How to integrate Api into flutter app

1: Introduction to API Integration in Flutter

- **What is an API?**
  - API stands for Application Programming Interface.
  - It allows communication between two software applications.
- **Importance of API in Mobile Apps**
  - Access data from external services.
  - Keep app data dynamic and updated.
- **Flutter's HTTP package**
  - Commonly used package: http (install with flutter pub add http).

2: Setting Up the Project

- **Step 1: Add the HTTP Package**
  - Add the dependency in pubspec.yaml:

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.14.0
```

Step 2: Import the Package

- Import http in your Dart file:

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

3: Making a GET Request

- **Explanation of GET Request**
  - Used to fetch data from an API.
- **Example Code:**

```
Future<void> fetchData() async {
  final response = await http.get(Uri.parse('https://api.example.com/data'));

  if (response.statusCode == 200) {
    // Successful response
    print(response.body);
  } else {
    // Error response
    throw Exception('Failed to load data');
  }
}
```

Handling Response

- Check the status code.
- Parse the JSON data using dart:convert.

4: Parsing JSON Data

- **Understanding JSON Structure**
    - JSON: JavaScript Object Notation, a lightweight data format.
- **Example Code:**

```
import 'dart:convert';


void parseJson(String responseBody) {
  final parsed = json.decode(responseBody);
  print(parsed['key']);
}
```

- **Using dart:convert**
    - json.decode() to convert JSON string into a Map.

5: Displaying Data in the UI

- **Connecting API to the UI**
  - Use FutureBuilder to handle async operations.
- **Example Code:**

```
FutureBuilder(
 future: fetchData(),
 builder: (context, snapshot) {
  if (snapshot.connectionState == ConnectionState.done) {
   if (snapshot.hasData) {
    return Text(snapshot.data.toString());
   } else {
    return Text('Error: ${snapshot.error}');
   }
  } else {
   return CircularProgressIndicator();
  }
 },
);
```

Best Practices

- Error handling.
- Data caching for performance.