ESCUELA DE SISTEMAS Y TECNOLOGÍAS

Transparencias de **ANALISTA DE SISTEMAS**
Edición 2023 - Materia: **Aplicaciones Móviles con Flutter**

TEMA: Consumo de APIs REST

Agenda

- Introducción
- El Paquete http
- Enviar Solicitudes HTTP
- Manejar la Respuesta
- Codificar y Decodificar JSON
- MultipartRequest

Introducción

- Para proporcionar a los usuarios una experiencia completa, las aplicaciones móviles a menudo requieren acceder y enviar datos a través de *Internet*.
- Una forma eficiente y ampliamente utilizada de lograr esta comunicación es mediante la integración de una *API REST*.
- En las siguientes transparencias exploraremos los conceptos básicos de la comunicación entre una aplicación móvil desarrollada en *Flutter* y una *API REST*.

El Paquete http (1)

- El paquete **http** es una biblioteca que permite realizar solicitudes *HTTP* desde una aplicación *Flutter*.
- Las solicitudes *HTTP* son esenciales para interactuar con servidores *web*, consumir *API REST* y obtener o enviar datos desde y hacia recursos en línea.
- El paquete *http* proporciona funciones para realizar varios tipos de solicitudes *HTTP*, como *GET*, *POST*, *PUT* y *DELETE*.
- Estas funciones permiten interactuar con servidores *web* y servicios *REST* de manera eficiente.

El Paquete http (2)

- El paquete *http* simplifica el proceso de manejar las respuestas *HTTP*.
- Se puede verificar el código de estado de la respuesta para determinar si la solicitud fue exitosa (código *200*) o si hubo un error.
- Además, se puede acceder al contenido de la respuesta en diferentes formatos, como texto o *JSON*.
- También es posible agregar encabezados personalizados a las solicitudes *HTTP* según sea necesario. Esto es útil para autenticación, envío de *tokens* de seguridad u otros requisitos específicos del servidor.

El Paquete http (3)

- El paquete *http* maneja excepciones relacionadas con problemas de red, como conexiones fallidas o solicitudes que no pueden completarse. Esto permite implementar lógica de manejo de errores para garantizar la robustez de la aplicación.
- Las solicitudes *HTTP* se realizan de forma asíncrona, lo que significa que no bloquean la interfaz de usuario de la aplicación mientras se espera una respuesta del servidor. Se puede utilizar *async / await* para realizar solicitudes sin bloquear la ejecución de otras tareas.
- El paquete *http* también admite conexiones seguras mediante el protocolo *HTTPS*.

El Paquete http (4)

- Para utilizar este paquete, primero debe agregarse a la sección de dependencias del archivo *pubspec.yaml*:

  dependencies:

  ...

  http: ^1.1.0

- Luego de lo cual podrá ser importado en el archivo de código *Dart* donde se vaya a utilizar:

- *import 'package:http/http.dart';*

Enviar Solicitudes HTTP (1)

- El método *GET* se utiliza para recuperar datos de un servidor. Se puede usar la función **http.get(...)** para realizar una solicitud *GET*.
- Parámetros:
  - **url**: Instancia de *Uri* con la *URL* del recurso que se desea recuperar.
  - **headers** (opcional): *Map<String, String>?* de encabezados *HTTP* personalizados que se desea enviar con la solicitud.

Enviar Solicitudes HTTP (2)

- El método *POST* se utiliza para enviar datos al servidor. Se puede usar la función **http.post(...)** para realizar una solicitud *POST*.
- Parámetros:
  - **url**: Instancia de *Uri* con la *URL* a donde se desea enviar los datos para crear un nuevo recurso.
  - **headers** (opcional): *Map<String, String>?* de encabezados *HTTP* personalizados que se desea enviar con la solicitud.
  - **body** (opcional): *Object?* con el cuerpo de la solicitud, que contiene los datos a enviar al servidor. Puede ser un String, un List<int> (bytes) o un Map<String, String> (campos de formulario).

Enviar Solicitudes HTTP (3)

- El método *PUT* se utiliza para actualizar datos en el servidor. Se puede usar la función **http.put(...)** para realizar una solicitud *PUT*.
- Parámetros:
  - **url**: Instancia de *Uri* con la *URL* a donde se desea enviar los datos a actualizar.
  - **headers** (opcional): *Map<String, String>?* de encabezados *HTTP* personalizados que se desea enviar con la solicitud.
  - **body** (opcional): *Object?* con el cuerpo de la solicitud, que contiene los datos a enviar al servidor para la actualización.

Enviar Solicitudes HTTP (4)

- El método *DELETE* se utiliza para eliminar recursos en el servidor. Se puede usar la función **http.delete(...)** para realizar una solicitud *DELETE*.
- Parámetros:
  - **url**: Instancia de *Uri* con la *URL* del recurso que se desea eliminar.
  - **headers** (opcional): *Map<String, String>?* de encabezados *HTTP* personalizados que se desea enviar con la solicitud.

Manejar la Respuesta (1)

- Cada uno de los métodos vistos anteriormente devuelve un *Future<http.Response>* que permite esperar la respuesta del servidor de manera asíncrona y manejarla de acuerdo a las necesidades
- El objeto http.Response se utiliza para representar la respuesta de una solicitud HTTP.
- Contiene información sobre la respuesta del servidor, incluyendo el código de estado, los encabezados y el cuerpo de la respuesta.

Manejar la Respuesta (2)

- Propiedades de *http.Response*:
  - **statusCode**: Esta propiedad *int* almacena el código de estado de la respuesta *HTTP*. Por lo general, los códigos de estado *2xx* indican una respuesta exitosa (por ejemplo, *200* para OK), mientras que los códigos *4xx* y *5xx* indican errores (por ejemplo, *404* para "*Not Found*" o *500* para "*Internal Server Error*"). La clase *HttpStatus* de la biblioteca *dart:io* define constantes con los códigos de estado *HTTP*.
  - **reasonPhrase**: *String?* con una breve descripción textual del código de estado. Por ejemplo, para un código de estado *404*, esto podría ser "*Not Found*".

Manejar la Respuesta (3)

- Propiedades de *http.Response* (cont.):
  - **headers**: *Map<String, String>* que contiene los encabezados de la respuesta *HTTP*. Se puede acceder a los encabezados individuales utilizando sus nombres como claves en este mapa.
  - **bodyBytes**: *Uint8List* con los *bytes* brutos del cuerpo de la respuesta. Se puede utilizar esta propiedad si se necesita trabajar con los datos en formato binario.
  - **body**: *String* con el cuerpo de la respuesta como una cadena de texto. La mayoría de las veces, éste es el formato en el que se trabaja con los datos de la respuesta.

Manejar la Respuesta (4)

- Métodos de *http.Response*:
  - **String toString()**: Devuelve la respuesta *HTTP* como una cadena de texto legible que incluye el código de estado y los encabezados.

Codificar y Decodificar JSON (1)

- Al consumir una *API REST* es habitual enviar y recibir datos estructurados en formato *JSON*.
- *JSON* (*JavaScript Object Notation*) es un formato de intercambio de datos comúnmente utilizado en aplicaciones web y en la comunicación entre sistemas debido a su facilidad de lectura, tanto para humanos como para máquinas.
- La biblioteca *dart:convert*, proporciona funciones para la codificación y decodificación de datos en formato *JSON*.
- En dicha biblioteca se encuentran las funciones *jsonEncode(...)* y *jsonDecode(...)* que permiten codificar y decodificar información en formato *JSON*.

Codificar y Decodificar JSON (2)

- La función **jsonEncode(...)** se utiliza para convertir objetos *Dart* en una representación de cadena *JSON*.
- Dicha función toma un *Object?* por parámetro, como un mapa (*Map*), una lista (*List*), o un valor escalar (número, cadena de texto, booleano o *null*), y lo convierte en un *String JSON* válido.
- Al utilizar jsonEncode(...), es importante que el objeto a convertir en *JSON* sea compatible con dicho formato.
- Por ejemplo, *Dart* no admite automáticamente la serialización de funciones o objetos personalizados, por lo que se debe trabajar con tipos de datos que se puedan convertir de manera segura en *JSON*.

Codificar y Decodificar JSON (3)

- La función **jsonDecode(...)** se utiliza para convertir una cadena de texto *JSON* en objetos *Dart*.
- Permite tomar un String *JSON* y convertirlo en un objeto (*dynamic*) que se pueda utilizar en la aplicación, como un mapa (*Map*), una lista (*List*), o un valor escalar (número, cadena de texto, booleano o *null*).
- Es importante tener en cuenta que la función *jsonDecode(...)* asume que la cadena *JSON* que se le proporciona por parámetro es válida.
- Si la cadena *JSON* no es válida, *jsonDecode(...)* generará una excepción *FormatException*.

Codificar y Decodificar JSON (4)

- Por lo tanto, es una buena práctica manejar errores y excepciones al trabajar con *jsonDecode(...)* para garantizar que la aplicación sea robusta ante datos *JSON* incorrectos o malformados.

MultipartRequest (1)

- La clase **MultipartRequest** de la biblioteca *http* se utiliza para crear y configurar una solicitud *HTTP* de tipo *multipart/form-data*.
- Este tipo de solicitud es comúnmente utilizada para enviar archivos binarios, como imágenes o archivos de audio, a través de una solicitud *HTTP POST*.
- Se puede construir una instancia indicando, mediante un *String*, el método *HTTP*, y un objeto *Uri* con la *URL* de la solicitud.
- Propiedades importantes de *MultipartRequest*:
  - **method**: Permite establecer el método *HTTP* para la solicitud. Generalmente se utiliza "*POST*" para enviar datos binarios.

MultipartRequest (2)

- Propiedades importantes de *MultipartRequest* (cont.):
  - **uri**: Objeto *Uri* que especifica la *URL* del servidor a la que se enviará la solicitud *multipart*.
  - **files**: Lista de objetos *http.MultipartFile*, que representa los archivos que se adjuntarán a la solicitud *multipart*. Cada archivo debe especificarse como un objeto *MultipartFile*, incluyendo su nombre, contenido y tipo de contenido (*MIME type*).
  - **fields**: *Map<String, String>* que permite agregar campos adicionales (clave - valor) a la solicitud *multipart*, si es necesario. Estos campos son datos adicionales que no son archivos binarios.

MultipartRequest (3)

- Propiedades importantes de *MultipartRequest* (cont.):
  - **headers**: *Map<String, String>* de encabezados *HTTP* personalizados que se desea enviar con la solicitud *multipart*.
- Para enviar la solicitud al servidor se utiliza el método *send()*.
- Este método envía la solicitud *multipart* al servidor y devuelve un objeto *Future<http.StreamedResponse>*.
- Se puede esperar la respuesta utilizando *await* para luego procesarla.