Module 7: Networking and API Integration

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

A. What is a RESTful API?

- REST = Representational State Transfer.
- A RESTful API is a way for software systems to communicate over the internet using HTTP.
- It follows a set of principles and conventions to make communication simple, scalable, and predictable.

In simple words:
A RESTful API lets your mobile app (client) talk to a server (backend) to fetch or send data (like user details, products, orders, etc.).

Key Characteristics of RESTful APIs

1. **Stateless →** Each request is independent; the server doesn't store client session.
2. **Client-Server Architecture →** The app (client) and backend (server) are separate.
3. **Use of HTTP Methods:**
   - **GET →** Retrieve data
   - **POST →** Create data
   - **PUT/PATCH →** Update data
   - **DELETE →** Remove data
4. **Resource-based URLs:**
   - **Example:**
     - **GET /users →** Fetch all users
     - **GET /users/1 →** Fetch user with ID 1
5. **Data Formats →** Usually JSON (lightweight, easy for mobile apps).

Importance of RESTful APIs in Mobile Applications

1. **Data Access & Exchange**
   - Mobile apps often need dynamic data (e.g., products, messages, weather).
   - REST APIs let apps fetch and send this data in real time.
2. **Scalability**
   - RESTful APIs are lightweight and stateless, making them efficient for high-traffic mobile apps.
3. **Platform Independence**
   - The same REST API can be used by iOS, Android, Web, or IoT devices.
4. **Modularity**
   - The frontend (mobile UI) is decoupled from the backend (database, business logic).
   - **This makes development faster and easier to maintain.**

5. **Integration with Third-Party Services**
    - Mobile apps use REST APIs to integrate with payment gateways (Stripe, PayPal), social media (Facebook, Twitter), maps (Google Maps), etc.
6. **Security & Authentication**
    - REST APIs often use tokens (JWT, OAuth) to secure communication between app and server.

Example in a Mobile App

Let's say you're building an e-commerce app:

- **To show products →** GET /products
- **To view details of a product →** GET /products/123
- **To add an item to the cart →** POST /cart
- **To update quantity in cart →** PUT /cart/123
- **To place order →** POST /orders

The mobile app doesn't directly talk to the database — instead, it communicates via RESTful APIs.

In summary:
A RESTful API is the backbone of modern mobile apps. It enables your app to talk to servers, fetch dynamic data, integrate with services, and stay scalable, secure, and platform-independent.

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

A. What is JSON?

- JSON = *JavaScript Object Notation* → lightweight format for exchanging data.
- Looks like a dictionary ({}) or list ([]).
- Example (user data):

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

How JSON Data is Parsed in Flutter

In Flutter (Dart), JSON is usually handled with the dart:convert library.

1. Decode JSON String → Dart Object

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

```
void main() {
  String jsonString = '{"id": 1, "name": "Alice"}';

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

  print(user['name']); // Alice
}
```

- jsonDecode() converts a JSON string into a Map (for objects) or List (for arrays).

---

2. Parse JSON into a Model Class

Instead of using raw Maps, you usually create a model class for cleaner code.

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

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

  // Factory constructor to create a User from JSON
  factory User.fromJson(Map<String, dynamic> json) {
    return User(
      id: json['id'],
      name: json['name'],
      email: json['email'],
    );
  }

  // Convert User back to JSON
  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'name': name,
      'email': email,
    };
  }
}
```

---

3. Using JSON with the Model

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

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

  print(user.name); // Alice
  print(user.toJson()); // {id: 1, name: Alice, email: alice@example.com}
}
```

---

4. Parsing a JSON List

If the API returns multiple objects:

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

```
String jsonString = '''
[
  {"id": 1, "name": "Alice"},
  {"id": 2, "name": "Bob"}
]
''';

List<dynamic> jsonList = jsonDecode(jsonString);

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

print(users[0].name); // Alice
```

Typical Flow in a Flutter App

1. Use http package to fetch data from a REST API.
2. Convert the response body (JSON string) into Dart objects using jsonDecode.
3. Map the JSON into your model classes (fromJson).
4. Use those models in your widgets (e.g., ListView to display data).

In summary:

- JSON in Flutter is parsed using dart:convert → jsonDecode().
- Best practice is to create model classes with fromJson and toJson methods.
- This makes your code clean, type-safe, and easy to maintain.

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

A. HTTP Methods and Their Purpose

1. GET

- **Purpose:** Retrieve (read) data from the server.
- **Characteristics:**
  - Doesn't change anything on the server.
  - Safe and idempotent (repeating it has no side effects).
- **When to Use:**
  - Fetching a list of items (e.g., products, users).
  - Loading details of a single resource.
- **Example:**
  - **GET /users →** fetch all users.
  - **GET /users/5 →** fetch user with ID 5.

---

2. POST

- **Purpose:** Create a new resource on the server.
- **Characteristics:**
  - Sends data in the request body (usually JSON).
  - Causes a change on the server (not idempotent).
- **When to Use:**
  - User registration.

  - Submitting a form.
  - Adding a new item to a database.
- **Example:**

```
POST /users with body:
{
  "name": "Alice",
  "email": "alice@example.com"
}
```

→ Creates a new user.

---

3. PUT

- **Purpose:** Update an existing resource (replace it entirely).
- **Characteristics:**
  - Requires sending the full updated data.
  - Idempotent (sending the same request multiple times gives the same result).
- **When to Use:**
  - Replacing an existing resource with new data.
  - Updating an entire object.
- **Example:**

```
PUT /users/5 with body:
{
  "id": 5,
  "name": "Alice Smith",
  "email": "alice@example.com"
}
```

→ Updates user 5 with the new info.

---

4. PATCH (often mentioned with PUT)

- **Purpose:** Update part of a resource.
- **When to Use:**
  - Updating only specific fields instead of replacing everything.
- **Example:**

```
PATCH /users/5 with body:
{ "email": "newalice@example.com" }
```

→ Updates only the email of user 5.

---

5. DELETE

- **Purpose:** Remove a resource from the server.
- **Characteristics:**
    - Idempotent (deleting the same resource again has no further effect).
- **When to Use:**
    - Removing a user account.
    - Deleting a product from inventory.
- **Example:**

  DELETE /users/5 → deletes user with ID 5.

Real Example in a Mobile App (E-commerce)

- **GET /products →** Load product list.
- **GET /products/10 →** Load product details.
- **POST /cart →** Add product to cart.
- **PUT /cart/3 →** Update item quantity in cart (replace old value).
- **PATCH /cart/3 →** Update just the quantity field.
- **DELETE /cart/3 →** Remove item from cart.

In summary:

- **GET →** Read
- **POST →** Create
- **PUT →** Replace/Update fully
- **PATCH →** Update partially
- **DELETE →** Remove