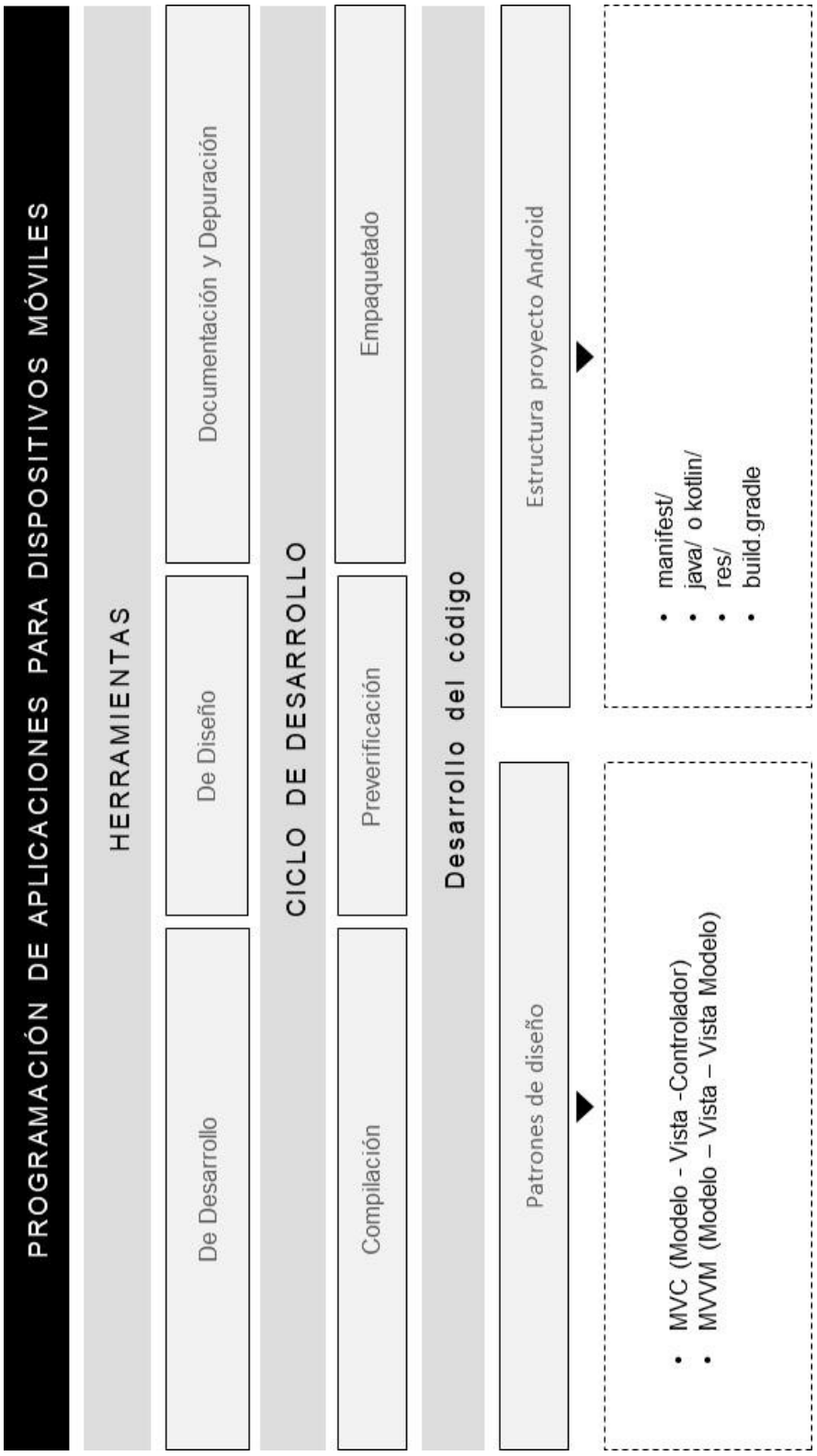
Programación multimedia y dispositivos móviles

Programación de aplicaciones para dispositivos móviles

Índice

Esquema


PROGRAMACIÓN DE APLICACIONES PARA DISPOSITIVOS MÓVILES
HERRAMIENTAS
De Desarrollo
De Diseño
Documentación y Depuración
CICLO DE DESARROLLO
Compilación
Preverificación
Empaquetado
Desarrollo del código
Patrones de diseño
MVC (Modelo - Vista -Controlador)
MVVM (Modelo – Vista – Vista Modelo)
Estructura proyecto Android
manifest/
java/ o kotlin/
res/
build.gradle

Material de estudio

3.1. Introducción y objetivos

En el transcurso de este tema se realizará una introducción al desarrollo de aplicaciones móviles con Android Studio, analizando sus características y componentes principales, a estructura y fases de los proyectos, así como las herramientas más relevantes dentro de Android Studio.

Durante el desarrollo de este tema se deben conseguir los siguientes objetivos:

- Comprender los conceptos básicos relacionados con las aplicaciones en dispositivos móviles.
- Comprender la estructura de un proyecto Android.
- Conocer las fases de construcción de un proyecto Android.
- Comprender el desarrollo de las interfaces gráficas en Android.
- Conocer los sistemas de documentación y depuración en un proyecto Android.

3.2. Herramientas y fases de construcción

Desarrollar aplicaciones móviles para Android es un proceso complejo que involucra una combinación de herramientas de desarrollo y un flujo de trabajo bien estructurado, con varias fases claramente diferenciadas.

Herramientas de Desarrollo para Android

Existen numerosas herramientas que los desarrolladores de aplicaciones Android pueden utilizar a lo largo del ciclo de vida del desarrollo de software. Android Studio

proporciona herramientas esenciales para escribir, depurar, probar y construir aplicaciones para Android, entre las que se incluyen:

- Editor de código con autocompletado.
- Emuladores de dispositivos: permite probar la aplicación en varios dispositivos y versiones de Android sin necesidad de hardware físico.
- Herramientas de depuración: como el Logcat, que muestra logs en tiempo real, y herramientas de inspección de memoria y CPU.
- Sistema de compilación Gradle: gestiona la construcción, configuración, y dependencias del proyecto.
- SDK de Android (Android Software Development Kit): El SDK incluye bibliotecas, herramientas de línea de comandos, depuradores y ejemplos de código necesarios para desarrollar aplicaciones Android. Algunas de las utilidades que provee son:
    - adb (Android Debug Bridge): permite la interacción con dispositivos o emuladores Android desde la terminal, ejecutar comandos remotos, instalar aplicaciones, etc.
    - avd (Android Virtual Device): gestiona los dispositivos virtuales que se utilizan en el emulador de Android.
- Kotlin: Kotlin es el lenguaje de programación oficial para el desarrollo de Android (además de Java). Se prefiere debido a su concisión, seguridad y interoperabilidad con Java. La mayoría de las aplicaciones modernas están desarrolladas o migrando a Kotlin.
- JetPack compose: es un framework declarativo moderno diseñado por Google para simplificar y mejorar el desarrollo de interfaces de usuario (UI) en Android. Es parte de la familia Jetpack de herramientas, bibliotecas y componentes que ayudan a los desarrolladores a crear aplicaciones Android más rápidamente, de manera eficiente y con un código más limpio.
- Java: Aunque Kotlin ha ganado popularidad, Java sigue siendo ampliamente utilizado y es uno de los lenguajes más estables y maduros para Android. Muchas bibliotecas, marcos y herramientas están escritas en Java.
- Firebase: Es una plataforma ofrecida por Google que proporciona una variedad de servicios backend, como bases de datos en tiempo real, autenticación,

almacenamiento de archivos, análisis y notificaciones push. Ayuda a los desarrolladores a enfocarse más en la lógica de la aplicación que en la infraestructura.

- Git: Herramienta de control de versiones que es esencial para el desarrollo colaborativo. Repositorios como GitHub, GitLab o Bitbucket permiten a los equipos compartir y gestionar versiones de su código.
- Retrofit: Biblioteca popular para manejar solicitudes HTTP, que facilita la integración de APIs RESTful en las aplicaciones. Retrofit, junto con bibliotecas como Gson o Moshi, convierte automáticamente las respuestas JSON en objetos Kotlin o Java.
- Glide/Picasso: Estas son bibliotecas populares para cargar y manejar imágenes dentro de las aplicaciones Android. Son útiles para gestionar la caché, la decodificación y el manejo de imágenes desde URLs o almacenamiento local.
- Dagger/Hilt: Dagger es una herramienta para la inyección de dependencias (Dependency Injection) que permite que las aplicaciones sigan el principio de inversión de control (IoC), haciendo que el código sea más modular y fácil de probar. Hilt es la solución optimizada para Android.
- JUnit y Espresso: Son dos herramientas fundamentales para la creación de pruebas en Android.
- JUnit: es el marco estándar para la creación de pruebas unitarias.
- Espresso: es un framework para pruebas de interfaz de usuario (UI), utilizado para escribir y ejecutar pruebas en la interfaz de usuario de la aplicación de manera automática.

Herramientas de Diseño para Android

En Android Studio, las herramientas de diseño juegan un papel crucial para crear interfaces de usuario atractivas y funcionales de manera eficiente. Estas herramientas permiten a los desarrolladores visualizar, editar y probar los diseños de la interfaz (layouts), asegurando que la UI se adapte a diferentes dispositivos y tamaños de pantalla.

1. Layout Editor (Editor de Layouts)

El Layout Editor es una de las herramientas más utilizadas para diseñar la interfaz de usuario en Android Studio. Permite editar de manera visual los archivos XML que definen los layouts, facilitando la creación de interfaces sin necesidad de escribir todo el código manualmente.

Características del Layout Editor:

- Modo de diseño visual: Te permite arrastrar y soltar componentes (botones, textos, imágenes, etc.) desde la paleta al layout, organizar los componentes, y ver cómo se verá la UI en tiempo real.
- Modo de diseño XML: Ofrece la posibilidad de alternar entre la vista visual y el código XML del layout, permitiendo editar los atributos y las propiedades manualmente.
- Vista dividida: El Layout Editor permite ver simultáneamente el diseño visual y el XML en una vista dividida, lo que facilita la edición del código mientras ves los cambios reflejados en el diseño.
- Propiedades dinámicas: A medida que seleccionas un componente de la interfaz, en el panel de propiedades se muestran todos sus atributos (como ancho, alto, margen, color, etc.), los cuales puedes modificar directamente.

2. Palette (Paleta de Componentes)

La Palette es una herramienta que contiene todos los componentes de la interfaz de usuario que puedes usar en el diseño del layout. Estos componentes están agrupados por categorías, como botones, widgets de texto, contenedores, layouts, y otros componentes avanzados.

3. Component Tree (Árbol de Componentes)

El Component Tree es una herramienta que muestra la jerarquía de vistas y layouts dentro del diseño de la interfaz. Te permite ver cómo están organizados los componentes y seleccionar rápidamente cualquier componente para editarlo o modificar su estructura.

Características del Component Tree:

- Jerarquía visual: Muestra la estructura del diseño en forma de árbol, con layouts que contienen vistas y otros componentes. Puedes expandir y colapsar nodos para explorar la jerarquía de la interfaz.
- Fácil navegación: Te permite seleccionar cualquier componente en el árbol, y se resaltará automáticamente en el layout, lo que facilita la edición directa de sus propiedades.
- Reorganización de vistas: Puedes arrastrar y soltar componentes en el árbol para reordenarlos en el layout, lo cual es especialmente útil en layouts complejos.

4. ConstraintLayout y el Editor de Restricciones

ConstraintLayout es el contenedor de layout más avanzado en Android, y permite crear interfaces complejas y responsivas utilizando un sistema de restricciones que definen la posición de los elementos en relación con otros componentes o con el contenedor principal.

5. Preview (Previsualización)

La Preview o ventana de previsualización te permite ver cómo se verá la interfaz de usuario en varios dispositivos, tamaños de pantalla y orientaciones (vertical u horizontal) en tiempo real mientras editas el layout.

Características de la Previsualización:

- Simulación de dispositivos: Puedes seleccionar diferentes dispositivos, como teléfonos, tablets, o pantallas plegables, para ver cómo se ajustará tu interfaz en cada uno de ellos.
- Múltiples resoluciones: La previsualización permite probar la UI en varias resoluciones, densidades de pantalla (dpi) y configuraciones de orientación.
- Temas y estilos: También es posible previsualizar cómo se verá la interfaz con diferentes temas de la aplicación (por ejemplo, tema claro u oscuro).

6. Theme Editor (Editor de Temas)

El Theme Editor es una herramienta que facilita la creación y personalización de los temas y estilos visuales de la aplicación. Los temas en Android controlan la apariencia de los componentes de la interfaz, como colores, tipografías, fondos, etc.

7. Layout Inspector (Inspector de Layout)

El Layout Inspector es una herramienta que te permite inspeccionar en detalle la interfaz de usuario de una aplicación mientras está en ejecución, tanto en un dispositivo físico como en un emulador. Es especialmente útil para depurar problemas de diseño.

8. Motion Editor (Editor de MotionLayout)

El Motion Editor es una herramienta visual para crear animaciones complejas y transiciones entre diferentes estados de la interfaz utilizando MotionLayout. Esta herramienta facilita la creación de animaciones sin necesidad de escribir manualmente las animaciones en XML.

Funciones del Motion Editor:

- Diseño visual de animaciones: Permite definir y ajustar animaciones de transición entre diferentes estados de un componente (por ejemplo, mover un botón de una posición a otra, cambiar su tamaño, o animar una imagen).
- Timeline de animaciones: El Motion Editor proporciona una línea de tiempo que permite ver y ajustar visualmente la duración y el comportamiento de las animaciones.
- Interfaz basada en estados: Puedes definir varios "estados" de un componente, y MotionLayout se encargará de animar la transición entre estos estados.

9. Vector Asset Studio

El Vector Asset Studio te permite importar y crear gráficos vectoriales en formato SVG o VectorDrawable, que son escalables sin perder calidad y ocupan menos espacio que los gráficos rasterizados.

10. Font Studio

El Font Studio permite integrar fácilmente fuentes personalizadas en tu aplicación. Puedes importar nuevas fuentes desde archivos externos o desde la biblioteca de fuentes de Google Fonts.

Fases de Construcción de Aplicaciones Móviles en Android

El proceso de desarrollo de una aplicación Android generalmente sigue un ciclo de vida estructurado. Cada fase del proceso de construcción es crítica para garantizar que la aplicación final sea de alta calidad, funcional y esté alineada con las expectativas del usuario. Las fases clave incluyen:

1. Planificación y Requerimientos

- Objetivos del proyecto: Se determinan los requisitos de la aplicación, sus funcionalidades principales y los problemas que la aplicación resolverá.
- Investigación de mercado: Se analiza la competencia, las necesidades del usuario y las posibles características diferenciadoras que pueden ofrecerse.
- Selección de tecnologías: Elegir las tecnologías más adecuadas para el desarrollo. Esto puede implicar decidir entre Kotlin o Java, Firebase o un backend personalizado, y qué bibliotecas utilizar.

2. Diseño de la Aplicación

- Diseño de la arquitectura: Se define la estructura de la aplicación. Por ejemplo, seguir patrones como MVC (Model-View-Controller), MVVM (Model-View-ViewModel) o MVP (Model-View-Presenter), es crucial para asegurar la escalabilidad y el mantenimiento del código.
- Diseño UI/UX: Los diseñadores crean la interfaz de usuario (UI) y el diseño de la experiencia de usuario (UX). Herramientas como Figma o Sketch suelen utilizarse para prototipos de diseño, y luego estos se traducen a componentes Android como Views, Fragments y Activities.
- Creación de Mockups y Wireframes: Estos son bocetos o prototipos de la aplicación que ayudan a visualizar la interfaz y el flujo de la aplicación antes de comenzar el desarrollo.

3. Desarrollo

Esta es la fase principal en la que se escribe el código y se integran las funcionalidades. Se divide en varias sub-etapas:

- Estructuración del Proyecto: Crear la estructura base, los paquetes y los módulos de la aplicación.
- Desarrollo Frontend: Implica crear la interfaz de usuario en XML o usando Jetpack Compose. Se implementan componentes visuales y se gestionan eventos del usuario.
- Desarrollo Backend: Implementación de la lógica de la aplicación, integración con bases de datos, APIs y otras funciones como autenticación, gestión de datos locales, notificaciones, etc.
- Pruebas: Mientras se desarrolla, se suelen escribir pruebas unitarias con JUnit para asegurar que las clases y métodos individuales funcionan correctamente. También se pueden crear pruebas automatizadas para UI con Espresso o Robolectric.

4. Pruebas y Debugging

- Pruebas de calidad: Se realizan pruebas de integración y regresión para asegurar que las nuevas funcionalidades no rompan las existentes.
- Pruebas en dispositivos reales: Es importante probar la aplicación en múltiples dispositivos y versiones de Android debido a la fragmentación del ecosistema. Aunque los emuladores son útiles, los dispositivos físicos son esenciales para verificar el rendimiento real.
- Pruebas de usabilidad: Se realiza una evaluación de la experiencia del usuario final con la aplicación. Esto incluye la facilidad de navegación, el tiempo de respuesta y el diseño de la interfaz.

5. Despliegue y Publicación

- Optimización: Se revisa la aplicación para garantizar un rendimiento óptimo, ajustando aspectos como la gestión de la memoria, el uso de red, o el tiempo de carga.

- Firmado de la Aplicación: Antes de lanzar una aplicación, es necesario firmarla con una clave privada. Esta es una medida de seguridad requerida por Google Play.
- Publicación en Google Play Store: El proceso de despliegue incluye la creación de una cuenta de desarrollador, subir el APK o AAB, escribir descripciones, agregar capturas de pantalla, especificar la segmentación de la audiencia y seleccionar opciones de precios.

6. Mantenimiento y Actualizaciones

- Corrección de errores: Después del lanzamiento, es probable que se descubran errores que no fueron encontrados durante las pruebas. Se emiten actualizaciones para corregirlos.
- Actualizaciones de funcionalidad: A medida que los usuarios proporcionan retroalimentación o evolucionan los requisitos del negocio, se añaden nuevas características o mejoras a la aplicación.
- Mantenimiento de compatibilidad: A medida que se lanzan nuevas versiones de Android, la aplicación debe mantenerse compatible, lo que puede requerir ajustes o actualizaciones de código.

7. Monitoreo y Mejora Continua

- Análisis de uso: Herramientas como Firebase Analytics o Google Analytics se utilizan para monitorizar el comportamiento de los usuarios dentro de la aplicación, lo que ayuda a entender cómo interactúan los usuarios y qué áreas necesitan mejoras.
- Optimización continua: Basado en los datos obtenidos, se optimiza el rendimiento de la aplicación, la usabilidad y la retención de usuarios mediante nuevas actualizaciones.

3.3. Desarrollo del código

El desarrollo de código en un proyecto Android en Android Studio sigue un proceso estructurado que combina la escritura de código para la lógica de la aplicación (en Kotlin o Java), la definición de la interfaz de usuario (en XML), y la integración con herramientas y bibliotecas que permiten manejar el ciclo de vida, las interacciones y las funcionalidades de la aplicación.

Android Studio soporta la implementación de arquitecturas como MVVM (Model-View-ViewModel) y el uso de bibliotecas de Jetpack, como LiveData y Room, que ayudan a mantener una estructura más organizada y escalable.

MVVM (Model-View-ViewModel)

Este patrón de arquitectura es muy utilizado porque separa claramente la lógica de la interfaz de usuario (View) de la lógica de negocio (Model), mientras que el ViewModel actúa como intermediario que gestiona la lógica y los datos de la UI.

- ViewModel: Maneja los datos de la interfaz de usuario y los cambios en el estado.
- LiveData: Es un componente observable que actualiza la interfaz automáticamente cuando cambian los datos.
- Repository: Es una capa que maneja el acceso a datos, ya sea de la base de datos o de la red.

Estructura de un proyecto Android

Al crear un proyecto en Android Studio, este genera un conjunto de directorios y archivos que forman la base del proyecto. La estructura básica de un proyecto Android se divide en módulos, siendo el más común el módulo "app". El directorio app/ es el módulo principal de la aplicación, que contiene todo lo relacionado con la lógica del código, los recursos y la configuración específica del módulo de la

aplicación Android. Este es el módulo que se compila y se ejecuta en un dispositivo o emulador.

Estructura interna del módulo app/:

- manifests/
- java/ o kotlin/
- res/
- build.gradle (nivel de módulo)

1. manifests/ (Manifest de la Aplicación)

- Dentro de este directorio está el archivo AndroidManifest.xml, que es crucial para definir aspectos importantes de la aplicación. Aquí se declaran los componentes de la aplicación (como Activities, Services, BroadcastReceivers), los permisos que la aplicación requiere (como acceso a internet, cámara, etc.), y otras configuraciones como el tema de la aplicación o la configuración de las intents.

Ejemplo de un archivo AndroidManifest.xml donde se declara la actividad principal de la aplicación (MainActivity), así como el ícono, el nombre de la aplicación, y otros aspectos importantes.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.MyApp">

        <activity android:name=".MainActivity">
            <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name="android.intent.category.LAUNCHER" />
```

```
            </intent-filter>
        </activity>

    </application>
</manifest>
```

2. java/ y kotlin/ (Código Fuente)

Este directorio contiene el código fuente de la aplicación, ya sea en Java o en Kotlin, dependiendo del lenguaje que estés utilizando. En esta carpeta se organizan los paquetes y las clases que definen la lógica de la aplicación. Se recomienda organizar el código en paquetes para seguir el patrón de arquitectura que estés utilizando (por ejemplo, view, model, viewmodel si sigues el patrón MVVM).

3. res/ (Recursos)

El directorio res/ almacena todos los recursos no de código que la aplicación necesita. Estos recursos incluyen los layouts de la interfaz, imágenes, cadenas de texto, colores, estilos, etc. Android gestiona automáticamente los recursos para diferentes dispositivos y configuraciones (como tamaños de pantalla, densidades, idiomas).

Subdirectorios comunes dentro de res/:

- layout/: Contiene los archivos XML que definen los layouts (interfaces de usuario).
    - Ejemplo: activity_main.xml, donde se definen los componentes visuales que conforman una pantalla.
- drawable/: Contiene imágenes y gráficos vectoriales.
    - Ejemplo: Archivos .png, .jpg, o VectorDrawable (.xml).
- values/: Contiene archivos XML que definen valores estáticos como cadenas de texto (strings.xml), colores (colors.xml), estilos (styles.xml), y dimensiones (dimens.xml).
- mipmap/: Contiene los iconos de la aplicación en diferentes densidades de pantalla (mdpi, hdpi, xhdpi, etc.).
- menu/: Define menús de la aplicación a través de archivos XML (menu.xml).

4. Archivos build.gradle

Los archivos build.gradle son scripts de configuración del sistema de construcción Gradle. En un proyecto Android típico, hay dos archivos build.gradle: uno a nivel del proyecto y otro a nivel del módulo app.

- build.gradle (Nivel de proyecto): Define configuraciones globales y dependencias compartidas entre módulos. Ejemplo:

```
buildscript {
    repositories {
        google()
        mavenCentral()
    }
    dependencies {
        classpath "com.android.tools.build:gradle:8.0.0"
    }
}
```

- build.gradle (Nivel de módulo app): Configura los detalles específicos de la aplicación, como el SDK, versiones, dependencias y tareas de compilación. Ejemplo:

```
android {
    compileSdkVersion 33
    defaultConfig {
        applicationId "com.example.myapp"
        minSdkVersion 21
        targetSdkVersion 33
        versionCode 1
        versionName "1.0"
    }
    buildTypes {
        release {
            minifyEnabled false
        }
    }
```

```
}

dependencies {
    implementation "androidx.core:core-ktx:1.8.0"
    implementation "com.google.android.material:material:1.5.0"
}
```

Aquí se configuran las dependencias, como bibliotecas de Android Jetpack, y las configuraciones del SDK y versiones de la aplicación.

5. Otros Archivos y Directorios

- settings.gradle: Lista los módulos que forman parte del proyecto. Por lo general, tiene una entrada que apunta al módulo app.
- gradle.properties: Configura propiedades globales de Gradle, como optimizaciones de compilación o configuraciones específicas del entorno de construcción.
- local.properties: Contiene configuraciones específicas del entorno local, como la ubicación del SDK de Android en tu máquina.

6. Módulos Adicionales

En proyectos grandes, es común que haya múltiples módulos dentro de un proyecto Android. Cada módulo puede representar una biblioteca o una parte independiente de la aplicación (como un módulo de red, un módulo de base de datos, etc.).

Tipos de módulos:

- Módulo de aplicación (app/): Es el módulo principal que contiene el código de la aplicación.
- Módulos de bibliotecas (library/): Permiten reutilizar código en diferentes partes de la aplicación o incluso compartirlo entre proyectos.

3.4. Compilación, preverificación, empaquetado y ejecución

En el desarrollo de aplicaciones Android en Android Studio, el proceso de construcción (build) y ejecución de una aplicación incluye varias fases clave: compilación, preverificación, empaquetado y ejecución. Estas fases aseguran que el código fuente de la aplicación se transforme en un archivo instalable (APK o AAB) que puede ejecutarse en dispositivos Android. Android Studio gestiona este ciclo de vida utilizando Gradle, que es la herramienta de automatización de construcción.

A continuación, se explica cada una de estas fases:

Compilación

La compilación es la fase en la que el código fuente (generalmente escrito en Java o Kotlin) se convierte en un formato que pueda entender la máquina virtual Android (Android Runtime o ART). Este proceso incluye varios subprocesos como la compilación del código fuente, la compilación de recursos y la generación de archivos intermedios.

Subfases del proceso de compilación:

1. Compilación de código fuente (Java/Kotlin):

- Los archivos .java y .kt son compilados en bytecode que puede ejecutarse en la Máquina Virtual de Java (JVM). Para Kotlin, se utiliza el compilador de Kotlin, mientras que para Java se usa el compilador de Java.
- Android no ejecuta directamente el bytecode de JVM, sino que convierte este bytecode en un formato más optimizado para dispositivos Android, llamado DEX (Dalvik Executable).

2. Generación de bytecode DEX:

- Una vez que el código se ha compilado en bytecode de la JVM, se convierte en archivos .dex. Estos archivos son específicos del entorno Android y están optimizados para ejecutarse en la máquina virtual Dalvik o Android Runtime (ART). Las herramientas de Android utilizan dx o d8 para convertir el bytecode en archivos DEX.
- Los archivos DEX son los que finalmente se ejecutan en el dispositivo Android, y es importante notar que a partir de Android 5.0, la ART optimiza el código DEX para mejorar el rendimiento utilizando un proceso llamado Ahead-of-Time (AOT).

3. Compilación de recursos:

- Android no solo compila el código fuente, sino también los recursos (layouts, imágenes, strings, etc.). En esta fase, los archivos XML como los layouts y las definiciones de actividades se procesan y se generan los archivos binarios correspondientes.
- AAPT (Android Asset Packaging Tool): Es la herramienta que se utiliza para compilar y empaquetar todos los recursos. Se encarga de convertir los archivos XML en un formato binario y comprimir imágenes y otros recursos.
- Esta fase también incluye la generación del archivo R.java o su equivalente en Kotlin, que actúa como un índice de todos los recursos de la aplicación, permitiendo a los desarrolladores acceder a estos recursos en el código.

4. Herramientas utilizadas:

- Gradle: Es el motor que organiza todas estas tareas de compilación. Gradle define la estructura del proyecto, las dependencias y las configuraciones de compilación.
- Compilador de Kotlin/Java: Convierte el código fuente en bytecode para JVM.
- D8/R8: Herramientas para la optimización y conversión de bytecode a DEX.

Preverificación

La preverificación es una fase en la que se verifica que el código generado es válido antes de empaquetarlo en un APK o AAB. Aunque no es un proceso separado y definido en todos los sistemas, en el contexto de Android se refiere a la validación de

varias reglas para asegurar que el código y los recursos generados cumplan con las especificaciones del sistema Android.

Verificaciones típicas:

- Validación de bytecode: Se asegura que el bytecode generado es válido para el entorno de ejecución de Android (ART o Dalvik). Esto puede incluir la validación del formato DEX, asegurando que no haya errores en la transformación del bytecode.
- Verificación de referencias: En esta fase se asegura que todas las referencias a métodos, clases y recursos son correctas, y que no se accede a bibliotecas o APIs que no sean compatibles con la versión objetivo de Android.
- Verificación de permisos: Verificación de que la aplicación tiene declarados correctamente los permisos necesarios en el archivo AndroidManifest.xml.

Empaquetado

El empaquetado es la fase en la que todo el código compilado, los archivos DEX, los recursos y las dependencias se combinan en un archivo ejecutable que se puede instalar en un dispositivo Android. Hay dos formatos principales de empaquetado:

- APK (Android Package Kit): Es el formato tradicional para distribuir aplicaciones Android. Contiene todos los archivos necesarios para instalar y ejecutar la aplicación en un dispositivo.
- AAB (Android App Bundle): Es un formato más reciente que optimiza el tamaño de la aplicación. AAB permite que Google Play genere APK específicos para cada dispositivo, lo que reduce el tamaño de descarga y la cantidad de recursos innecesarios en el dispositivo final.

Pasos en el empaquetado:

1. Firma de la aplicación:

- Android requiere que todas las aplicaciones estén firmadas digitalmente antes de que se puedan instalar en un dispositivo. La firma garantiza la autenticidad e integridad de la aplicación.
- Durante el desarrollo, Android Studio utiliza una clave de depuración para firmar la aplicación automáticamente. Sin embargo, para publicar una aplicación en Google Play, se requiere firmar la aplicación con una clave de producción.

2. Generación de manifest:

El archivo AndroidManifest.xml es procesado durante esta fase. Este archivo define las configuraciones principales de la aplicación, como los permisos necesarios, las actividades principales y los servicios. Gradle se asegura de que el AndroidManifest.xml contenga toda la información correcta y esté optimizado.

3. Compresión y empaquetado de archivos:

- Todos los archivos DEX, los recursos compilados (imágenes, XML binarios, etc.), y las bibliotecas (JAR o AAR) se empaquetan juntos en el APK o AAB.
- AAPT empaqueta los recursos y los archivos compilados, y finalmente crea el APK. En el caso de AAB, este proceso genera un archivo modular que contiene todos los recursos optimizados para diferentes dispositivos.

Ejecución

Una vez que el APK o AAB ha sido generado, la fase de ejecución se encarga de instalar la aplicación en un dispositivo físico o emulador y lanzarla.

Detalles del proceso:

1. Instalación:

- En Android Studio, la instalación de la aplicación en el dispositivo es automática cuando se presiona el botón "Run". Esto utiliza ADB (Android Debug Bridge), una herramienta que permite comunicarse con dispositivos Android conectados por USB o red.

- ADB instala el APK en el dispositivo o emulador y, opcionalmente, inicia la aplicación una vez que se completa la instalación.

2. Inicio de la aplicación:

- Android Studio envía un comando a través de ADB para iniciar la actividad principal definida en el archivo AndroidManifest.xml.
- Durante la ejecución, Android Studio se puede conectar al proceso de la aplicación para proporcionar herramientas de depuración en tiempo real como el depurador, Logcat, e inspección de memoria.

3. Depuración en tiempo real:

Durante la fase de ejecución, Android Studio permite realizar una depuración en vivo del código utilizando herramientas como breakpoints, Logcat y el Android Profiler para monitorear el rendimiento de la aplicación.

Resumen del Ciclo de Construcción de una Aplicación Android

1. Compilación: El código fuente se transforma en bytecode JVM y luego en archivos DEX específicos de Android. Los recursos también se compilan y se integran en el paquete final.
2. Preverificación: Validación del código generado y los recursos, asegurando que cumplen con los requisitos de Android.
3. Empaquetado: El código compilado, los recursos y las dependencias se empaquetan en un archivo APK o AAB, firmado digitalmente para garantizar la seguridad.
4. Ejecución: El APK se instala y ejecuta en un dispositivo Android o emulador, permitiendo la depuración en tiempo real.

El uso de Gradle como herramienta de compilación automatiza todo este proceso y proporciona configuraciones específicas para diferentes entornos (depuración, liberación, variantes de la aplicación, etc.), facilitando el manejo y la personalización del flujo de trabajo de construcción en Android Studio.

3.5. Interfaces de usuario. Clases asociadas

Las interfaces de usuario (UI) en un proyecto Android juegan un papel fundamental, ya que determinan cómo los usuarios interactúan con la aplicación. En Android, el diseño de la interfaz de usuario se basa en una combinación de layouts, vistas y componentes de IU que son definidos en archivos XML o programáticamente en el código Kotlin/Java con Jetpack compose. Estas interfaces están ligadas a clases asociadas que permiten gestionar la lógica y el comportamiento de los elementos de la interfaz.

Layouts (Diseños) en Android

Los layouts son la estructura básica de las interfaces en Android y definen cómo se organizan visualmente los componentes dentro de la pantalla. Un layout es un contenedor que puede contener otros layouts y vistas, organizando su disposición y comportamiento.

Tipos comunes de Layouts:

- LinearLayout: Organiza los elementos en una sola fila (horizontal) o en una sola columna (vertical). Es útil para interfaces simples que requieren una alineación ordenada de componentes.
- RelativeLayout: Permite organizar componentes relativos entre sí o relativos al contenedor principal. Es flexible para posicionar elementos con relación a otros, pero puede volverse complejo cuando hay muchos componentes.
- ConstraintLayout: Es el layout más avanzado y flexible, que permite alinear vistas utilizando restricciones (constraints). Es ideal para crear interfaces complejas con un rendimiento optimizado, ya que minimiza la cantidad de anidamientos entre vistas.

- FrameLayout: Es un layout simple que contiene una sola vista, pero las vistas adicionales se superponen unas sobre otras. Útil para situaciones donde las vistas deben aparecer una encima de otra.
- RecyclerView: No es un layout en sí, pero es una vista muy usada para mostrar listas grandes de datos de forma eficiente, utilizando un patrón de reciclado de vistas.

En un proyecto Android, las clases relacionadas con la interfaz de usuario se encargan de controlar la interacción, gestionar la lógica y actualizar los elementos visuales. Las clases más importantes son Activity, Fragment, y componentes como View y ViewModel.

1. Activity

Una Activity representa una pantalla completa en una aplicación Android. Es una de las clases principales que controla la interfaz de usuario y gestiona el ciclo de vida de los componentes de la pantalla.

- Cada actividad tiene un archivo de layout asociado, que define la UI, y la actividad se encarga de manejar las interacciones del usuario con los componentes de esa pantalla.
- La clase Activity en sí misma extiende de Context, lo que le permite interactuar con el sistema operativo para cosas como iniciar nuevas actividades o acceder a recursos.

Ejemplo básico de una Activity:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val button: Button = findViewById(R.id.button)
        button.setOnClickListener {
            // Acción cuando el botón es presionado
```

```
            Toast.makeText(this,            "Botón            presionado",
Toast.LENGTH_SHORT).show()
        }
    }
}
```

2. Fragment

Un Fragment es un componente reutilizable de la interfaz de usuario que representa una parte de la interfaz dentro de una actividad. Los fragments permiten dividir una actividad en componentes más pequeños y modulares, lo que facilita el manejo de pantallas complejas o la reutilización de la UI.

- Los fragments tienen su propio ciclo de vida y su propia lógica de UI, lo que permite que diferentes pantallas se gestionen dentro de una única actividad.
- Son útiles en interfaces dinámicas y pantallas que deben cambiar sin necesidad de recrear toda la actividad.

3. View y Custom Views

La clase View es la clase base de todos los componentes visuales. Cada componente visual, como botones, textos o imágenes, hereda de View.

También es posible crear vistas personalizadas (Custom Views) al extender la clase View o subclases existentes y definir comportamientos o diseños personalizados.

Ejemplo básico de Custom View:

```
class MyCustomView(context: Context, attrs: AttributeSet?) : View(context,
attrs) {
    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        // Dibujar contenido personalizado
        canvas.drawColor(Color.RED)
    }
}
```

4. ViewModel

En el patrón MVVM (Model-View-ViewModel), que es comúnmente utilizado en aplicaciones Android con Jetpack, el ViewModel actúa como intermediario entre la interfaz de usuario (Vista) y los datos (Modelo). El ViewModel mantiene el estado de la UI a través de cambios de configuración, como la rotación de la pantalla, y gestiona la lógica de negocio.

Los ViewModels son particularmente útiles para evitar que las actividades o los fragments manejen directamente los datos, promoviendo una separación más limpia de responsabilidades.

En este ejemplo, el ViewModel mantiene un contador que se puede actualizar y observar desde la actividad o fragment.

```
class MyViewModel : ViewModel() {
    val counter = MutableLiveData<Int>()

    fun increment() {
        counter.value = (counter.value ?: 0) + 1
    }
}
```

5. LiveData

LiveData es una clase que permite manejar datos que pueden ser observados. Los componentes de la UI, como una actividad o fragment, pueden suscribirse a los cambios en un LiveData, y este actualizará la UI automáticamente cuando los datos cambien.

3.6. Depuración y documentación

La depuración es una parte crucial del desarrollo, ya que permite identificar y corregir errores en el código y mejorar el rendimiento general de la aplicación. Android Studio

proporciona varias herramientas integradas que ayudan a los desarrolladores a encontrar y solucionar errores de manera eficiente.

1. Logcat

Logcat es una herramienta de registro en tiempo real que muestra mensajes generados por el sistema Android y las aplicaciones en ejecución. A través de Logcat, los desarrolladores pueden ver errores, advertencias y mensajes personalizados que ayudan a identificar problemas.

Puedes filtrar los mensajes por nivel de severidad (info, warning, error), por nombre de paquete, o por palabras clave específicas.

Los mensajes de Logcat se generan utilizando la clase Log en el código Java o Kotlin, que permite escribir mensajes en diferentes niveles: Log.d() (debug), Log.e() (error), Log.w() (warning), entre otros.

```
Log.d("MainActivity", "This is a debug message")
Log.e("MainActivity", "An error occurred", exception)
```

2. Debugger (Depurador)

El depurador de Android Studio permite pausar la ejecución de la aplicación, inspeccionar el estado de las variables y objetos, y avanzar paso a paso a través del código. Las características clave del depurador incluyen:

- Breakpoints (Puntos de interrupción): Permiten detener la ejecución del programa en una línea específica para inspeccionar el estado de las variables o ver qué camino está tomando el código.
- Step Into, Step Over y Step Out: Estas opciones permiten avanzar paso a paso dentro de las funciones, saltar sobre ellas o salir de una función para inspeccionar el flujo de ejecución.
- Evaluar expresiones: Puedes evaluar variables o expresiones directamente desde el depurador para ver su valor en tiempo real.

- Depuración Condicional: Los breakpoints condicionales permiten detener el programa solo si se cumplen ciertas condiciones, como cuando una variable tiene un valor específico.
- Depuración Remota: Android Studio permite depurar aplicaciones que se ejecutan en dispositivos físicos conectados o en emuladores, e incluso depurar dispositivos de forma remota conectándose a ellos a través de ADB (Android Debug Bridge).

Ejemplo de uso básico del debugger:

- Coloca un breakpoint haciendo clic en el margen izquierdo de una línea de código.
- Inicia la aplicación en modo depuración haciendo clic en el botón "Debug" en lugar de "Run".
- Cuando la ejecución se detiene en el breakpoint, puedes inspeccionar el estado de las variables, avanzar en el código, y modificar el estado directamente si es necesario.

3. Android Profiler

Android Profiler es un conjunto de herramientas avanzadas de análisis de rendimiento, que permiten a los desarrolladores medir y optimizar la utilización de recursos como CPU, memoria, red y energía.

- CPU Profiler: Mide el uso de la CPU y permite visualizar qué métodos están consumiendo más recursos. Permite hacer un perfil de la ejecución del código, detectar cuellos de botella y optimizar el rendimiento.
- Memory Profiler: Ayuda a rastrear el uso de la memoria en tiempo real, detectando fugas de memoria (memory leaks) y gestionando el uso de objetos. Permite realizar un análisis de "heap" para ver qué objetos están en uso y cuáles han sido recolectados por el Garbage Collector.
- Network Profiler: Mide el uso de la red, monitorea solicitudes y respuestas HTTP, y verifica el tamaño y la latencia de los paquetes de datos. Esto es esencial para identificar problemas relacionados con la eficiencia de las conexiones de red.

- Energy Profiler: Permite analizar cómo la aplicación consume energía, lo cual es vital para mejorar la duración de la batería en los dispositivos.

4. Inspección de Layout

- Layout Inspector: Es una herramienta que permite inspeccionar visualmente la jerarquía de vistas de una aplicación en ejecución. Ayuda a ver cómo se estructuran los elementos visuales y permite ajustar sus propiedades. Es útil cuando se trata de depurar problemas de diseño, como vistas superpuestas, márgenes incorrectos o elementos fuera de lugar.
- Layout Validation: Permite visualizar cómo se ve la interfaz de usuario en diferentes dispositivos y tamaños de pantalla, ayudando a detectar problemas de compatibilidad entre distintos tamaños o densidades de pantalla.

5. Emulador de Android

El emulador de Android en Android Studio permite ejecutar y depurar aplicaciones en un entorno virtual sin la necesidad de usar un dispositivo físico. Esto es útil para probar la aplicación en diferentes versiones de Android y tamaños de pantalla. El emulador incluye soporte para herramientas como Logcat y el depurador.

6. Crashlytics (Firebase)

Crashlytics, parte de Firebase, es una herramienta útil para monitorear y rastrear fallos en la aplicación una vez que está en producción. Proporciona información detallada sobre los errores que ocurren en los dispositivos de los usuarios, lo que permite a los desarrolladores diagnosticar y solucionar problemas en versiones lanzadas de la aplicación.

7. Lint (Herramienta de análisis estático)

Android Studio tiene integrado Lint, una herramienta de análisis estático que analiza el código fuente para detectar errores potenciales, problemas de rendimiento, y problemas de accesibilidad. Lint puede identificar patrones peligrosos o problemáticos antes de ejecutar la aplicación, ayudando a los desarrolladores a mantener un código limpio y eficiente.

Documentación en proyectos Android

La documentación es esencial para asegurar que el código sea comprensible y fácil de mantener a lo largo del tiempo, tanto para el propio desarrollador como para otros miembros del equipo.

1. Comentarios en el Código

- Comentarios básicos: Son anotaciones que explican partes específicas del código. Se utilizan para hacer que el código sea más legible y comprensible, especialmente en casos donde la lógica es compleja.
- Comentarios de bloque: Se utilizan para explicar partes más largas del código o para hacer anotaciones de alto nivel.

2. KDoc (Kotlin Documentation)

KDoc es el sistema estándar para escribir documentación en proyectos Kotlin. Se parece mucho a Javadoc (usado en Java), pero con algunas diferencias específicas para Kotlin. KDoc permite generar documentación a partir de comentarios estructurados, haciendo que sea fácil de leer y comprender.

Los comentarios de KDoc se colocan encima de clases, métodos y propiedades, y pueden incluir descripciones, detalles de parámetros y tipos de retorno.

Ejemplo de KDoc:

```
/**
 * Calcula el área de un rectángulo dado su ancho y su altura.
 *
 * @param width El ancho del rectángulo
 * @param height La altura del rectángulo
 * @return El área calculada del rectángulo
 */
fun calculateArea(width: Int, height: Int): Int {
    return width * height
}
```

Las etiquetas comunes en KDoc incluyen:

@param para describir parámetros de la función.
@return para describir lo que la función devuelve.
@throws para documentar excepciones que una función puede lanzar.

3. Documentación en la Wiki o README

Para proyectos más grandes, es común mantener documentación adicional fuera del código, como en un archivo README.md o en una Wiki dentro del repositorio de control de versiones (GitHub, GitLab). Esta documentación puede incluir:

- Instrucciones de configuración: Cómo clonar, configurar y ejecutar el proyecto localmente.
- Guías de arquitectura: Explicaciones de cómo está estructurada la aplicación (por ejemplo, usando patrones como MVVM o MVP).
- Dependencias: Listado de bibliotecas utilizadas y su propósito.
- Guías para contribución: Normas para colaborar en el proyecto, incluyendo estándares de código, revisión de pull requests, etc.

4. Javadoc

Aunque KDoc es la herramienta de documentación nativa para Kotlin, Javadoc sigue siendo útil en proyectos Java o en proyectos que mezclan Java y Kotlin. Javadoc genera automáticamente la documentación HTML a partir de comentarios en el código.

3.7. Referencias bibliográficas

Eixarch, R. P. (2023). *Kotlin y Jetpack Compose. Desarrollo de aplicaciones Android (Profesional).* Ra-Ma, S.A.

Google. (11 de 07 de 2024). *Android Developers*. Obtenido de https://developer.android.com/

JetBrains. (11 de 07 de 2024). *Kotlin*. Obtenido de https://kotlinlang.org/

Leiva, A. (2016). *Kotlin for Android Developers: Learn Kotlin the easy way while developing and Android App.* CreateSpace Independent Publishing Platform.

Trivedi, H. (2020). *Android application development with Kotlin: Build Your First Android App In No Time.* BPB Publications.

A fondo

Layout Editor

https://developer.android.com/studio/write/layout-editor?hl=es-419 Documentación en línea. (2024)

Documentación oficial sobre el Layout Editor, herramienta oficial de Android Studio para el diseño de interfaces.

Testing en Android

https://cursokotlin.com/testing-en-android-test-unitarios/ Documentación en línea. (2022)

El testing en Android es un aspecto crucial del desarrollo de aplicaciones, ya que garantiza la calidad, estabilidad y funcionalidad de una aplicación antes de su lanzamiento. A medida que las aplicaciones móviles crecen en complejidad, las pruebas se vuelven fundamentales para asegurar que funcionen correctamente en una amplia variedad de dispositivos, versiones de Android, y situaciones del mundo real.

Opciones para desarrolladores en dispositivos Android

https://developer.android.com/studio/debug/dev-options?hl=es-419 Documentación en línea. (2024)

Las Opciones para desarrolladores en dispositivos Android son un conjunto de configuraciones avanzadas que permiten a los desarrolladores acceder a funcionalidades y herramientas necesarias para probar, depurar y optimizar sus aplicaciones. Estas opciones se encuentran desactivadas de manera predeterminada

en la mayoría de los dispositivos, ya que están diseñadas principalmente para tareas técnicas durante el desarrollo de software.

Entrenamientos

Entrenamiento 1

- Desarrollar una app Android Studio con Kotlin que disponga de un TexView y un botón. Cada vez que se pulsa el botón el TextView debe de indicar cuantas pulsaciones se han realizado.
- Desarrollo paso a paso:
    - Crear proyecto Android de Activity vacia con el diseño basado en Views.
    - Usar ConstraintLayout
    - Colocar el Texview centrado y en la parte superior.
    - Definir un texto inicial
    - Definir un margen superior para el TextView.
    - Definir un id para el TextView.
    - Colocar un Button en el centro de la pantalla y definir tu texto, por ejemplo Púlsame !!
    - Cambiar el color del background.
    - Definir un id para el Button.
    - Definir en la MainActivity una variable pulsaciones.
    - Capturar el evento Click del botón.
    - Mostrar en el TextView el numero de pulsaciones acumuladas.
- Solución: https://github.com/Anuar-UNIR/PMDM_2024-2025/tree/main/Tema%203/Entrenamiento_1

Entrenamiento 2

- Desarrollar una app Android Studio con Kotlin que simule la vista de un log in:

- Desarrollo paso a paso:
    - Crear proyecto Android de Activity vacía con el diseño basado en Views.
    - Usar ConstraintLayout para todos los componentes
    - La App consta de los siguientes componentes.
        - View
        - TextView (LOG IN)
        - TextView (Nombre de usuario)
        - EditTextView (para introducir el email)
        - TextView (Contraseña)
        - EditTextView (para introducir la contraseña)
        - Button

- Personalizar los colores.
- Usar los ficheros colors.xml y strings.xml.
- Definir en la MainActivity una variable pulsaciones.
- Capturar el evento Click del botón y mostrar la información del email y contraseña

► Solución: https://github.com/Anuar-UNIR/PMDM_2024-2025/tree/main/Tema%203/Entrenamiento_2

Entrenamiento 3

► A partir de la App del entrenamiento 2, hay que añadir una nueva activity que se cargue después de pulsar el botón de logIn.

► Desarrollo paso a paso:

- Generar una nueva empty Activity.
- Añadir un TextView.
- Generar un objeto de Intent()
- Añadir los parámetros para pasar a la otra Activity
- Leer los parámetros
- Mostrar los valores del login en el TextView

- Solución: https://github.com/Anuar-UNIR/PMDM_2024-2025/tree/main/Tema%203/Entrenamiento_3

Entrenamiento 4

- Desarrollar una app Android para practicar con los estilos y temas en Android Studio
- Desarrollo paso a paso: https://developer.android.com/codelabs/basic-android-kotlin-training-change-app-theme?hl=es-419#0
- Solución: https://github.com/Anuar-UNIR/PMDM_2024-2025/tree/main/Tema%203/Entrenamiento_4

Entrenamiento 5

- Desarrollar una app Android libre con todo lo visto hasta ahora, usando diferentes componentes, experimentando con sus atributos.
- Desarrollo paso a paso: Se trata de un proyecto libre pero la app debería tener:
    - Uso de temas y estilos
    - Botones, textViews, EditText, y/o cardviews
    - Uso de Constrains Layout
- Solución: https://github.com/Anuar-UNIR/PMDM_2024-2025/tree/main/Tema%203/Entrenamiento_5