Módulo: Programación Multimedia y dispositivos móviles.
Unidad de Trabajo Nº3: Comunicaciones y contenido multimedia.
Actividad de Clase 1: Obtener datos de Internet con Retrofit.

1. **Objetivo general.**

   Crear una aplicación que use Retrofit 2 para recuperar datos de una API y utilice JetPack Compose.

2. **Metodología.**

   Cada alumno creará una app utilizando retrofit y los controles necesarios para mostrar los datos obtenidos desde la API.

3. **Descripción.**

Introducción

¿Qué es Retrofit?

Es un cliente de servidores REST para Android y Java desarrollado por Square, muy simple y muy fácil de aprender. Permite hacer peticiones al servidor tipo: GET, POST, PUT, PATCH, DELETE y HEAD, y gestionar diferentes tipos de parámetros, paseando automáticamente la respuesta a un tipo de datos. Y nos va a permitir consumir APIs. Podemos definir API REST como un servicio que nos provee de las funciones que necesitamos para poder obtener información de un cliente externo, como por ejemplo, una base de datos alojada en cualquier parte del mundo desde dentro de nuestra propia aplicación.

Realiza lo siguiente:

Configuración del Proyecto.

1. Vamos a crear nuestra App con Retrofit y Compose. Para ello Crea un nuevo proyecto y selecciona Empty Compose Activity.
2. Vamos a utilizar la siguiente API https://jsonplaceholder.typicode.com/ que está diseñada para realizar pruebas.
3. En primer lugar vamos a añadir el permiso de acceso a Internet en el fichero **AndroidManifest.xml** como podemos ver en la siguiente imagen:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools">

    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:dataExtractionRules="@xml/data_extraction_rules"
        android:fullBackupContent="@xml/backup_rules"
```

4. Ahora vamos a añadir las dependencias en el build.gradle.kts(Module: app). Además de la de Retrofit, también aparecen las de Gson Serialización, OkHttp y View Model. Una vez añadidos le damos a Sincronizar (Sync Now)

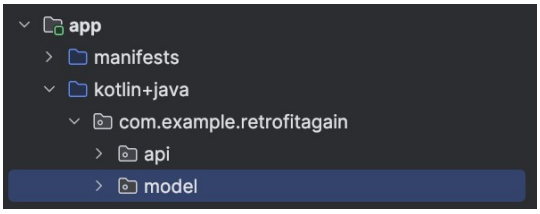
```
dependencies {  this: DependencyHandlerScope

    implementation(libs.androidx.core.ktx)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.ui.graphics)
    implementation(libs.androidx.ui.tooling.preview)
    implementation(libs.androidx.material3)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform(libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.ui.test.junit4)
    debugImplementation(libs.androidx.ui.tooling)
    debugImplementation(libs.androidx.ui.test.manifest)

    // Retrofit
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
    implementation("com.squareup.retrofit2:converter-gson:2.9.0")
    // Gson serializacion
    implementation("com.google.code.gson:gson:2.8.8")
    // OkHttp
    implementation("com.squareup.okhttp3:logging-interceptor:4.9.1")
    // View Model
    implementation("androidx.lifecycle:lifecycle-viewmodel-compose:2.8.7")
}
```

Creación del modelo de datos (objetos) que vamos a recibir de la API .

5. En el Proyecto dentro de la carpeta Kotlin+Java, selecciona la carpeta con el nombre completo de tu proyecto y pulsa con el botón derecho y a continuación sobre **New → Package** y si quieres le puedes poner el nombre **model** (o si lo prefieres network) te debe aparecer de forma similar a la siguiente:



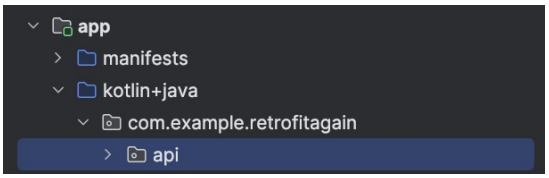
6. Selecciona el package que acabas de crear y con el botón derecho pulsa sobre **New → Kotlin Class/File** y selecciona **Data class** y ponle como nombre **Post**. En dicha

data class vamos a crear el modelo de datos que utilizaremos para representar los objetos que recibiremos de la API. Como puedes ver a continuación.

```
data class Post(
    val body: String,
    val id: Int,
    val title: String,
    val userId: Int
)
```

Creación de la API (ApiService).

7. En el Proyecto dentro de la carpeta Kotlin+Java, selecciona la carpeta con el nombre completo de tu proyecto y pulsa con el botón derecho y a continuación sobre **New → Package** y si quieres le puedes poner el nombre **api** (o si lo prefieres network) te debe aparecer de forma similar a la siguiente:



8. Selecciona el package que acabas de crear y con el botón derecho pulsa sobre **New → Kotlin Class/File** y selecciona **Interface** y ponle como nombre **ApiService**. En dicha interfaz vamos a definir el tipo de consumo de API y lo que nos va a devolver.

```
interface ApiService {
    @GET("posts")
    suspend fun getPosts(): List<Post>
}
```

9. Vamos a explicar el código anterior, para empezar vamos a seleccionar el tipo de llamada que es. En este caso será de tipo GET y para ello usamos la etiqueta @GET y en nuestro ejemplo vamos a solicitar los posts por eso lo ponemos entre “ dentro del paréntesis de la petición GET. Podríamos poner más. En JSONPlaceholder pueden aparecer hasta 6 recursos comunes (EndPoints) que son los siguientes:

- /posts 100 posts
- /comments 500 comments
- /albums 100 albums
- /photos 5000 photos
- /todos 200 todos
- /users 10 users

10. Lo siguiente es que antes de la palabra **fun** hay una nueva palabra reservada llamada **suspend**, esta será necesaria para trabajar con corrutinas, es decir, siempre que queramos hacer llamadas en segundo plano usando corrutinas tendremos que añadirla para que funcione nuestro código.

11. A continuación dentro del package de api (o network según le hayas llamado). Vamos a crear un instancia de Retrofit. Para ello con el botón derecho pulsa sobre **New → Kotlin Class/File** y selecciona **Object** y ponle como nombre **ApiClient**. En dicho objeto vamos a definir una constante con la url base y luego vamos a crear la instancia de Retrofit. A continuación, puedes observar el código.

```
object ApiClient {

    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"

    val retrofit: Retrofit = Retrofit.Builder()
        .baseUrl(BASE_URL)
        .addConverterFactory(GsonConverterFactory.create())
        .build()
    val apiService: ApiService = retrofit.create(ApiService::class.java)
}
```

12. Breve explicación del código. En primer lugar, creamos una constante que va a ser la dirección base de API, luego dependiendo del punto final (EndPoint) que seleccionemos, en nuestro caso posts. Luego creamos un objeto retrofit y creamos la instancia utilizando la interfaz que creamos anteriormente.

Creación del ViewModel

¿Qué es ViewModel?

ViewModel: Se encarga de acceder a los datos y realizar las transformaciones necesarias para mostrarlas en la vista.

13. En el Proyecto dentro de la carpeta Kotlin+Java, selecciona la carpeta con el nombre completo de tu proyecto y pulsa con el botón derecho y a continuación sobre **New → Package** y si quieres le puedes poner el nombre **viewmodel**

14. Selecciona el package que acabas de crear y con el botón derecho pulsa sobre **New → Kotlin Class/File** y selecciona **Class** y ponle como nombre **PostViewModel**. En dicha clase vamos a definir el acceso a Retrofit. Y será la clase que pase los datos a nuestra vista. A continuación, puedes ver una imagen del código. Dicho código está comentado para ahorrar la explicación.

```
// La clase debe heredar de ViewModel()
class PostViewModel: ViewModel() {
    // Propiedades de nuestra clase
    // Definimos los estados de los post a obtener privada para que no se pueda acceder desde fuera
    private val _posts = mutableStateListOf<Post>()
    // Los asociamos a nuestro viewmodel para que se pueda acceder desde fuera
    val posts: List<Post> get() = _posts

    // Inicializamos las propiedades con la función getPost
    init {
        getPost()
    }

    // Inicializamos _posts obteniendo los post desde retrofit lo hacemos utilizan try {} catch() {}
    private fun getPost() {
        viewModelScope.launch { this: CoroutineScope
            try {
                _posts.clear()
                _posts.addAll(ApiClient.apiService.getPosts())
            } catch (e: Exception) {
                Log.e( tag: "PostViewModel",  msg: "Error recuperando Post", e)
            }
        }
    }
}
```

Creación de la Vista

15. En el Proyecto dentro de la carpeta **Kotlin+Java**, selecciona la carpeta con el nombre completo de tu proyecto y pulsa con el botón derecho y a continuación sobre **New → Package** y si quieres le puedes poner el nombre **view**

16. Selecciona el package que acabas de crear y con el botón derecho pulsa sobre **New → Kotlin Class/File** y selecciona **File** y ponle como nombre **Screens** (o Pantallas según prefieras).

17. En el código que te voy a mostrar como ejemplo, es una presentación muy sencilla, mostrando un listado con LazyColumn de los post que hayamos obtenido a través de Retrofit. Utilizando una función para mostrar cada uno de los datos del post. Lo único diferente es que las actividades anteriores es que la función Composable tiene como parámetro un viewmodel, para poder acceder a los posts obtenidos por retrofit. Tú si quieres puedes utilizar otra presentación, con otros componentes diferentes, pero siempre debes acceder a los datos obtenidos por Retrofit mediante el viewmodel.

```kotlin
// Tenemos como parámetro el viewmodel para poder acceder a los posts obtenidos por Retrofit
@Composable
fun PostListScreen(viewmodel: PostViewModel) {
    val posts = viewmodel.posts
    Column(
        modifier = Modifier
            .padding(8.dp, 32.dp)
    ) { this: ColumnScope
        Text(
            text = "Post",
            fontSize = 30.sp,
            fontWeight = FontWeight.Bold
        )
        LazyColumn { this: LazyListScope
            items(posts) { this: LazyItemScope post ->
                PostItem(post)
            }
        }
    }
}


@Composable
fun PostItem(post: Post) {
    Card(
        modifier = Modifier
            .padding(8.dp)
    ) { this: ColumnScope
        Text(text = post.id.toString())
        Spacer(modifier = Modifier.height(8.dp))
        Text(text = post.title)
        Spacer(modifier = Modifier.height(8.dp))
        Text(text = post.body)
    }
}
```

Actualización del MainActivity

18. Modificamos el MainActivity. En primer lugar, elimina la función @Composable Greeting y si quieres puedes eliminar también la función @Preview. Y en la MainActivity, en Scaffold, creamos el viewmodel y llamamos a PostListScreen pasandole como parámetro el viewmodel creado. A continuación puedes ver un ejemplo del código.

```kotlin
Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
    val viewModel = PostViewModel()
    PostListScreen(viewModel)
}
```

19. En último lugar, ejecuta la aplicación y comprueba su funcionamiento. Puedes ver un ejemplo en el siguiente video.

https://drive.google.com/file/d/139UkNdOBgSvEi4CIS6MaiVF5OxWR6qjz/view?usp=sharing