o modificaciones por parte de futuros desarrolladores.

- **Requisitos para la ejecución de dicha aplicación**

Listado detallado de los requisitos necesarios, tales como hardware, software, bibliotecas, entornos de ejecución y configuraciones previas que deben ser cumplidos para que la aplicación funcione correctamente.

- **Diseño de la aplicación**

Se mostraran los diseños conceptuales de la aplicación mediante diagramas UML de las clases, atributos, métodos y sus relaciones.

También se incluyen diagramas de estados para visualizar los estados de la aplicación y las transiciones entre ellos en función de eventos y acciones.

2. Alcance

Este documento, como se detalló en el objetivo, está diseñado específicamente para orientar al programador encargado del mantenimiento y actualización de la aplicación.

Dado el nivel de especialización que esta tarea implica, a continuación se presenta una lista de requisitos imprescindibles que el programador debe tener en cuenta para desempeñar esta responsabilidad.

Requisitos de Conocimiento

1. Programación en Android:

- Familiaridad con Android Studio como entorno de desarrollo integrado (IDE).
- Conocimiento de Kotlin como lenguaje de programación principal.
- Experiencia en el manejo de componentes Android, como Activities y ViewModels.

2. Entrenamiento de modelos de clasificación de imágenes mediante redes convolucionales:

- Conocimiento de los frameworks TensorFlow Keras y PyTorch para la creación, entrenamiento y exportación de modelos.
- Capacidad para integrar modelos pre-entrenados y personalizados en aplicaciones Android utilizando herramientas como TensorFlow Lite.
- Habilidades en el pre procesamiento de datos, ajuste de hiperparámetros y evaluación del rendimiento del modelo.

3. Requerimientos Técnicos

Los requisitos técnicos son importantes para garantizar que el sistema se construya, implemente y mantenga correctamente. A continuación, se detallan los aspectos de hardware y software, necesarios para el proyecto.

3.1. Requerimientos Mínimos de Hardware

- CPU: Microprocesador de al menos 1 GHz o superior, con 2 o más núcleos de procesamiento y compatible con 64-bit.
- Memoria RAM: 4 GB.
- Almacenamiento: 128 GB o más disponible para la instalación del sistema operativo, actualizaciones, y software adicional.
- GPU: Compatible con DirectX 12 o superior.
- Pantalla: Pantalla de 1024 x 768 como mínimo con 256 colores.

3.2. Requerimientos Mínimos de Software

- Windows 7 / Windows 10 / Windows 11 de 64-bit.
- Como alternativa, cualquier distribución de Linux de 64 bits.
- SDK de Java 17.
- SDK de Android versión 21, como mínimo.
- Intérprete Python versión 3.11 requerida.
- Virtualizador del computador activado desde la BIOS.

4. Herramientas Utilizadas para el Desarrollo

El proceso de desarrollo de la aplicación y del modelo de inteligencia artificial ha involucrado diversas herramientas y tecnologías. A continuación, se describen las principales herramientas utilizadas.

- Entorno de Desarrollo Integrado (IDE): Android Studio, para el desarrollo de la aplicación Android.
- Visual Studio Code: Para la programación y entrenamiento de modelos de inteligencia artificial en Python.
- ElevenLabs: Para la creación de voces para la aplicación, con la voz de Jessica.
- DeepL: Para la traducción del idioma de la interfaz gráfica a distintos lenguajes.
- RoboFlow Universe: Para almacenar y explorar, datasets y modelos de reconocimiento de imágenes en la nube.
- Distintas bibliotecas para Python, que están listadas en los requerimientos del proyecto.

5. Instalación

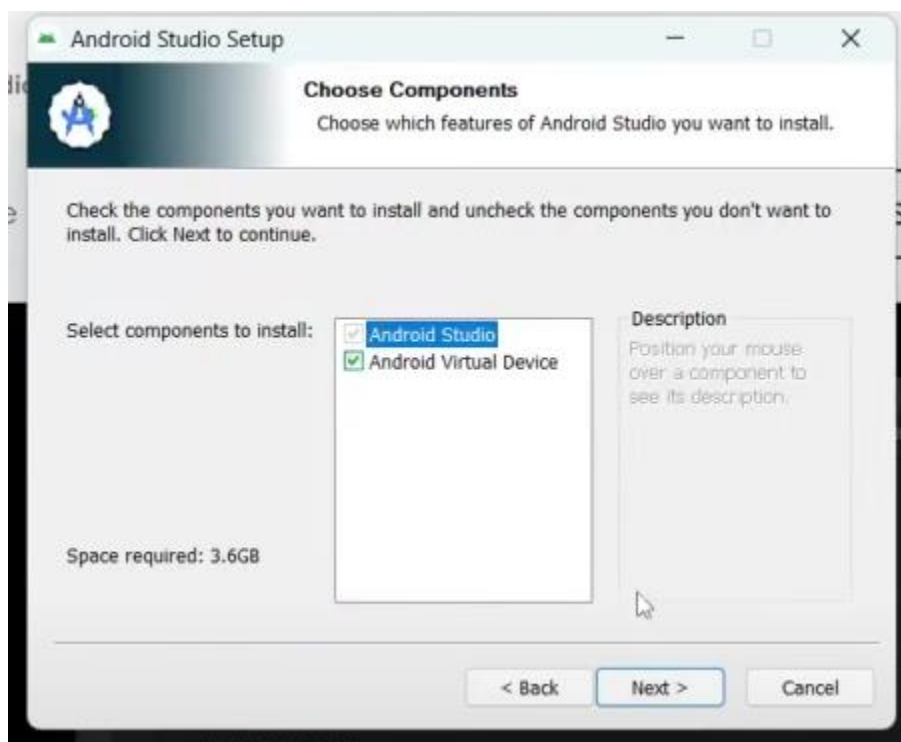
A continuación se detallaran los pasos para instalar los programas y entornos de desarrollo para el proyecto, tanto en Windows como en distribuciones Linux.

5.1. Android Studio

1. Instalación en Windows

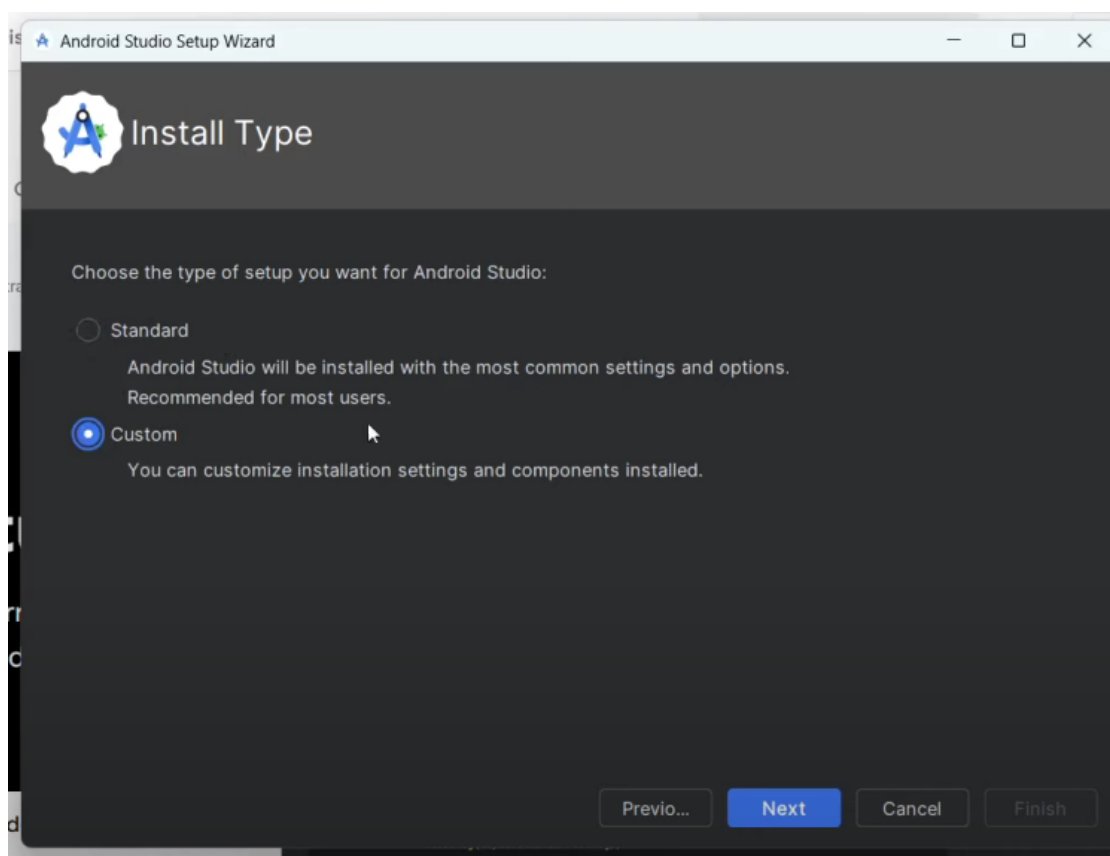
Accedemos a la página oficial de Android Studio (<https://developer.android.com/studio>) y hacemos clic en el botón de descarga correspondiente.

Se mostrará un cuadro de diálogo con los términos y condiciones correspondientes a la Licencia de Android Software Development Kit (SDK). Una vez revisado el contenido completo del acuerdo, será necesario marcar la casilla de verificación que indica la aceptación de los términos: "He leído y acepto los términos y condiciones mencionados anteriormente". Posteriormente, se procederá a iniciar la descarga de Android Studio al presionar el botón correspondiente.

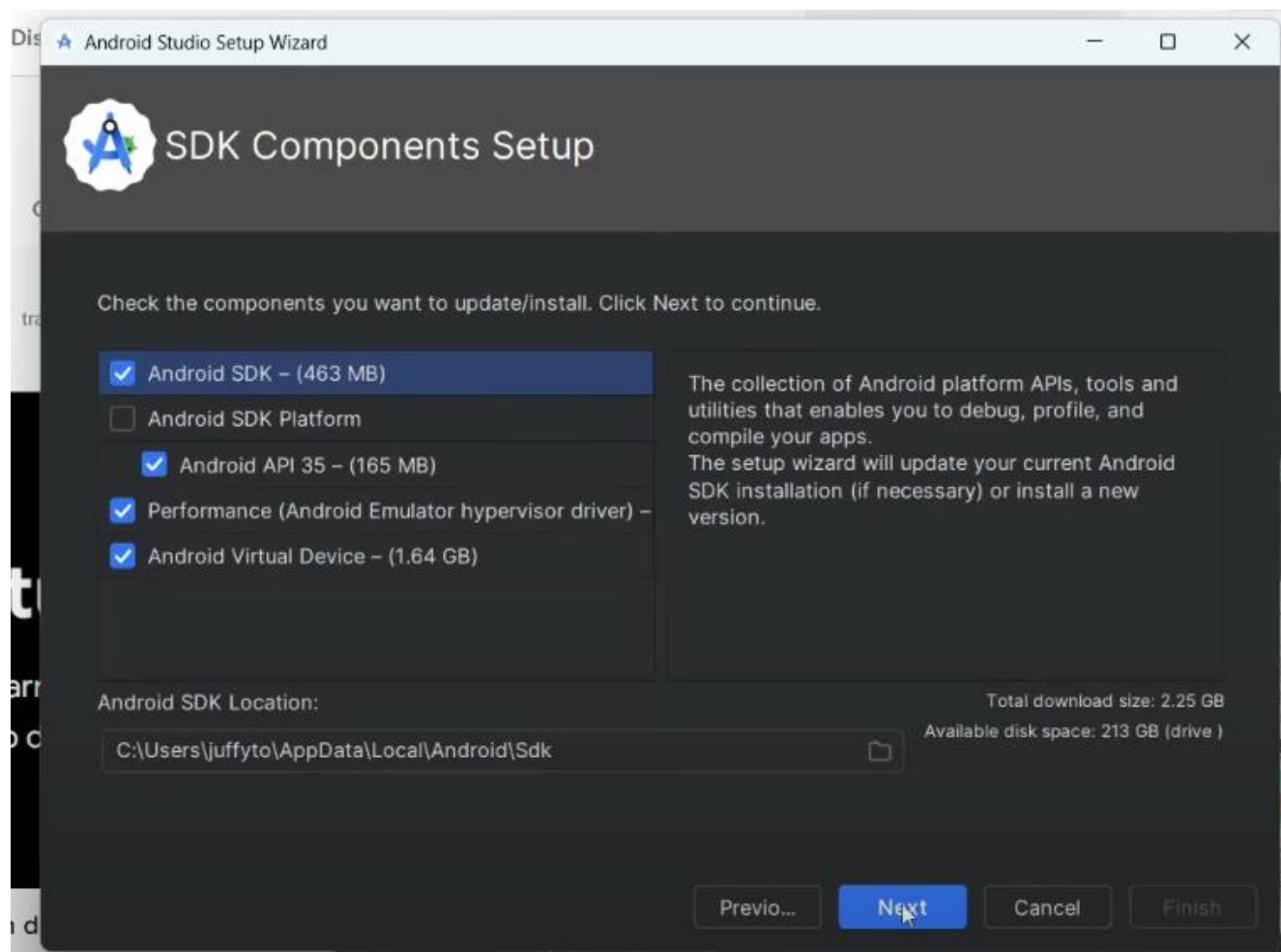


En Windows, se descargará un archivo ejecutable. Al abrirlo, el asistente de instalación solicitará confirmar si se desea instalar Android Studio junto con el emulador para dispositivos Android. Es fundamental asegurarse de que esta opción esté seleccionada durante la instalación, ya que el emulador es una herramienta esencial para realizar pruebas y depuración en el desarrollo de la aplicación.

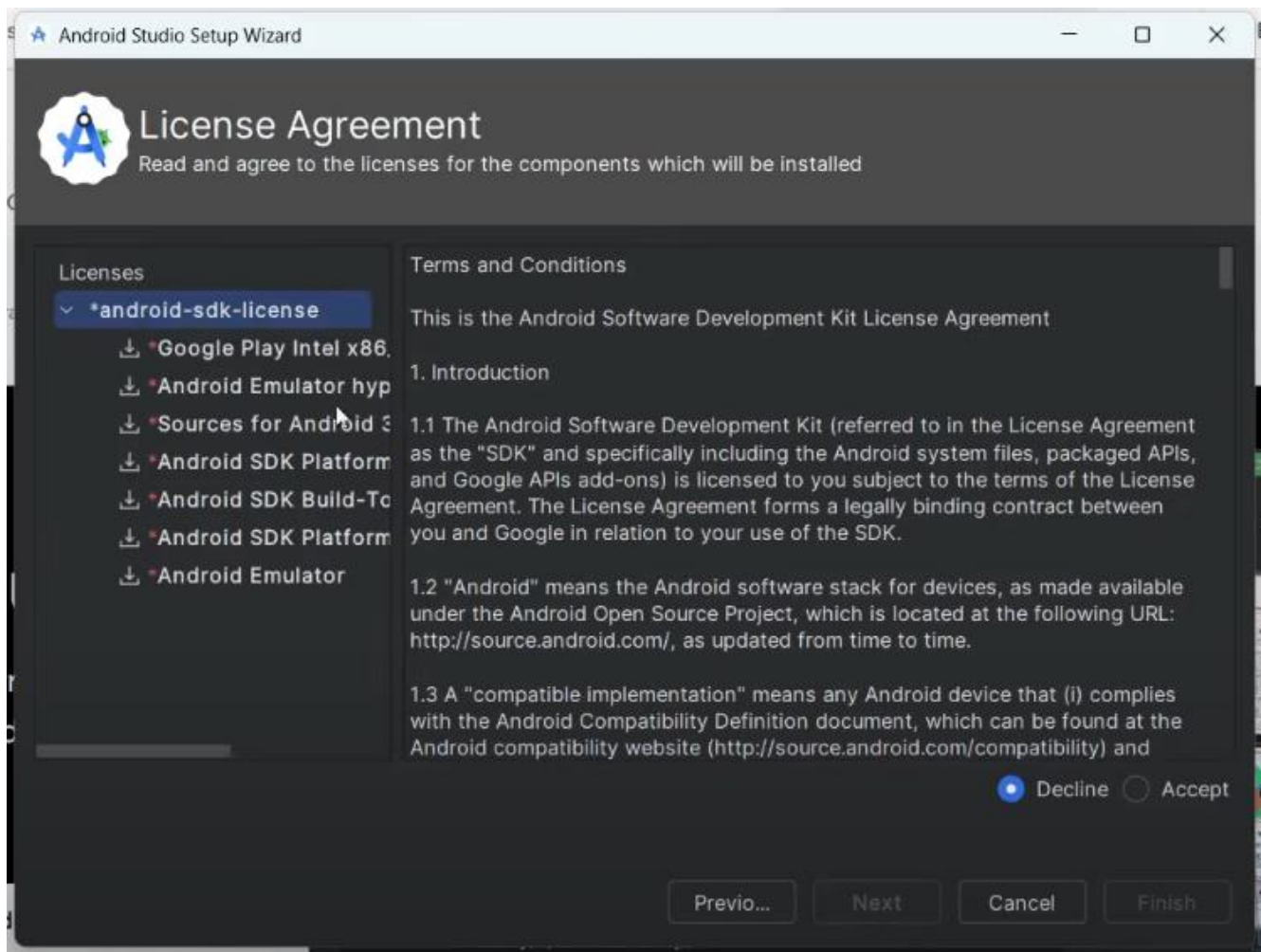
Una vez dado en siguiente, seleccionamos la ubicación de instalación, e instalamos el IDE.



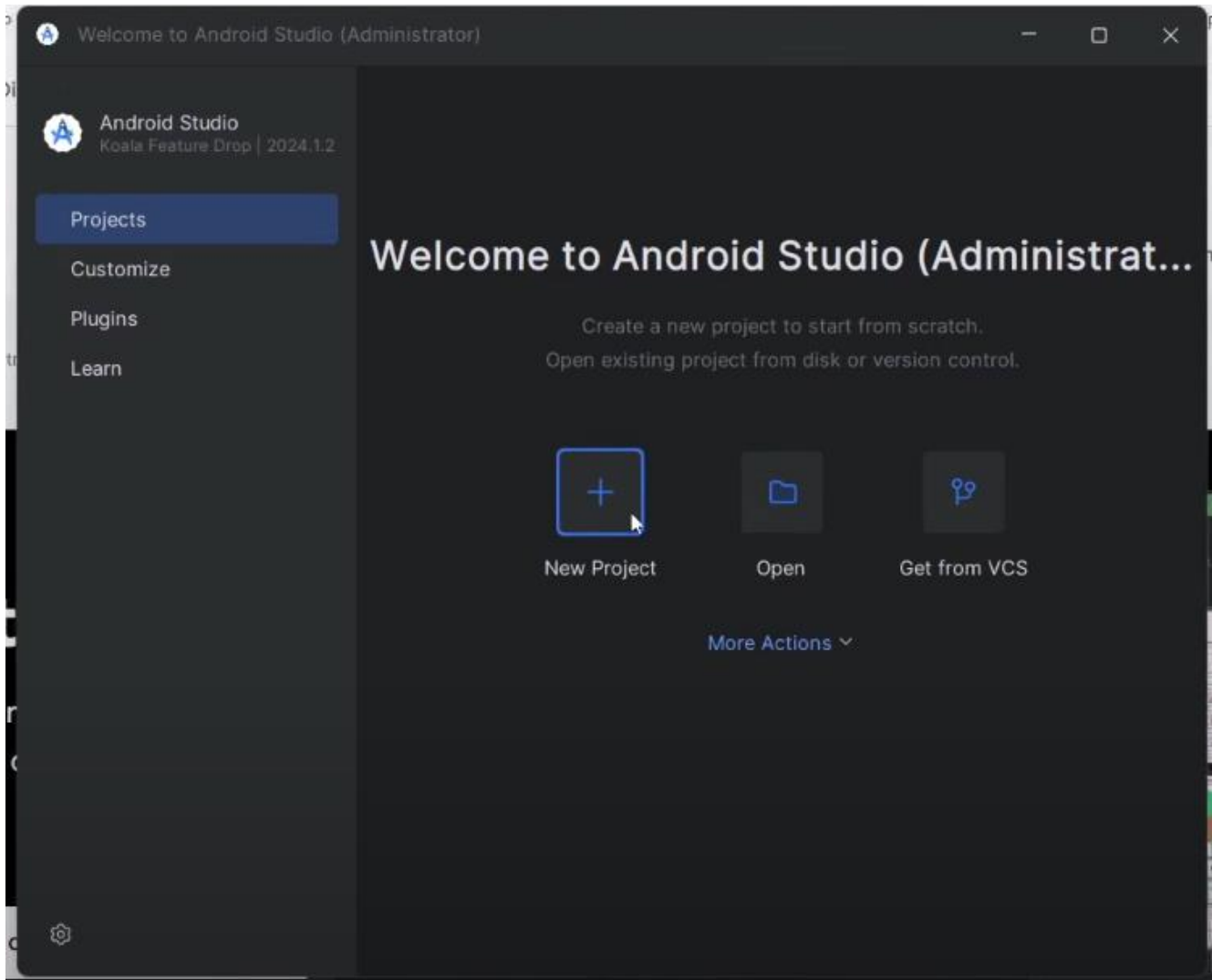
Al abrir Android Studio por primera vez, se iniciará automáticamente el asistente de instalación del Android SDK. En el primer paso, es recomendable seleccionar la opción de instalación personalizada (Custom) en lugar de la estándar. Esto permite configurar detalladamente las herramientas a instalar y elegir las versiones específicas del SDK de Android que se utilizarán durante el desarrollo de la aplicación, asegurando que sean compatibles con los requerimientos del proyecto.



Se debe seleccionar la opción para instalar el Android SDK, junto con todas las dependencias adicionales que este requiere para su correcto funcionamiento. Posteriormente, se especifica la ubicación donde se desea realizar la instalación, asegurándose de contar con suficiente espacio en el disco para alojar los componentes seleccionados.



Durante el proceso de instalación, se presenta nuevamente una sección que detalla los términos y condiciones previamente vistos en la página principal de Android Studio. Es necesario aceptar estos términos para continuar con la instalación. Una vez aceptados, el instalador procede a descargar e instalar todos los componentes seleccionados.



Al completar el proceso de instalación, el instalador mostrará una pantalla de finalización, indicando que Android Studio se ha instalado correctamente y está listo para su uso. Al abrirlo veremos la pantalla principal del IDE, crear y cargar proyectos.

2. Linux

En sistemas operativos basados en Linux, es necesario instalar previamente las dependencias de Java para garantizar la compatibilidad con Android Studio. Los comandos requeridos son los siguientes:

```
$ sudo add-apt-repository universe  
$ sudo apt update  
$ sudo apt install openjdk-17-jdk -y
```

```
$ sudo apt install openjdk-17-jre -y
```

Después de haber instalado Java, accedemos a la página oficial de Android Studio, la cual detectará automáticamente el sistema operativo Linux y proporcionará el archivo de instalación correspondiente. Tras la descarga, descomprimos el archivo y, a continuación, abrimos la terminal en la misma ubicación donde se encuentra el archivo descomprimido. En la terminal, ejecutamos los siguientes comandos:

```
$ sudo mv android-studio /opt/
```

```
$ sudo ln -sf /opt/android-studio/bin/studio.sh /bin/android-studio
```

Una vez que haya ejecutado estos comandos, copia un fragmento de código para permitir que la aplicación Android Studio esté disponible en el acceso directo del menú de la aplicación en Linux:

```
$ sudo nano /usr/share/applications/android-studio.desktop
```

Una vez dentro del editor de texto nano, copiamos el siguiente texto haciendo Ctrl + C y lo pegamos con Mayus + Ctrl + v.

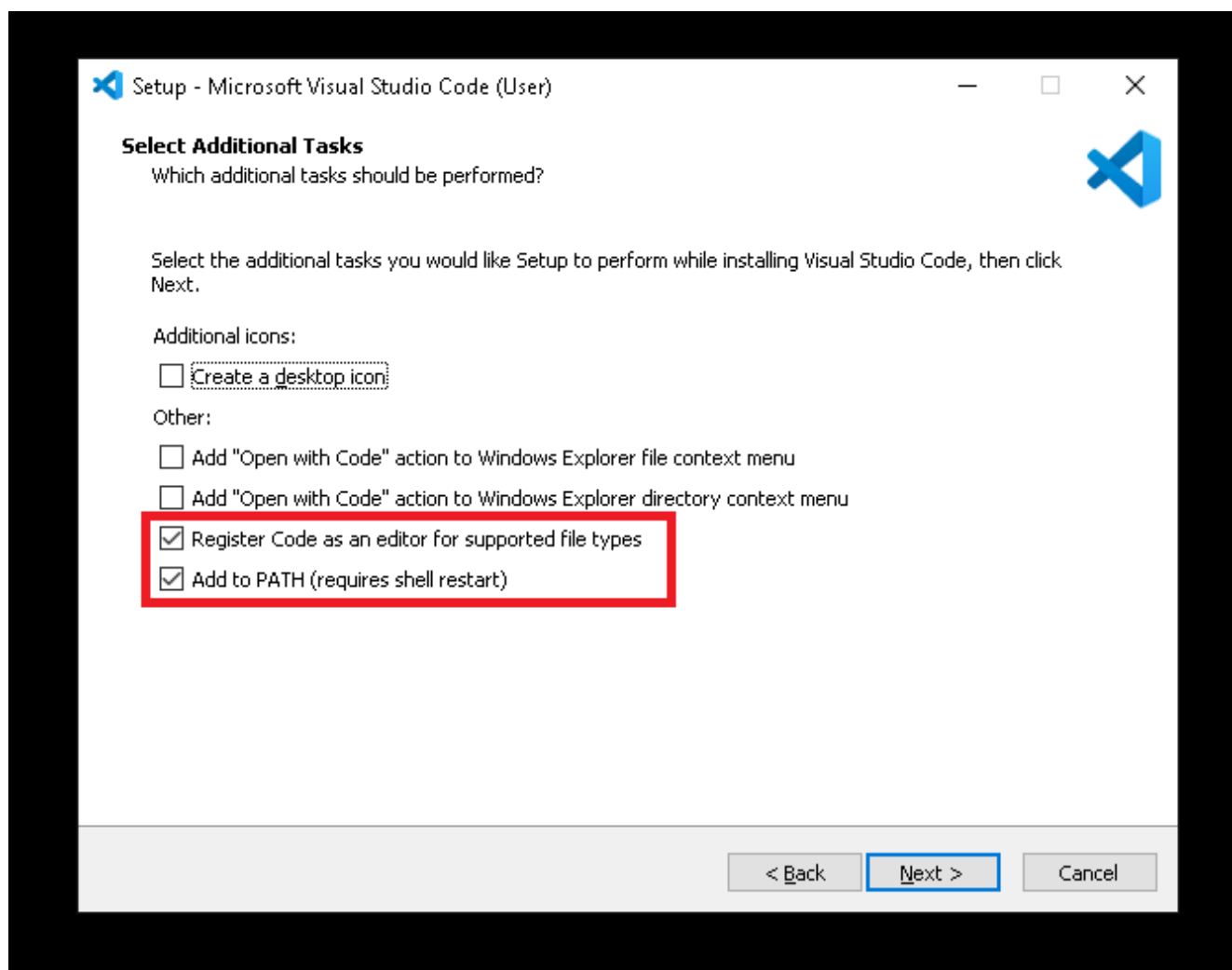
```
[Desktop Entry  
Version=1.0  
Type=Application  
Name=Android Studio  
Comment=Android Studio  
Exec=bash -i "/opt/android-studio/bin/studio.sh" %f  
Icon=/opt/android-studio/bin/studio.png  
Categories=Development;IDE;  
Terminal=false  
StartupNotify=true  
StartupWMClass=jetbrains-android-studio  
Name[en_GB]=android-studio.desktop
```

Una vez realizado el cambio, guardamos y cerramos el archivo presionando Ctrl + X, luego confirmamos la acción presionando Enter para guardar los cambios realizados. Ahora, Android Studio estará disponible para ser ejecutado desde el menú de aplicaciones o actividades. A partir de este punto, el proceso de configuración y ejecución sigue los mismos pasos que en la instalación para Windows.

5.2. Entorno de Desarrollo Python

1. Windows

Para proceder con la instalación de Visual Studio Code y configurar el entorno de programación en Python, nos dirigimos a su página de instalación (<https://code.visualstudio.com/>) y presionamos en el botón de descargar para Windows. Una vez descargado el archivo de instalación, continúa con los pasos de instalación.

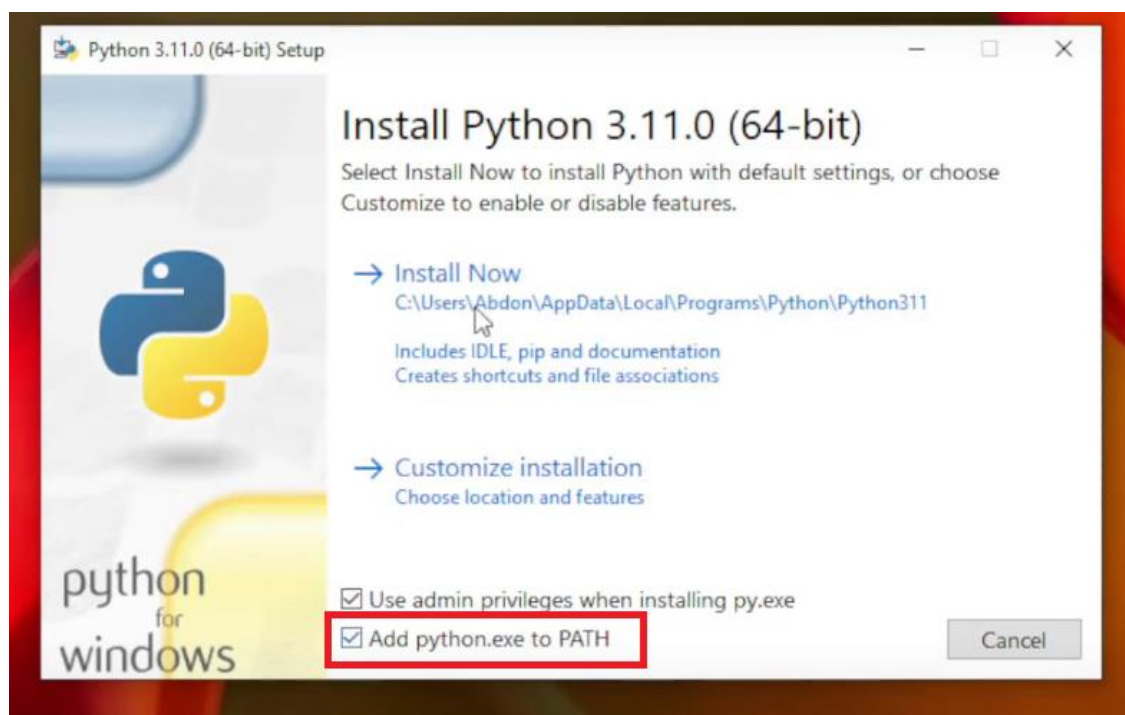


Durante el proceso de instalación, asegúrese de seleccionar la opción "Agregar a PATH" para facilitar el acceso desde la línea de comandos. Luego, elige la ubicación de instalación deseada y procede con la instalación del programa.

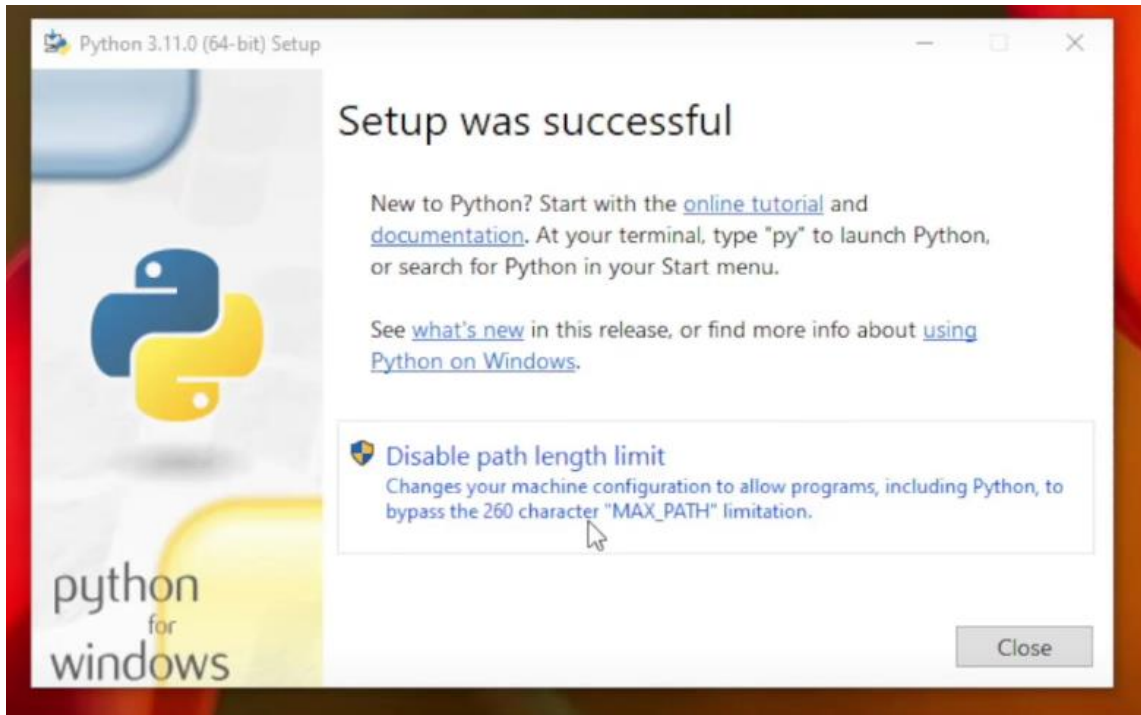
Ahora Nos dirigimos a la página de descarga de Python 3.11 (<https://www.python.org/downloads/release/python-3110/>) y seleccionamos el instalador recomendado.

Files						
Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore
Gzipped source tarball	Source release		c5f77f1ea256dc5bdb0897eeb4d35bb0	25.1 MB	SIG	.sigstore
XZ compressed source tarball	Source release		fe92acfa0db9b9f5044958edb451d463	18.9 MB	SIG	.sigstore
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	98fa94815780c9330fc2154559365834	40.6 MB	SIG	CRT SIG
Windows installer (64-bit)	Windows	Recommended	4fe11b2b0bb0c744cf74aff537f7cd7f	24.0 MB	SIG	CRT SIG
Windows installer (32-bit)	Windows		e369a267acaa d62487223bd835279bb9	22.9 MB	SIG	CRT SIG
Windows installer (ARM64)	Windows	Experimental	18e5bd9a4854109a df3b77c7c9dc1ded	23.2 MB	SIG	CRT SIG
Windows embeddable package (64-bit)	Windows		7df0f4244e5a66760b7caa ed58e86c93	10.1 MB	SIG	CRT SIG
Windows embeddable package (32-bit)	Windows		0888959642cc8af087d88da3866490a5	9.1 MB	SIG	CRT SIG
Windows embeddable package (ARM64)	Windows		e3dbbd5d63c6cb203a dc6c0c8ca5f5f7	9.3 MB	SIG	CRT SIG

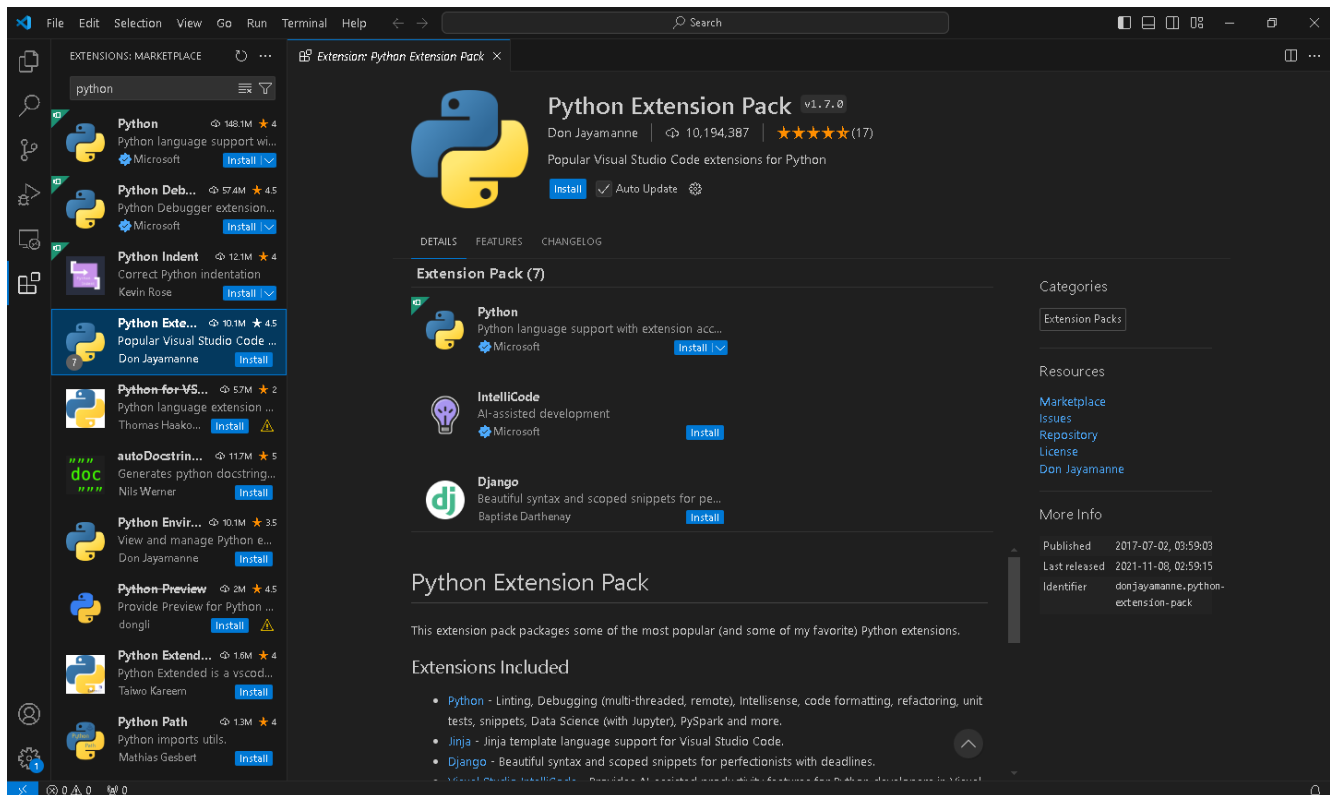
Al abrir el instalador de Python, asegúrate de marcar la casilla "Add Python.exe to PATH" antes de continuar con la instalación. Esto permitirá que Python se pueda ejecutar desde la línea de comandos sin necesidad de especificar la ruta completa. Luego, selecciona la opción "Install Now" para iniciar el proceso de instalación.



Al finalizar la instalación de Python, haz clic en el botón "Disable path length limit". Esta opción eliminará la restricción de longitud de ruta en Windows, lo que evita problemas al trabajar con directorios o archivos con rutas largas en proyectos de Python.

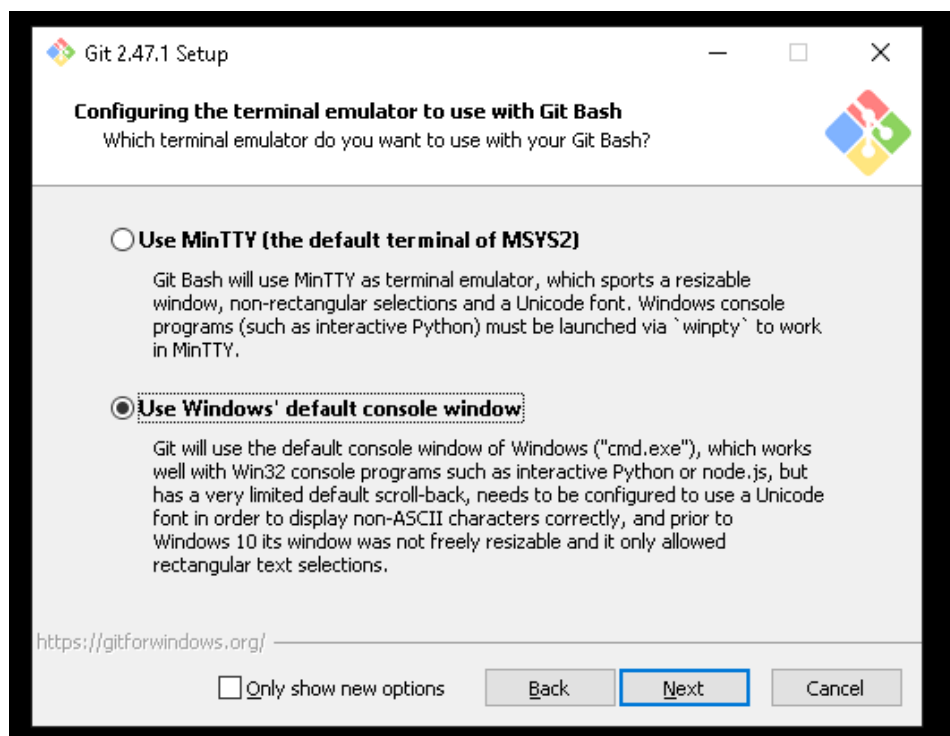


A continuación, abrimos Visual Studio Code e instalamos el paquete de extensiones para desarrollo en Python. Para ello, accedemos la opción de extensiones en Visual Studio Code o al siguiente enlace en el Marketplace del editor de texto (<https://marketplace.visualstudio.com/items?itemName=donjayamanne.python-extension-pack>). Este paquete incluye soporte completo para el lenguaje Python, herramientas de depuración, IntelliCode, administrador de entornos virtuales y funcionalidades de indentación, como se ve en la imagen.

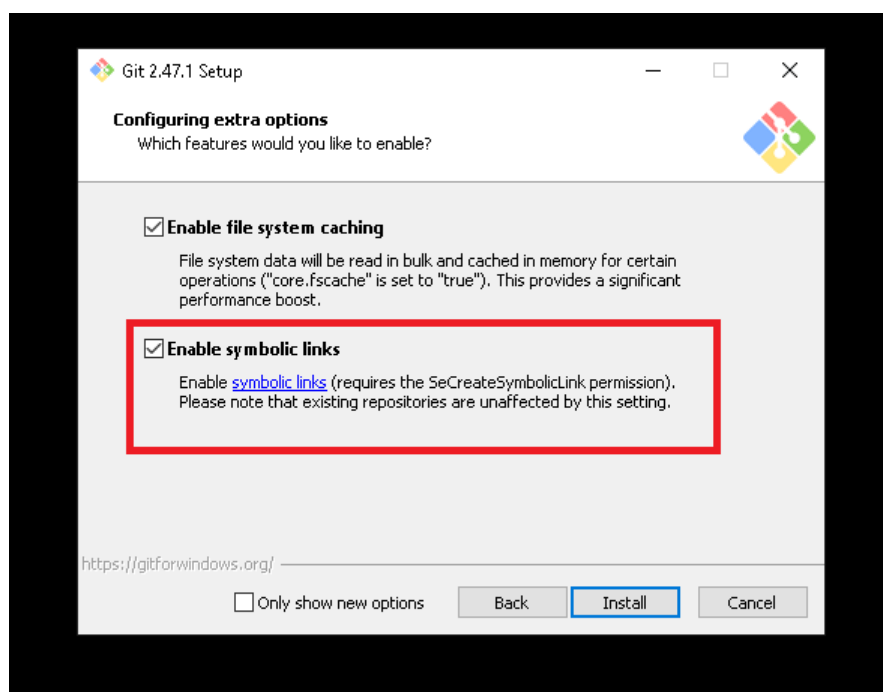


Es fundamental descargar Git para Windows. Para ello, accedemos a su página oficial de descarga (<https://git-scm.com/downloads>) y seleccionamos el instalador correspondiente a Windows 64 bits.

Durante el proceso de instalación, se presentarán varios mensajes que configuran Git para el sistema operativo. En la mayoría de los casos, no es necesario realizar ajustes adicionales para esta guía de instalación. A continuación, se describen los pasos relevantes que deben seguirse.



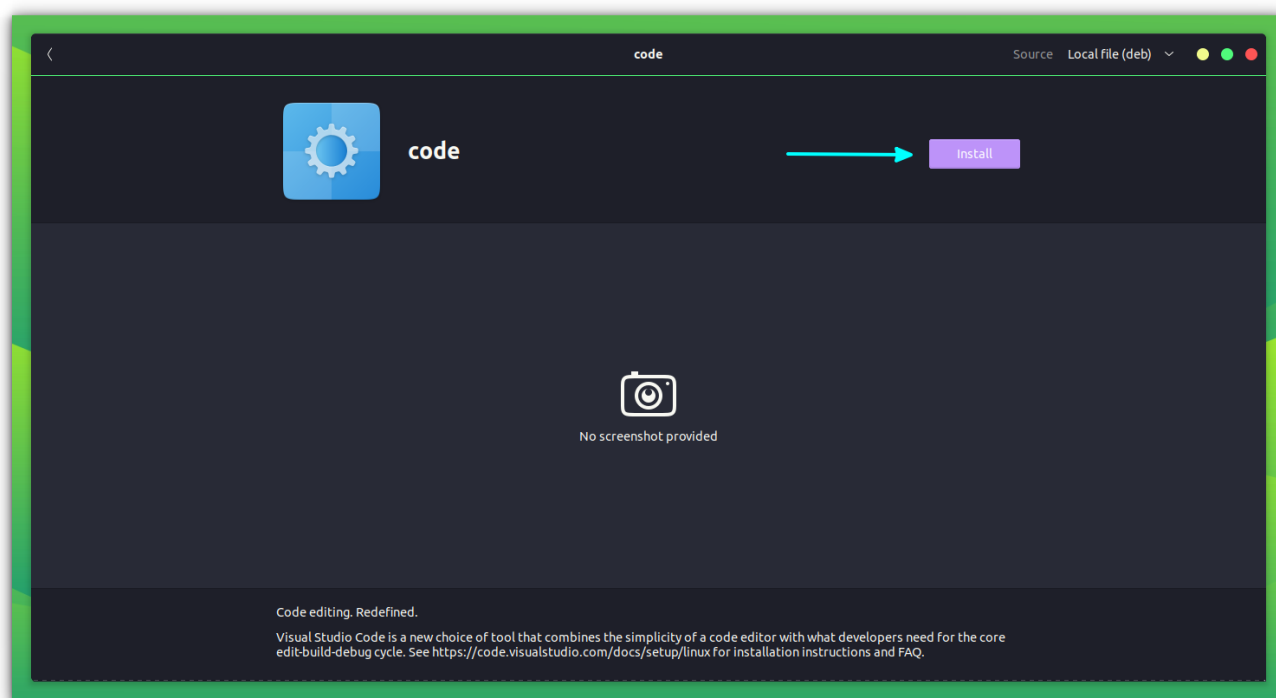
En el apartado de selección de terminal para Git, elegiremos la opción "Use Window's default console window", usar Git desde la consola de Windows por defecto, tal como se muestra en la imagen. Esta opción asegura que Git se pueda ejecutar desde la terminal CMD estándar de Windows sin necesidad de configuraciones adicionales.



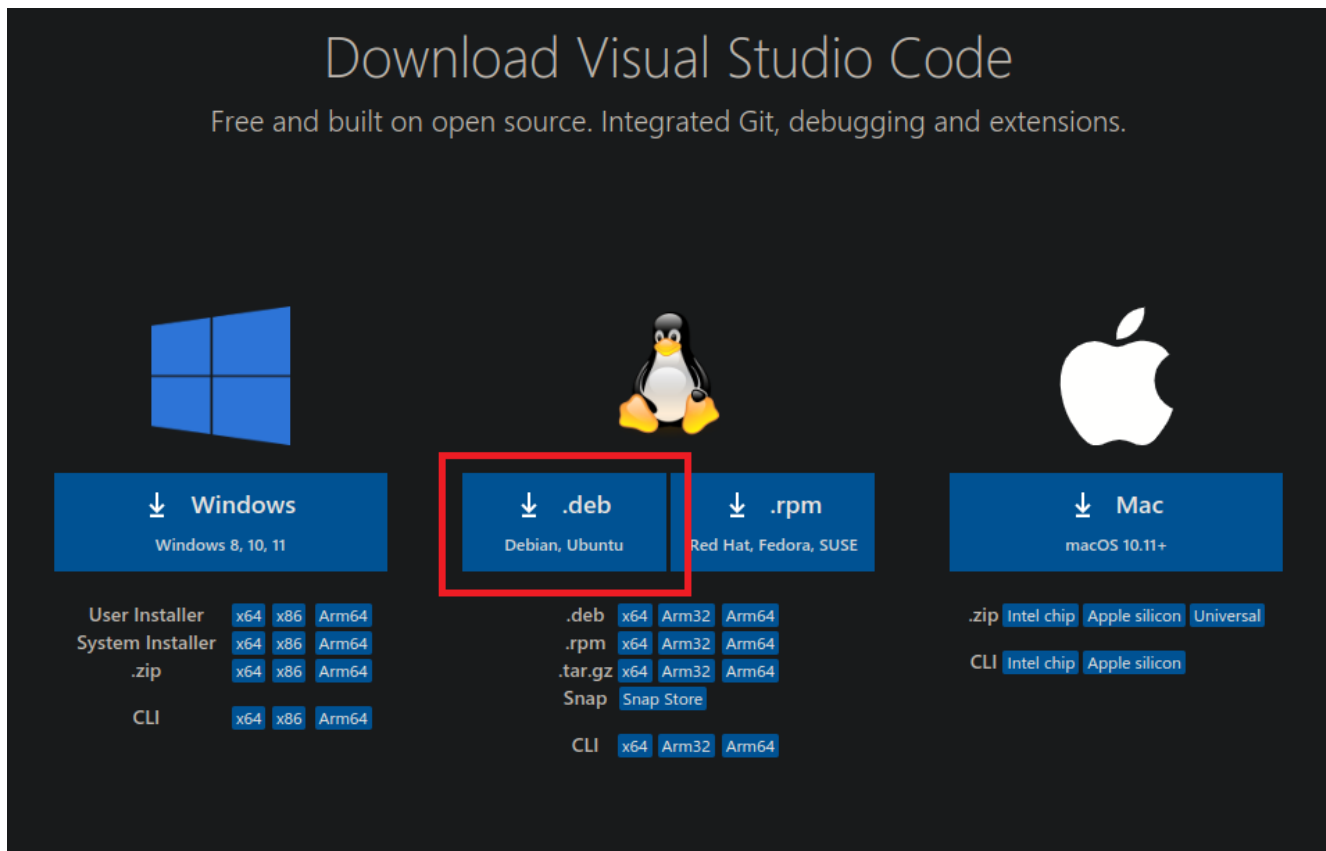
A continuación, activamos la opción de compatibilidad con Symbolic Links en Windows. Las demás opciones las dejamos con la configuración predeterminada. Una vez seleccionadas, procedemos con la instalación de Git, y finalizamos con la instalación.

2. Linux

En Linux, para instalar Visual Studio Code, accedemos a la página oficial del programa, la cual detecta automáticamente el sistema operativo. Descargamos el archivo ejecutable .deb para arquitectura de 64 bits y procedemos con la instalación del paquete.



Como se puede observar en la imagen, esta es la pantalla del instalador de Visual Studio Code cuando se utiliza el gestor de paquetes correspondiente a la distribución de Linux que se esté utilizando.



Como alternativa, en la página de descarga de Visual Studio Code, se ofrece la opción de descargar el archivo adecuado para sistemas Linux basados en Debian (.deb). En caso de estar utilizando una distribución basada en Red Hat, se debe descargar el archivo .rpm correspondiente.

El resto de los pasos de configuración de Visual Studio Code son los mismos. Git ya está preinstalado en la mayoría de las distribuciones de Linux, por lo que no es necesario instalarlo por separado. Aunque Python también suele estar instalado por defecto en Linux, para este proyecto es necesario utilizar una versión 3.11, para garantizar la compatibilidad con SDKs de Android antiguos.

Para instalar Python 3.11 en Linux, es necesario abrir la terminal y ejecutar el siguiente comando:

```
$ sudo apt install python3.11
```

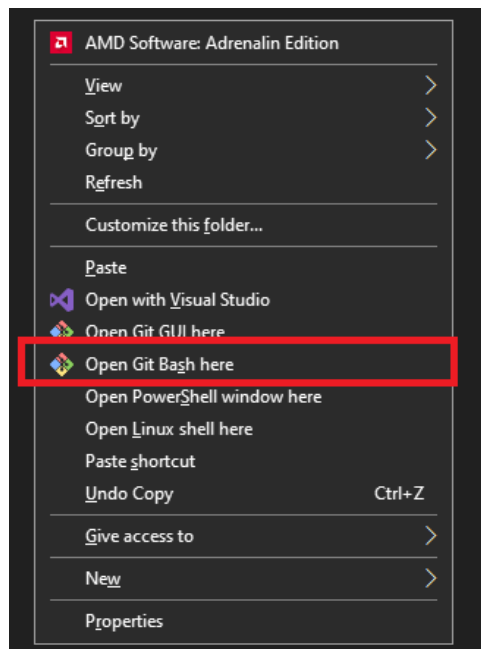
Una vez completada la instalación de Python 3.11, se habrá configurado correctamente el entorno de desarrollo con Android Studio y Visual Studio Code, con el entorno de programación en Python listo para ser utilizado en el proyecto.

6. Configuración

Para configurar Android Studio para el desarrollo de la aplicación, siga los pasos a continuación:

1. Clonar el repositorio del proyecto

Seleccione una carpeta donde desee guardar los archivos del proyecto. Luego, realice Shift + Clic derecho en el directorio y elija la opción de abrir la terminal de Git, como se ilustra en la imagen.



Una vez dentro de la terminal de Git, proceda a clonar el repositorio del proyecto utilizando el siguiente comando:

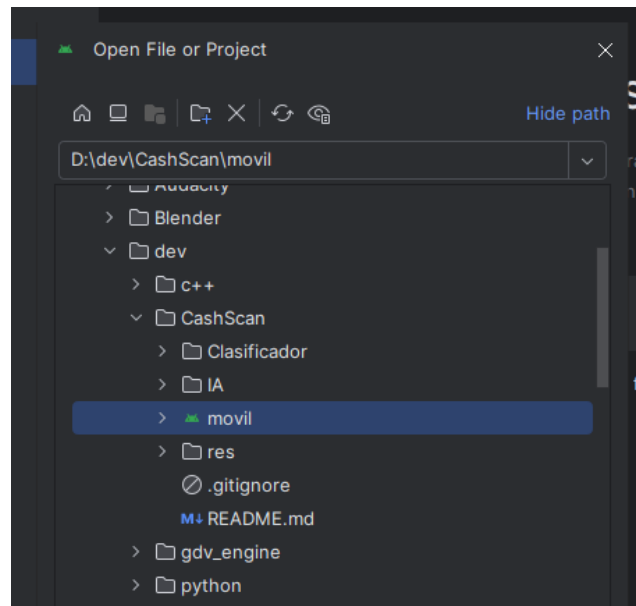
git clone <https://github.com/macasteghione/CashScan>

Después de ejecutarlo, observará que se ha creado una nueva carpeta en el directorio seleccionado, la cual contiene todos los archivos del proyecto.

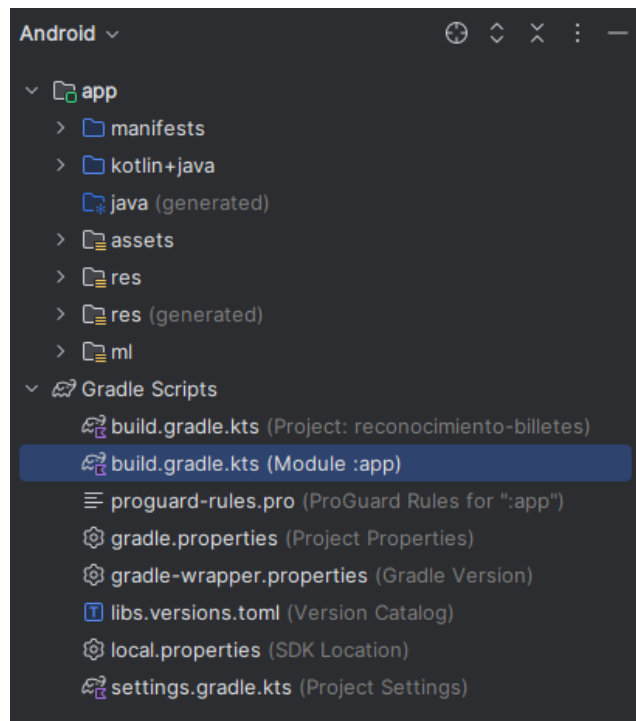
2. Configurar Android Studio

Inicie Android Studio y, en el menú principal, seleccione la opción "Abrir un proyecto". Alternativamente, puede ir a Archivo → Abrir.

Navegue hasta la carpeta donde se encuentra el repositorio que clonó en el paso anterior.



Seleccione la carpeta denominada "móvil" y haga clic en Aceptar. Android Studio procederá a configurar automáticamente el entorno de desarrollo para el proyecto. Este proceso puede tardar varios minutos en completarse, dependiendo de las dependencias y configuraciones necesarias.



Una vez que Android Studio complete el proceso de carga del proyecto, la estructura de archivos debería visualizarse como se muestra en la imagen.

- Build.gradle.kts

Ubique el archivo build.gradle.kts (Module: app) dentro de la estructura del proyecto y ábralo.

En la línea 78 del archivo, se configura la integración de Python en el proyecto. En esta sección, se debe especificar la ruta del intérprete de Python. También se pueden modificar las bibliotecas que se requieren instalar y la ubicación de los archivos .py correspondientes al proyecto.

```

78  chaquopy {
79      defaultConfig {
80          buildPython( ...bp: "C:\\Users\\maty\\AppData\\Local\\Programs\\Python\\Python311\\python.exe") // Ubicación de python
81          version = "3.8"
82
83          // Para instalar librerías de Python
84          pip {
85              install(...args: "requests")
86          }
87      }
88
89      sourceSets {
90          getByName( name: "main") {
91              srcDir( srcPath: "src/main/python") // Ubicación de los archivos python
92          }
93      }
94  }

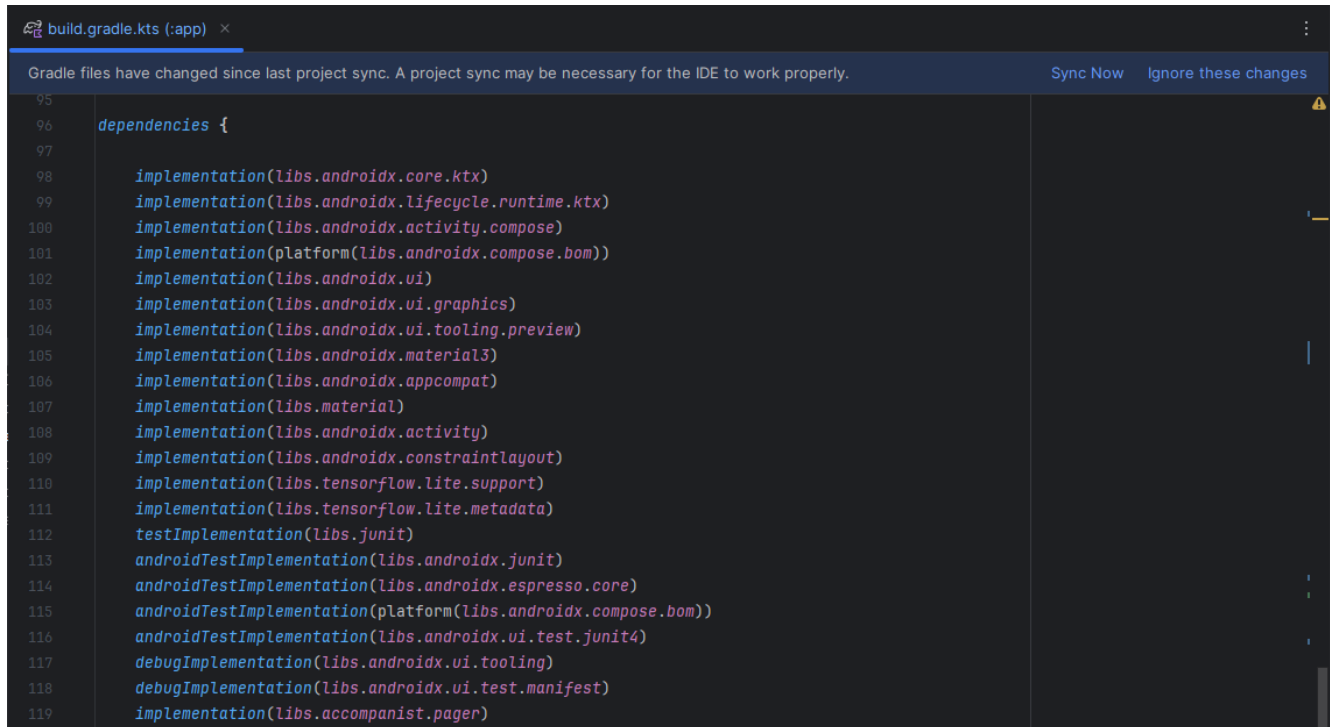
```

Para que la aplicación compile correctamente, localizamos la ubicación del intérprete de Python 3.11, que por defecto se encuentra en la ruta:

C:/Users/[nombre_de_usuario]/AppData/Local/Programs/Python/Python311/

Dentro de esta carpeta, identificamos el archivo ejecutable de Python, llamado python.exe. Aseguramos que la dirección completa al ejecutable sea configurada correctamente.

En el caso de usar Linux, el intérprete se encuentra en la ruta: **/usr/bin/python3**



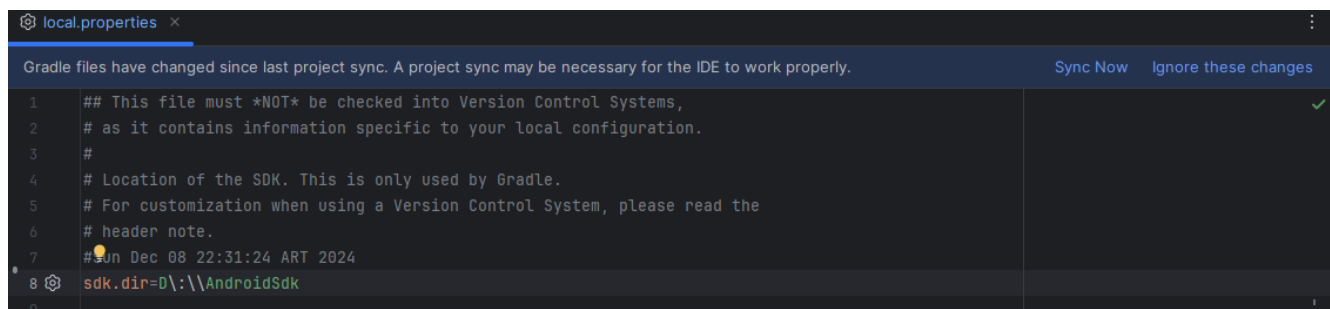
```

95
96 dependencies {
97
98     implementation(libs.androidx.core.ktx)
99     implementation(libs.androidx.lifecycle.runtime.ktx)
100    implementation(libs.androidx.activity.compose)
101    implementation(platform(libs.androidx.compose.bom))
102    implementation(libs.androidx.ui)
103    implementation(libs.androidx.ui.graphics)
104    implementation(libs.androidx.ui.tooling.preview)
105    implementation(libs.androidx.material3)
106    implementation(libs.androidx.appcompat)
107    implementation(libs.material)
108    implementation(libs.androidx.activity)
109    implementation(libs.androidx.constraintlayout)
110    implementation(libs.tensorflow.lite.support)
111    implementation(libs.tensorflow.lite.metadata)
112    testImplementation(libs.junit)
113    androidTestImplementation(libs.androidx.junit)
114    androidTestImplementation(libs.androidx.espresso.core)
115    androidTestImplementation(platform(libs.androidx.compose.bom))
116    androidTestImplementation(libs.androidx.ui.test.junit4)
117    debugImplementation(libs.androidx.ui.tooling)
118    debugImplementation(libs.androidx.ui.test.manifest)
119    implementation(libs.accompanist.pager)

```

Dentro del mismo archivo, es posible añadir dependencias y bibliotecas adicionales para el proyecto, las cuales pueden ser necesarias en el futuro. Para ello, se debe agregar la función `implementation()` y dentro de ella, especificar el nombre de la biblioteca que se desea incluir.

- Local.properties



```

1  ## This file must *NOT* be checked into Version Control Systems,
2  # as it contains information specific to your local configuration.
3  #
4  # Location of the SDK. This is only used by Gradle.
5  # For customization when using a Version Control System, please read the
6  # header note.
7  #Sun Dec 08 22:31:24 ART 2024
8  sdk.dir=D:\\AndroidSdk
9

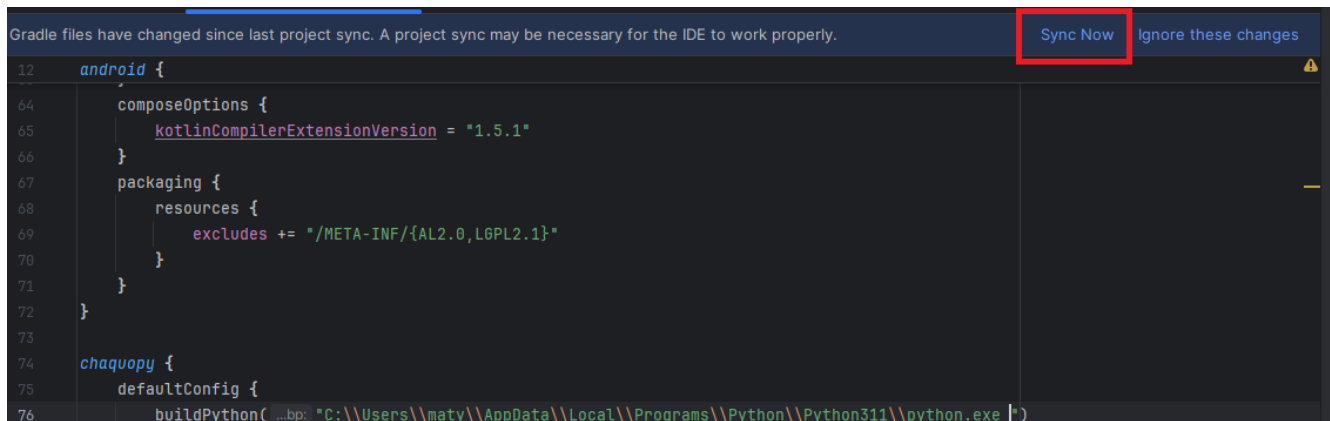
```

En el archivo `local.properties`, se configura la ubicación del SDK de Android necesario para compilar la aplicación. Por defecto, la ubicación estándar en sistemas Windows es:

`C:\Users\[nombre_de_usuario]\AppData\Local\Android\sdk`

Si el SDK se encuentra instalado en un directorio diferente, es necesario actualizar la propiedad `sdk.dir` en el archivo `local.properties` con la ruta correspondiente.

Asegúrese de usar la sintaxis adecuada para las rutas en Windows (dobles barras invertidas \\).



Sincronizamos los cambios en Android Studio para asegurarnos de que todas las configuraciones sean aplicadas. Una vez completado, el entorno está configurado y listo para comenzar con el desarrollo y la programación de la aplicación.

3. Configuración de Visual Studio Code

Para abrir el proyecto de Python con Visual Studio Code, primero acceda a la terminal de Windows y navegue hasta la carpeta del proyecto previamente clonado.

Dentro del proyecto, encontrará dos carpetas principales, además de la carpeta móvil: Clasificador e IA. La carpeta Clasificador contiene el código fuente del algoritmo de entrenamiento MCBIA7, mientras que la carpeta IA incluye scripts destinados a pruebas y evaluaciones del modelo. Para abrir uno de estos proyectos, navegue a la carpeta deseada utilizando el comando `cd [nombre_de_la_carpeta]`, y luego ejecute el comando `code .` para abrir Visual Studio Code directamente en el directorio del proyecto seleccionado.

En el archivo README.md que se encuentra en la carpeta IA, contiene un listado de comandos que son necesarios al momento del desarrollo, que son los siguientes:

3.1. Crear y activar un entorno virtual desde la terminal

```
python -m venv env
source env/bin/activate
```

3.2. Instalar las dependencias

Si ya tienes un archivo requirements.txt, usa el siguiente comando para instalar las dependencias necesarias:

```
pip install -r requirements.txt
```

Si necesitas generar el archivo requirements.txt desde tu entorno actual:

```
pip freeze > requirements.txt
```

3.3. Crear las carpetas de salida

Crear la carpeta donde se guardarán los resultados de los scripts:

```
mkdir src/pesos
```

```
mkdir src/pesos/100
```

O el número del peso: 100_contours, 100_data, 100_canny, 100_resized

3.4. Ejecución del proyecto

Para ejecutar el programa principal:

```
python src/preprocess/resize.py
```

O, si es en Linux:

```
python3 src/preprocess/resize.py
```

3.5. Dependencia qt6

Para utilizar la biblioteca matplotlib se necesitan instalar las siguientes dependencias:

```
sudo apt install qt6-base-dev
```

```
sudo apt install libxcb-cursor0
```

3.6. Otros Comandos

Para salir del entorno virtual:

```
deactivate
```

Puedes listar los paquetes instalados en el entorno actual usando:

```
pip list
```

7. Diseño de la Arquitectura

En esta sección, se explicará la arquitectura del proyecto mediante diagramas de clases UML y diagramas de estados de todas las clases y actividades del proyecto.

7.1. Base de Datos

La clase SQLiteHelper extiende de SQLiteOpenHelper, proporcionando métodos para gestionar la base de datos SQLite que almacena información sobre billetes. Esta base de datos es utilizada para guardar, recuperar, actualizar y eliminar datos de los billetes.

- UML

Atributos

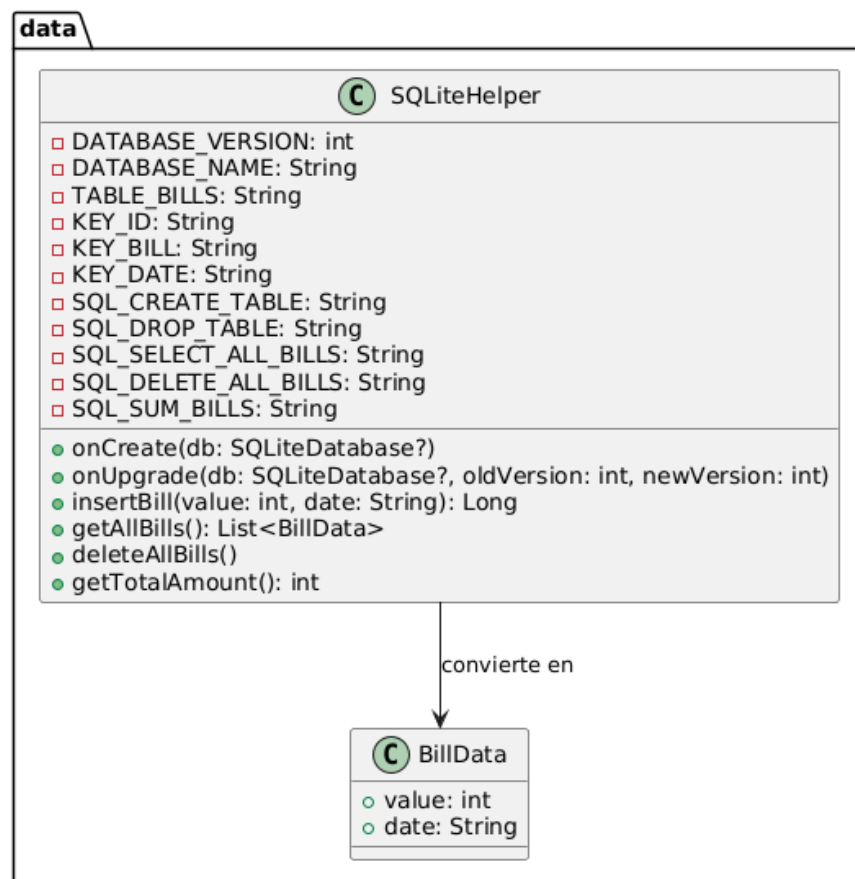
- DATABASE_VERSION: La versión de la base de datos.
- DATABASE_NAME: Nombre del archivo de la base de datos.
- TABLE_BILLS: Nombre de la tabla en la base de datos donde se guardan los billetes.
- KEY_ID, KEY_BILL, KEY_DATE: Nombres de las columnas de la tabla.

Se definen varias consultas como constantes para crear, eliminar, seleccionar y sumar los billetes.

Métodos

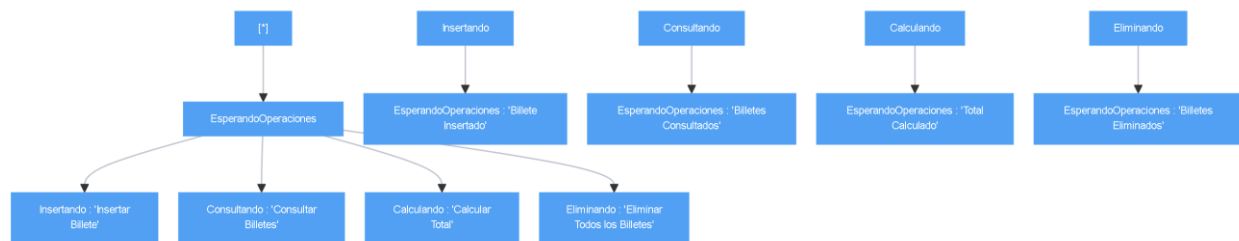
- onCreate: Método para crear la base de datos y la tabla cuando la base de datos se inicializa por primera vez.

- **onUpgrade:** Se ejecuta cuando se cambia la versión de la base de datos. En este caso, elimina la tabla anterior y crea una nueva.
- **insertBill:** Inserta un nuevo billete con su valor y fecha en la base de datos.
- **getAllBills:** Recupera todos los billetes almacenados en la base de datos y los devuelve como una lista de objetos BillData.
- **deleteAllBills:** Elimina todos los registros de billetes de la base de datos.
- **getTotalAmount:** Calcula y devuelve la suma de los valores de todos los billetes almacenados.
- **android.database.Cursor.toBillData:** Método auxiliar para convertir cada fila del resultado de la consulta SQL a un objeto BillData.



- Diagrama de Estados

- **EsperandoOperaciones:** Estado inicial donde la base de datos está esperando una operación.
- **Insertando:** Cuando se inserta un nuevo billete en la base de datos. Después de la inserción, el sistema regresa al estado "EsperandoOperaciones".
- **Consultando:** Cuando se consultan todos los billetes en la base de datos. Al finalizar la consulta, el sistema regresa al estado "EsperandoOperaciones".
- **Calculando:** Cuando se calcula la suma total de los billetes. Después de calcular el total, el sistema regresa al estado "EsperandoOperaciones".
- **Eliminando:** Cuando se eliminan todos los billetes de la base de datos. Al finalizar esta operación, el sistema regresa al estado "EsperandoOperaciones".



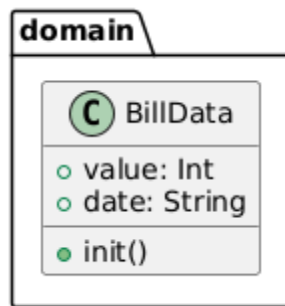
7.2. Billeto

La clase BillData es una clase de datos simple que representa los datos de un billete reconocido por la aplicación.

- UML

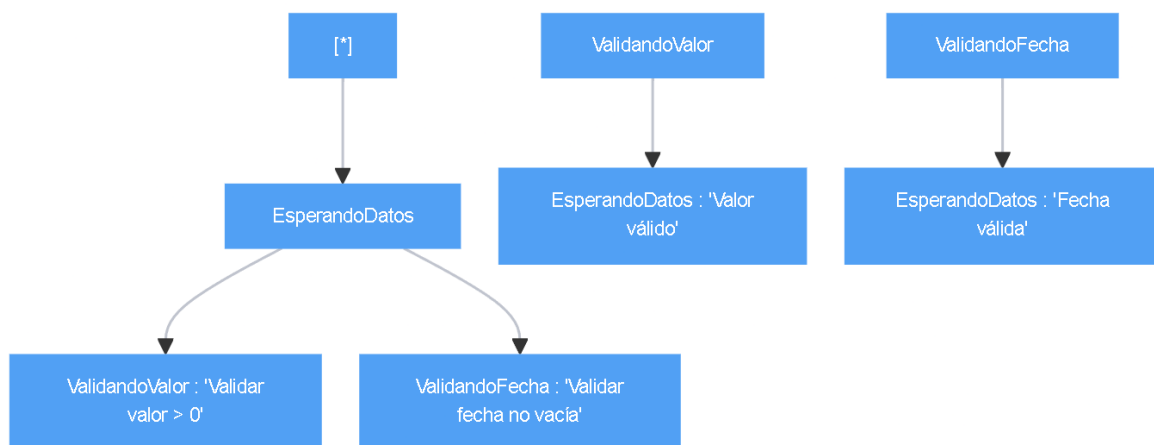
Atributos

- **value:** El valor monetario del billete, representado como un entero.
- **date:** La fecha en que se reconoció el billete, almacenada como una cadena.



- Diagrama de Estados

- **EsperandoDatos:** Este es el estado inicial cuando se crea un objeto **BillData**. El sistema está esperando que se le proporcionen los datos del billete (valor y fecha).
- **ValidandoValor:** Después de recibir los datos, el sistema valida que el valor sea mayor que cero.
- **ValidandoFecha:** Luego, el sistema valida que la fecha no esté vacía.
- **Transiciones:** Si ambos valores son válidos, el sistema regresa al estado **EsperandoDatos**, lo que significa que el objeto **BillData** se ha creado exitosamente.



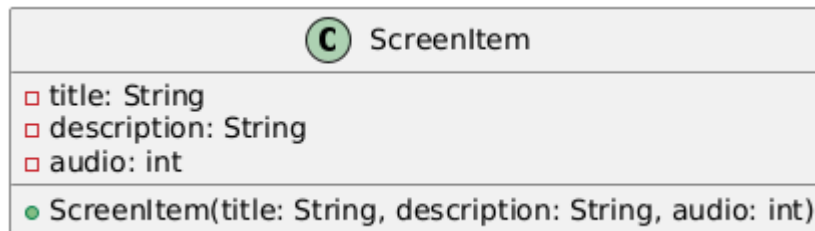
7.3. Item de cada Billeto

La clase ScreenItem representa un elemento de la pantalla en el tutorial de la aplicación.

- UML

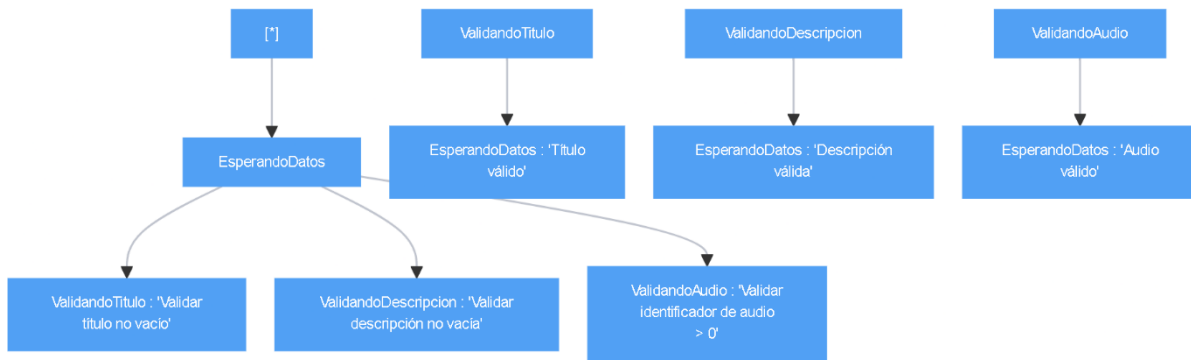
Atributos

- title: El título del elemento de la pantalla, representado como una cadena de texto.
- description: Una breve descripción del elemento de la pantalla, también representada como una cadena de texto.
- audio: El identificador de recurso de audio asociado a la pantalla, representado como un número entero.



- Diagrama de Estados

- EsperandoDatos: Este es el estado inicial cuando se crea un objeto ScreenItem. El sistema está esperando los datos del título, la descripción y el identificador de audio.
- ValidandoTitulo: El sistema valida que el título no sea vacío.
- ValidandoDescripcion: Luego, valida que la descripción no sea vacía.
- ValidandoAudio: Después, valida que el identificador de audio sea un número positivo.
- Transiciones: Si todas las validaciones son exitosas, el sistema regresa al estado EsperandoDatos, lo que indica que el objeto ScreenItem se ha creado correctamente.



7.4. Adaptador de interfaz para los Billetes

La clase BillsAdapter es un adaptador para mostrar una lista de billetes en un RecyclerView. Este adaptador maneja la visualización de los datos de tipo BillData en la interfaz de usuario.

- UML

Atributos

- bills: Es una lista de BillData donde contiene la lista inicial de billetes escaneados.

Métodos

- ViewHolder: La clase BillViewHolder contiene referencias a los TextViews en la vista de cada ítem del RecyclerView, que muestran el valor del billete y la fecha asociada.
- onCreateViewHolder: Este método infla el diseño del ítem (un archivo XML) cuando se necesita un nuevo ViewHolder para mostrar un billete. Esto optimiza el rendimiento de la vista, ya que reutiliza las vistas según sea necesario.
- onBindViewHolder: Este método es responsable de vincular los datos del objeto BillData a los TextViews correspondientes. Los valores del billete son formateados como moneda y la fecha es mostrada en el formato adecuado.
- getItemCount: Devuelve el número total de elementos (billetes) en la lista.

- **updateBills:** Este método utiliza DiffUtil para calcular las diferencias entre la lista actual y la nueva lista de billetes. Esto mejora la eficiencia al actualizar solo los elementos que han cambiado en el RecyclerView.
- **formatCurrency:** Este método formatea un valor monetario (representado como un Int) según la configuración regional del dispositivo, convirtiéndolo en un formato adecuado de moneda.

Clase Interna

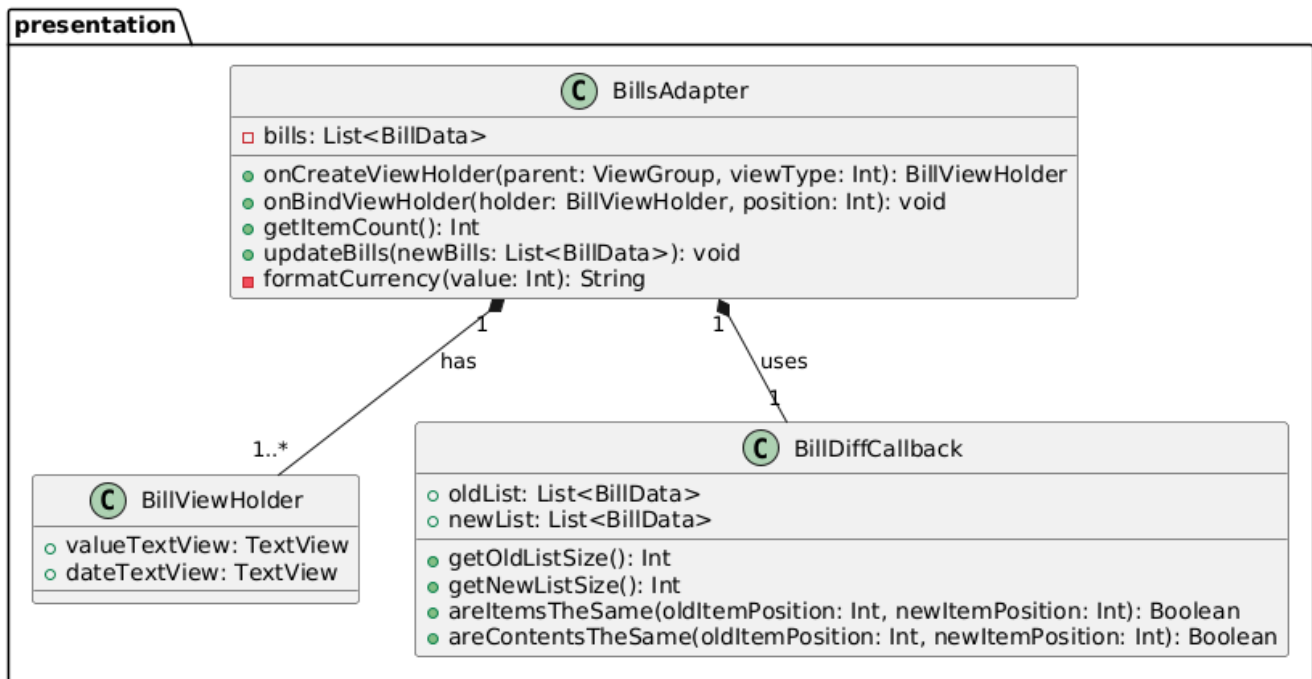
BillDiffCallback: Esta clase es usada por DiffUtil para comparar dos listas de BillData y determinar qué elementos han cambiado, agregado o eliminado. Esto permite actualizaciones más eficientes en el RecyclerView.

Atributos

- **oldList:** Contiene la posición del ítem en la lista antigua.
- **newList:** Contiene la posición del ítem en la lista nueva.

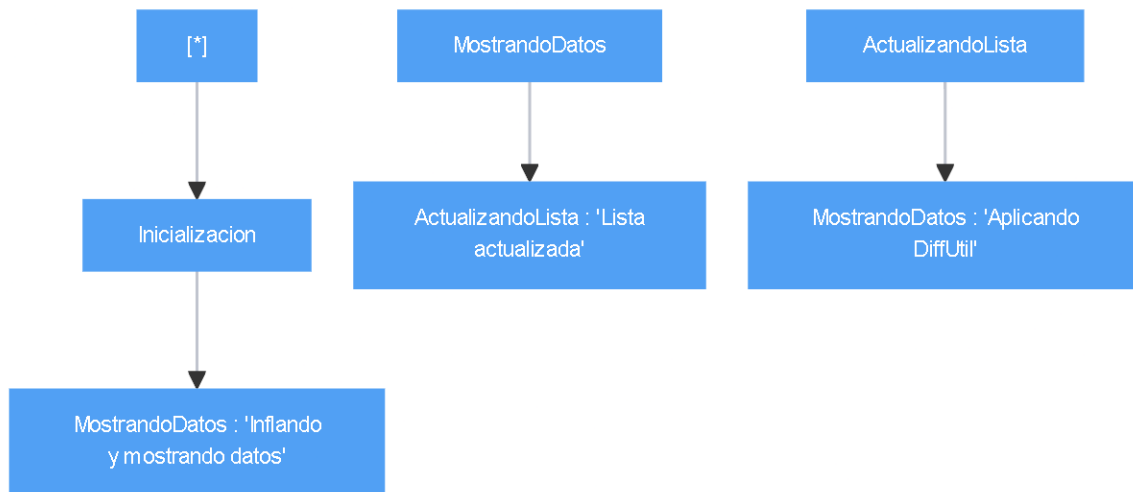
Metodos

- **areItemsTheSame:** Compara si dos elementos de la lista son el mismo objeto. En este caso, se considera que los elementos son los mismos si tienen la misma fecha. Esto se usa para determinar si un billete en la lista vieja corresponde al mismo billete en la nueva lista.
- **areContentsTheSame:** Compara si los contenidos de dos elementos son los mismos.



- Diagrama de Estados

- **Inicializacion:** El adaptador comienza en este estado cuando se crea. El RecyclerView aún no ha mostrado ningún dato.
- **MostrandoDatos:** Este es el estado en el que el BillsAdapter está mostrando la lista de billetes en la interfaz de usuario. Los billetes se muestran a través de los ViewHolders que han sido inflados.
- **ActualizandoLista:** Cuando la lista de billetes se actualiza (usando el método `updateBills`), el adaptador entra en este estado. Aquí, DiffUtil calcula las diferencias y solo actualiza los elementos que han cambiado, mejorando el rendimiento.



7.5. Adaptador Gráfico del Tutorial.

- UML

La clase IntroViewPagerAdapter es un adaptador personalizado para el ViewPager en una aplicación Android. Este adaptador maneja la visualización de las pantallas introductorias, donde cada pantalla muestra un título, una descripción y un recurso de audio asociado. La clase utiliza un ViewPager para navegar entre estas pantallas de forma interactiva.

Atributos

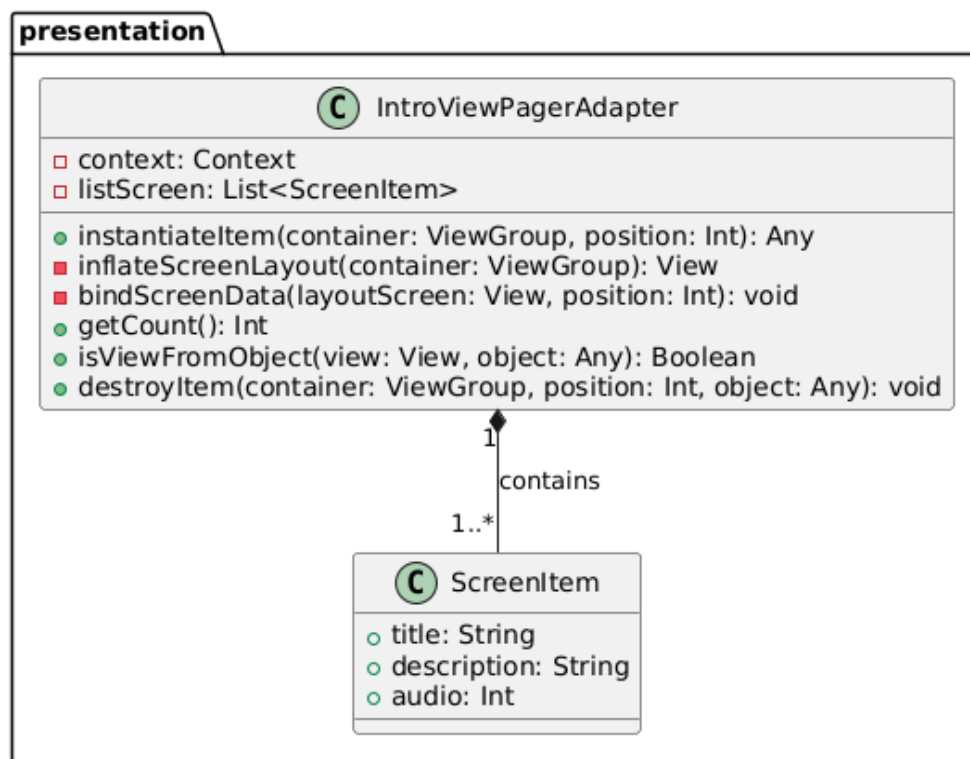
- listScreen: Lista de elementos de tipo ScreenItem para mostrar en las pantallas.

Métodos

- Constructor: El constructor recibe un Context y una lista de objetos ScreenItem, que contiene los datos que se mostrarán en cada pantalla (título, descripción y recurso de audio).
- instantiateltem: Este método es responsable de crear la vista de cada ítem del ViewPager. Para cada pantalla (ítem) en el ViewPager, el adaptador infla una vista,

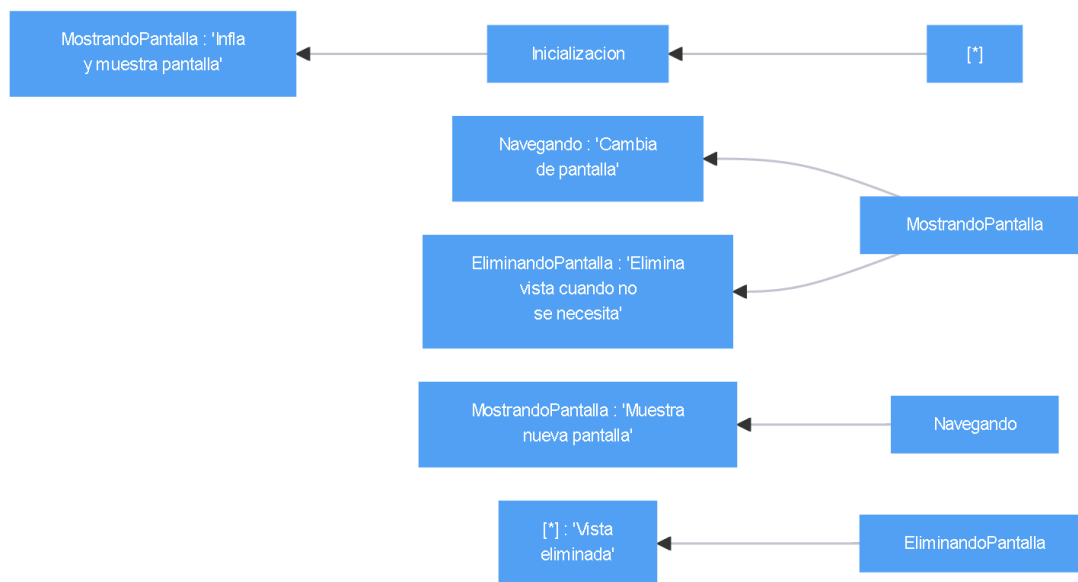
asocia los datos correspondientes (título y descripción) y la agrega al contenedor de vistas (container).

- **inflateScreenLayout:** Este método infla la vista de la pantalla utilizando el `LayoutInflater`. La vista inflada es el diseño XML que contiene los elementos visuales de cada pantalla, como el título y la descripción.
- **bindScreenData:** Este método se encarga de asignar los datos del `ScreenItem` a las vistas correspondientes en el diseño de la pantalla (como el `TextView` para el título y la descripción).
- **getCount:** Este método devuelve el número total de pantallas que debe mostrar el `ViewPager`, es decir, el tamaño de la lista `listScreen`.
- **isViewFromObject:** Este método se usa para comprobar si una vista pertenece a un objeto específico. Se asegura de que el `ViewPager` reconozca correctamente las vistas en las que se encuentra.
- **destroyItem:** Este método elimina las vistas que ya no se necesitan del contenedor, optimizando así el uso de memoria.



- Diagrama de Estados

- **Inicializacion:** El adaptador comienza en este estado cuando se crea y se configura con el contexto y la lista de pantallas.
- **MostrandoPantalla:** En este estado, el adaptador infla y muestra una pantalla del ViewPager. Se asignan los datos (título, descripción) y la pantalla se presenta al usuario.
- **Navegando:** Cuando el usuario navega a una nueva pantalla del ViewPager, el adaptador se mueve a este estado y muestra la siguiente pantalla.
- **EliminandoPantalla:** Este estado se alcanza cuando una vista del ViewPager ya no es necesaria (por ejemplo, si el usuario navega fuera de ella). El adaptador elimina la vista del contenedor para optimizar el uso de memoria.



7.6. Adaptador para los Modelos de Clasificación

La clase `ModelListAdapter` es un adaptador para un `RecyclerView` en una aplicación Android que muestra una lista de modelos. Cada modelo es un `String`, y el adaptador maneja la visualización de los modelos, permitiendo que el usuario seleccione un modelo de la lista.

Cuando un modelo es seleccionado, se ejecuta un callback con la información del modelo seleccionado.

- UML

Métodos:

- Constructor: El constructor recibe una lista de String que contiene los modelos a mostrar y un callback que se ejecuta cuando un modelo es seleccionado, y pasa el modelo y su posición como parámetros.
- onCreateViewHolder: Este método infla el diseño del item (R.layout.item_model) para cada elemento del RecyclerView y devuelve un nuevo ViewHolder.
- onBindViewHolder: Asocia los datos (nombre del modelo) con el ViewHolder en función de la posición en la lista de modelos.
- getItemCount: Retorna el número total de modelos en la lista, indicando cuántos elementos deben ser mostrados en el RecyclerView.
- setSelectedPosition: Este método permite establecer un modelo como seleccionado. Cambia el color o el fondo del modelo seleccionado y notifica al adaptador que el ítem ha cambiado para actualizar la vista.

Clase Interna

ViewHolder: Mantiene las vistas de cada item dentro del RecyclerView. Contiene una referencia a modelName (nombre del modelo) y selectionIndicator (indicador visual de selección).

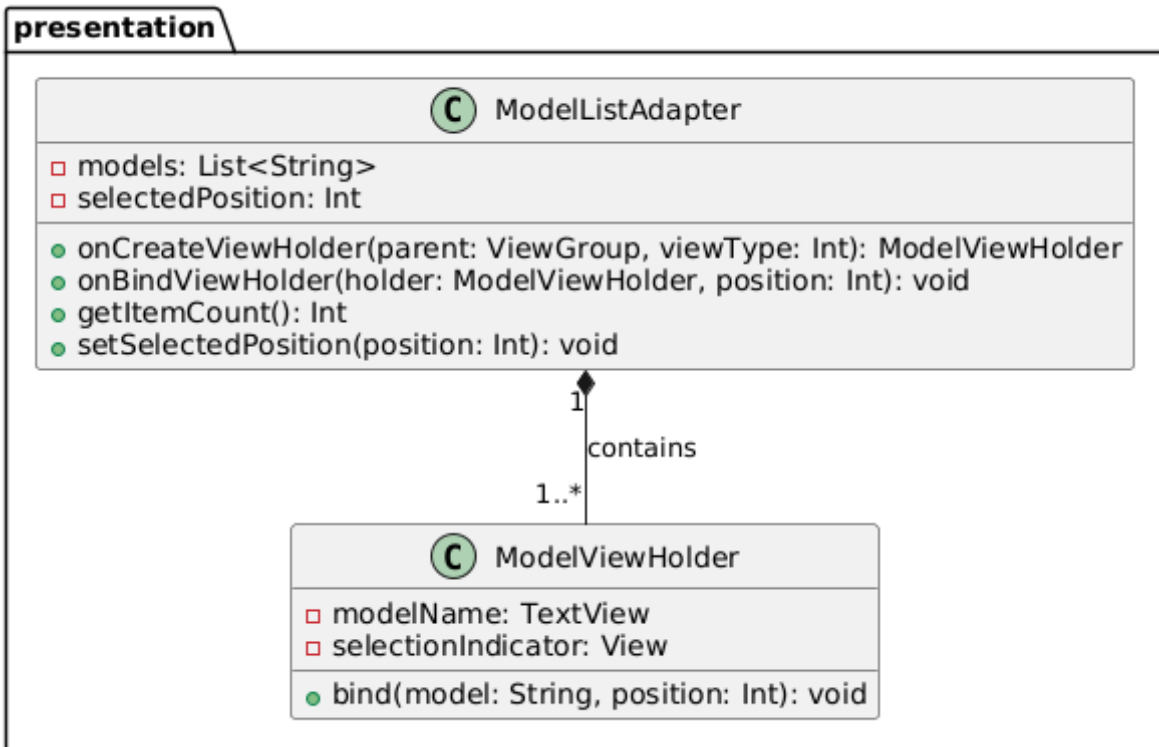
Atributos

- modelName: El nombre del modelo clasificador.
- selectionIndicator: El indicador del modelo seleccionado.

Métodos

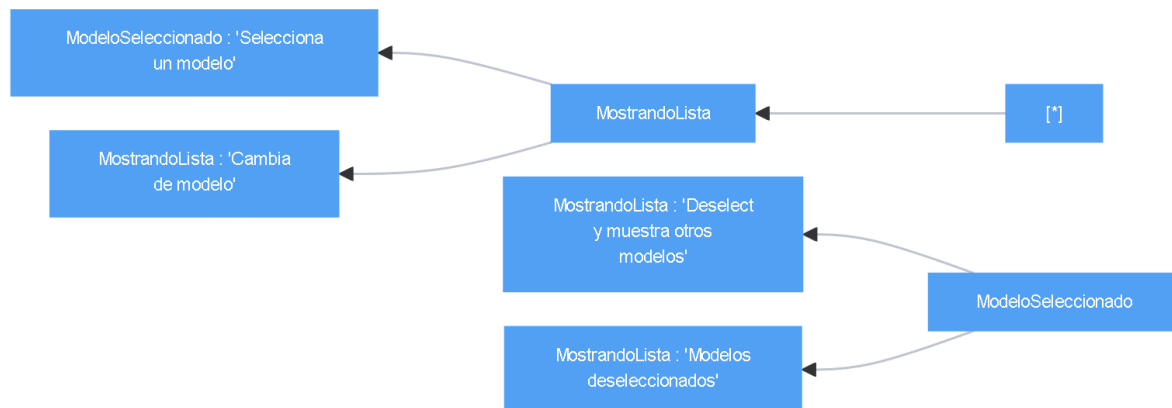
- bind: Establece el nombre del modelo y cambia el color del indicador de selección según si el modelo está seleccionado o no. Además, maneja el evento de clic, y

cuando el ítem es clicado, se actualiza la selección y se ejecuta el callback `onModelSelected`.



- Diagrama de Estados

- **MostrandoLista:** Este es el estado inicial, donde el RecyclerView muestra la lista de modelos. El usuario puede seleccionar un modelo haciendo clic en él.
- **ModeloSeleccionado:** Este estado representa cuando un modelo ha sido seleccionado por el usuario. El fondo del modelo seleccionado cambia visualmente, y se actualiza la posición seleccionada.
- **Transition entre estados:** Al seleccionar un modelo, se pasa del estado de "MostrandoLista" a "ModeloSeleccionado". Cuando un modelo es deseleccionado (por ejemplo, cuando se hace clic en otro modelo), el estado vuelve a "MostrandoLista". La transición refleja la interacción del usuario con la lista de modelos y la actualización visual de la selección.



7.7. Configuración

ConfigActivity es una actividad de Android que permite a los usuarios seleccionar un modelo de una lista en un RecyclerView, navegar mediante gestos en la pantalla, y escuchar sonidos relacionados con las selecciones.

- UML

Atributos

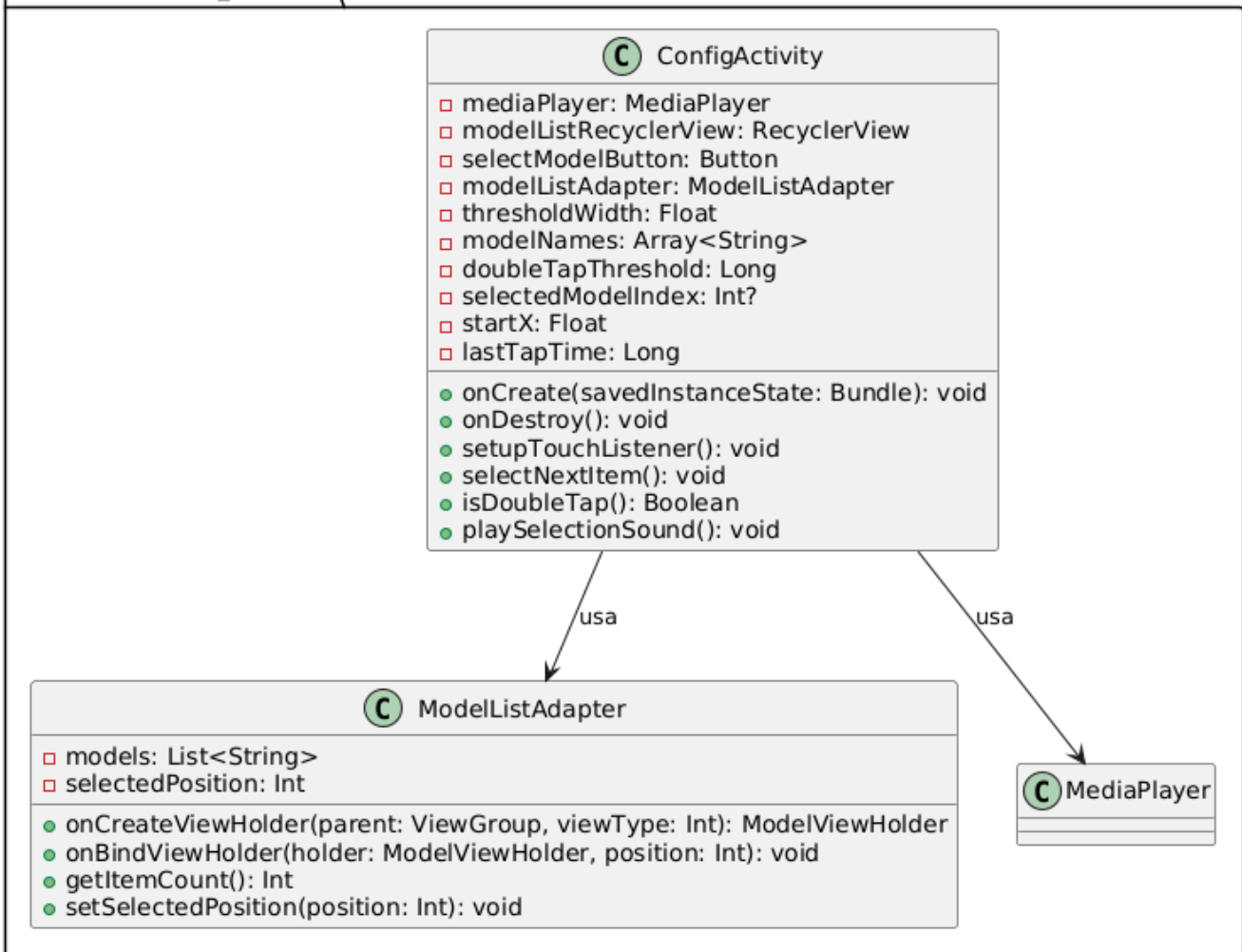
- mediaPlayer: Reproduce sonidos relacionados con las interacciones del usuario.
- modelListRecyclerView: Un RecyclerView que muestra una lista de modelos.
- selectModelButton: Un botón que confirma la selección y navega a la siguiente actividad.
- modelListAdapter: Adaptador que gestiona la lista de modelos en el RecyclerView.
- thresholdWidth: Determina el umbral para reconocer gestos de deslizamiento horizontal.
- modelNames: Lista de nombres de modelos cargados desde recursos.
- doubleTapThreshold: Tiempo límite para reconocer un doble toque.

- `selectedModelIndex`: Índice del modelo actualmente seleccionado.
- `startX`: Coordenada inicial del toque en el eje X.
- `lastTapTime`: Marca temporal del último toque en pantalla.

Métodos

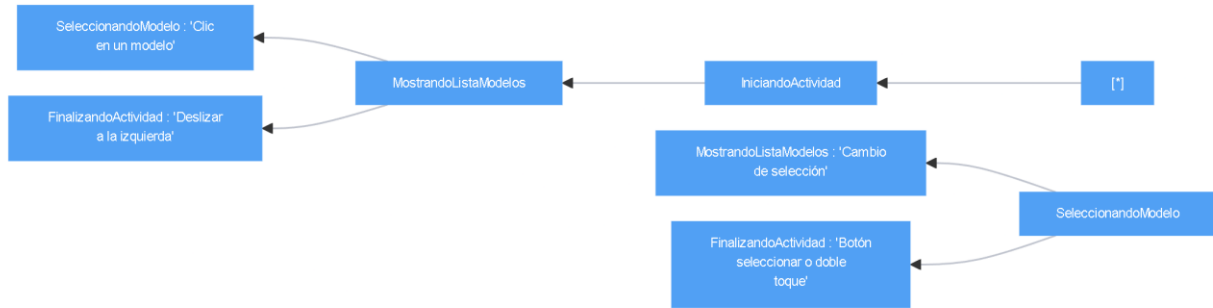
- `onCreate`: Configura la interfaz de usuario y el reproductor de audio inicial. Configura el RecyclerView y el adaptador de modelos. Recupera el índice del modelo previamente seleccionado (si existe). Configura un OnClickListener para el botón de selección de modelos.
- `onDestroy`: Libera los recursos asociados al MediaPlayer al destruir la actividad.
- `setupTouchListener`: Configura un listener táctil que detecta gestos de deslizamiento y doble toque. Deslizamiento hacia la derecha: selecciona el siguiente modelo. Deslizamiento hacia la izquierda: cierra la actividad. Doble toque: navega a la siguiente pantalla con el modelo seleccionado.
- `selectNextItem`: Cambia la selección al siguiente modelo en la lista. Si se llega al final, vuelve al primer modelo.
- `isDoubleTap`: Determina si el usuario realizó un doble toque en la pantalla.
- `playSelectionSound`: Reproduce un sonido asociado al modelo seleccionado.

reconocimiento_billetes



- Diagrama de Estados

- **IniciandoActividad:** La actividad se inicializa y configura los elementos visuales y funcionales.
- **MostrandoListaModelos:** El RecyclerView muestra la lista de modelos disponibles para seleccionar.
- **SeleccionandoModelo:** El usuario selecciona un modelo mediante clic, actualizando la posición seleccionada y reproduciendo un sonido.
- **FinalizandoActividad:** La actividad se cierra, ya sea por deslizamiento hacia la izquierda o al confirmar la selección.



7.8. Historial de Billetes

La clase CountBillActivity es una actividad de Android diseñada para gestionar el historial de billetes contados. Ofrece funcionalidades para visualizar el historial, eliminarlo, compartirlo mediante un archivo de texto, y manejar interacciones como toques rápidos y gestos de deslizamiento.

- UML

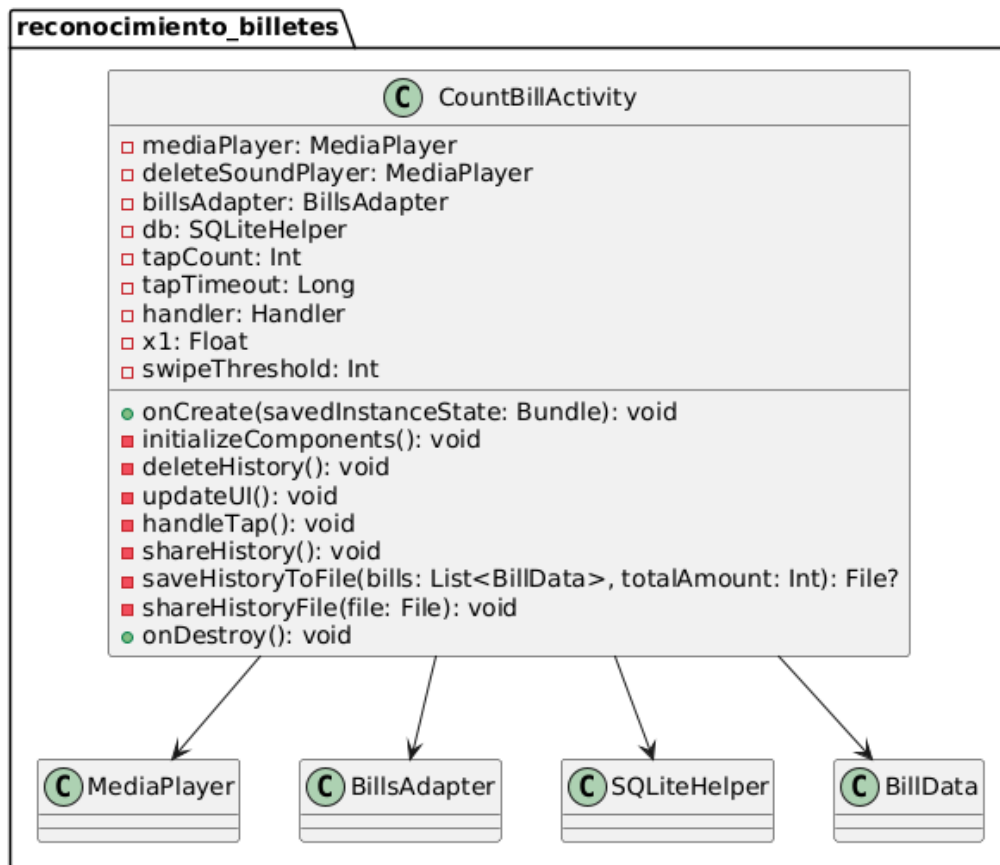
Atributos

- mediaPlayer y deleteSoundPlayer: Controlan la reproducción de sonidos al iniciar la actividad y al eliminar el historial.
- billsAdapter: Adaptador del RecyclerView que muestra la lista de billetes contados.
- db: Instancia de SQLiteHelper para manejar la base de datos local.
- tapCount, tapTimeout y handler: Controlan los toques rápidos para activar ciertas funcionalidades, como borrar el historial tras 5 toques rápidos.
- x1 y swipeThreshold: Controlan la detección de gestos de deslizamiento horizontal.

Métodos

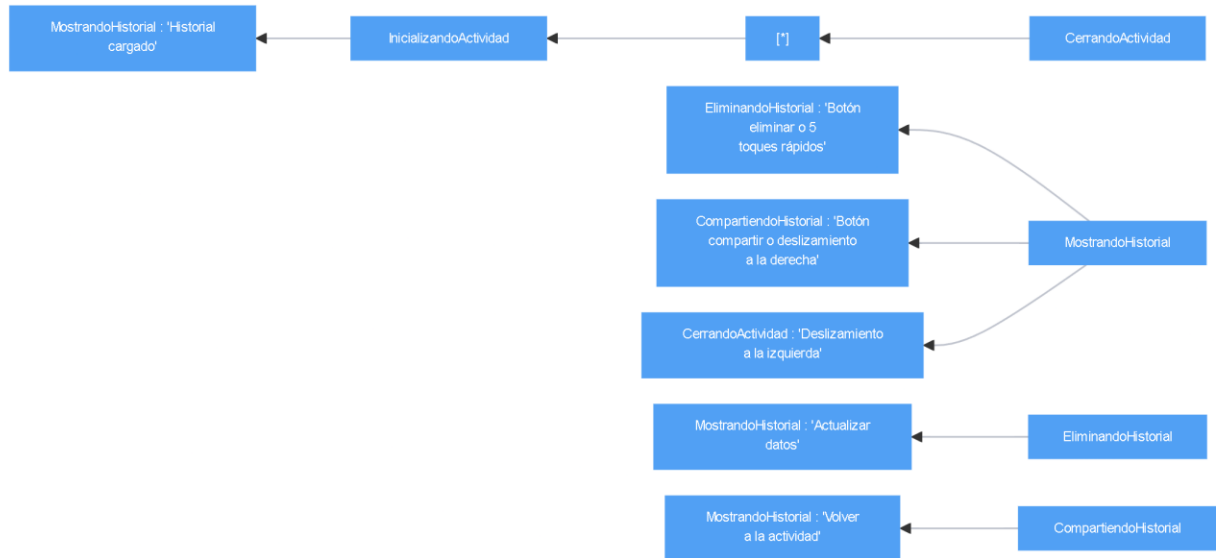
- onCreate: Configura la interfaz de usuario y eventos táctiles. Inicializa la base de datos, los reproductores de sonido, y el adaptador del RecyclerView. Carga los datos iniciales desde la base de datos.

- `initializeComponents`: Configura el reproductor de audio y establece la conexión con la base de datos.
- `deleteHistory`: Borra el historial de billetes de la base de datos y actualiza la interfaz gráfica.
- `updateUI`: Refresca los datos en el RecyclerView y actualiza el total mostrado en pantalla.
- `handleTap`: Detecta si se han realizado 5 toques rápidos para eliminar el historial.
- `shareHistory`: Guarda el historial en un archivo de texto y lo comparte mediante un Intent.
- `saveHistoryToFile`: Genera un archivo de texto con el historial de billetes, incluyendo sus valores y fechas.
- `shareHistoryFile`: Configura y lanza un Intent para compartir el archivo de texto generado.
- `onDestroy`: Libera los recursos utilizados por los reproductores de sonido y el Handler.



- Diagrama de Estados

- Visualizar el historial de billetes: Muestra los datos almacenados en una base de datos SQLite mediante un RecyclerView.
- Eliminar el historial: Ocurre al presionar un botón o realizar 5 toques rápidos consecutivos.
- Compartir el historial: Genera un archivo de texto con los datos del historial y lo comparte mediante un Intent.
- Interacción táctil avanzada: Responde a gestos de deslizamiento y toques múltiples, mejorando la experiencia del usuario.



7.9. Introducción de la Aplicación

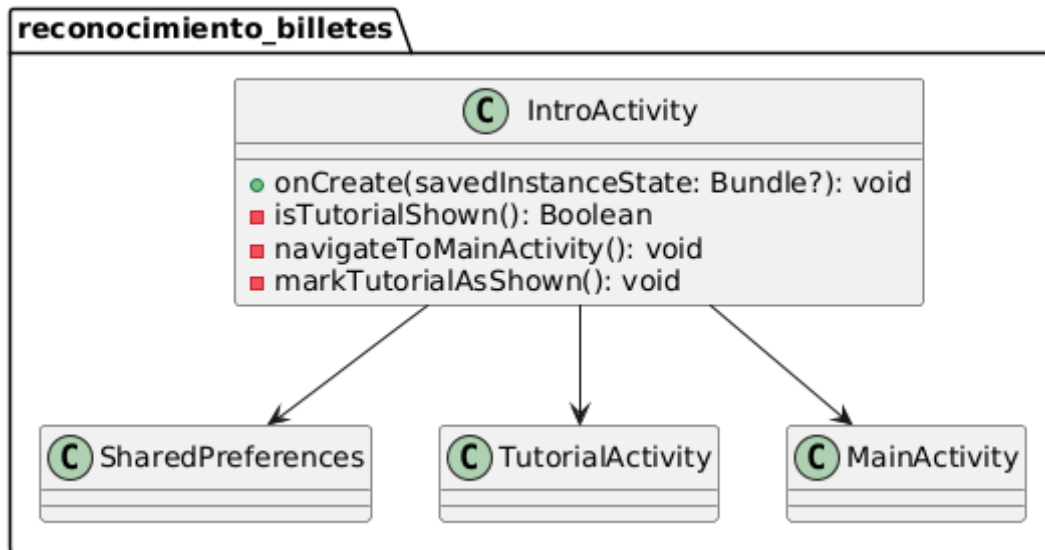
La clase IntroActivity es la actividad inicial de una aplicación de Android que implementa una pantalla de bienvenida o Splash Screen. Su principal propósito es determinar si el usuario ya ha visto el tutorial inicial y, en función de ello, redirigirlo a la actividad correspondiente: el tutorial o la actividad principal.

- UML

Métodos

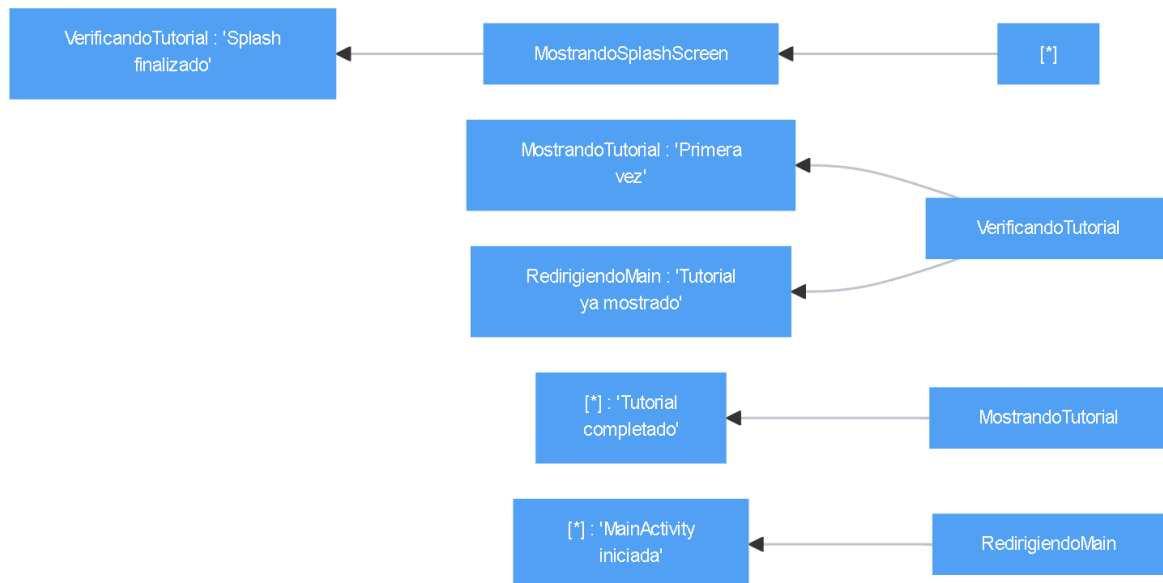
- **onCreate:** Se ejecuta al iniciar la actividad. Muestra el Splash Screen. Verifica si el tutorial ha sido mostrado previamente. Finaliza la actividad después de la navegación.
- **isTutorialShown:** Verifica en SharedPreferences si el tutorial ha sido mostrado en sesiones anteriores. Devuelve true si la clave isIntroOpened está configurada como true, o false en caso contrario.
- **navigateToMainActivity:** Inicia la actividad principal (MainActivity) y finaliza la actividad actual.

- `markTutorialAsShown`: Guarda en `SharedPreferences` que el tutorial ha sido mostrado, configurando la clave `isIntroOpened` como `true`.



- Diagrama de Estados

- **Mostrar Splash Screen**: Utiliza la nueva API de `SplashScreen` para mostrar una pantalla de introducción de corta duración.
- **Verificar el estado del tutorial**: Consulta `SharedPreferences` para determinar si el usuario ya completó el tutorial en sesiones anteriores.
- **Redirigir al usuario**: Si es la primera vez que abre la aplicación, inicia la actividad del tutorial. Si ya lo completó, redirige directamente a la actividad principal.
- **Persistir el estado**: Usa `SharedPreferences` para guardar el estado de si el tutorial ha sido mostrado.



7.10. Menú Principal

La clase MainActivity es la actividad principal de una aplicación que muestra un menú con opciones para navegar hacia diferentes funciones. Utiliza botones y gestos táctiles para redirigir al usuario a otras actividades. También implementa audio en segundo plano que se reproduce cuando la actividad está en primer plano.

- UML

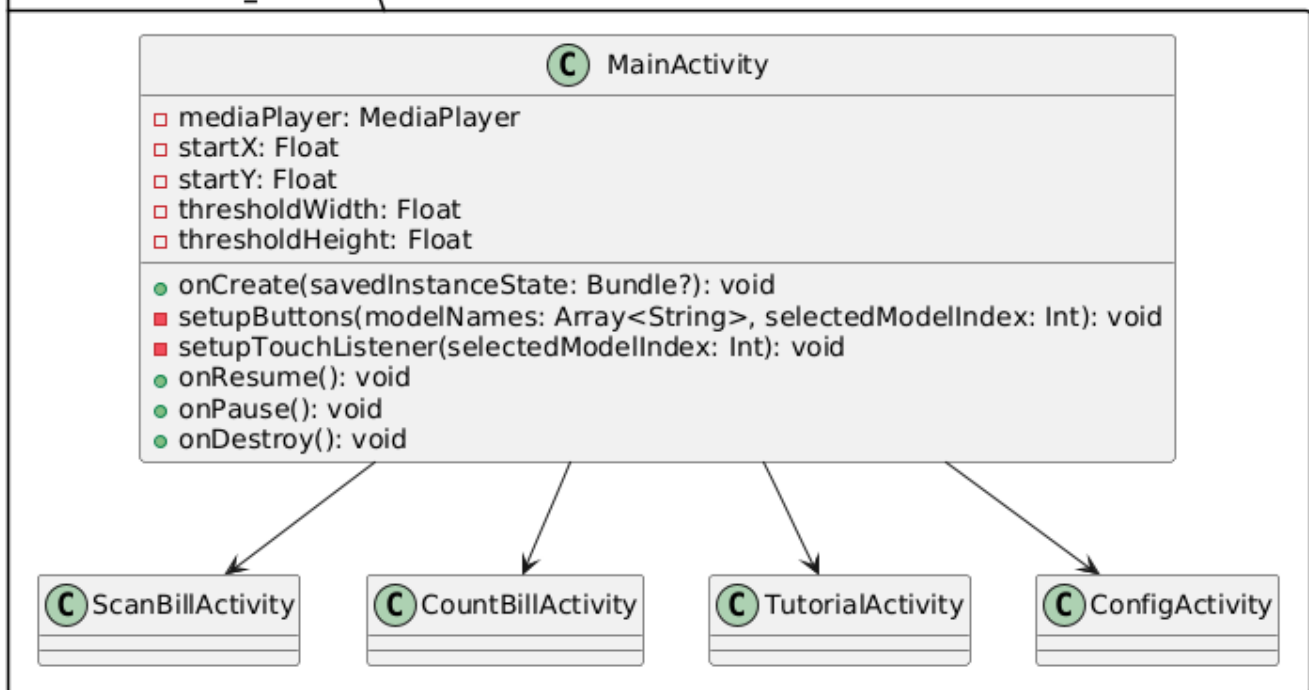
Atributos

- mediaPlayer: Controla la reproducción de un archivo de audio en el menú principal.
- startX y startY: Coordenadas iniciales del toque en la pantalla, utilizadas para detectar deslizamientos.
- thresholdWidth y thresholdHeight: Umbrales que determinan si un gesto táctil es considerado un deslizamiento horizontal o vertical.

Métodos

- onCreate: Configura la UI, inicializa el audio, y configura los botones y los detectores de deslizamientos.
- setupButtons: Configura las acciones de los botones.
- setupTouchListener: Detecta deslizamientos en las direcciones horizontal y vertical.
- onResume: Reproduce el audio cuando la actividad vuelve al primer plano.
- onPause: Pausa el audio si está en reproducción.
- onDestroy: Libera los recursos utilizados por el reproductor de audio.

reconocimiento_billetes

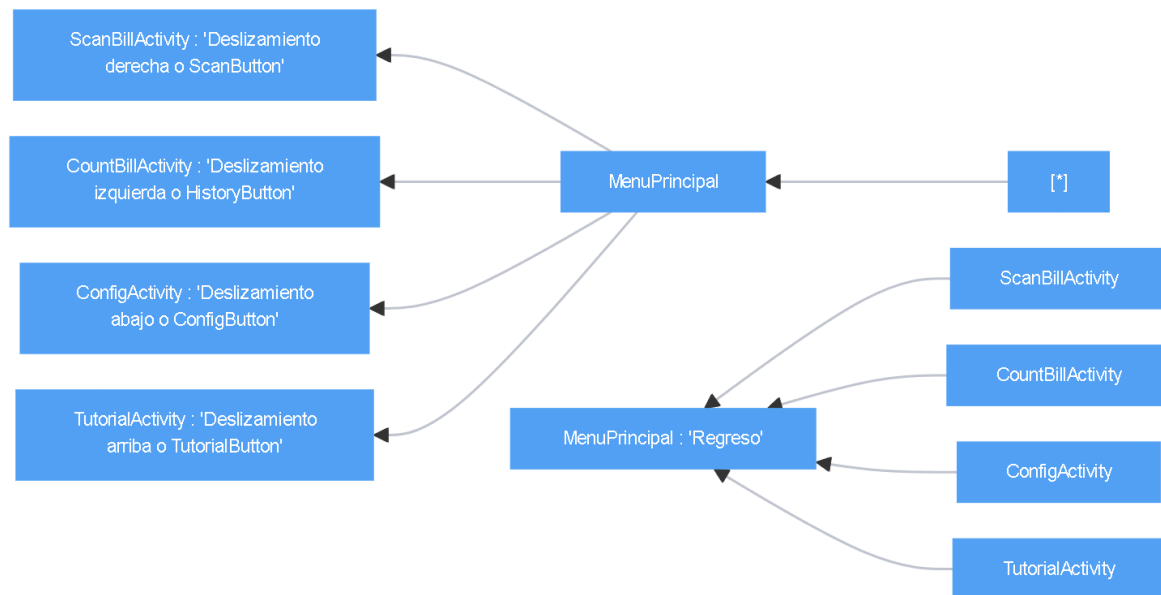


- Diagrama de Estados

Los estados de MainActivity son los puntos de entrada para las demás actividades:

- Escanear Billetes (ScanBillActivity).
- Consultar el Historial (CountBillActivity).

- Ver el Tutorial (TutorialActivity).
- Configurar la Aplicación (ConfigActivity).



7.11. Escaneo de Billetes

La clase **ScanBillActivity** está diseñada para escanear y clasificar billetes utilizando un modelo de aprendizaje automático. La funcionalidad principal incluye capturar imágenes desde la cámara, procesarlas mediante un modelo TensorFlow Lite, y devolver el resultado al usuario mediante diálogos y sonidos.

- UML

Atributos

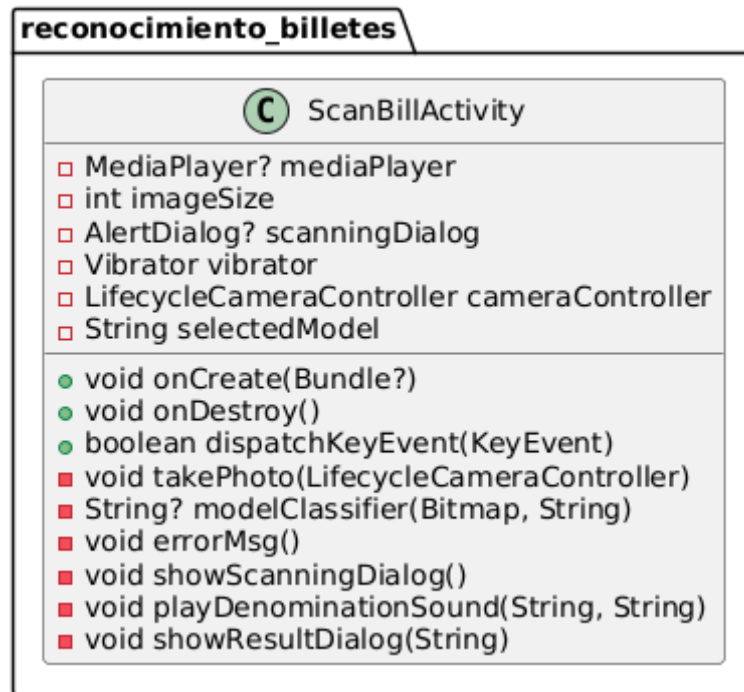
- **mediaPlayer**: Reproduce sonidos para mejorar la experiencia del usuario, como notificaciones y resultados de clasificación.

- **imageSize:** Dimensión en píxeles a la que se escala cada imagen capturada para cumplir con los requisitos del modelo de aprendizaje automático (224x224 píxeles).
- **scanningDialog:** Diálogo que se muestra mientras se procesa una imagen.
- **vibrator:** Proporciona retroalimentación háptica para confirmar interacciones.
- **cameraController:** Controla las funciones de la cámara, como capturar imágenes y encender el flash.
- **selectedModel:** Nombre del modelo de aprendizaje automático seleccionado por el usuario para clasificar las imágenes.

Métodos

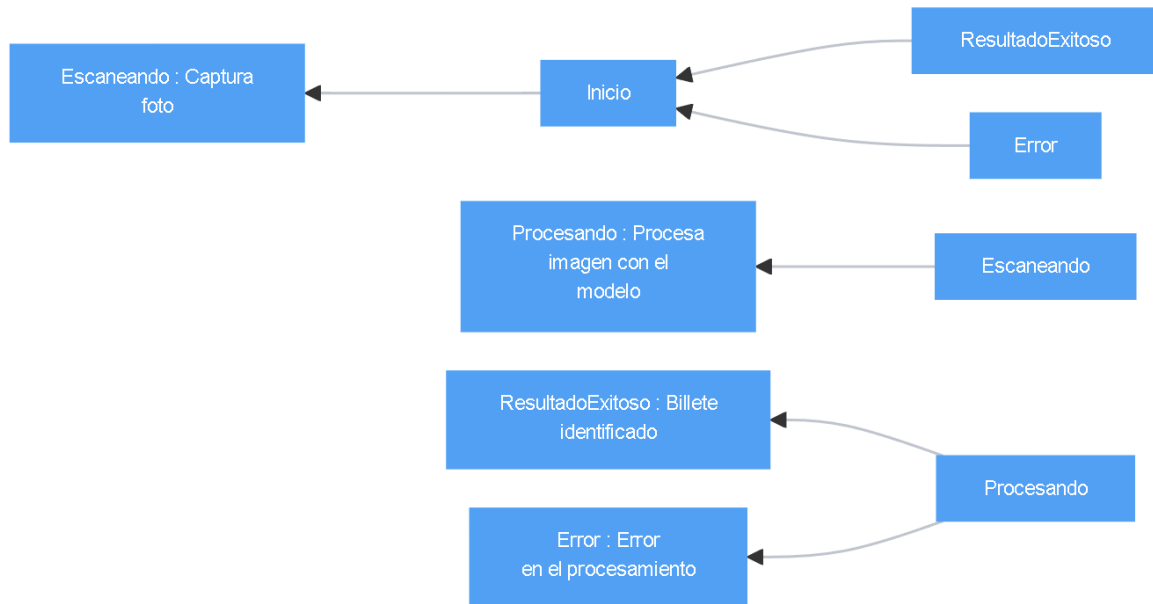
- **onCreate:** Inicializa la actividad y sus componentes, incluyendo: Configuración de la cámara. Reproducción de audio introductorio. Configuración del control táctil para salir de la actividad.
- **onDestroy:** Libera recursos como MediaPlayer y el controlador de la cámara para evitar fugas de memoria.
- **dispatchKeyEvent:** Captura eventos de teclas físicas (como los botones de volumen). Al presionar el botón de bajar volumen, se toma una foto.
- **takePhoto:** Captura una foto, la procesa, y la clasifica usando el modelo seleccionado.
- **Escala y rota la imagen para ajustarla a las especificaciones del modelo.**
- **modelClassifier:** Realiza la clasificación de una imagen: Carga un modelo TensorFlow Lite. Preprocesa la imagen para generar un tensor. Devuelve el nombre de la clase (denominación del billete) con la mayor confianza.
- **errorMsg:** Muestra un mensaje de error si el proceso de clasificación falla.
- **showScanningDialog:** Muestra un diálogo y reproduce un sonido mientras se procesa una imagen.
- **playDenominationSound:** Reproduce un sonido correspondiente a la denominación del billete detectado.

- `showResultDialog`: Muestra un diálogo con el resultado de la clasificación (denominación del billete).



- Diagrama de Estados

- **Inicio:** La actividad comienza configurando los componentes necesarios, incluyendo la cámara y el reproductor de audio.
- **Escaneando:** El usuario toma una foto de un billete, que se escala y rota según sea necesario.
- **Procesando:** La imagen capturada se pasa al modelo de aprendizaje automático para su clasificación.
- **ResultadoExitoso:** Si el modelo identifica correctamente el billete, se muestra un diálogo con el resultado y se reproduce un sonido.
- **Error:** Si la clasificación falla, se muestra un mensaje de error.



7.12. Tutorial

La clase TutorialActivity implementa un tutorial interactivo para una aplicación de reconocimiento de billetes. El tutorial está compuesto por varias pantallas que los usuarios pueden navegar. Cada pantalla incluye texto, audio y botones interactivos.

- UML

Atributos

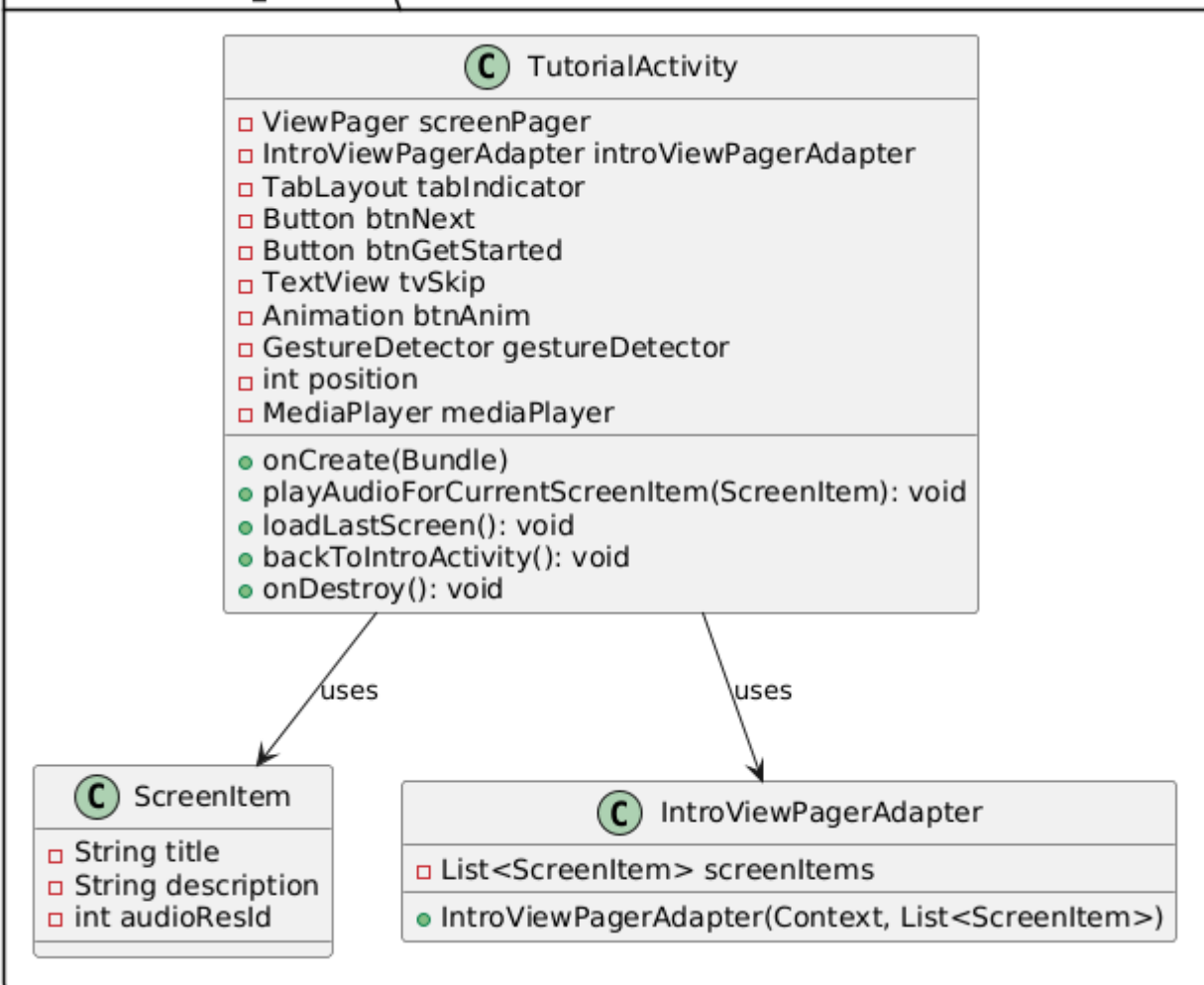
- screenPager: Controlador de vistas para la navegación de pantallas.
- introViewPagerAdapter: Adaptador para gestionar las pantallas.
- tabIndicator: Indicador de pestañas.
- btnNext, btnGetStarted: Botones para avanzar o finalizar el tutorial.
- tvSkip: Texto para omitir el tutorial.

- `btnAnim`: Animación para el botón de inicio.
- `position`: Índice de la pantalla actual.
- `mediaPlayer`: Reproductor de audio.
- `gestureDetector`: Gestor de gestos para interacciones avanzadas.
- `mList`: Lista de objetos `ScreenItem`, que contienen los datos del tutorial (título, descripción, audio asociado).

Métodos

- `onCreate`: Inicializa la actividad y configura los elementos de la interfaz.
- `playAudioForCurrentScreenItem`: Reproduce el audio correspondiente a la pantalla actual.
- `loadLastScreen`: Configura los elementos visuales al llegar a la última pantalla.
- `backToIntroActivity`: Finaliza el tutorial y redirige al usuario a la actividad principal.
- `onDestroy`: Libera los recursos asociados al `MediaPlayer` para evitar fugas de memoria.

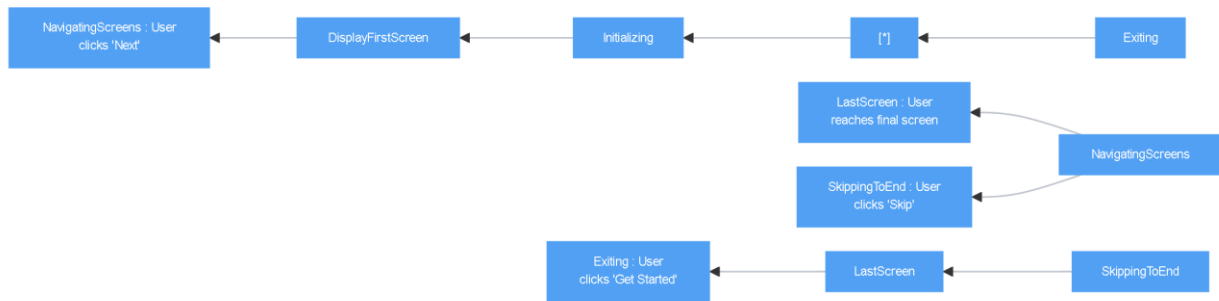
reconocimiento_billetes



- Diagrama de Estados

- Estado Inicial: La actividad comienza en el estado Initializing, donde se configuran los elementos del tutorial. El audio de la primera pantalla comienza automáticamente.
- Navegación entre Pantallas: El usuario puede hacer clic en el botón "Next" para avanzar pantalla por pantalla, moviéndose al estado NavigatingScreens. Al cambiar de pantalla, el audio correspondiente se reproduce automáticamente.
- Finalización del Tutorial: Al llegar a la última pantalla, la actividad entra al estado LastScreen, donde se ocultan elementos innecesarios y se destaca el botón "Get Started". El usuario puede finalizar el tutorial al hacer clic en este botón, moviéndose al estado Exiting.

- Omisión del Tutorial: Si el usuario selecciona "Skip", se mueve directamente al estado SkippingToEnd y luego a LastScreen.
- Liberación de Recursos: En cualquier punto, si la actividad es destruida, el recurso MediaPlayer es liberado para evitar fugas de memoria.



8. Desarrollo

En esta sección se presenta todo el código fuente de la aplicación desarrollada utilizando el lenguaje de programación Kotlin.

8.1. SQLiteHelper.kt

```

package com.example.reconocimiento_billetes.data

import android.content.ContentValues
import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import com.example.reconocimiento_billetes.domain.BillData

/**
 * Esta clase ayuda a interactuar con una base de datos SQLite para almacenar,
 * recuperar y gestionar
 * información sobre billetes en la aplicación.
 * Hereda de [SQLiteOpenHelper], proporcionando métodos para crear y actualizar la
 * base de datos.
 */
class SQLiteHelper(context: Context) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

```

```

companion object {
    private const val DATABASE_VERSION = 1
    private const val DATABASE_NAME = "billDatabase.db"

    // Nombre de la tabla en la base de datos
    private const val TABLE_BILLS = "Bills"

    // Columnas de la tabla
    private const val KEY_ID = "id"
    private const val KEY_BILL = "bill"
    private const val KEY_DATE = "date"

    // Consultas SQL como constantes
    private const val SQL_CREATE_TABLE = """
        CREATE TABLE $TABLE_BILLS (
            $KEY_ID INTEGER PRIMARY KEY AUTOINCREMENT,
            $KEY_BILL INTEGER,
            $KEY_DATE TEXT
        )
    """

    private const val SQL_DROP_TABLE = "DROP TABLE IF EXISTS $TABLE_BILLS"
    private const val SQL_SELECT_ALL_BILLS = "SELECT * FROM $TABLE_BILLS"
    private const val SQL_DELETE_ALL_BILLS = "DELETE FROM $TABLE_BILLS"
    private const val SQL_SUM_BILLS = "SELECT SUM($KEY_BILL) FROM
$TABLE_BILLS"
}

/**
 * Aquí se define la estructura de la base de datos y se ejecuta la consulta
de creación de la tabla.
 */
override fun onCreate(db: SQLiteDatabase?) {
    db?.execSQL(SQL_CREATE_TABLE)
}

/**
 * Se ejecuta cuando se actualiza la base de datos (cuando se cambia la
versión).
 * En este caso, elimina la tabla existente y crea una nueva.
 */
override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int)
{
    db?.execSQL(SQL_DROP_TABLE)
    onCreate(db)
}

/**
 * Inserta un nuevo billete en la base de datos.
 *
 * @param value El valor del billete que se inserta.
 * @param date La fecha asociada al billete.
 * @return El ID del nuevo registro insertado en la base de datos.
 */
fun insertBill(value: Int, date: String): Long {
    // Crea un ContentValues para almacenar los datos del billete.

```



```

        val contentValues = ContentValues().apply {
            put(KEY_BILL, value)
            put(KEY_DATE, date)
        }

        // Inserta el billete en la base de datos y retorna el ID del nuevo
registro.
        return writableDatabase.use { db ->
            db.insert(TABLE_BILLS, null, contentValues)
        }
    }

    /**
     * Obtiene todos los billetes almacenados en la base de datos.
     *
     * @return Una lista de objetos [BillData] representando todos los billetes.
     */
    fun getAllBills(): List<BillData> {
        val billList = mutableListOf<BillData>()

        // Consulta la base de datos y recorre el cursor para obtener todos los
billetes.
        readableDatabase.use { db ->
            db.rawQuery(SQL_SELECT_ALL_BILLS, null).use { cursor ->
                while (cursor.moveToNext())
                    billList.add(cursor.toBillData())
            }
        }

        return billList
    }

    /**
     * Elimina todos los billetes de la base de datos.
     */
    fun deleteAllBills() {
        writableDatabase.use { db ->
            db.execSQL(SQL_DELETE_ALL_BILLS)
        }
    }

    /**
     * Calcula el total de los valores de todos los billetes almacenados.
     *
     * @return El total de los valores de los billetes.
     */
    fun getTotalAmount(): Int {
        readableDatabase.use { db ->
            db.rawQuery(SQL_SUM_BILLS, null).use { cursor ->
                // Si hay un resultado, retorna la suma; de lo contrario, retorna
0.
                return if (cursor.moveToFirst()) cursor.getInt(0) else 0
            }
        }
    }
}

```

```

/**
 * Convierte un cursor de la base de datos a un objeto [BillData].
 * Este método ayuda a transformar cada fila del cursor en un objeto que
 * represente los datos del billete.
 *
 * @return Un objeto [BillData] con los datos extraídos del cursor.
 */
private fun android.database.Cursor.toBillData(): BillData {
    return BillData(
        value = getInt(getColumnIndexOrThrow(KEY_BILL)),
        date = getString(getColumnIndexOrThrow(KEY_DATE))
    )
}

```

8.2. BillData.kt

```

package com.example.reconocimiento_billetes.domain

/**
 * Representa los datos de un billete reconocidos por la aplicación.
 *
 * @property value El valor monetario del billete.
 * @property date La fecha en que se reconoció el billete, en formato de cadena.
 */
data class BillData(
    val value: Int,
    val date: String
) {
    init {
        require(value > 0) { "El valor del billete debe ser mayor que cero." }
        require(date.isNotEmpty()) { "La fecha no puede estar vacía." }
    }
}

```

8.3. ScreenItem.kt

```

package com.example.reconocimiento_billetes.domain

/**
 * Representa un elemento de la pantalla en el tutorial de la aplicación.
 *
 * @property title El título del elemento de la pantalla.
 * @property description Una breve descripción del elemento.
 * @property audio El identificador del recurso de audio asociado.
 */
data class ScreenItem(
    val title: String,
    val description: String,

```

```

        val audio: Int
    ) {
        init {
            require(title.isNotEmpty()) { "El título no puede estar vacío." }
            require(description.isNotEmpty()) { "La descripción no puede estar vacía." }
        }

        require(audio > 0) { "El identificador de audio debe ser un número positivo válido." }
    }
}

```

8.4. BillsAdapter.kt

```

package com.example.reconocimiento_billetes.presentation

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.RecyclerView
import com.example.reconocimiento_billetes.R
import com.example.reconocimiento_billetes.domain.BillData
import java.text.NumberFormat
import java.util.Locale

/**
 * Adaptador para mostrar una lista de billetes en un RecyclerView.
 *
 * Este adaptador toma una lista de objetos [BillData] y los muestra en un
 * RecyclerView.
 * Utiliza [DiffUtil] para actualizar la lista de manera eficiente cuando los
 * datos cambian.
 *
 * @param bills Lista inicial de billetes.
 */
class BillsAdapter(private var bills: List<BillData>) :
    RecyclerView.Adapter<BillsAdapter.BillViewHolder>() {

    /**
     * ViewHolder que contiene las referencias a los elementos de la vista.
     *
     * Este ViewHolder es responsable de referenciar las vistas individuales
     * dentro de
     * cada item del RecyclerView para el valor y la fecha del billete.
     */
    class BillViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val valueTextView: TextView = itemView.findViewById(R.id.valueTextView)
        val dateTextView: TextView = itemView.findViewById(R.id.dateTextView)
    }
}

```

```

    * Infla la vista del ítem en el RecyclerView.
    *
    * Este método se invoca para crear un nuevo ViewHolder cuando se necesita.
    Infla el diseño de cada ítem
    * utilizando el archivo XML `layout_bill` y crea un objeto `BillViewHolder`
    para esa vista.
    *
    * @param parent El contenedor donde se colocará el nuevo ítem.
    * @param viewType El tipo de vista del ítem (no utilizado en este caso).
    * @return Un nuevo ViewHolder con la vista inflada.
    */
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
    BillViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.layout_bill, parent, false)
        return BillViewHolder(view)
    }

    /**
    * Vincula los datos de un billete a su vista correspondiente.
    *
    * Este método es responsable de llenar las vistas de un ítem del RecyclerView
    con los datos del objeto
    * [BillData] correspondiente a la posición actual.
    *
    * @param holder El ViewHolder que contiene las vistas a llenar.
    * @param position La posición del ítem dentro de la lista de billetes.
    */
    override fun onBindViewHolder(holder: BillViewHolder, position: Int) {
        val bill = bills[position]
        holder.valueTextView.text = formatCurrency(bill.value)
        holder.dateTextView.text = bill.date
    }

    /**
    * Devuelve el número total de ítems en la lista de billetes.
    *
    * @return El tamaño de la lista de billetes.
    */
    override fun getItemCount(): Int = bills.size

    /**
    * Actualiza la lista de billetes utilizando DiffUtil para optimizar las
    actualizaciones.
    *
    * Este método compara la lista actual con la nueva lista de billetes y
    calcula las diferencias
    * para aplicar solo los cambios necesarios, mejorando el rendimiento al
    actualizar el RecyclerView.
    *
    * @param newBills Lista actualizada de billetes.
    */
    fun updateBills(newBills: List<BillData>) {
        val diffResult = DiffUtil.calculateDiff(
            BillDiffCallback(
                bills,

```

```

        newBills
    )
    )
    bills = newBills
    diffResult.dispatchUpdatesTo(this) // Aplica los cambios
}

/**
 * Formatea un valor monetario según la configuración regional actual.
 *
 * Este método toma un valor de tipo `Int` y lo formatea como una cadena de
 texto en el formato de
 * moneda correspondiente a la configuración regional del dispositivo.
 *
 * @param value El valor monetario que se desea formatear.
 * @return Una cadena de texto que representa el valor monetario formateado.
 */
private fun formatCurrency(value: Int): String {
    val formatter = NumberFormat.getCurrencyInstance(Locale.getDefault())
    return formatter.format(value)
}

/**
 * Clase para calcular las diferencias entre dos listas de billetes.
 *
 * Esta clase usa [DiffUtil] para comparar dos listas de objetos [BillData] y
 determinar qué elementos
 * han cambiado, se han agregado o se han eliminado, lo que permite actualizar
 el RecyclerView de manera
 * más eficiente.
 */
class BillDiffCallback(
    private val oldList: List<BillData>,
    private val newList: List<BillData>
) : DiffUtil.Callback() {

    /**
     * Devuelve el tamaño de la lista antigua.
     *
     * @return El tamaño de la lista antigua.
     */
    override fun getOldListSize(): Int = oldList.size

    /**
     * Devuelve el tamaño de la nueva lista.
     *
     * @return El tamaño de la nueva lista.
     */
    override fun getNewListSize(): Int = newList.size

    /**
     * Compara si dos elementos de la lista son el mismo objeto.
     *
     * En este caso, se considera que los elementos son los mismos si tienen
 la misma fecha.
     * Esto se usa para determinar si un billete en la lista vieja corresponde

```

```

al mismo billete en la nueva lista.
    *
    * @param oldItemPosition La posición del ítem en la lista antigua.
    * @param newItemPosition La posición del ítem en la lista nueva.
    * @return `true` si los ítems son el mismo (por ejemplo, tienen la misma
fecha), `false` de lo contrario.
    */
    override fun areItemsTheSame(oldItemPosition: Int, newItemPosition: Int):
Boolean {
        return oldList[oldItemPosition].date == newList[newItemPosition].date
    }

    /**
    * Compara si los contenidos de dos elementos son los mismos.
    *
    * @param oldItemPosition La posición del ítem en la lista antigua.
    * @param newItemPosition La posición del ítem en la lista nueva.
    * @return `true` si los contenidos de los ítems son los mismos, `false`
de lo contrario.
    */
    override fun areContentsTheSame(oldItemPosition: Int, newItemPosition:
Int): Boolean {
        return oldList[oldItemPosition] == newList[newItemPosition]
    }
}

```

8.5. IntroViewPagerAdapter.kt

```

package com.example.reconocimiento_billetes.presentation

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.viewpager.widget.PagerAdapter
import com.example.reconocimiento_billetes.R
import com.example.reconocimiento_billetes.domain.ScreenItem

/**
 * Adaptador para el ViewPager que maneja las pantallas introductorias.
 *
 * @param context El contexto de la aplicación.
 * @param listScreen Lista de elementos de tipo [ScreenItem] para mostrar en las
pantallas.
 */
class IntroViewPagerAdapter(
    private val context: Context,
    private val listScreen: List<ScreenItem>
) : PagerAdapter() {

```

```

/**
 * Infla la vista de cada ítem del pager y la rellena con los datos
correspondientes.
 *
 * @param container El contenedor en el que se añadirá la vista.
 * @param position La posición actual del ítem.
 * @return El objeto asociado con la vista inflada.
 */
override fun instantiateItem(container: ViewGroup, position: Int): Any {
    val layoutScreen = inflateScreenLayout(container)

    // Asignar los valores del título y la descripción a las vistas
correspondientes
    bindScreenData(layoutScreen, position)

    container.addView(layoutScreen)
    return layoutScreen
}

/**
 * Infla el diseño de cada pantalla usando el LayoutInflater.
 *
 * @param container El contenedor que recibe la vista inflada.
 * @return La vista inflada.
 */
private fun inflateScreenLayout(container: ViewGroup): View {
    val inflater = context.getSystemService(Context.LAYOUT_INFLATER_SERVICE)
as LayoutInflater
    return inflater.inflate(R.layout.layout_screen, container, false)
}

/**
 * Asigna el título y la descripción a la vista correspondiente.
 *
 * @param layoutScreen La vista inflada a la que se asignan los datos.
 * @param position La posición del elemento actual en la lista.
 */
private fun bindScreenData(layoutScreen: View, position: Int) {
    val title: TextView = layoutScreen.findViewById(R.id.intro_title)
    val description: TextView =
layoutScreen.findViewById(R.id.intro_description)

    val screenItem = listScreen[position]
    title.text = screenItem.title
    description.text = screenItem.description
}

/**
 * Devuelve el número de pantallas en el adaptador.
 *
 * @return El tamaño de la lista de pantallas.
 */
override fun getCount(): Int = listScreen.size

/**
 * Compara si una vista corresponde al objeto de un ítem.

```

```

    *
    * @param view La vista del ítem a comparar.
    * @param `object` El objeto de referencia para comparar.
    * @return `true` si la vista corresponde al objeto, `false` de lo contrario.
    */
    override fun isViewFromObject(view: View, `object`: Any): Boolean = view ==
`object`

    /**
    * Elimina una vista del contenedor cuando ya no se necesita.
    *
    * @param container El contenedor que contiene la vista.
    * @param position La posición de la vista a eliminar.
    * @param `object` El objeto asociado con la vista a eliminar.
    */
    override fun destroyItem(container: ViewGroup, position: Int, `object`: Any) {
        container.removeView(`object` as View)
    }
}

```

8.6. ModelListAdapter.kt

```

package com.example.reconocimiento_billetes.presentation

import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView
import com.example.reconocimiento_billetes.R

/**
 * Adaptador para mostrar una lista de modelos en un RecyclerView.
 *
 * @param models Lista de modelos a mostrar.
 * @param onModelSelected Callback que se ejecuta cuando se selecciona un modelo.
 */
class ModelListAdapter(
    private val models: List<String>,
    private val onModelSelected: (String, Int) -> Unit
) : RecyclerView.Adapter<ModelListAdapter.ModelViewHolder>() {

    private var selectedPosition: Int = RecyclerView.NO_POSITION

    /**
     * Crea una nueva instancia de ViewHolder y la inicializa con la vista
     correspondiente.
     *
     * @param parent El ViewGroup en el que se va a añadir la nueva vista.
     * @param viewType El tipo de vista que se va a crear.
     * @return Un nuevo ModelViewHolder.
     */
}

```



```

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
ModelViewHolder {
            val view = LayoutInflater.from(parent.context)
                .inflate(R.layout.item_model, parent, false)
            return ModelViewHolder(view)
        }

        /**
         * Asigna los datos a las vistas del ViewHolder.
         *
         * @param holder El ViewHolder que contiene las vistas a las que se asignarán
los datos.
         * @param position La posición actual en la lista de modelos.
         */
        override fun onBindViewHolder(holder: ModelViewHolder, position: Int) {
            val model = models[position]
            holder.bind(model, position)
        }

        /**
         * Devuelve el número total de modelos en la lista.
         *
         * @return El tamaño de la lista de modelos.
         */
        override fun getItemCount(): Int = models.size

        /**
         * Actualiza la posición del modelo seleccionado y notifica al adaptador.
         *
         * @param position La posición del modelo seleccionado.
         */
        fun setSelectedPosition(position: Int) {
            val previousPosition = selectedPosition
            selectedPosition = position
            notifyItemChanged(previousPosition) // Notificar cambio en la posición
anterior
            notifyItemChanged(selectedPosition) // Notificar cambio en la nueva
posición
        }

        /**
         * ViewHolder que contiene las referencias a las vistas de cada ítem.
         */
        inner class ModelViewHolder(view: View) : RecyclerView.ViewHolder(view) {
            private val modelName: TextView = view.findViewById(R.id.model_name)
            private val selectionIndicator: View =
view.findViewById(R.id.selection_indicator)

            /**
             * Asigna el modelo y la posición al ViewHolder y establece el listener
para clics.
             *
             * @param model El modelo que se va a mostrar.
             * @param position La posición del modelo en la lista.
             */
            fun bind(model: String, position: Int) {

```

```

        modelName.text = model
        selectionIndicator.setBackgroundResource(
            if (position == selectedPosition)
                R.drawable.circle_selected
            else
                R.drawable.circle_unselected
        )

        itemView.setOnClickListener {
            setSelectedPosition(position)
            onModelSelected(model, position)
        }
    }
}

```

8.7. ConfigActivity.kt

```

package com.example.reconocimiento_billetes

import android.annotation.SuppressLint
import android.content.Intent
import android.media.MediaPlayer
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.widget.Button
import androidx.appcompat.app.AppCompatActivity
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.reconocimiento_billetes.presentation.ModelListAdapter
import com.example.reconocimiento_billetes.presentation.getLocalizedAudioResId

/**
 * ConfigActivity permite al usuario seleccionar un modelo desde una lista y
 * escuchar un sonido
 * cuando realiza una selección. Además, gestiona la navegación entre pantallas
 * con un botón
 * y el deslizamiento de la pantalla.
 */
class ConfigActivity : AppCompatActivity() {

    private lateinit var mediaPlayer: MediaPlayer
    private lateinit var modelListRecyclerView: RecyclerView
    private lateinit var selectModelButton: Button
    private lateinit var modelListAdapter: ModelListAdapter

    private val thresholdWidth get() = (resources.displayMetrics.widthPixels *
0.25f)
    private val modelNames: Array<String> by lazy {
resources.getStringArray(R.array.model_names) }
    private val doubleTapThreshold: Long = 300

```

```

private var selectedIndex: Int? = null
private var startX = 0f
private var lastTapTime: Long = 0

/**
 * Método de inicialización de la actividad.
 * Configura los elementos de la interfaz y reproduce el sonido inicial.
 */
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_config)

    mediaPlayer = MediaPlayer.create(this, getLocalizedAudioResId(this,
"configuracion"))
    mediaPlayer.start()

    // Recupera el índice del modelo seleccionado
    selectedIndex = intent.getIntExtra("selectedIndex", -1)

    modelListRecyclerView = findViewById(R.id.model_list)
    selectModelButton = findViewById(R.id.select_model_button)

    // Configura el RecyclerView con un LinearLayoutManager
    modelListRecyclerView.layoutManager = LinearLayoutManager(this)

    modelListAdapter = ModelListAdapter(modelNames.toList()) { _, position ->
        selectedIndex = position
        playSelectionSound()
        modelListAdapter.setSelectedPosition(
            selectedIndex ?: 0
        )
    }

    modelListRecyclerView.adapter = modelListAdapter

    selectedIndex?.let { index ->
        if (index >= 0) modelListAdapter.setSelectedPosition(index)
    }

    selectModelButton.setOnClickListener {
        selectedIndex?.let { index ->
            val intent = Intent(this, MainActivity::class.java)
            intent.putExtra("selectedIndex", index)
            startActivity(intent)
            finish()
        }
    }

    setupTouchListener()
}

/**
 * Libera el reproductor de medios al destruir la actividad.
 */
override fun onDestroy() {

```

```

        super.onDestroy()
        mediaPlayer.release()
    }

    /**
     * Configura un listener táctil para detectar deslizamientos y toques en la
     * pantalla.
     * Permite navegar entre elementos o finalizar la actividad mediante gestos.
     */
    @SuppressWarnings("ClickableViewAccessibility")
    private fun setupTouchListener() {
        val touchListener = View.OnTouchListener { _, event ->
            when (event.action) {
                MotionEvent.ACTION_DOWN -> {
                    startX = event.x
                }

                MotionEvent.ACTION_UP -> {
                    val deltaX = event.x - startX

                    if (deltaX > thresholdWidth) selectNextItem()

                    if (deltaX < -thresholdWidth) finish()

                    if (isDoubleTap()) {
                        selectedModelIndex?.let { index ->
                            val intent = Intent(this, MainActivity::class.java)
                            intent.putExtra("selectedModelIndex", index)
                            startActivity(intent)
                            finish()
                        }
                    }
                }
            }
            true
        }
    }

    findViewById<RecyclerView>(R.id.model_list).setOnTouchListener(touchListener)
}

/**
 * Cambia la selección al siguiente ítem de la lista.
 */
private fun selectNextItem() {
    selectedModelIndex = (selectedModelIndex ?: -1) + 1
    selectedModelIndex = selectedModelIndex?.takeIf { it <
modelListAdapter.itemCount } ?: 0
    modelListAdapter.setSelectedPosition(selectedModelIndex ?: 0)
    playSelectionSound()
}

/**
 * Detecta si ha ocurrido un doble toque en la pantalla.
 * @return true si se detecta un doble toque, false en caso contrario.
 */

```

```

private fun isDoubleTap(): Boolean {
    val currentTime = System.currentTimeMillis()
    val isDoubleTap = currentTime - lastTapTime < doubleTapThreshold
    lastTapTime = currentTime
    return isDoubleTap
}

/**
 * Reproduce un sonido asociado al modelo seleccionado.
 * El sonido se selecciona dinámicamente según el nombre del modelo.
 */
private fun playSelectionSound() {
    try {
        // Se crea un nuevo reproductor de medios con el sonido
        // correspondiente al modelo
        mediaPlayer = MediaPlayer.create(
            this,
            getLocalizedAudioResId(
                this,
                "configuracion_${modelNames[selectedModelIndex!!].split("
                ") [0].lowercase()}"
            )
        )
        mediaPlayer.start()
    } catch (e: Exception) {
        e.printStackTrace()
    }
}
}

```

8.8. CountBillActivity.kt

```

package com.example.reconocimiento_billetes

import android.annotation.SuppressLint
import android.content.Intent
import android.media.MediaPlayer
import android.net.Uri
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.view.MotionEvent
import android.view.View
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity
import androidx.core.content.FileProvider
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.example.reconocimiento_billetes.data.SQLiteHelper
import com.example.reconocimiento_billetes.domain.BillData
import com.example.reconocimiento_billetes.presentation.BillsAdapter

```

```

import com.example.reconocimiento_billetes.presentation.getLocalizedAudioResId
import java.io.File
import java.io.FileOutputStream
import kotlin.math.abs

/**
 * Actividad que permite gestionar el historial de billetes contados.
 * Muestra la lista de billetes contados, permite eliminar el historial,
 * compartir el historial y manejar interacciones como los toques rápidos para
 borrar el historial.
 */
class CountBillActivity : AppCompatActivity() {

    private lateinit var mediaPlayer: MediaPlayer
    private lateinit var deleteSoundPlayer: MediaPlayer
    private lateinit var billsAdapter: BillsAdapter
    private lateinit var db: SQLiteHelper

    private var tapCount = 0
    private val tapTimeout = 500L
    private val handler = Handler(Looper.getMainLooper())

    private var x1: Float = 0f
    private val swipeThreshold = 300

    /**
     * Método que se ejecuta al crear la actividad. Inicializa los componentes,
     * configura el RecyclerView, los botones y los eventos de toque.
     */
    @SuppressWarnings("ClickableViewAccessibility", "SetTextI18n")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_count_bill)

        initializeComponents()

        val recyclerView = findViewById<RecyclerView>(R.id.billsRecyclerView)
        billsAdapter = BillsAdapter(emptyList())
        recyclerView.apply {
            adapter = billsAdapter
            layoutManager = LinearLayoutManager(this@CountBillActivity)
        }

        findViewById<Button>(R.id.deleteButton).setOnClickListener {
            deleteHistory()
        }

        findViewById<Button>(R.id.shareButton).setOnClickListener {
            shareHistory()
        }

        val touchListener = View.OnTouchListener { v, event ->
            v.performClick()
            when (event.action) {
                MotionEvent.ACTION_DOWN -> {
                    x1 = event.x

```

```

        handleTap()
        true
    }

    MotionEvent.ACTION_UP -> {
        val x2 = event.x
        val xDiff = x2 - x1

        if (abs(xDiff) > swipeThreshold) {
            if (xDiff > 0)
                shareHistory()
            else finish()
        }
        true
    }

    else -> false
}

}

findViewById<RecyclerView>(R.id.billsRecyclerView).setOnTouchListener(touchListene
r)

val bills = db.getAllBills().reversed()
billsAdapter.updateBills(bills)
findViewById<TextView>(R.id.totalTextView).text =
    "${getString(R.string.totalBilletes)}${db.getTotalAmount()}"
}

/**
 * Inicializa los componentes como los reproductores de audio y la base de
datos.
 */
private fun initializeComponents() {
    mediaPlayer = MediaPlayer.create(this, getLocalizedAudioResId(this,
"historial_billetes"))
    deleteSoundPlayer =
        MediaPlayer.create(this, getLocalizedAudioResId(this,
"borrar_historial"))
    mediaPlayer.start()

    db = SQLiteHelper(this)
}

/**
 * Elimina todos los billetes del historial de la base de datos
 * y actualiza la interfaz de usuario.
 */
private fun deleteHistory() {
    db.deleteAllBills()
    updateUI()
    deleteSoundPlayer.start()
}

/**

```

```

    * Actualiza la interfaz de usuario con los billetes más recientes
    * y muestra el total de dinero.
    */
@SuppressLint("SetTextI18n")
private fun updateUI() {
    val bills = db.getAllBills().reversed()
    billsAdapter.updateBills(bills)
    findViewById<TextView>(R.id.totalTextView).text =
        "${getString(R.string.totalBilletes)}${db.getTotalAmount()}"
}

/**
 * Maneja el conteo de toques rápidos. Si se tocan más de 4 veces en un corto
periodo,
 * elimina el historial.
 */
private fun handleTap() {
    tapCount++
    handler.removeCallbacksAndMessages(null)

    if (tapCount >= 5) {
        deleteHistory()
        tapCount = 0
    } else handler.postDelayed({ tapCount = 0 }, tapTimeout)
}

/**
 * Comparte el historial de billetes mediante un archivo.
 * Guarda el historial en un archivo y lo comparte mediante una intención.
 */
private fun shareHistory() {
    val bills = db.getAllBills().reversed()
    val totalAmount = db.getTotalAmount()
    val file = saveHistoryToFile(bills, totalAmount)
    if (file != null) shareHistoryFile(file)
}

/**
 * Guarda el historial de billetes en un archivo de texto en el almacenamiento
interno.
 */
private fun saveHistoryToFile(
    bills: List<BillData>,
    totalAmount: Int
): File? {
    val fileName = "cashScan.txt"
    val file = File(this.filesDir, fileName)

    try {
        FileOutputStream(file).use { output ->
output.write("${getString(R.string.totalBilletesArchivo)}$totalAmount\n\n".toByteArray())

output.write("${getString(R.string.historialDeBilletes)}:\n".toByteArray())

```



```

        for (bill in bills) {
            val line =
                "${getString(R.string.valorBillete)}${bill.value},
${getString(R.string.fechaBillete)}${bill.date}\n"
            output.write(line.toByteArray())
        }
    }
} catch (e: Exception) {
    e.printStackTrace()
    return null
}

return file
}

/**
 * Comparte el archivo de historial utilizando un intent.
 */
private fun shareHistoryFile(file: File) {
    val uri: Uri = FileProvider.getUriForFile(
        this,
        "${this.packageName}.fileprovider",
        file
    )

    val intent = Intent(Intent.ACTION_SEND).apply {
        type = "text/plain"
        putExtra(Intent.EXTRA_STREAM, uri)
        addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION)
    }

    startActivity(Intent.createChooser(intent,
        getString(R.string.compartirHistorial)))
}

/**
 * Libera los recursos utilizados por los reproductores de audio y el handler.
 */
override fun onDestroy() {
    super.onDestroy()
    mediaPlayer.release()
    deleteSoundPlayer.release()
    handler.removeCallbacksAndMessages(null)
}
}

```

8.9. IntroActivity.kt

```

package com.example.reconocimiento_billetes

import android.content.Intent
import android.os.Bundle

```

```

import androidx.appcompat.app.AppCompatActivity
import androidx.core.splashscreen.SplashScreen.Companion.installSplashScreen

/**
 * Actividad inicial de la aplicación que maneja la pantalla de bienvenida (Splash
 * Screen).
 * Esta actividad verifica si es la primera vez que el usuario abre la aplicación.
 * Si es la primera vez, muestra el tutorial, de lo contrario, redirige
 * directamente
 * a la actividad principal.
 */
class IntroActivity : AppCompatActivity() {

    /**
     * Método que se ejecuta cuando la actividad es creada.
     * Muestra el Splash Screen, verifica si el tutorial ya fue mostrado,
     * y redirige a la actividad correspondiente (Tutorial o Main).
     */
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Muestra el Splash Screen durante 1 segundo
        Thread.sleep(1000)
        installSplashScreen()

        if (isTutorialShown()) {
            navigateToMainActivity()
        } else {
            markTutorialAsShown()
            startActivity(Intent(this, TutorialActivity::class.java))
        }

        finish()
    }

    /**
     * Verifica si el tutorial ya ha sido mostrado en sesiones anteriores.
     * Utiliza SharedPreferences para almacenar el estado de esta preferencia.
     *
     * @return `true` si el tutorial ya fue mostrado, `false` si no lo ha sido.
     */
    private fun isTutorialShown(): Boolean {
        val sharedPrefs = getSharedPreferences("myPrefs", MODE_PRIVATE)
        return sharedPrefs.getBoolean("isIntroOpened", false)
    }

    /**
     * Inicia la actividad principal de la aplicación (MainActivity).
     */
    private fun navigateToMainActivity() {
        startActivity(Intent(this, MainActivity::class.java))
        finish()
    }

    /**
     * Marca en SharedPreferences que el tutorial ha sido mostrado.
     */
}

```

```

    */
    private fun markTutorialAsShown() {
        val sharedPrefs = getSharedPreferences("myPrefs", MODE_PRIVATE)
        val editor = sharedPrefs.edit()
        editor.putBoolean("isIntroOpened", true)
        editor.apply()
    }
}

```

8.10. MainActivity.kt

```

package com.example.reconocimiento_billetes

import android.annotation.SuppressLint
import android.content.Intent
import android.media.MediaPlayer
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.widget.RelativeLayout
import androidx.activity.ComponentActivity
import androidx.cardview.widget.CardView
import androidx.core.content.ContextCompat
import com.example.reconocimiento_billetes.presentation.getLocalizedAudioResId

/**
 * Actividad principal que muestra el menú con opciones para acceder a diferentes
 * funciones.
 * Incluye navegación mediante botones y deslizamientos táctiles.
 */
class MainActivity : ComponentActivity() {

    private lateinit var mediaPlayer: MediaPlayer

    private var startX = 0f
    private var startY = 0f

    private val thresholdWidth get() = (resources.displayMetrics.widthPixels *
0.25f)
    private val thresholdHeight get() = (resources.displayMetrics.heightPixels *
0.25f)

    /**
     * Se ejecuta cuando se crea la actividad. Inicializa los componentes de la UI
     y maneja
     * la navegación y los gestos táctiles.
     */
    @SuppressLint("ClickableViewAccessibility")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}

```

```

        // Obtiene el índice del modelo seleccionado y los nombres de los modelos
        val selectedIndex = intent.getIntExtra("selectedIndex", 0)
        val modelNames = resources.getStringArray(R.array.model_names)

        window.statusBarColor = ContextCompat.getColor(this, R.color.green)
        mediaPlayer = MediaPlayer.create(this, getLocalizedAudioResId(this,
"menu_principal"))

        setupButtons(modelNames, selectedIndex)
        setupTouchListener(selectedModelIndex)
    }

    /**
     * Configura los botones del menú principal con sus respectivas acciones.
     */
    private fun setupButtons(modelNames: Array<String>, selectedIndex: Int) {
        findViewById<CardView>(R.id.ScanButton).setOnClickListener {
            val intent = Intent(this, ScanBillActivity::class.java)
            // Envía el modelo seleccionado a la actividad ScanBillActivity
            intent.putExtra(
                "selectedIndex",
                modelNames[selectedIndex].split(" ")[0].lowercase()
            )
            startActivity(intent)
        }

        findViewById<CardView>(R.id.HistoryButton).setOnClickListener {
            startActivity(Intent(this, CountBillActivity::class.java))
        }

        findViewById<CardView>(R.id.TutorialButton).setOnClickListener {
            startActivity(Intent(this, TutorialActivity::class.java))
        }

        findViewById<CardView>(R.id.ConfigButton).setOnClickListener {
            val intent = Intent(this, ConfigActivity::class.java)
            intent.putExtra("selectedIndex", selectedIndex)
            startActivity(intent)
        }
    }

    /**
     * Configura el escuchador de deslizamientos táctiles en la pantalla
principal.
     * Detecta los deslizamientos en las direcciones horizontal y vertical para
navegar.
     */
    @SuppressWarnings("ClickableViewAccessibility")
    private fun setupTouchListener(selectedModelIndex: Int) {
        val touchListener = View.OnTouchListener { _, event ->
            when (event.action) {
                MotionEvent.ACTION_DOWN -> {
                    startX = event.x
                    startY = event.y
                }
            }
        }
    }

```

```

        MotionEvent.ACTION_UP -> {
            // Calcula las diferencias en las coordenadas para detectar el
deslizamiento
            val deltaX = event.x - startX
            val deltaY = event.y - startY

            if (deltaX > thresholdWidth)
                startActivity(Intent(this, ScanBillActivity::class.java))
            else if (deltaX < -thresholdWidth)
                startActivity(Intent(this, CountBillActivity::class.java))

            if (deltaY > thresholdHeight) {
                val intent = Intent(this, ConfigActivity::class.java)
                intent.putExtra("selectedModelIndex", selectedModelIndex)
                startActivity(intent)
            } else if (deltaY < -thresholdHeight)
                startActivity(Intent(this, TutorialActivity::class.java))
        }
    }
    true
}

findViewById<RelativeLayout>(R.id.main_layout).setOnTouchListener(touchListener)
}

/**
 * Se ejecuta cuando la actividad se pone en primer plano. Reproduce el audio.
 */
override fun onResume() {
    super.onResume()
    mediaPlayer.start()
}

/**
 * Se ejecuta cuando la actividad entra en segundo plano. Pausa el audio si
está en reproducción.
 */
override fun onPause() {
    super.onPause()
    if (mediaPlayer.isPlaying) mediaPlayer.pause()
}

/**
 * Se ejecuta cuando la actividad se destruye. Libera el reproductor de
medios.
 */
override fun onDestroy() {
    super.onDestroy()
    mediaPlayer.release()
}
}

```

8.11. ScanBillActivity.kt

```
package com.example.reconocimiento_billetes

import android.Manifest
import android.content.Context
import android.graphics.Bitmap
import android.graphics.Matrix
import android.media.MediaPlayer
import android.media.ThumbnailUtils
import android.os.Bundle
import android.os.SystemClock
import android.os.Vibrator
import android.view.KeyEvent
import androidx.activity.compose.setContent
import androidx.appcompat.app.AlertDialog
import androidx.appcompat.app.AppCompatActivity
import androidx.camera.core.ImageCapture
import androidx.camera.core.ImageProxy
import androidx.camera.view.CameraController
import androidx.camera.view.LifecycleCameraController
import androidx.compose.foundation.gestures.detectDragGestures
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.padding
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.PhotoCamera
import androidx.compose.material3.BottomSheetScaffold
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.rememberBottomSheetScaffoldState
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableFloatStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.input.pointer.pointerInput
import androidx.compose.ui.platform.LocalConfiguration
import androidx.compose.ui.platform.LocalDensity
import androidx.compose.ui.unit.dp
import androidx.core.app.ActivityCompat
import androidx.core.content.ContextCompat
import com.chaquo.python.Python
import com.chaquo.python.android.AndroidPlatform
import com.example.reconocimiento_billetes.presentation.CameraPreview
import com.example.reconocimiento_billetes.presentation.getLocalizedAudioResId
import com.example.reconocimiento_billetes.presentation.guardarBaseDeDatos
import com.example.reconocimiento_billetes.presentation.hasCameraPermission
import com.example.reconocimiento_billetes.presentation.loadClassNames
```

```

import com.example.reconocimiento_billetes.presentation.loadModel
import com.example.reconocimiento_billetes.presentation.vibrateDevice
import com.example.reconocimiento_billetes.ui.theme.ReconocimientobilletesTheme
import org.tensorflow.lite.DataType
import org.tensorflow.lite.support.tensorbuffer.TensorBuffer
import java.io.File
import java.io.FileOutputStream
import java.nio.ByteBuffer
import java.nio.ByteOrder
import kotlin.math.min

/**
 * Actividad para escanear billetes y clasificarlos mediante un modelo de
 * aprendizaje automático.
 */
class ScanBillActivity : AppCompatActivity() {

    private var mediaPlayer: MediaPlayer? = null
    private val imageSize = 224
    private var scanningDialog: AlertDialog? = null
    private lateinit var vibrator: Vibrator
    private lateinit var cameraController: LifecycleCameraController
    private lateinit var selectedModel: String

    @OptIn(ExperimentalMaterial3Api::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        mediaPlayer = MediaPlayer.create(this, getLocalizedAudioResId(this,
"escaneo_billetes"))
        mediaPlayer?.start()

        if (!hasCameraPermission(this))
            ActivityCompat.requestPermissions(this,
arrayOf(Manifest.permission.CAMERA), 0)

        if (!Python.isStarted()) Python.start(AndroidPlatform(this))

        vibrator = getSystemService(Context.VIBRATOR_SERVICE) as Vibrator

        cameraController = LifecycleCameraController(this).apply {
            setEnabledUseCases(CameraController.IMAGE_CAPTURE)
        }

        selectedModel = intent.getStringExtra("selectedModel").toString()

        setContent {
            ReconocimientobilletesTheme {
                var offsetX by remember { mutableFloatStateOf(0f) }

                val screenWidth = LocalConfiguration.current.screenWidthDp.dp
                val percentage = 0.37f
                val thresholdWidth = with(LocalDensity.current) {
screenWidth.toPx() * percentage }

                val scaffoldState = rememberBottomSheetScaffoldState()

```

```

        BottomSheetScaffold(scaffoldState = scaffoldState,
            sheetPeekHeight = 0.dp,
            sheetContent = {}) { padding ->
                Box(modifier = Modifier
                    .fillMaxSize()
                    .padding(padding)
                    .pointerInput(Unit) {
                        detectDragGestures(onDragEnd = {
                            if (offsetX > -thresholdWidth) {
                                offsetX = 0f
                            }
                        }, onDragCancel = {
                            offsetX = 0f
                        }) { _, dragAmount ->
                            offsetX += dragAmount.x
                            if (offsetX < -thresholdWidth) finish()
                        }
                    }) {
                    CameraPreview(
                        controller = cameraController,
                        modifier = Modifier.fillMaxSize()
                    )

                    Row(
                        modifier = Modifier
                            .fillMaxWidth()
                            .align(Alignment.BottomCenter)
                            .padding(16.dp),
                        horizontalArrangement = Arrangement.SpaceAround
                    ) {
                        IconButton(onClick = {
                            showScanningDialog()
                            takePhoto(cameraController)
                        }) {
                            Icon(
                                imageVector = Icons.Default.PhotoCamera,
                                contentDescription = "Tomar foto"
                            )
                        }
                    }
                }
            }
        }
    }

/**
 * Liberar recursos cuando la actividad se destruye.
 */
override fun onDestroy() {
    super.onDestroy()
    mediaPlayer?.release()
    mediaPlayer = null
    cameraController.unbind()
}

```



```

/**
 * Capturar la foto al presionar el botón de volumen.
 */
override fun dispatchKeyEvent(event: KeyEvent): Boolean {
    val keyCode = event.keyCode
    val action = event.action

    if (keyCode == KeyEvent.KEYCODE_VOLUME_DOWN && action ==
KeyEvent.ACTION_DOWN) {
        showScanningDialog()
        takePhoto(cameraController)
    }

    return super.dispatchKeyEvent(event)
}

/**
 * Tomar una foto con la cámara.
 */
private fun takePhoto(controller: LifecycleCameraController) {
    controller.enableTorch(true)
    SystemClock.sleep(2000) // Esperar un momento antes de capturar

    controller.takePicture(
        ContextCompat.getMainExecutor(this),
        object : ImageCapture.OnImageCapturedCallback() {
            override fun onCaptureSuccess(image: ImageProxy) {
                super.onCaptureSuccess(image)
                controller.enableTorch(false)

                // Procesar la imagen capturada
                val matrix = Matrix().apply {
                    postRotate(image.imageInfo.rotationDegrees.toFloat())
                }

                var bitmap = Bitmap.createBitmap(
                    image.toBitmap(),
                    0,
                    0,
                    image.width,
                    image.height,
                    matrix,
                    true
                )
                val dimension = min(bitmap.width, bitmap.height)
                bitmap = ThumbnailUtils.extractThumbnail(bitmap, dimension,
dimension)
                bitmap = Bitmap.createScaledBitmap(bitmap, imageSize,
imageSize, false)

                try {
                    // Guardar la imagen temporalmente
                    val tempFile = File.createTempFile("temp_image", ".jpg",
cacheDir)
                    val fos = FileOutputStream(tempFile)

```

```

        bitmap.compress(Bitmap.CompressFormat.JPEG, 100, fos)
        fos.flush()
        fos.close()

        val classificationResult = modelClassifier(bitmap,
selectedModel)

        // Mostrar resultados del escaneo
        scanningDialog?.dismiss()
        if (classificationResult != null) {
showResultDialog("${getString(R.string.billeteDetectado)}$classificationResult")
            playDenominationSound(classificationResult.toString(),
selectedModel)
            guardarBaseDeDatos(classificationResult.toString(),
applicationContext)
        } else {
showResultDialog(getString(R.string.noSeDetectoBillete))
            playDenominationSound(classificationResult.toString(),
selectedModel)
        }

        tempFile.delete()
    } catch (e: Exception) {
        e.printStackTrace()
        scanningDialog?.dismiss()
        errorMsg()
    } finally {
        image.close()
    }
}

}))

}

/**
 * Clasificar la imagen usando el modelo especificado.
 */
private fun modelClassifier(image: Bitmap, modelName: String): String? {
    val model = loadModel(modelName, this)

    // Preparar la imagen para la entrada del modelo
    val inputFeature0 =
        TensorBuffer.createFixedSize(intArrayOf(1, imageSize, imageSize, 3),
DataType.FLOAT32)
    val byteBuffer = ByteBuffer.allocateDirect(4 * imageSize * imageSize *
3).apply {
        order(ByteOrder.nativeOrder())
    }

    // Convertir la imagen en datos que el modelo pueda procesar
    val intValues = IntArray(imageSize * imageSize)
    image.getPixels(intValues, 0, image.width, 0, 0, image.width,
image.height)
    var pixel = 0

```

```

        for (i in 0 until imageSize) {
            for (j in 0 until imageSize) {
                val `val` = intValues[pixel++]
                byteBuffer.putFloat(((`val` shr 16) and 0xFF) * (1f / 1))
                byteBuffer.putFloat(((`val` shr 8) and 0xFF) * (1f / 1))
                byteBuffer.putFloat((`val` and 0xFF) * (1f / 1))
            }
        }

        inputFeature0.loadBuffer(byteBuffer)

        // Obtener las predicciones del modelo
        val outputs = model.process(inputFeature0)
        val outputFeature0 = outputs.outputFeature0AsTensorBuffer
        val confidences = outputFeature0.floatArray

        // Encontrar la clase con la mayor confianza
        var maxPos = 0
        var maxConfidence = 0f
        for (i in confidences.indices) {
            if (confidences[i] > maxConfidence) {
                maxConfidence = confidences[i]
                maxPos = i
            }
        }

        // Cargar los nombres de las clases y devolver el resultado
        val classes = loadClassNames(this, modelName)
        model.close()

        return if (maxConfidence >= .85) classes[maxPos] else null
    }

    /**
     * Mostrar un mensaje de error si la imagen no pudo ser procesada.
     */
    private fun errorMsg() {
        showResultDialog(getString(R.string.errorProcesarImagen))
        mediaPlayer = MediaPlayer.create(this, getLocalizedAudioResId(this,
"error"))
        mediaPlayer?.start()
    }

    /**
     * Mostrar el diálogo de escaneo.
     */
    private fun showScanningDialog() {
        vibrateDevice(vibrator)
        val dialogBuilder = AlertDialog.Builder(this)
        dialogBuilder.setTitle(getString(R.string.escaneando))
        dialogBuilder.setMessage(getString(R.string.esperaProcesoImagen))
        dialogBuilder.setCancelable(false)

        scanningDialog = dialogBuilder.create()
        scanningDialog?.show()
    }

```

```

        mediaPlayer = MediaPlayer.create(this, getLocalizedAudioResId(this,
"escaneando"))
        mediaPlayer?.start()
    }

    /**
     * Reproducir el sonido correspondiente a la denominación del billete.
     */
    private fun playDenominationSound(denomination: String, modelName: String) {
        var audioResId = getLocalizedAudioResId(this, "no_se_detecto")
        val localizedAudioResId =
            getLocalizedAudioResId(this, "answer_${denomination}_${modelName}")

        if (localizedAudioResId != 0) audioResId = localizedAudioResId

        audioResId.let {
            mediaPlayer?.release()
            mediaPlayer = MediaPlayer.create(this, it)
            mediaPlayer?.start()
        }
    }

    /**
     * Mostrar un diálogo con el resultado de la clasificación.
     */
    private fun showResultDialog(resultText: String) {
        val dialogBuilder = AlertDialog.Builder(this)
        dialogBuilder.setTitle(getString(R.string.resultadoClasificacion))
        dialogBuilder.setMessage(resultText)
        dialogBuilder.setPositiveButton("OK") { dialog, _ -> dialog.dismiss() }

        val alertDialog = dialogBuilder.create()
        alertDialog.show()
    }
}

```

8.12. TutorialActivity.kt

```

package com.example.reconocimiento_billetes

import android.annotation.SuppressLint
import android.content.Intent
import android.media.MediaPlayer
import android.os.Bundle
import android.view.GestureDetector
import android.view.MotionEvent
import android.view.View
import android.view.animation.Animation
import android.view.animation.AnimationUtils
import android.widget.Button
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

```

```

import androidx.viewpager.widget.ViewPager
import com.example.reconocimiento_billetes.domain.ScreenItem
import com.example.reconocimiento_billetes.presentation.IntroViewPagerAdapter
import com.example.reconocimiento_billetes.presentation.getLocalizedAudioResId
import com.google.android.material.tabs.TabLayout

/**
 * TutorialActivity muestra una secuencia de pantallas introductorias con audio y
 * animaciones.
 * Permite navegar entre pantallas y realizar acciones como saltar, continuar o
 * finalizar el tutorial.
 */
class TutorialActivity : AppCompatActivity() {

    private lateinit var viewPager: ViewPager
    private lateinit var introViewPagerAdapter: IntroViewPagerAdapter
    private lateinit var tabIndicator: TabLayout
    private lateinit var btnNext: Button
    private lateinit var btnGetStarted: Button
    private lateinit var tvSkip: TextView
    private lateinit var btnAnim: Animation
    private lateinit var gestureDetector: GestureDetector

    private var position = 0
    private var mediaPlayer: MediaPlayer? = null

    /**
     * Método de inicialización de la actividad.
     * Configura los elementos de la vista, la navegación y la reproducción de
     * audio.
     */
    @SuppressWarnings("ClickableViewAccessibility")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_intro)

        btnNext = findViewById(R.id.btn_next)
        btnGetStarted = findViewById(R.id.btn_get_started)
        tabIndicator = findViewById(R.id.tab_indicator)
        tvSkip = findViewById(R.id.tv_skip)
        btnAnim = AnimationUtils.loadAnimation(applicationContext,
R.anim.button_animation)

        val mList = listOf(
            ScreenItem(
                getString(R.string.intro1Title),
                getString(R.string.intro1),
                getLocalizedAudioResId(this, "intro_1")
            ),
            ScreenItem(
                getString(R.string.intro2Title),
                getString(R.string.intro2),
                getLocalizedAudioResId(this, "intro_2")
            ),
            ScreenItem(
                getString(R.string.intro3Title),

```

```

        getString(R.string.intro3),
        getLocalizedAudioResId(this, "intro_3")
    ),
    ScreenItem(
        getString(R.string.historialDeBilletes),
        getString(R.string.intro4),
        getLocalizedAudioResId(this, "intro_4")
    ),
    ScreenItem(
        getString(R.string.intro5Title),
        getString(R.string.intro5),
        getLocalizedAudioResId(this, "intro_5")
    )
)

screenPager = findViewById(R.id.screen_viewpager)
introViewPagerAdapter = IntroViewPagerAdapter(this, mList)
screenPager.adapter = introViewPagerAdapter
tabIndicator.setupWithViewPager(screenPager)

// Reproduce el audio para la primera pantalla
playAudioForCurrentScreenItem(mList[0])

// Configura el comportamiento al cambiar de pantalla en el ViewPager
screenPager.addOnPageChangeListener(object :
ViewPager.OnPageChangeListener {
    override fun onPageScrolled(
        position: Int,
        positionOffset: Float,
        positionOffsetPixels: Int
    ) {
    }

    override fun onPageSelected(position: Int) {
        this@TutorialActivity.position = position
        playAudioForCurrentScreenItem(mList[position])
    }

    override fun onPageScrollStateChanged(state: Int) {}
})

btnNext.setOnClickListener {
    position = screenPager.currentItem
    if (position < mList.size - 1) {
        position++
        screenPager.currentItem = position
    }
    if (position == mList.size - 1) loadLastScreen()
}

tabIndicator.addTabSelectedListener(object :
TabLayout.BaseOnTabSelectedListener<TabLayout.Tab> {
    override fun onTabSelected(tab: TabLayout.Tab?) {
        if (tab?.position == mList.size - 1) loadLastScreen()
    }
})

```

```

        override fun onTabUnselected(tab: TabLayout.Tab?) {}
        override fun onTabReselected(tab: TabLayout.Tab?) {}
    })

    btnGetStarted.setOnClickListener {
        backToIntroActivity()
    }

    tvSkip.setOnClickListener {
        screenPager.currentItem = mList.size
    }

    gestureDetector = GestureDetector(this, object :
GestureDetector.SimpleOnGestureListener() {
        override fun onDoubleTap(e: MotionEvent): Boolean {
            if (position == mList.size - 1) {
                backToIntroActivity()
                return true
            }
            return false
        }
    })

    screenPager.setOnTouchListener { _, event ->
        gestureDetector.onTouchEvent(event)
        false
    }
}

/**
 * Reproduce el audio correspondiente al ítem actual del tutorial.
 * @param screenItem El ítem que contiene la información de la pantalla
actual.
 */
private fun playAudioForCurrentScreenItem(screenItem: ScreenItem) {
    mediaPlayer?.release()
    mediaPlayer = MediaPlayer.create(this, screenItem.audio)
    mediaPlayer?.start()
}

/**
 * Configura la interfaz al llegar a la última pantalla del tutorial.
 * Cambia la visibilidad de los botones y oculta el indicador de tabulador.
 */
private fun loadLastScreen() {
    btnNext.visibility = View.INVISIBLE
    btnGetStarted.visibility = View.VISIBLE
    tvSkip.visibility = View.INVISIBLE
    tabIndicator.visibility = View.INVISIBLE
    btnGetStarted.animation = btnAnim
}

/**
 * Navega hacia la actividad principal (IntroActivity) cuando se finaliza el
tutorial.
 */

```

```

private fun backToIntroActivity() {
    val intent = Intent(this, IntroActivity::class.java)
    startActivity(intent)
    finish() // Finaliza la actividad actual
}

/**
 * Libera los recursos del reproductor de medios cuando la actividad es
 * destruida.
 */
override fun onDestroy() {
    super.onDestroy()
    mediaPlayer?.release()
}
}

```

8.13. MCBIA7.py

Código de entrenamiento del modelo de clasificación de billetes.

```

import json
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping
import os

# Ruta de imágenes
ruta_imagenes = r'D:\IA-32\env\Data'

# Parámetros de entrenamiento
tamano_imagen = (224, 224)
batch_size = 32
epochs = 20 # Vueltas totales

# Aumentación de datos (No tocar)
datagen = ImageDataGenerator(
    rescale=1./255,           # Normalización de las imágenes
    validation_split=0.2,     # Dividir datos en entrenamiento y validación
    rotation_range=20,        # Rotaciones aleatorias
    width_shift_range=0.2,     # Desplazamientos horizontales aleatorios
    height_shift_range=0.2,    # Desplazamientos verticales aleatorios
    shear_range=0.2,          # Cizallamiento aleatorio
    zoom_range=0.2,           # Zoom aleatorio
    horizontal_flip=True      # Invertir horizontalmente
)

```



```

)

# Dataset de entrenamiento
train_generator = datagen.flow_from_directory(
    ruta_imagenes,
    target_size=tamano_imagen,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

# Dataset de validación
validation_generator = datagen.flow_from_directory(
    ruta_imagenes,
    target_size=tamano_imagen,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)

# Mapeo de clases
class_indices = train_generator.class_indices
with open('mapeo_clases.json', 'w') as f:
    json.dump(class_indices, f)

print(f'Mapeo de clases guardado: {class_indices}')

# Definición de modelo de red neuronal con Dropout y regularización L2
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),

```

```

        layers.Dense(512, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)), # Regularización L2
        layers.Dropout(0.5), # Dropout para evitar el sobreajuste

        layers.Dense(len(train_generator.class_indices), activation='softmax') # Número
de clases de salida
])

# Compilacion del modelo
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

#Callback de EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)

train_dataset = tf.data.Dataset.from_generator(
    lambda: train_generator,
    output_signature=(
        tf.TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(None, len(train_generator.class_indices)),
dtype=tf.float32)
    )
).repeat()

validation_dataset = tf.data.Dataset.from_generator(
    lambda: validation_generator,
    output_signature=(
        tf.TensorSpec(shape=(None, 224, 224, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(None, len(validation_generator.class_indices)),
dtype=tf.float32)
    )
).repeat()

# Entrenamiento
history = model.fit(
    train_dataset,
    steps_per_epoch=train_generator.samples // batch_size,
    validation_data=validation_dataset,
    validation_steps=validation_generator.samples // batch_size,
    epochs=epochs,
    callbacks=[early_stopping] # Callback EarlyStopping
)

```

```
# Guardar el modelo entrenado
model.save('modelo_billetes.keras')

# Transformar el modelo
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Guardar el modelo
with open('modelo_billetes.tflite', 'wb') as f:
    f.write(tflite_model)

print("Modelo entrenado y convertido a TensorFlow Lite")
```

9. Privilegios

Los privilegios del proyecto para el reconocimiento de billetes, por parte del sistema operativo son:

- Acceso a la cámara: El proyecto requiere permisos para acceder a la cámara del dispositivo, ya que la aplicación captura imágenes de los billetes para clasificarlos.
- Acceso a archivos y almacenamiento: La aplicación necesita acceso al almacenamiento interno del dispositivo para guardar los resultados, imágenes y configuraciones.
- Acceso a servicios del sistema: La aplicación utiliza servicios como el vibrador del dispositivo, la reproducción de audio, y la linterna de la cámara para alumbrar en oscuridad, lo que le otorga privilegios para interactuar con estos recursos.
- Permisos de red: La aplicación utiliza la funcionalidad en línea, como la descarga de modelos.

Para el usuario, los privilegios de este proyecto son los siguientes:

- Interfaz de usuario interactiva: Los usuarios pueden navegar fácilmente por el tutorial de la aplicación, visualizar las pantallas introductorias y acceder a funciones como tomar fotos y obtener resultados de escaneo de billetes.

- Reconocimiento de billetes: El usuario puede usar la cámara de su dispositivo para escanear billetes y clasificarlos mediante un modelo de reconocimiento de imágenes.
- Audio personalizado: El usuario recibe retroalimentación auditiva personalizada a través de sonidos que corresponden a las acciones realizadas en la aplicación, como los sonidos para las denominaciones de billetes detectados o cuando cambia de actividad.
- Vibración del dispositivo: Cuando el usuario escanea un billete, el dispositivo vibra como retroalimentación táctil.
- Tutorial guiado: Al iniciar la aplicación, el usuario tiene acceso a un tutorial paso a paso que los guía sobre cómo usar la app. Este tutorial incluye explicaciones auditivas para garantizar que el usuario comprenda cómo utilizar la aplicación correctamente.
- Navegación fácil entre pantallas: Los usuarios pueden navegar entre las pantallas utilizando gestos en la pantalla del dispositivo.
- Personalización del modelo: Los usuarios pueden seleccionar el modelo de reconocimiento de billetes que mejor se adapte a sus necesidades.
- Acceso a resultados y almacenamiento: Una vez que el billete ha sido identificado, el usuario puede mantener el resultado de la clasificación y también se puede guardar la información relacionada en la base de datos, lo que permite llevar un historial de los billetes escaneados.