Consumo de API REST

Imágenes en Servidores (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?

- *REST y autenticación*
- *Buenas prácticas de autenticación en requests HTTP*

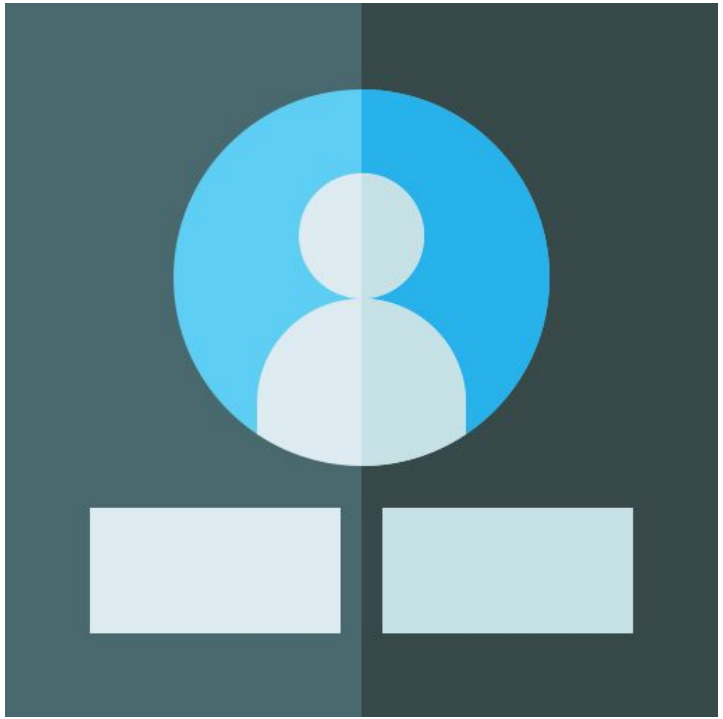
¿Sabes cuál es la principal diferencia entre autenticación y autorización?

{desafío} latam_

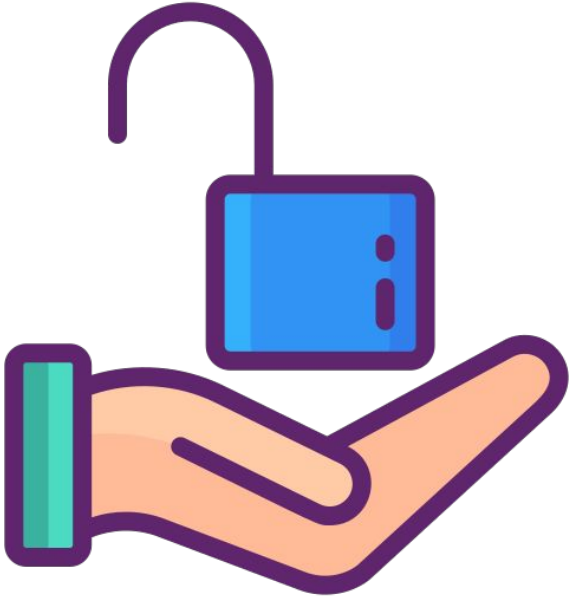
/* Autenticación y Autorización */

Autenticación y Autorización

La **autenticación** es el proceso de verificar la identidad de un usuario, dispositivo o sistema. Implica proporcionar un conjunto de credenciales, como un nombre de usuario y una contraseña, para demostrar que el usuario es quien dice ser. El propósito de la autenticación es confirmar la identidad del usuario y garantizar que solo los usuarios autorizados tengan acceso a los recursos protegidos.



Autenticación y Autorización



La **autorización**, por otro lado, es el proceso de determinar si un usuario autenticado tiene acceso a un recurso específico. Una vez que se ha autenticado a un usuario, se realizan verificaciones de autorización para determinar si el usuario puede realizar ciertas acciones, como leer, escribir o eliminar datos. La autorización a menudo se implementa mediante listas de control de acceso, control de acceso basado en roles u otros mecanismos que definen los permisos asociados con un usuario o grupo de usuarios.

Autenticación y Autorización

En resumen, la autenticación verifica la identidad del usuario, mientras que la autorización determina qué puede hacer el usuario una vez que se ha establecido su identidad.

/* REST y autenticación */

REST y autenticación

La autenticación en un servicio REST normalmente funciona mediante el envío de credenciales de autenticación (como un nombre de usuario y una contraseña) con cada solicitud al servidor. Luego, el servidor usa estas credenciales para verificar la identidad del usuario y determinar si tiene acceso a los recursos solicitados.

REST y autenticación

Aquí hay una descripción general del proceso:

1. El cliente envía una solicitud al servidor con las credenciales de autenticación incluidas en los encabezados o el cuerpo de la solicitud.
2. El servidor recibe la solicitud y valida las credenciales.
3. Si las credenciales son válidas, el servidor genera un token de autenticación, que es un identificador único que representa al usuario autenticado.
4. El servidor devuelve el token de autenticación al cliente.
5. El cliente almacena el token de autenticación y lo usa para solicitudes posteriores al servidor.
6. El servidor usa el token de autenticación para verificar la identidad del usuario para cada solicitud posterior.

REST y autenticación

Este proceso se implementa comúnmente mediante un sistema de autenticación basado en token, como OAuth2 o JWT. En este sistema, el servidor genera un token que se devuelve al cliente, y el cliente usa este token para autenticar solicitudes posteriores al servidor.

Es importante tener en cuenta que la autenticación en los servicios REST debe realizarse a través de una conexión segura, como HTTPS, para evitar el espionaje o la manipulación de información confidencial durante la transmisión.

/* Buenas prácticas de autenticación en requests HTTP */

Buenas prácticas de autenticación en requests HTTP

Use protocolos seguros	Use métodos de autenticación seguros	Almacene las credenciales de forma segura	Enviar credenciales solo a través de canales seguros
Use siempre protocolos seguros como HTTPS o SSL para transmitir datos confidenciales, como credenciales de autenticación. Esto evitará el espionaje y la manipulación de datos durante la transmisión.	Use métodos de autenticación seguros como OAuth2, JWT o autenticación básica sobre SSL para autenticar al usuario. Evite el uso de métodos de autenticación débiles, como enviar contraseñas en texto claro.	Almacene las credenciales de autenticación, como contraseñas y tokens, de forma segura en el lado del cliente. Por ejemplo, use Android Keystore para almacenar información confidencial.	Envíe credenciales de autenticación solo a través de canales seguros, como HTTPS. No envíe credenciales a través de canales no cifrados, como HTTP.

Buenas prácticas de autenticación en requests HTTP

Use tokens para la autenticación

Use tokens para la autenticación en lugar de enviar credenciales con cada solicitud. Los tokens pueden ser de corta o larga duración y se pueden actualizar según sea necesario.

Utilice la validación del lado del servidor

Valide los tokens de autenticación en el lado del servidor para garantizar que el cliente tenga los permisos necesarios para acceder a los recursos solicitados.

Use el almacenamiento seguro de tokens

Almacene tokens de forma segura en el lado del cliente y evite almacenar tokens en texto sin formato en preferencias compartidas, bases de datos SQLite u otras ubicaciones inseguras.

Demostración "Autenticación"

{desafío} latam_

Autenticación

Primero, comencemos con un ejemplo simple de cómo crear un cliente Retrofit en Kotlin. Asumiremos que tenemos un endpoint de API `/api/login` que devuelve un token JWT cuando hacemos POST con un objeto JSON con nuestras credenciales de inicio de sesión:

```
interface ApiService {
    @POST("/api/login")
    suspend fun login(@Body credentials: LoginCredentials): Response<AuthToken>
}
```

Aquí, **`LoginCredentials`** es una data class que representa nuestras credenciales de inicio de sesión y **`AuthToken`** es otra clase de datos que representa nuestro token JWT.

Autenticación

Ahora, supongamos que tenemos un modelo de vista que necesita usar esta API para iniciar sesión y obtener un token de autenticación. Podemos crear un cliente Retrofit y usarlo para hacer la llamada a la API de esta manera:

```
class LoginViewModel(private val api: ApiService) : ViewModel() {
    private var authToken: AuthToken? = null

    suspend fun login(username: String, password: String): Boolean {
        val response = api.login(LoginCredentials(username,
password))
        if (response.isSuccessful) {
            authToken = response.body()
            return true
        }
        return false
    }

    fun getAuthToken(): String? {
        return authToken?.token
    }
}
```

En este viewModel, estamos usando `api` para realizar la llamada a la API de inicio de sesión. Si la respuesta es exitosa, almacenamos el AuthToken en `authToken`. También tenemos una función `getAuthToken` para recuperar el token JWT, que usaremos para llamadas API posteriores.

Autenticación

Finalmente, escribamos un Unit Test para este ViewModel. Usaremos Mockito para simular el `ApiService` y usaremos la función `runBlocking` para ejecutar la prueba en una rutina suspendida:

```
class LoginViewModelTest {
    private lateinit var api: ApiService
    private lateinit var viewModel: LoginViewModel

    @Before
    fun setup() {
        api = mock(ApiService::class.java)
        viewModel = LoginViewModel(api)
    }
 . . .
}
```

los test están en la siguiente slide

{desafío} latam_

Autenticación - Unit Test

```
@Test
fun `login success`() = runBlocking {
    val authToken = AuthToken("dummy_token")
    `when`(api.login(LoginCredentials("username", "password")))
        .thenReturn(Response.success(authToken))

    val result = viewModel.login("username", "password")

    assertTrue(result)
    assertEquals("dummy_token", viewModel.getAuthToken())
}

@Test
fun `login failure`() = runBlocking {
    `when`(api.login(LoginCredentials("username", "password")))
        .thenReturn(Response.error(401, ResponseBody.create(MediaType.parse("application/json"), "")))

    val result = viewModel.login("username", "password")

    assertFalse(result)
    assertNull(viewModel.getAuthToken())
}
```

Autenticación

- En este Unit Test, estamos probando la función de inicio de sesión de `LoginViewModel`. Estamos simulando el `ApiService` usando Mockito, y estamos usando `thenReturn` para especificar la respuesta que la API debe devolver para un conjunto determinado de parámetros de entrada. Luego llamamos al inicio de sesión con algunas credenciales ficticias y verificamos que el AuthToken esté almacenado correctamente.

- También estamos probando el caso en el que la API devuelve una respuesta de error, en cuyo caso esperamos que la función de inicio de sesión devuelva falso y la función `getAuthToken` devuelva nulo.

- Ten en cuenta que, en este ejemplo, usamos la clase `Response` de Retrofit para representar la respuesta de la API. La propiedad `isSuccessful` de Response devuelve verdadero si la respuesta tiene un código de estado en el rango [200..300].

Explica con tus palabras ¿Cómo funciona la autenticación en Retrofit?

Próxima sesión...

- *Authoken, jwt, token-bearer*
- *Añadir el authtoken a un header usando Retrofit*

{desafío}
latam_

Academia de
talentos digitales