Consumo de API REST

REST y Android (Parte II)

{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

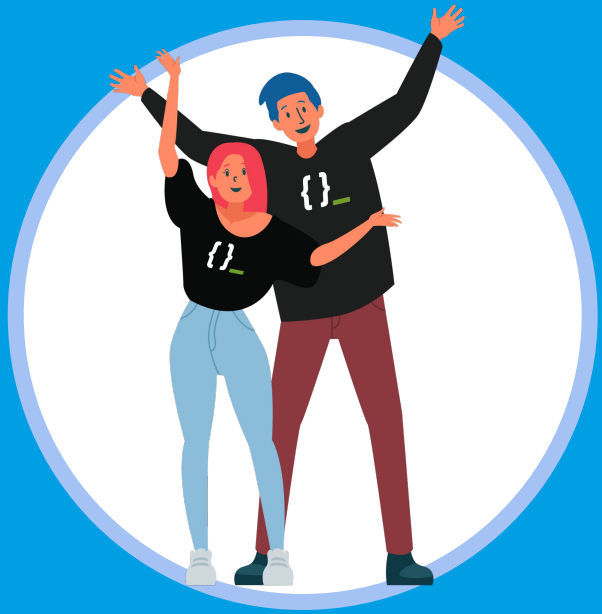
- Unidad 1:
  Acceso a datos en Android
- Unidad 2:
  Consumo de API REST
- Unidad 3:
  Testing
- Unidad 4:
  Distribución del aplicativo Android

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Los verbos HTTP con Retrofit y sus anotaciones*
- *QueryParams con Retrofit*
- *Headers con Retrofit*

¿Cómo crees que Retrofit maneja los HTTP methods?



/* Los verbos HTTP con Retrofit y sus anotaciones */

Los verbos HTTP con Retrofit y sus anotaciones

Retrofit admite varias anotaciones de métodos HTTP que se pueden usar para definir el tipo de solicitud que se realiza al servidor. Estas son las anotaciones de métodos HTTP más comunes en Retrofit:

- @GET: esta anotación se utiliza para definir una solicitud GET. Una solicitud GET se utiliza para recuperar información del servidor sin realizar ningún cambio.
- @POST: esta anotación se utiliza para definir una solicitud POST. Una solicitud POST se usa para enviar datos al servidor para su procesamiento, por ejemplo, creando un nuevo recurso en el servidor.
- @PUT: esta anotación se utiliza para definir una solicitud PUT. Una solicitud PUT se utiliza para actualizar un recurso existente en el servidor.

Los verbos HTTP con Retrofit y sus anotaciones

- @DELETE: esta anotación se utiliza para definir una solicitud DELETE. Una solicitud DELETE se utiliza para eliminar un recurso en el servidor.
- @PATCH: esta anotación se utiliza para definir una solicitud de PATCH. Una solicitud PATCH se utiliza para actualizar una parte de un recurso existente en el servidor.
- Además de estos métodos HTTP, Retrofit también admite otras anotaciones, como @Headers, @Query, @QueryMap, @Path, @Body y otras, que se pueden usar para definir los parámetros, los encabezados y el cuerpo de la solicitud.

/* QueryParams con Retrofit */

QueryParams con Retrofit

En Retrofit, los parámetros de consulta se utilizan para especificar información adicional para incluir en la solicitud de API. Estos parámetros se agregan al final de la URL del extremo de la API como pares clave-valor, con las claves y los valores separados por un signo igual (=) y múltiples parámetros separados por un ampersand (&).

Por ejemplo, considere un punto final de API que devuelve una lista de usuarios según ciertos filtros, como la edad y la ubicación. El punto final se puede definir como:

```
@GET("users")
fun getUsers(
    @Query("min_age") minAge: Int,
    @Query("max_age") maxAge: Int,
    @Query("location") location: String
): Call<List<User>>
```

QueryParams con Retrofit

Puede llamar a este punto final proporcionando los valores para los parámetros de consulta:

```
val api = retrofit.create(API::class.java)
val call = api.getUsers(18, 25, "NYC")
```

Esto hará una solicitud GET a la siguiente URL:
`https://your-base-url.com/users?min_age=18&max_age=25&location=NYC`

También puede usar la anotación @QueryMap para pasar un mapa de parámetros de consulta

```
@GET("users")
fun getUsers(@QueryMap params: Map<String, String>): Call<List<User>>
```

QueryParams con Retrofit

Luego, llámalo de la siguiente forma:

```
val params = mapOf("min_age" to 18, "max_age" to 25, "location" to "NYC")
val call = api.getUsers(params)
```

También es posible utilizar la anotación @QueryName para especificar la clave del parámetro de consulta.

```
@GET("users")
fun getUsers(@QueryName("min_age") minAge: Int): Call<List<User>>
```

Es importante tener en cuenta que los parámetros de consulta son opcionales y, si no se proporcionan, el punto final seguirá funcionando.

/* Headers con Retrofit */

Headers con Retrofit

- En Retrofit, los headers se utilizan para pasar información adicional junto con una solicitud de API. Estos headers se envían como pares clave-valor en los encabezados de solicitud.
- Los headers se pueden usar para pasar varios tipos de información, como tokens de autenticación, tipos de contenido preferido o cadenas de agentes de usuario personalizadas.
- Retrofit le permite agregar encabezados a sus solicitudes de API de varias maneras:
- Uso de una anotación @Headers: puede usar esta anotación para especificar una lista de encabezados que deben enviarse con cada solicitud para un punto final específico.

Headers con Retrofit

```
@Headers("Accept: application/json", "User-Agent: MyApp")
@GET("users")
fun getUsers(): Call<List<User>>
```

Uso del parámetro @Header en el método: puede usar la anotación @Header para especificar un encabezado que debe incluirse con una solicitud específica.

```
@GET("users")
fun getUsers(@Header("Authorization") token: String): Call<List<User>
```

Headers con Retrofit - Interceptor

Uso de Interceptor: puede usar un Interceptor para agregar encabezados a cada solicitud realizada por el cliente Retrofit.

```
val client = OkHttpClient.Builder()
    .addInterceptor { chain ->
        val newRequest = chain.request().newBuilder()
            .addHeader("Accept", "application/json")
            .addHeader("User-Agent", "MyApp")
            .build()
        chain.proceed(newRequest)
    }
    .build()
```

Vale la pena señalar que Retrofit agrega automáticamente algunos encabezados a las solicitudes, como el encabezado Content-Type cuando se envía el cuerpo de una solicitud y el encabezado Accept-Encoding cuando se usa el cliente OkHttp.

Demostración "HTTP Request a GithubApiService"

{desafío} latam_

HTTP Request a GithubApiService

Para realizar un HTTP request a "`https://api.github.com`" usando Retrofit, puedes seguir estos pasos:

Crea una interfaz para la API: define los endpoints de la API y los métodos HTTP correspondientes en una interfaz. Por ejemplo, para recuperar información sobre un usuario específico, puede crear una interfaz de la siguiente manera:

```
interface GithubApiService {
    @GET("users/{username}")
    suspend fun getUser(@Path("username") username: String): Response<User>
}
```

HTTP Request a GithubApiService

Configura Retrofit: Inicialice Retrofit creando una instancia de la clase Retrofit y configurándola con la URL base de la API y una fábrica de convertidores (como `GsonConverterFactory`).

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

HTTP Request a GithubApiService

Cree una instancia de la interfaz API: use el método de creación de la clase Retrofit para crear una instancia de la interfaz Github ApiService.

```
val githubApiService = retrofit.create(GithubApiService::class.java)
```

Realiza la solicitud de API: llame al método deseado en la instancia de GithubApiService para realizar la solicitud de API. Por ejemplo, para recuperar información sobre un usuario con el nombre de usuario octocat, llamaría al método getUser de la siguiente manera:

```
val response = githubApiService.getUser("octocat")
```

HTTP Request a GithubApiService

Maneja la respuesta: la variable de respuesta contendrá la respuesta del servidor, que se puede manejar usando los métodos proporcionados por el objeto Response. Por ejemplo, puede verificar si la respuesta fue exitosa llamando al método `isSuccessful` y accediendo al cuerpo de la respuesta usando la propiedad body.

```
if (response.isSuccessful) {
    val user = response.body()
    // Use the user data
} else {
    // Handle the error
}
```

Este es un ejemplo básico de cómo realizar una solicitud HTTP a la API de Github usando Android Kotlin y Retrofit. Puedes ampliar este ejemplo agregando más puntos finales de API y métodos HTTP a la interfaz de Github ApiService y realizando solicitudes de API adicionales según sea necesario.

¿Cuál es la importancia de QueryParams y Headers en Retrofit para Android?

{desafío} latam_

Próxima sesión...

- *Guía de ejercicios*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales