Flutter API Integration: Fetching Data & Displaying in App

Subtitle: Learn how to connect your Flutter app with REST APIs to fetch and display dynamic data.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Most real-world apps rely on **external data**. Understanding how to connect your Flutter app with APIs is crucial for building functional apps. This guide covers **fetching data, parsing JSON, and displaying it in your Flutter UI**.

Step 1: Add HTTP Package

1. Open `pubspec.yaml`
2. Add dependency:

```
dependencies:
  http: ^0.14.0
```

3. Run:

```
flutter pub get
```

Step 2: Import HTTP Library

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

Step 3: Fetch Data from API

Example using JSONPlaceholder API:

```
Future<List> fetchPosts() async {
  final response = await http.get(Uri.parse('https://jsonplaceholder.typicode.co
```

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

Exercise: Try fetching data from any public API.

Step 4: Display Data Using FutureBuilder

```
FutureBuilder<List>(
  future: fetchPosts(),
  builder: (context, snapshot) {
    if (snapshot.connectionState == ConnectionState.waiting) {
      return CircularProgressIndicator();
    } else if (snapshot.hasError) {
      return Text('Error: ${snapshot.error}');
    } else {
      final posts = snapshot.data!;
      return ListView.builder(
        itemCount: posts.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(posts[index]['title']),
            subtitle: Text(posts[index]['body']),
          );
        },
      );
    }
  },
)
```

Exercise: Display the first 10 posts with title and body.

Step 5: Handling Loading & Errors

- Show **CircularProgressIndicator** while fetching
- Handle errors gracefully using `snapshot.hasError`

- Avoid empty UI states

Step 6: Using Models for Data

Define a **Post model** for cleaner code:

```
class Post {
  final int id;
  final String title;
  final String body;

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

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

Convert JSON list to List:

```
List<Post> posts = (jsonDecode(response.body) as List)
    .map((json) => Post.fromJson(json))
    .toList();
```

Exercise: Refactor your API code to use models instead of raw JSON.

Step 7: Mini Project

Build a **Posts App**:

- Fetch posts from API
- Display in a scrollable ListView
- Show loading spinner while fetching
- Handle errors gracefully

Key Takeaways

- Use `http` package to fetch data from APIs
- Parse JSON and display in UI
- Use FutureBuilder to manage async data
- Use models to organize API data
- Always handle loading states and errors

API integration makes your apps **dynamic and real-world ready**. Without this, your apps are static and not usable beyond demo purposes.

Visit **haas.dev** for more Flutter guides, tutorials, and complete project resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app
