Module 7: Networking and API Integration

Theory Assignment

Name:Chandvaniya Abhay

1. Explain what a RESTful API is and its importance in mobile applications.

What is a RESTful API?

A RESTful API:

- Uses **HTTP protocols** (the same protocol used by web browsers)
- Communicates via standard methods:
  - GET – retrieve data
  - POST – create data
  - PUT / PATCH – update data
  - DELETE – remove data
- Exchanges data typically in **JSON format**
- Treats data as **resources** (e.g., users, products, messages)
- Is **stateless** (each request contains all the information needed)

Example:

If a mobile app wants to get user data:

GET https://api.example.com/users/123

The server responds with JSON:

```
{
 "id": 123,
 "name": "John Doe",
 "email": "john@example.com"
}
```

Importance of RESTful APIs in Mobile Applications

Mobile apps (Android, iOS) do not usually store large amounts of data locally. Instead, they rely on RESTful APIs to communicate with backend servers.

1 Backend Communication

Mobile apps use REST APIs to:

- Log in users
- Fetch user profiles
- Upload photos
- Process payments
- Send messages

Without APIs, apps would not be able to connect to databases or servers.

2 Platform Independence

RESTful APIs work over HTTP, meaning:

- Android apps
- iOS apps
- Web apps
- Desktop apps

All can use the same backend service.

3 Scalability

REST is stateless, which makes it:

- Easier to scale
- Better for handling many mobile users
- Suitable for cloud environments

4 Faster Development

Developers can:

- Build backend and mobile app separately
- Reuse APIs across multiple platforms
- Update backend without changing the entire app

5 Lightweight Data Exchange

Since REST commonly uses JSON:

- Data transfer is small
- Mobile bandwidth is used efficiently
- Apps perform better on slower networks

2. Describe how JSON data is parsed and used in Flutter.

1 What is JSON in Flutter?

When a Flutter app calls a backend API (using packages like http), the server usually returns data in **JSON format**.

Example JSON response:

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```

Flutter (which uses the Dart language) cannot directly use this as an object — it must first **parse** the JSON.

2 How JSON is Parsed in Flutter

Flutter uses Dart’s built-in dart:convert library to parse JSON.

Step 1: Import the library

```
import 'dart:convert';
```

Step 2: Decode JSON string into a Map

```
String jsonString = '{"id":1,"name":"Alice","email":"alice@example.com"}';

Map<String, dynamic> userMap = jsonDecode(jsonString);

print(userMap['name']); // Alice
```

jsonDecode() converts:

- JSON → Map<String, dynamic> (for objects)
- JSON → List<dynamic> (for arrays)

3 Converting JSON to a Dart Model (Best Practice)

Instead of using raw Maps, Flutter developers create **model classes**.

Example JSON:

```
{
  "id": 1,
  "name": "Alice",
  "email": "alice@example.com"
}
```

Step 1: Create a Model Class

```
class User {
  final int id;
  final String name;
  final String email;

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

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

Step 2: Convert JSON to Object

```
Map<String, dynamic> jsonMap = jsonDecode(jsonString);
User user = User.fromJson(jsonMap);
```

```
print(user.name); // Alice
```

This process is called **deserialization** (JSON → Dart object).

4 Parsing JSON from an API in Flutter

Usually, JSON comes from an API call.

Example using http package:

```
import 'package:http/http.dart' as http;

Future<User> fetchUser() async {
  final response = await http.get(Uri.parse('https://api.example.com/user/1'));

  if (response.statusCode == 200) {
    return User.fromJson(jsonDecode(response.body));
  } else {
    throw Exception('Failed to load user');
  }
}
```

5 Handling JSON Arrays

If the API returns a list:

```
[
  {"id":1,"name":"Alice"},
  {"id":2,"name":"Bob"}
]
```

Parse it like this:

```
List<dynamic> jsonList = jsonDecode(response.body);
```

```
List<User> users =
   jsonList.map((json) => User.fromJson(json)).toList();
```

6 Using Parsed Data in Flutter UI

Once converted into Dart objects, the data can be used in widgets:

```
Text(user.name)
```

Or inside a ListView.builder for multiple items.

7 Why Parsing JSON Properly is Important

Type safety
Cleaner code
Easier debugging
Better scalability
Separation of concerns (API layer vs UI layer)

3. Explain the purpose of HTTP methods (GET, POST, PUT, DELETE) and when to use each.

1 GET – Retrieve Data

Purpose:

Used to **fetch or read data** from the server.

When to use:

- Getting a list of users
- Viewing a single product
- Fetching posts
- Loading profile information

Example:

GET /users
GET /users/10

Key Characteristics:

- Does **not modify** data
- Safe and idempotent
- Can be cached
- Parameters are usually sent in the URL

Use GET when you only want to **read data**.

2 POST – Create New Data

Purpose:

Used to **create a new resource** on the server.

When to use:

- Registering a new user
- Creating a new order
- Uploading a post
- Submitting a form

Example:

POST /users

Request body:

```
{
  "name": "Alice",
  "email": "alice@email.com"
}
```

Key Characteristics:

- Changes server data
- Not idempotent (multiple requests create multiple records)
- Data is sent in the request body

Use POST when you want to **add new data**.

3 PUT – Update Existing Data

Purpose:

Used to **update an existing resource** (usually replaces the entire resource).

When to use:

- Updating user profile
- Changing order details
- Editing product information

Example:

PUT /users/10

Request body:

```
{
  "name": "Alice Updated",
  "email": "alice_new@email.com"
}
```

Key Characteristics:

- Modifies existing data
- Idempotent (sending the same request multiple times results in the same outcome)
- Usually replaces the full resource

Use PUT when you want to **fully update existing data**.

4 DELETE – Remove Data

Purpose:

Used to **delete a resource** from the server.

When to use:

- Deleting a user account
- Removing a product
- Cancelling an order

Example:

DELETE /users/10

Key Characteristics:

- Removes data
- Idempotent (deleting the same resource multiple times has the same effect)

Use DELETE when you want to **remove data permanently**.