Consumo de API REST

REST y Android (Parte I)

{desafío}
latam_

Construir una aplicación Android que consume un servicio REST actualizando la interfaz de usuario, acorde al lenguaje Kotlin y a la librería Retrofit

- Unidad 1:
  Acceso a datos en Android
- Unidad 2:
  Consumo de API REST
- Unidad 3:
  Testing
- Unidad 4:
  Distribución del aplicativo Android

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *HTTP request threads*
- *Retrofit librería para requests HTTP*
- *Hacer requests HTTP con Retrofit*

¿Qué es una solicitud HTTP y por qué es importante en el desarrollo de apps en Android?

{desafío}
latam_

/* HTTP request threads */

HTTP request threads

- Un HTTP request threads, también conocido como HTTP threads, es un subproceso responsable de manejar una sola solicitud HTTP. En una aplicación web, cada solicitud HTTP entrante es manejada por un hilo separado, lo que permite que la aplicación maneje varias solicitudes simultáneamente.

- Cuando un cliente (como un navegador web o una aplicación móvil) realiza una solicitud HTTP a un servidor, la solicitud se envía a través de la red al servidor. Luego, el servidor procesa la solicitud, genera una respuesta y la envía de regreso al cliente a través de la red. El subproceso que es responsable de manejar la solicitud en el lado del servidor se denomina subproceso de solicitud HTTP.

HTTP request threads

- El subproceso de solicitud HTTP lee la solicitud, la analiza y luego realiza las acciones necesarias para generar la respuesta. Una vez que se genera la respuesta, el subproceso la envía de vuelta al cliente y luego está disponible para manejar otra solicitud.

- Es importante tener en cuenta que para cada solicitud, se crea un nuevo hilo. Esto significa que si el servidor recibe una gran cantidad de solicitudes al mismo tiempo, creará una gran cantidad de subprocesos, lo que puede provocar un problema de rendimiento. Por lo tanto, los servidores usan grupos de subprocesos para manejar la gran cantidad de solicitudes simultáneamente y evitar sobrecargar el sistema.

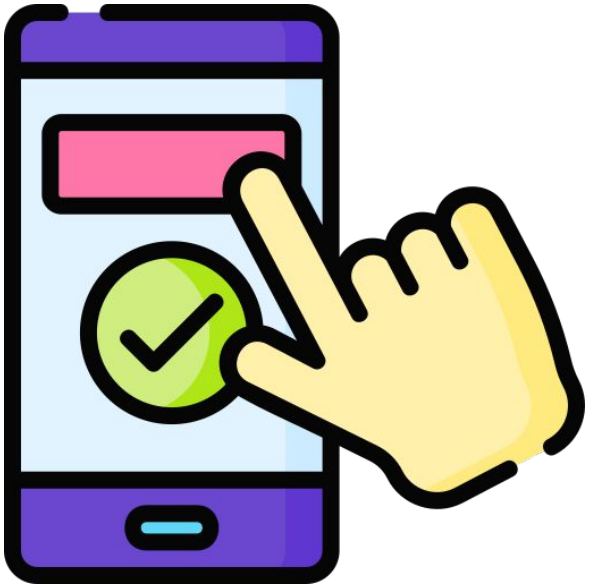
/* Retrofit librería para requests HTTP */

Retrofit librería para requests HTTP

- Retrofit es un cliente HTTP con seguridad de tipos para Android y Java, desarrollado por Square. Permite a los desarrolladores realizar fácilmente llamadas de red, al abstraerse de las complejidades del manejo de solicitudes y respuestas HTTP. Retrofit utiliza anotaciones para definir la estructura de las llamadas de red, lo que ayuda a crear una API clara y fácil de usar.

- Retrofit usa OkHttp, un popular cliente HTTP de código abierto, para manejar los detalles de bajo nivel de las llamadas de red. También incluye soporte integrado para manejar tareas comunes, como analizar JSON y XML, serializar y deserializar datos, y más.

Retrofit librería para requests HTTP



- Retrofit proporciona una API simple y limpia para realizar solicitudes síncronas y asíncronas. También admite el almacenamiento en caché de solicitudes y respuestas, el registro y también permite agregar convertidores personalizados para manejar otros tipos de datos.
- Retrofit también permite usar corrutinas y también integrarse con otras bibliotecas como Room o LiveData.
- Retrofit facilita a los desarrolladores la creación de una capa de red robusta y escalable para su aplicación, al reducir la cantidad de código necesario para gestionar las llamadas de red.

/* Qué es Gson */

¿Qué es Gson?

Gson es una biblioteca de Java que se puede utilizar para convertir objetos de Java a su representación JSON y viceversa. Fue creado por Google y es ampliamente utilizado en la comunidad Java para trabajar con datos JSON. Gson puede convertir automáticamente entre objetos JSON y Java, incluidas listas y mapas. También puede manejar objetos anidados y manejar diferentes tipos de datos.

Gson es una biblioteca poderosa que facilita el trabajo con JSON en Java; es fácil de usar, rápida y eficiente, también es muy liviana y no tiene dependencias externas. Una de las características clave de Gson es que puede manejar la mayoría de las conversiones de tipos de datos automáticamente, lo que lo hace muy conveniente de usar.

/* Hacer requests HTTP con Retrofit */

¿Cómo hacer requests HTTP con Retrofit?

Antes de hacer un request, es necesario tener claro cuales son los pasos necesarios. Estos son los pasos para realizar una solicitud HTTP usando Retrofit con Kotlin:

1. Agregue Retrofit y sus dependencias a su archivo de compilación de Gradle.

2. Defina una interfaz para el endpoint de la API al que desea llamar. Esta interfaz debe incluir anotaciones que describen el tipo de solicitud HTTP y la URL del punto final.

3. Cree una instancia de Retrofit usando un Retrofit.Builder.

4. Cree una implementación de la interfaz API utilizando la instancia Retrofit.

¿Cómo hacer requests HTTP con Retrofit?

5. Llame al método del endpoint deseado en la implementación de la interfaz API y pasa los parámetros necesarios.

6. Maneje la respuesta de la llamada API, ya sea procesando los datos de respuesta o manejando cualquier error.

7. Recuerde lo aprendido sobre Clean Architecture

```
// Paso 1: Agrega Retrofit a tu Gradle build.
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.9.0'

// Paso 2: Define la interfaz y el endpoint.
interface ApiService {
    @GET("users/{user}")
    fun getUser(@Path("user") user: String): Call<User>
}

// Paso 3: Crea una instancia de Retrofit.
val retrofit = Retrofit.Builder()
    .baseUrl("https://api.github.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()

// Paso 4: Crea una implementación de la interfaz.
val api = retrofit.create(ApiService::class.java)
```

```
// Paso 5: Llama al método correspondiente al endpoint al que quieres hacer el request.

val call = api.getUser("user")
call.enqueue(object : Callback<User> {
    override fun onResponse(call: Call<User>, response: Response<User>) {
        // Paso 6: Maneja la respuesta del llamado de la API.
        if (response.isSuccessful) {
            val user = response.body()
            // Maneja la respuesta
        } else {
            // Maneja el error
        }
    }

    override fun onFailure(call: Call<User>, t: Throwable) {
        // Controla el caso en el que el request falla, no confundir caso de error anterior
    }
})
```

Ejercicio
"HTTP request con Retrofit"

En el siguiente ejercicio vamos a hacer un request a Github, para esto, se requiere que tengas una cuenta creada o al menos el nombre de un usuario

¡Empecemos!

- Define una interfaz Retrofit para el endpoint de la API:

```
interface ApiService {
    @GET("users/{user}")
    fun getUser(@Path("user") user: String): Call<User>
}
```

- Crea una instancia de la clase de Retrofit:

```
val retrofit = Retrofit.Builder()
    .baseUrl("https://api.github.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build()

val api = retrofit.create(ApiService::class.java)
```

- **Haz el llamado a la API**

```
val call = api.getUser("user")
call.enqueue(object : Callback<User> {
    override fun onResponse(call: Call<User>, response: Response<User>) {
        if (response.isSuccessful) {
            val user = response.body()
            // reemplaza con printl o log.d
        } else {
            // reemplaza con printl o log.d
        }
    }

    override fun onFailure(call: Call<User>, t: Throwable) {
        // reemplaza con printl o log.d
    }
})
```

¿Qué es Retrofit y qué problema soluciona?

Próxima sesión...

- *Rest y Android Parte II*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales