**Connecting a Flutter Application to an API**

**Step-by-Step Guide**

1. **Add Dependencies:**
   * Add the http package to your pubspec.yaml file.
   * Run the command to install the dependencies.
2. **Create a Service Class:**
   * Create a class dedicated to handling API requests.
   * Include methods for different types of HTTP requests (GET, POST, etc.).
3. **Set Up the Service Class:**
   * Implement methods to make network requests.
   * Handle responses and potential errors.
4. **Use the Service Class in Your App:**
   * Create instances of the service class where needed.
   * Call the service class methods to fetch or send data.
5. **Update the UI Based on API Data:**
   * Use the data fetched from the API to update the UI.
   * Handle loading states and errors gracefully.
6. **Test API Calls:**
   * Test the API calls to ensure they return the expected data.
   * Debug any issues that arise during API communication.
7. **Implement State Management (Optional):**
   * Use state management solutions like Provider to manage API data and state across the app.
   * Ensure the UI updates correctly in response to state changes.

**Before We Proceed…**

**1. Future**

* **Definition:** Represents a potential value or error that will be available at some point in the future.
* **Importance:** Futures are essential for handling asynchronous operations, such as network requests, without blocking the main thread.

**2. async**

* **Definition:** A keyword that allows a function to run asynchronously and return a Future.
* **Importance:** Enables writing asynchronous code that looks synchronous, making it easier to read and maintain. Functions marked with async can use await to pause execution until the awaited Future completes.

**3. await**

* **Definition:** A keyword used to pause the execution of an async function until the Future it is waiting on completes.
* **Importance:** Simplifies handling asynchronous operations by allowing code to wait for the result of a Future before proceeding. This helps avoid complex callback chains and improves code readability.

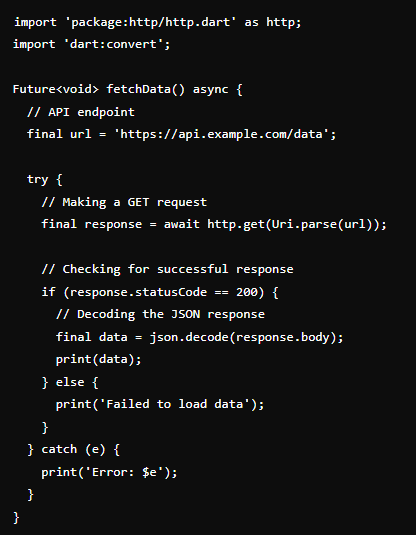
**4. HTTP Methods ( GET, POST, PUT, DELETE)**

**5. JSON Encode/Decode**

* **json.encode:** Converts a Dart object to a JSON string.
  + **Importance:** Used when sending data to the server in a format that the server can understand.
* **json.decode:** Converts a JSON string to a Dart object.
  + **Importance:** Used when receiving data from the server and converting it into a Dart object for further processing.

**Code Example:**

Get Request



Post Request

