78. Programación Android

Para programar en Android podemos utilizar Java y la descripción del Interfaz de Usuario (IU) en ficheros XML. Desde hace unos años Google eligió Kotlin como lenguaje recomendado para los desarrollos Android y tenmos la posibilidad de usar Jetpack Compose para el IU

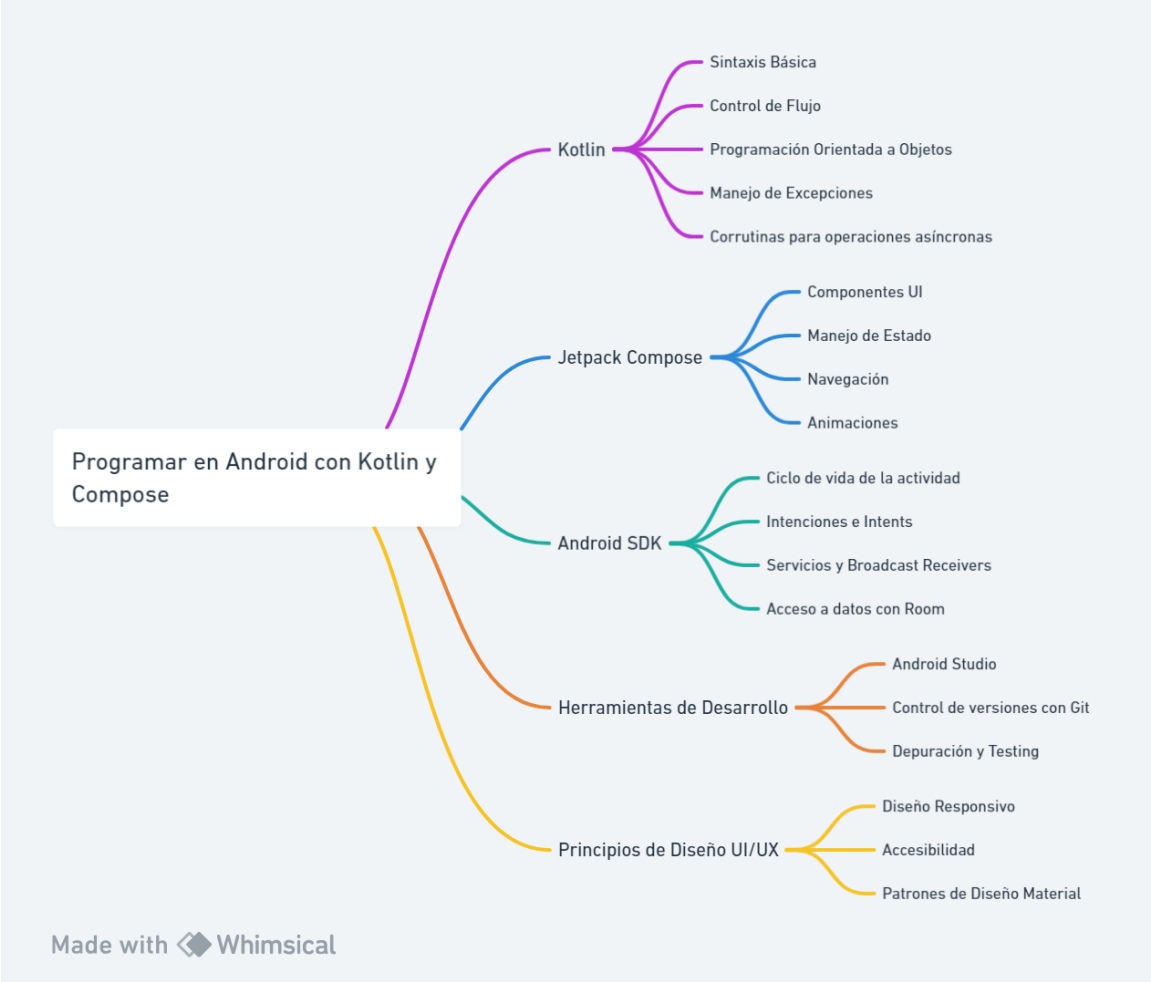
En la web Developer Android se encuentran tanto la documentación como tutoriales y recursos para desarrollar en Android.

78.1 Mapa del curso

En este módulo veremos como programar aplicaciones en Android, el interfaz de usuario y demás técnicas utilizando Kotlin y Jetpack Compose. También veremos como se construyen IU usando XML.

78.2 Indice en Android Developer

General:



(G. interactivo)

Compose:

78.3 Desarrollo de aplicaciones Android. IDE Android Studio

Para empezar nos debemos familiarizar con el IDE utilizado para el desarrollo en Android.

78.3.1 Instalar Android Studio, el IDE para desarrollar en Android

Una vez descargado, se instala siguiendo estas instrucciones, que para Linux sigue estos pasos: 1. Descomprimir el fichero .zip y copiarlo a */usr/local* o bien */opt/* .
2. Se inicia Android Studio ejecutando **studio.sh** 3. Continuar con el asistene "wizard" de instalacion

78.3.2 Introducción a Android Studio

Está basado tanto el editor como las herramientas en Intelij Idea, con estas características:

- Un sistema de compilación flexible basado en **Gradle**
- Un **emulador** rápido y cargado de funciones

- Un **entorno unificado** donde puedes desarrollar para todos los dispositivos Android
- Ediciones en vivo para actualizar elementos componibles en emuladores y dispositivos físicos, en tiempo real
- Integración con **GitHub** y plantillas de código para ayudarte a compilar funciones de apps comunes y también importar código de muestra
- Variedad de marcos de trabajo (frameworks) y herramientas de prueba, por ejemplo **JUnit**
- Herramientas de **Lint** para identificar problemas de rendimiento, usabilidad y compatibilidad de versiones, entre otros
- Compatibilidad con C++ y NDK
- Compatibilidad integrada con Google Cloud Platform, que facilita la integración con Google Cloud Messaging y App Engine (Herramientas Firebase de Google)

78.3.3 Sobre Git y github

El sistema de control de versiones más utilizado e integrado en Android Studio es **git**

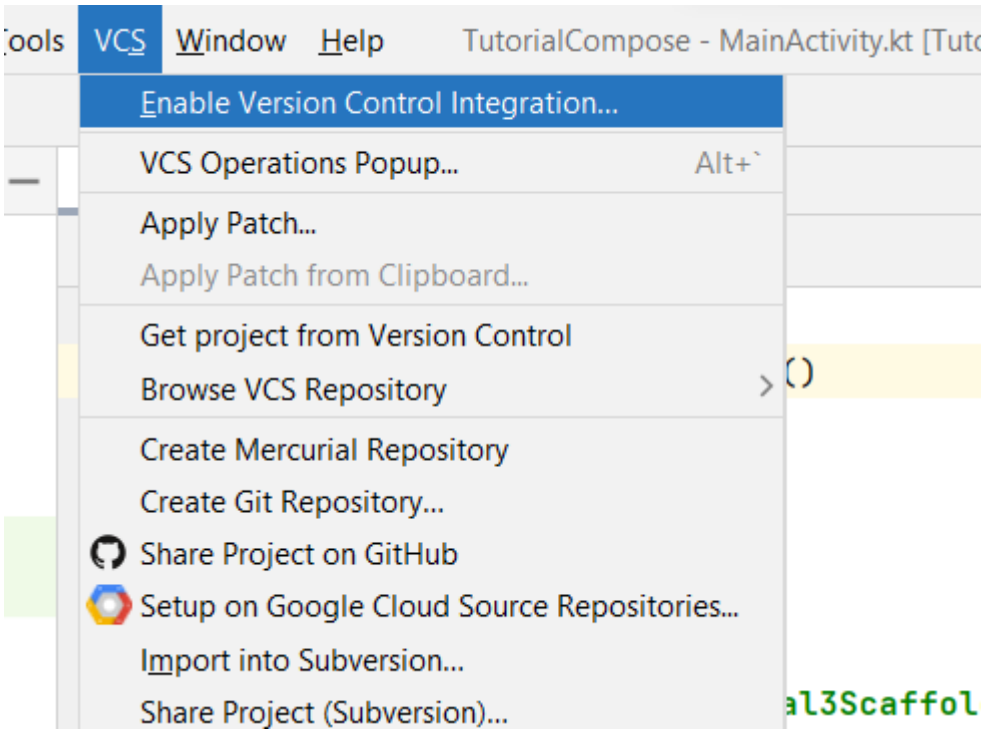
Conviene tener claros los conceptos y como funciona, para ello seguir esta web: Git + inmersión , que es un tutorial paso a paso con los conceptos básicos.

GitHub es una plataforma de desarrollo colaborativo que utiliza el sistema de control de versiones Git. Permite a los desarrolladores almacenar repositorios de código, colaborar con otros, y facilita el seguimiento de cambios en el código. GitHub ofrece funcionalidades como la gestión de proyectos, la documentación, y opciones de integración continua, entre otros. Es ampliamente utilizado en la industria del software para el desarrollo de proyectos de código abierto y privados.
Podemos usar Github como servidor git. En Android Studio tenemos la herramienta que permite utilizar git desde el IDE.

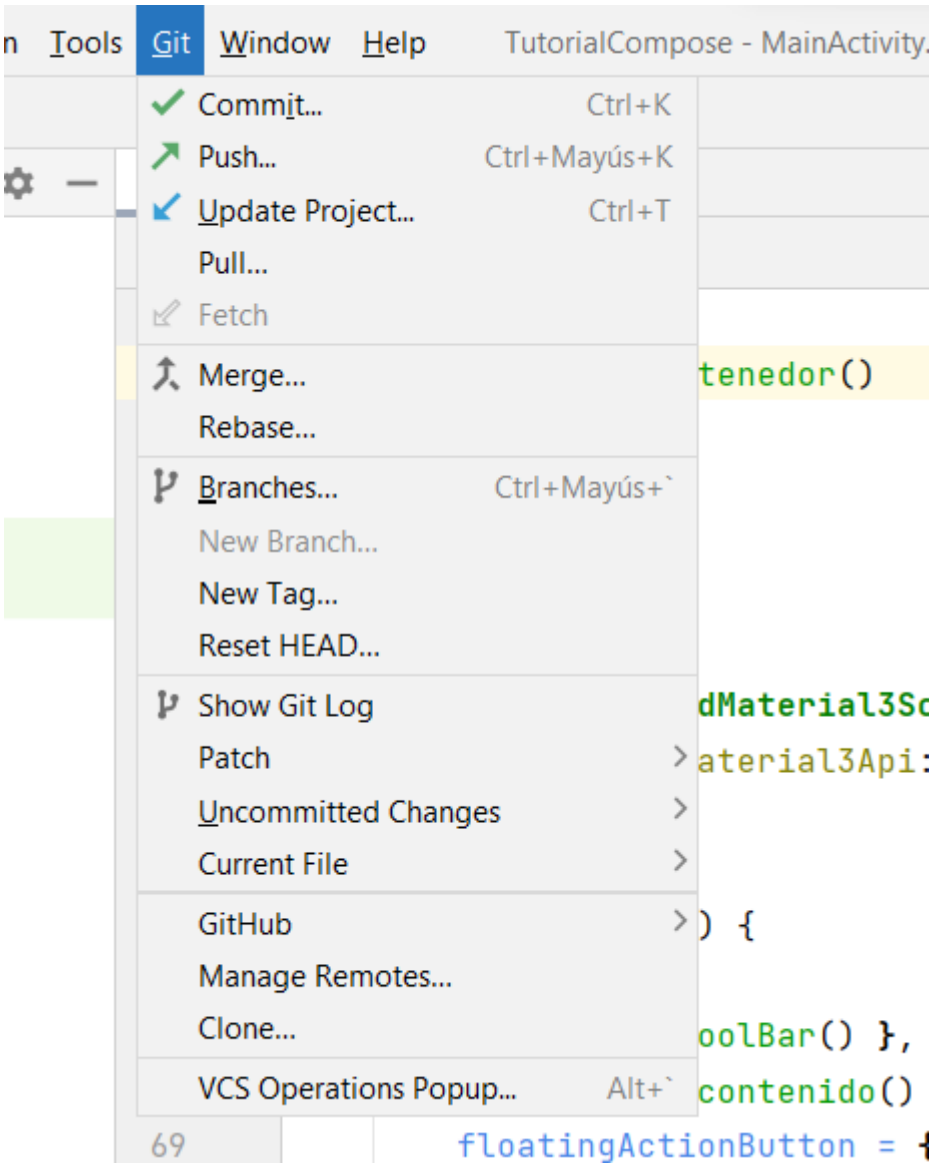
[TODO] Configurar git como repositorio de Github.

78.3.3.1 Git y Github desde Android Studio

Primeramente es necesario activar la integración con el control de versiones desde el menú de *VSC*



Una vez activado se pueden usar desde el menú *git* todas las funciones git:



78.3.3.1.1 SUBIR PROYECTOS A GITHUB

Para este curso utilizamos la la organización **Villablanca-dam**

78.3.4 Sobre Gradle

Gradle ss una herramienta de construcción automática y se utiliza para automatizar tareas repetitivas, como la compilación. Otra herramienta similar es **Maven**

Ver en esta página

78.4 Primer Proyecto Android

Vamos a crear el primer proyecto siguiendo esta guía

Con los siguientes cambios: * package name edu.villablanca * Minimun SDK API 24 Nougat Android 7.0 * Save Location: Utilizar un directorio para todos los proyectos de este módulo y un directorio "greeting" para este proyecto.

Cuando termine la sincronización del proyecto, se habrán creado unos ficheros y descargados librerías jar.

78.4.1 Código inicial

El punto de entrada en la clase MainActivity.

```
package edu.villablanca.greeting

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.ui.Modifier
import androidx.compose.ui.tooling.preview.Preview
import edu.villablanca.greeting.ui.theme.GreetingTheme

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            GreetingTheme {
                // A surface container using the 'background' color from the
theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("Android")
                }
            }
        }
    }
}

@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
    Text(
        text = "Hello $name!",
        modifier = modifier
    )
}

@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
```

```
    GreetingTheme {
        Greeting("Android")
    }
}
```

Veamos cada una de las partes de este ejemplo:

- Se define una clase **MainActivity** que hereda de **ComponentActivity**. Esta es la actividad principal de la aplicación.
- Se sobrescribe el método **onCreate** que se invoca cuando se crea la actividad. Dentro de este método, se inicializa la UI de la actividad. La firma de esta función es **override fun onCreate(savedInstanceState: Bundle?)**
- Se utiliza la función **setContent** para establecer el contenido UI de la actividad usando Jetpack Compose.La firma de esta función es :

  ```
  fun ComponentActivity.setContent(content: @Composable () -> Unit)
  ```

  content: @Composable () -> Unit: Es el parámetro que recibe la función. Representa una función componible (denotada por la anotación @Composable) que no recibe argumentos y no devuelve ningún valor (Unit en Kotlin es similar a void en otros lenguajes de programación).
- Se utiliza un tema personalizado llamado **GreetingTheme**. Dentro de este tema, se tiene un contenedor Surface que ocupa todo el espacio disponible (**fillMaxSize()**) y tiene un color de fondo definido por el esquema de colores del tema. Dentro de Surface, se invoca la función componible Greeting con el parámetro "Android".

La clase **GreetingTheme** esta definida en `import edu.villablanca.greeting.ui.theme.GreetingTheme` y ha sido creada por el asistente del proyecto. con la siguiente firma:

```
@Composable
fun GreetingTheme(
    darkTheme: Boolean = isSystemInDarkTheme(),
    // Dynamic color is available on Android 12+
    dynamicColor: Boolean = true,
    content: @Composable () -> Unit
)
```

* Función Composable Greeting: muestra un texto "Hello [nombre]!". Recibe un parámetro name y un modificador opcional * * **Vista Previa**: Se define una función componible para la vista previa llamada GreetingPreview. Esta función permite visualizar cómo se verá Greeting en el editor de Android Studio sin necesidad de ejecutar la aplicación en un dispositivo o emulador.

78.4.2 Vista previa.

Seleccionamos la ventana **Split** y tenemos que forzar "Build and Refresh"

78.5 Jetpack Compose

Jetpack Compose es un conjunto de bibliotecas de Android que facilita la creación de interfaces de usuario de manera declarativa y eficiente. Con Jetpack Compose, puedes construir la interfaz de usuario de tu aplicación de manera más rápida y fácil que utilizando XML y Java/Kotlin tradicionales

Jetpack Compose se compone de varios componentes clave:

- **Composables**: Son funciones de Kotlin que representan elementos de la interfaz de usuario, como botones, texto o diseños complejos. Los Composables se organizan en un árbol jerárquico para construir la interfaz.
- **State**: Jetpack Compose facilita la gestión del estado de la aplicación mediante el uso de remember y mutableStateOf, lo que permite que los componentes reaccionen automáticamente a los cambios en los datos.
- **Navigation**: Proporciona herramientas para la navegación entre diferentes pantallas de la aplicación.
- **Material** Design: Jetpack Compose se integra perfectamente con el diseño de Material Design, lo que facilita la creación de interfaces atractivas y coherentes.
- **ViewModels**: Puedes utilizar ViewModels en combinación con Jetpack Compose para separar la lógica de la interfaz de usuario de la lógica de negocio.

78.6 Comparación Jetpack Compose y diseño tradicional poe vistas con XML

Aspecto	**Jetpack Compose**	**Diseño Tradicional con XML**
Lenguaje	Utiliza Kotlin para definir Composables y lógica de interfaz de usuario.	Utiliza XML para definir la estructura y atributos de la interfaz de usuario.
Declarativo	Programación declarativa: defines cómo se ve la interfaz de usuario en función del estado y los datos.	Programación imperativa: defines la estructura y los atributos de cada vista de manera detallada.

Aspecto	Jetpack Compose	Diseño Tradicional con XML
Componentes Reutilizables	Utiliza Composables reutilizables que se pueden componer en diferentes diseños.	Utiliza diseños XML reutilizables (layouts) con `include` o `merge` para reutilizar vistas.
Modificadores	Aplica modificadores directamente a los Composables para ajustar su apariencia y comportamiento.	Utiliza atributos XML como `layout_width`, `layout_height`, `gravity`, etc., para ajustar las vistas.
Responsividad	Los Composables son responsivos por defecto y se adaptan automáticamente al tamaño de la pantalla y la orientación.	Debes crear recursos de diseño específicos para diferentes tamaños de pantalla y orientaciones.
Creación Dinámica	Puedes crear y modificar dinámicamente Composables en respuesta a eventos o cambios de estado.	La creación dinámica de vistas en XML es más compleja y a menudo requiere manipulación programática.
Preview en Tiempo Real	Puedes previsualizar la interfaz de usuario en tiempo real directamente en Android Studio.	La vista previa en tiempo real en XML es más limitada y a menudo requiere ejecutar la aplicación.
Separación de Responsabilidades	Fomenta la separación de la lógica de la interfaz de usuario y la lógica de negocio mediante ViewModels y Composables.	Requiere un manejo cuidadoso de la separación de lógica en actividades, fragmentos y controladores.
Curva de Aprendizaje	Puede requerir una curva de aprendizaje inicial debido a la programación declarativa.	Es más accesible para desarrolladores que ya están familiarizados con XML y vistas.
Ventajas Clave	Flexibilidad, mantenibilidad, legibilidad, previsualización en	Tradicional y ampliamente adoptado, especialmente en aplicaciones más antiguas.

Aspecto	Jetpack Compose	Diseño Tradicional con XML
	tiempo real, y mejor soporte para la creación dinámica de UI.	

78.7 Apendice

Enlaces: * Developer Android

ver 0.8 2-11-23

¿Fue útil esta página?