Lección 2: Consumo de APIs REST con Flutter y Manejo de Datos

Objetivos de la Lección

- **Comprender** cómo realizar peticiones HTTP en Flutter utilizando el paquete `http`.
- **Implementar** modelos de datos para usuarios, publicaciones y comentarios basados en los endpoints de JSONPlaceholder.
- **Manejar** datos obtenidos de las APIs REST y mapearlos a objetos de Dart.
- **Desarrollar** servicios para encapsular la lógica de comunicación con las APIs.
- **Visualizar** datos obtenidos de las APIs en la interfaz básica de la aplicación.

Introducción a la Lección

El consumo de APIs REST es una habilidad esencial en el desarrollo de aplicaciones móviles modernas. Permite a las aplicaciones interactuar con servicios web y obtener datos dinámicos. En esta lección, nos enfocaremos en cómo consumir los endpoints de **JSONPlaceholder** para obtener información de usuarios, publicaciones y comentarios. Aprenderemos a realizar peticiones HTTP, manejar respuestas y mapear datos JSON a modelos de Dart.

Esta lección es fundamental para construir las funcionalidades básicas de nuestra aplicación de red social simulada, ya que necesitamos obtener y mostrar datos reales en nuestra interfaz.

Desarrollo de Conceptos

¿Qué es una API REST?

Una API REST (Representational State Transfer) es un servicio web que utiliza los métodos HTTP estándar para permitir la comunicación entre un cliente y un servidor.

En nuestro caso, el cliente es nuestra aplicación Flutter, y el servidor es JSONPlaceholder.

El paquete http en Flutter

El paquete `http` es una biblioteca proporcionada por Dart que facilita la realización de peticiones HTTP y la recepción de respuestas. Permite enviar solicitudes GET, POST, PUT, DELETE, entre otras.

Mapeo de JSON a Objetos Dart

Los datos obtenidos de una API REST generalmente están en formato JSON. Necesitamos convertir este JSON en objetos de Dart para manejarlos fácilmente en nuestra aplicación.

Secciones Técnicas Específicas

1. Configuración del Paquete http

Ya hemos agregado el paquete `http` en el archivo `pubspec.yaml` en la lección anterior:

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.13.4
  shared_preferences: ^2.0.7
```

Asegúrate de que las dependencias estén actualizadas ejecutando:

```
flutter pub get
```

2. Creación de Modelos de Datos

2.1. Modelo `User`

Ya creamos el modelo `User` en la lección anterior. Verifiquemos que está completo:

```
// lib/models/user.dart
class User {
  final int id;
  final String name;
  final String username;
  final String email;
  // Puedes añadir más campos si lo deseas

  User({
    required this.id,
    required this.name,
    required this.username,
    required this.email,
  });

  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'],
      name: json['name'],
      username: json['username'],
      email: json['email'],
    );
  }
}
```

2.2. Modelo Post

```
// lib/models/post.dart
class Post {
  final int userId;
  final int id;
  final String title;
  final String body;

  Post({
    required this.userId,
    required this.id,
    required this.title,
    required this.body,
  });

  factory Post.fromJson(Map<String, dynamic> json) {
    return Post(
      userId: json['userId'],
      id: json['id'],
      title: json['title'],
      body: json['body'],
    );
  }
}
```

2.3. Modelo Comment

```
// lib/models/comment.dart
class Comment {
  final int postId;
```

```
  final int id;
  final String name;
  final String email;
  final String body;

  Comment({
    required this.postId,
    required this.id,
    required this.name,
    required this.email,
    required this.body,
  });

  factory Comment.fromJson(Map<String, dynamic> json) {
    return Comment(
      postId: json['postId'],
      id: json['id'],
      name: json['name'],
      email: json['email'],
      body: json['body'],
    );
  }
}
```

3. Creación de Servicios para Consumir las APIs

Para organizar mejor nuestro código, crearemos una clase `ApiService` dentro de **lib/services/api_service.dart**.

3.1. Servicio `ApiService`

```dart
// lib/services/api_service.dart
import 'dart:convert';
import 'package:http/http.dart' as http;
import '../models/user.dart';
import '../models/post.dart';
import '../models/comment.dart';

class ApiService {
  final String baseUrl = 'https://jsonplaceholder.typicode.com';

  // Obtener lista de usuarios
  Future<List<User>> getUsers() async {
    final response = await
http.get(Uri.parse('$baseUrl/users'));
    if (response.statusCode == 200) {
      List jsonResponse = json.decode(response.body);
      return jsonResponse.map((user) =>
User.fromJson(user)).toList();
    } else {
      throw Exception('Error al cargar usuarios');
    }
  }

  // Obtener publicaciones
  Future<List<Post>> getPosts() async {
    final response = await
http.get(Uri.parse('$baseUrl/posts'));
    if (response.statusCode == 200) {
      List jsonResponse = json.decode(response.body);
```

```
      return jsonResponse.map((post) =>
Post.fromJson(post)).toList();
    } else {
      throw Exception('Error al cargar publicaciones');
    }
  }

  // Obtener comentarios de una publicación
  Future<List<Comment>> getComments(int postId) async {
    final response = await
http.get(Uri.parse('$baseUrl/posts/$postId/comments'));
    if (response.statusCode == 200) {
      List jsonResponse = json.decode(response.body);
      return jsonResponse.map((comment) =>
Comment.fromJson(comment)).toList();
    } else {
      throw Exception('Error al cargar comentarios');
    }
  }

  // Añadir comentario a una publicación
  Future<Comment> addComment(int postId, String name, String
email, String body) async {
    final response = await http.post(
      Uri.parse('$baseUrl/comments'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: jsonEncode(<String, dynamic>{
        'postId': postId,
```

```
        'name': name,
        'email': email,
        'body': body,
      }),
    );

    if (response.statusCode == 201) {
      return Comment.fromJson(json.decode(response.body));
    } else {
      throw Exception('Error al añadir comentario');
    }
  }
}
```

Explicación

- **getUsers()**: Realiza una petición GET al endpoint `/users` y devuelve una lista de usuarios.
- **getPosts()**: Realiza una petición GET al endpoint `/posts` y devuelve una lista de publicaciones.
- **getComments(int postId)**: Obtiene los comentarios de una publicación específica.
- **addComment(...)**: Simula añadir un comentario a una publicación utilizando una petición POST.

4. Implementación de la Lógica para Consumir las APIs

4.1. Modificar la Pantalla Principal para Mostrar Publicaciones

Creamos un nuevo archivo **lib/screens/home_screen.dart**.

```
// lib/screens/home_screen.dart
```

```dart
import 'package:flutter/material.dart';
import '../models/post.dart';
import '../services/api_service.dart';

class HomeScreen extends StatefulWidget {
  final Function(ThemeMode) onThemeChanged;

  HomeScreen({required this.onThemeChanged});

  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  ApiService apiService = ApiService();
  late Future<List<Post>> futurePosts;

  @override
  void initState() {
    super.initState();
    futurePosts = apiService.getPosts();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Publicaciones'),
        actions: [
          PopupMenuButton<ThemeMode>(
```

```
            onSelected: widget.onThemeChanged,
            itemBuilder: (context) => [
              PopupMenuItem(
                value: ThemeMode.system,
                child: Text('Predeterminado'),
              ),
              PopupMenuItem(
                value: ThemeMode.light,
                child: Text('Modo Claro'),
              ),
              PopupMenuItem(
                value: ThemeMode.dark,
                child: Text('Modo Oscuro'),
              ),
            ],
          ),
        ],
      ),
      body: FutureBuilder<List<Post>>(
        future: futurePosts,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            List<Post> posts = snapshot.data!;
            return ListView.builder(
              itemCount: posts.length,
              itemBuilder: (context, index) {
                return Card(
                  margin: EdgeInsets.symmetric(vertical: 8,
horizontal: 16),
                  child: ListTile(
```

```
                    title: Text(posts[index].title),
                    subtitle: Text(posts[index].body),
                    onTap: () {
                      // Navegar a la pantalla de detalles de la
publicación
                    },
                  ),
                );
              },
            );
          } else if (snapshot.hasError) {
            return Center(child: Text('Error:
${snapshot.error}'));
          }
          return Center(child: CircularProgressIndicator());
        },
      ),
    );
  }
}
```

Explicación

- **FutureBuilder**: Utilizamos `FutureBuilder` para manejar datos asíncronos.
- **ListView.builder**: Construye una lista de publicaciones obtenidas de la API.
- **Card y ListTile**: Widgets para mostrar información de manera organizada y estilizada.

4.2. Modificar `main.dart` para Usar `HomeScreen`

En **lib/main.dart**, reemplaza `MyHomePage` por `HomeScreen`.

```dart
// lib/main.dart
import 'package:flutter/material.dart';
import 'screens/home_screen.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatefulWidget {
  // Este widget es la raíz de la aplicación.
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  ThemeMode _themeMode = ThemeMode.system;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Red Social Básica',
      theme: ThemeData(
        brightness: Brightness.light,
        primarySwatch: Colors.blue,
      ),
      darkTheme: ThemeData(
        brightness: Brightness.dark,
        primarySwatch: Colors.blue,
      ),
      themeMode: _themeMode, // Modo de tema seleccionado
```

```
      home: HomeScreen(
        onThemeChanged: _toggleThemeMode,
      ),
    );
  }

  void _toggleThemeMode(ThemeMode mode) {
    setState(() {
      _themeMode = mode;
    });
    // Aquí guardaríamos la preferencia del usuario
  }
}
```

5. Manejo de Errores y Excepciones

Es importante manejar posibles errores durante las peticiones HTTP.

- **Timeouts**: Podemos establecer un tiempo máximo de espera.
- **Errores de Red**: Manejar casos donde no hay conexión a Internet.
- **Códigos de Estado HTTP**: Actuar según el código de estado recibido.

Ejemplo modificando `getUsers()`:

```
Future<List<User>> getUsers() async {
  try {
    final response = await http
        .get(Uri.parse('$baseUrl/users'))
        .timeout(Duration(seconds: 10));
    if (response.statusCode == 200) {
      // Procesar la respuesta
```

```
    } else {
      throw Exception('Error al cargar usuarios');
    }
  } on Exception catch (e) {
    throw Exception('Error de red: $e');
  }
}
```

6. Actualizar la Interfaz para Mostrar Usuarios

Podemos crear una pantalla para mostrar la lista de usuarios.

6.1. Crear `users_screen.dart`

```
// lib/screens/users_screen.dart
import 'package:flutter/material.dart';
import '../models/user.dart';
import '../services/api_service.dart';

class UsersScreen extends StatefulWidget {
  @override
  _UsersScreenState createState() => _UsersScreenState();
}

class _UsersScreenState extends State<UsersScreen> {
  ApiService apiService = ApiService();
  late Future<List<User>> futureUsers;

  @override
  void initState() {
    super.initState();
```

```
    futureUsers = apiService.getUsers();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Usuarios'),
      ),
      body: FutureBuilder<List<User>>(
        future: futureUsers,
        builder: (context, snapshot) {
          if (snapshot.hasData) {
            List<User> users = snapshot.data!;
            return ListView.builder(
              itemCount: users.length,
              itemBuilder: (context, index) {
                return ListTile(
                  title: Text(users[index].name),
                  subtitle: Text('@${users[index].username}'),
                  onTap: () {
                    // Navegar al perfil del usuario
                  },
                );
              },
            );
          } else if (snapshot.hasError) {
            return Center(child: Text('Error:
${snapshot.error}'));
          }
```

```
          return Center(child: CircularProgressIndicator());
        },
      ),
    );
  }
}
```

6.2. Añadir Navegación entre Pantallas

Modificamos el `HomeScreen` para añadir un botón que nos lleve a la lista de usuarios.

```
// En el AppBar del HomeScreen
actions: [
  IconButton(
    icon: Icon(Icons.people),
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => UsersScreen()),
      );
    },
  ),
  // ... resto del código
],
```

7. Mostrar Detalles de una Publicación y sus Comentarios

7.1. Crear `post_detail_screen.dart`

```
// lib/screens/post_detail_screen.dart
import 'package:flutter/material.dart';
```

```
import '../models/post.dart';
import '../models/comment.dart';
import '../services/api_service.dart';

class PostDetailScreen extends StatefulWidget {
  final Post post;

  PostDetailScreen({required this.post});

  @override
  _PostDetailScreenState createState() =>
_PostDetailScreenState();
}

class _PostDetailScreenState extends State<PostDetailScreen> {
  ApiService apiService = ApiService();
  late Future<List<Comment>> futureComments;

  @override
  void initState() {
    super.initState();
    futureComments = apiService.getComments(widget.post.id);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Detalles de la Publicación'),
      ),
```

```
      body: SingleChildScrollView(
        child: Column(
          children: [
            ListTile(
              title: Text(widget.post.title),
              subtitle: Text(widget.post.body),
            ),
            Divider(),
            Text('Comentarios', style: TextStyle(fontSize: 18)),
            FutureBuilder<List<Comment>>(
              future: futureComments,
              builder: (context, snapshot) {
                if (snapshot.hasData) {
                  List<Comment> comments = snapshot.data!;
                  return ListView.builder(
                    shrinkWrap: true,
                    physics: NeverScrollableScrollPhysics(),
                    itemCount: comments.length,
                    itemBuilder: (context, index) {
                      return ListTile(
                        title: Text(comments[index].name),
                        subtitle: Text(comments[index].body),
                      );
                    },
                  );
                } else if (snapshot.hasError) {
                  return Center(child: Text('Error:
${snapshot.error}'));
                }
```

```
                return Center(child:
CircularProgressIndicator());
              },
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          // Navegar a la pantalla para añadir un comentario
        },
        child: Icon(Icons.add_comment),
      ),
    );
  }
}
```

7.2. Navegar a los Detalles de la Publicación

En **HomeScreen**, modifica el onTap del `ListTile`:

```
onTap: () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) =>
PostDetailScreen(post: posts[index])),
  );
},
```

8. Añadir Comentarios a una Publicación

La funcionalidad para añadir comentarios será implementada en lecciones posteriores, pero podemos preparar la navegación.

```
// En PostDetailScreen
onPressed: () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) =>
AddCommentScreen(postId: widget.post.id)),
  );
},
```

Ejemplos en Código

Ejemplo: Realizar una Petición GET

```
Future<List<Post>> getPosts() async {
  final response = await http.get(Uri.parse('$baseUrl/posts'));
  if (response.statusCode == 200) {
    List jsonResponse = json.decode(response.body);
    return jsonResponse.map((post) =>
Post.fromJson(post)).toList();
  } else {
    throw Exception('Error al cargar publicaciones');
  }
}
```

Explicación:

- **http.get**: Realiza una petición GET al endpoint especificado.

- **json.decode**: Convierte la respuesta JSON en una estructura de datos de Dart.
- **map**: Itera sobre la lista y convierte cada elemento en una instancia de `Post`.

Relación con Otros Temas

Esta lección se relaciona con la programación orientada a objetos al crear modelos de datos y con el manejo de APIs RESTful. También refuerza conceptos de programación asíncrona y manejo de futuros en Dart. Además, sienta las bases para las siguientes lecciones, donde implementaremos la interacción del usuario con los datos obtenidos y añadiremos funcionalidades más avanzadas.

Resumen de la Lección

En esta lección, hemos aprendido a consumir APIs REST utilizando el paquete `http` en Flutter. Creamos modelos de datos para usuarios, publicaciones y comentarios, y desarrollamos un servicio que encapsula la lógica de comunicación con las APIs. Implementamos la visualización de publicaciones y usuarios en la interfaz básica de la aplicación y establecimos la estructura para manejar comentarios y detalles de publicaciones.

Actividad de la Lección

Objetivo de la Actividad:

- Verificar que has comprendido cómo consumir APIs REST en Flutter.
- Demostrar tu capacidad para manejar datos y presentarlos en la interfaz.

- Prepararte para las siguientes lecciones, donde añadiremos interactividad y funcionalidades adicionales a la aplicación.

Tarea Práctica:

1. **Implementa los modelos de datos** para `User`, `Post` y `Comment` en tu proyecto.
2. **Crea el servicio `ApiService`** siguiendo los ejemplos proporcionados.
3. **Modifica la pantalla principal** para mostrar una lista de publicaciones obtenidas de la API.
4. **Añade una pantalla** para mostrar la lista de usuarios.
5. **Implementa la navegación** entre las diferentes pantallas (publicaciones, detalles de publicación, usuarios).
6. **Prueba la aplicación** en un emulador o dispositivo físico para verificar que los datos se cargan correctamente.
7. **Documenta tu proceso**: Crea un documento en PDF que incluya capturas de pantalla de las pantallas implementadas y explica cualquier problema que hayas encontrado y cómo lo solucionaste.
8. **Entrega**: Sube el PDF y el código fuente actualizado del proyecto en una carpeta comprimida.