TUGAS 8 Rest API

PEMROGRAMAN PERANGKAT BERGERAK

Disusun Oleh :

Ahmad Junaidi / 2211104002

SE-06-01

Dosen Pengampu :

Yudha Islami Sulistya, S.Kom., M.Cs.

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

REST API kali ini saya mengguakan https://jsonplaceholder.typicode.com/ yang merupakan layanan dummy untuk pengujian API.

Berikut adalah penjelasan mengenai apa saja yang digunakan dalam program saya dan bagaimana kaitannya dengan REST API:

1. Metode HTTP yang digunakan

Program Anda memanfaatkan beberapa metode HTTP standar yang umum dalam REST API:

1. GET: Mengambil data dari server.
    a. Anda menggunakan ini di fungsi `fetchPosts` untuk mengambil daftar posting dari server.
2. POST: Menambahkan data baru ke server.
    a. Anda menggunakan ini di fungsi `createPost` untuk membuat posting baru.
3. PUT: Memperbarui data yang ada di server.
    a. Anda menggunakan ini di fungsi `updatePost` untuk mengubah data pada posting tertentu.
4. DELETE: Menghapus data dari server.
    a. Anda menggunakan ini di fungsi `deletePost` untuk menghapus posting tertentu.

2. Library yang Digunakan

Program Anda memanfaatkan beberapa library Flutter/Dart untuk mengimplementasikan REST API:

- **`http`**: Library HTTP yang digunakan untuk melakukan request (GET, POST, PUT, DELETE) ke server.
- **`get`**: Library GetX untuk manajemen state dan navigasi. Anda menggunakan **Obx** untuk memantau perubahan state data (misalnya, `posts` dan `isLoading`).

3. . Struktur Program

a. ApiService

File ini berisi semua logika untuk melakukan request ke REST API. Setiap metode seperti fetchPosts, createPost, updatePost, dan deletePost bertugas berinteraksi langsung dengan server.

b. ApiController

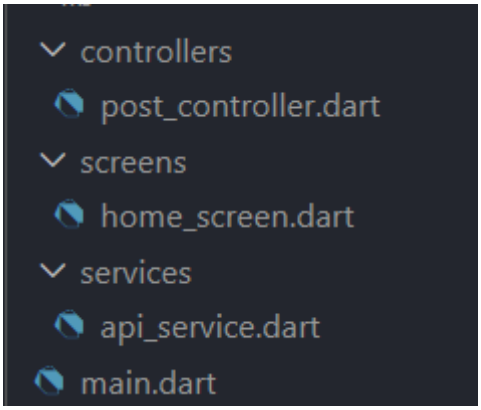
Controller ini bertindak sebagai penghubung antara **API service** dan **UI** (User Interface). Controller ini:

- Memantau perubahan data melalui **Reactive Variables** seperti posts dan isLoading.

- Menampilkan **Snackbar** untuk memberi tahu pengguna mengenai status request.

c. HomeScreen

Halaman utama aplikasi yang memanfaatkan controller untuk memicu request ke API dan menampilkan data di layar. Data yang ditampilkan berupa daftar posting yang diperoleh dari REST API.



4. Komponen REST API

Saya menggunakan endpoint REST API dari https://jsonplaceholder.typicode.com. Contoh:

- GET /posts: Mengambil semua posting.
- POST /posts: Membuat posting baru.
- PUT /posts/1: Memperbarui posting dengan ID 1.
- DELETE /posts/1: Menghapus posting dengan ID 1.

File Program:

Main.dart

```dart
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:modul_14/screens/home_screen.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      title: 'Flutter GetX Demo',
```

```
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const HomeScreen(),
    );
  }
}
```

api.service.dart

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

class ApiService {
  final String baseUrl = "https://jsonplaceholder.typicode.com";
  List<dynamic> posts = [];

  // HTTP GET
  Future<void> fetchPosts() async {
    final response = await http.get(Uri.parse('$baseUrl/posts'));

    if (response.statusCode == 200) {
      posts = json.decode(response.body);
    } else {
      throw Exception('Failed to load posts');
    }
  }

  // HTTP POST
  Future<void> createPost() async {
    final response = await http.post(
      Uri.parse('$baseUrl/posts'),
      headers: {'Content-Type': 'application/json'},
      body: json.encode(
          {'title': 'Flutter Post', 'body': 'Contoh POST', 'userId': 1}),
    );

    if (response.statusCode == 201) {
      posts.add({
        'title': 'Flutter Post',
        'body': 'Contoh POST',
        'id': posts.length + 1
      });
    } else {
      throw Exception('Failed to create post');
```

```
    }
  }

  // HTTP PUT
  Future<void> updatePost() async {
    final response = await http.put(
      Uri.parse('$baseUrl/posts/1'),
      body: json.encode(
          {'title': 'Updated Title', 'body': 'Updated Body', 'userId': 1}),
    );

    if (response.statusCode == 200) {
      final updatedPost = posts.firstWhere((post) => post['id'] == 1);
      updatedPost['title'] = 'Updated Title';
      updatedPost['body'] = 'Updated Body';
    } else {
      throw Exception('Failed to update post');
    }
  }

  // HTTP DELETE
  Future<void> deletePost() async {
    final response = await http.delete(Uri.parse('$baseUrl/posts/1'));

    if (response.statusCode == 200) {
      posts.removeWhere((post) => post['id'] == 1);
    } else {
      throw Exception('Failed to delete post');
    }
  }
}
```

Home_screen.dart

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:modul_14/controllers/post_controller.dart';

class HomeScreen extends StatelessWidget {
  const HomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    final ApiController controller = Get.put(ApiController());

    return Scaffold(
```

```
      appBar: AppBar(
        title: const Text('HTTP Request Example with GetX'),
        centerTitle: true,
        backgroundColor: Colors.blue,
      ),
      body: Padding(
        padding: const EdgeInsets.all(12.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Obx(() => controller.isLoading.value
                ? const Center(child: CircularProgressIndicator())
                : controller.posts.isEmpty
                    ? const Text(
                        "Tekan tombol GET untuk mengambil data",
                        style: TextStyle(fontSize: 12),
                      )
                    : Expanded(
                        child: ListView.builder(
                          itemCount: controller.posts.length,
                          itemBuilder: (context, index) {
                            return Card(
                              elevation: 4,
                              child: ListTile(
                                title: Text(
                                  controller.posts[index]['title'],
                                  style: const TextStyle(
                                      fontWeight: FontWeight.bold,
                                      fontSize: 12),
                                ),
                                subtitle: Text(
                                  controller.posts[index]['body'],
                                  style: const TextStyle(fontSize: 12),
                                ),
                              ),
                            );
                          },
                        ),
                      )),
            const SizedBox(height: 20),
            ElevatedButton(
              onPressed: controller.fetchPosts,
              style: ElevatedButton.styleFrom(backgroundColor:
Colors.orange),
              child: const Text('GET'),
            ),
            ElevatedButton(
              onPressed: controller.createPost,
```

```
              style: ElevatedButton.styleFrom(backgroundColor:
Colors.green),
              child: const Text('POST'),
            ),
            ElevatedButton(
              onPressed: controller.updatePost,
              style: ElevatedButton.styleFrom(backgroundColor: Colors.blue),
              child: const Text('UPDATE'),
            ),
            ElevatedButton(
              onPressed: controller.deletePost,
              style: ElevatedButton.styleFrom(backgroundColor: Colors.red),
              child: const Text('DELETE'),
            ),
          ],
        ),
      ),
    );
  }
}
```

Post_controller.dart

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

class ApiController extends GetxController {
  final String baseUrl = "https://jsonplaceholder.typicode.com";

  var posts = <dynamic>[].obs;
  var isLoading = false.obs;

  // Snackbar helper
  void showSuccessSnackBar(String message) {
    Get.snackbar(
      'Success',
      message,
      backgroundColor: Colors.green,
      colorText: Colors.white,
      snackPosition: SnackPosition.BOTTOM,
      duration: const Duration(seconds: 2),
    );
  }
```

```dart
void showErrorSnackBar(String message) {
  Get.snackbar(
    'Error',
    message,
    backgroundColor: Colors.red,
    colorText: Colors.white,
    snackPosition: SnackPosition.BOTTOM,
    duration: const Duration(seconds: 2),
  );
}

// GET Posts
Future<void> fetchPosts() async {
  isLoading.value = true;
  try {
    final response = await http.get(Uri.parse('$baseUrl/posts'));
    if (response.statusCode == 200) {
      posts.value = json.decode(response.body);
      showSuccessSnackBar('Data berhasil diambil!');
    } else {
      throw Exception('Failed to load posts');
    }
  } catch (e) {
    showErrorSnackBar('Error: $e');
  } finally {
    isLoading.value = false;
  }
}

// POST Data
Future<void> createPost() async {
  isLoading.value = true;
  try {
    final response = await http.post(
      Uri.parse('$baseUrl/posts'),
      headers: {'Content-Type': 'application/json'},
      body: json.encode({
        'title': 'Flutter Post',
        'body': 'Ini contoh POST.',
        'userId': 1,
      }),
    );
    if (response.statusCode == 201) {
      posts.add({
        'title': 'Flutter Post',
        'body': 'Ini contoh POST.',
        'id': posts.length + 1,
      });
```

```
        showSuccessSnackBar('Data berhasil ditambahkan!');
      } else {
        throw Exception('Failed to create post');
      }
    } catch (e) {
      showErrorSnackBar('Error: $e');
    } finally {
      isLoading.value = false;
    }
  }

  // UPDATE Data
  Future<void> updatePost() async {
    isLoading.value = true;
    try {
      final response = await http.put(
        Uri.parse('$baseUrl/posts/1'),
        body: json.encode({
          'title': 'Updated Title',
          'body': 'Updated Body',
          'userId': 1,
        }),
      );
      if (response.statusCode == 200) {
        var updatedPost = posts.firstWhere((post) => post['id'] == 1);
        updatedPost['title'] = 'Updated Title';
        updatedPost['body'] = 'Updated Body';
        showSuccessSnackBar('Data berhasil diperbarui!');
      } else {
        throw Exception('Failed to update post');
      }
    } catch (e) {
      showErrorSnackBar('Error: $e');
    } finally {
      isLoading.value = false;
    }
  }

  // DELETE Data
  Future<void> deletePost() async {
    isLoading.value = true;
    try {
      final response = await http.delete(Uri.parse('$baseUrl/posts/1'));
      if (response.statusCode == 200) {
        posts.removeWhere((post) => post['id'] == 1);
        showSuccessSnackBar('Data berhasil dihapus!');
      } else {
        throw Exception('Failed to delete post');
```

```
      }
    } catch (e) {
      showErrorSnackBar('Error: $e');
    } finally {
      isLoading.value = false;
    }
  }
}
```

pubspec.yaml

```
name: modul_14
description: "A new Flutter project."
# The following line prevents the package from being accidentally published
to
# pub.dev using `flutter pub publish`. This is preferred for private
packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as
versionCode.
# Read more about Android versioning at
https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-
number is used as CFBundleVersion.
# Read more about iOS versioning at
#
https://developer.apple.com/library/archive/documentation/General/Reference/
InfoPlistKeyReference/Articles/CoreFoundationKeys.html
# In Windows, build-name is used as the major, minor, and patch parts
# of the product and file versions while build-number is used as the build
suffix.
version: 1.0.0+1

environment:
  sdk: ^3.5.4

# Dependencies specify other packages that your package needs in order to
work.
# To automatically upgrade your package dependencies to the latest versions
# consider running `flutter pub upgrade --major-versions`. Alternatively,
```

```yaml
# dependencies can be manually updated by changing the version numbers below
to
# the latest version available on pub.dev. To see which dependencies have
newer
# versions available, run `flutter pub outdated`.
dependencies:
  flutter:
    sdk: flutter


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.8
  http: ^1.2.2
  get: ^4.6.6

dev_dependencies:
  flutter_test:
    sdk: flutter

  # The "flutter_lints" package below contains a set of recommended lints to
  # encourage good coding practices. The lint set provided by the package is
  # activated in the `analysis_options.yaml` file located at the root of
your
  # package. See that file for information about deactivating specific lint
  # rules and activating additional ones.
  flutter_lints: ^4.0.0

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  # assets:
  #   - images/a_dot_burr.jpeg
  #   - images/a_dot_ham.jpeg

  # An image asset can refer to one or more resolution-specific "variants",
see
  # https://flutter.dev/to/resolution-aware-images

  # For details regarding adding assets from package dependencies, see
```

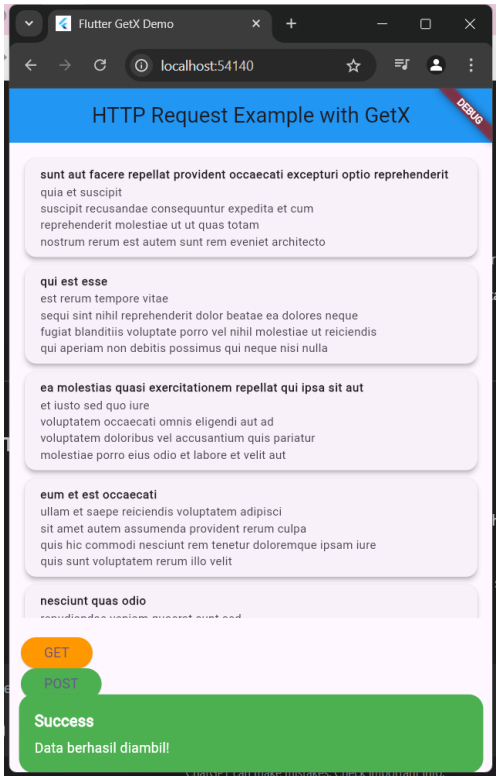
```
# https://flutter.dev/to/asset-from-package

# To add custom fonts to your application, add a fonts section here,
# in this "flutter" section. Each entry in this list should have a
# "family" key with the font family name, and a "fonts" key with a
# list giving the asset and other descriptors for the font. For
# example:
# fonts:
#   - family: Schyler
#     fonts:
#       - asset: fonts/Schyler-Regular.ttf
#       - asset: fonts/Schyler-Italic.ttf
#         style: italic
#   - family: Trajan Pro
#     fonts:
#       - asset: fonts/TrajanPro.ttf
#       - asset: fonts/TrajanPro_Bold.ttf
#         weight: 700
#
# For details regarding fonts from package dependencies,
# see https://flutter.dev/to/font-from-package
```

Output:

Tampilan Awal:

Fitur get



penjelasan get:

Fungsi:

- Tombol ini digunakan untuk mengambil data dari server melalui HTTP GET request.
- Menggunakan ApiService.fetchPosts() di dalam PostController untuk mendapatkan data dari API.

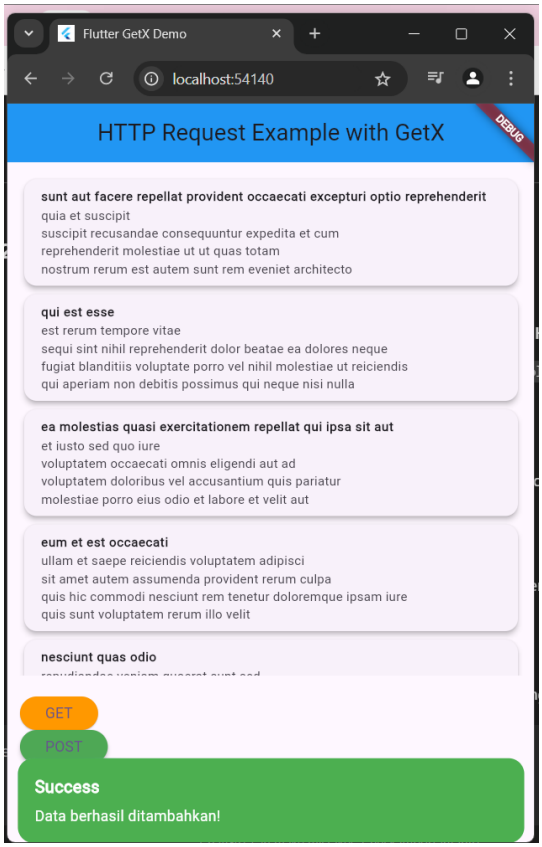
Tampilan Hasil:

- Ketika tombol GET ditekan, data dari API akan ditampilkan dalam bentuk ListView.
- Jika data berhasil diambil, akan muncul Snackbar berisi pesan: "Berhasil: Data berhasil diambil".
- Jika gagal, muncul Snackbar dengan warna merah: "Error: Gagal mengambil data".

Penggunaan State Management:

- posts.assignAll() digunakan untuk menyimpan data ke dalam state GetX, sehingga tampilan ListView akan diperbarui otomatis dengan Obx.

  Fitur POST



Penjelasan POST:

Fungsi:

- Tombol ini digunakan untuk menambah data ke server melalui HTTP POST request.
- Menggunakan ApiService.createPost() di dalam PostController untuk menambahkan data baru.

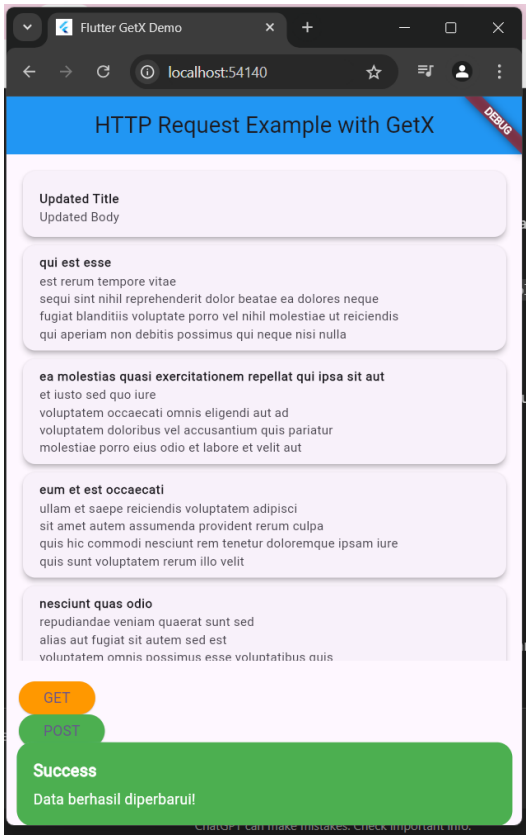
Tampilan Hasil:

- Setelah tombol POST ditekan, data baru akan ditambahkan ke daftar ListView.
- Snackbar sukses akan muncul dengan pesan: "Berhasil: Data berhasil ditambahkan".
- Jika terjadi kesalahan, akan muncul Snackbar dengan warna merah: "Error: Gagal menambah data".

Penjelasan Teknis:

- Setelah data ditambahkan, fungsi fetchPosts() dipanggil ulang agar tampilan diperbarui dengan data terbaru.
- State posts diperbarui otomatis berkat penggunaan GetX.

Fitur PUT/UPDATE



Penjelasan Update:

Fungsi:

- Tombol ini digunakan untuk memperbarui data yang sudah ada di server menggunakan HTTP PUT request.
- Menggunakan ApiService.updatePost() di dalam PostController untuk mengubah data.

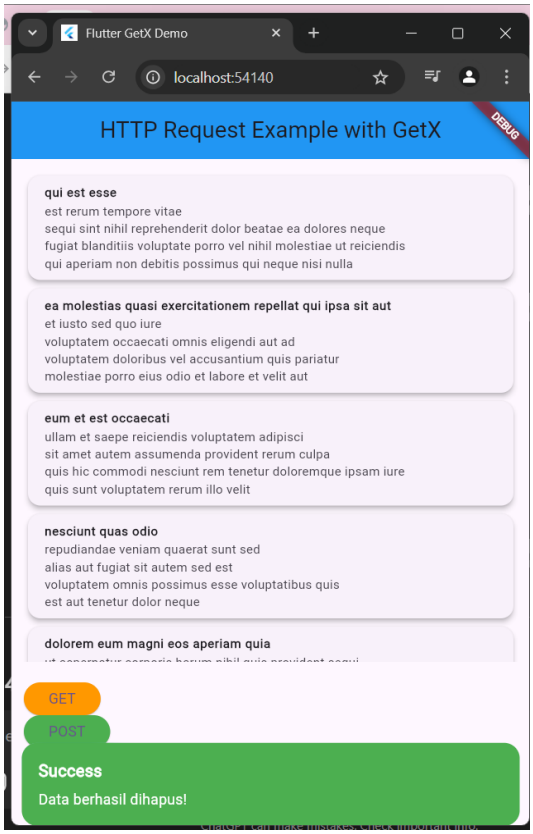
Tampilan Hasil:

- Setelah tombol PUT ditekan, data pada ListView akan diperbarui.
- Snackbar sukses akan muncul dengan pesan: "Berhasil: Data berhasil diperbarui".
- Jika ada kesalahan, muncul Snackbar dengan warna merah: "Error: Gagal memperbarui data".

Penjelasan Teknis:

- Data dengan ID tertentu akan diperbarui di server.
- Setelah berhasil, fetchPosts() dipanggil ulang untuk memperbarui state GetX.

Fitur DELETE



Penjelasan DELETE:

- Fungsi:
    - Tombol ini digunakan untuk menghapus data dari server melalui HTTP DELETE request.
    - Menggunakan ApiService.deletePost() di dalam PostController untuk menghapus data.
- Tampilan Hasil:
    - Setelah tombol DELETE ditekan, data dengan ID tertentu akan dihapus.
    - Data yang dihapus tidak akan lagi muncul di ListView.
    - Snackbar sukses akan muncul dengan pesan: "Berhasil: Data berhasil dihapus".
    - Jika ada kesalahan, muncul Snackbar dengan warna merah: "Error: Gagal menghapus data".
- Penjelasan Teknis:
    - Fungsi ini memanggil deletePost() yang menghapus data dari server.
    - Setelah sukses, data diperbarui melalui fetchPosts(), sehingga tampilan ListView ikut diperbarui otomatis.

Deskripsi

Aplikasi ini adalah CRUD sederhana (Create, Read, Update, Delete) yang menggunakan State Management GetX untuk mengelola data secara efisien. Aplikasi ini berinteraksi dengan API eksternal untuk mengambil, menambah, memperbarui, dan menghapus data. Dengan dukungan Snackbar, pengguna mendapatkan respon instan setelah setiap operasi berhasil atau gagal, sehingga memberikan pengalaman pengguna yang lebih baik dan informatif.

Aplikasi ini cocok untuk memahami konsep dasar integrasi API dengan Flutter, penerapan GetX sebagai state management, dan feedback notifikasi menggunakan Snackbar.

Fitur Utama Aplkasi

GET Data (Read)

- Mengambil data dari API dan menampilkannya dalam bentuk ListView.
- Data diperbarui secara otomatis berkat penggunaan GetX State Management.

POST Data (Create)

- Menambahkan data baru ke server menggunakan HTTP POST request.
- Data baru akan langsung tampil di ListView setelah berhasil ditambahkan.
- Menampilkan Snackbar sukses sebagai notifikasi umpan balik.

PUT Data (Update)

- Memperbarui data yang ada di server menggunakan HTTP PUT request.
- Data yang diperbarui akan ditampilkan di ListView secara real-time.
- Menampilkan Snackbar sukses atau error jika operasi gagal.

DELETE Data (Delete)

- Menghapus data dari server menggunakan HTTP DELETE request.
- Data yang dihapus akan langsung hilang dari ListView.
- Menampilkan Snackbar sukses atau error untuk memberikan notifikasi hasil operasi.

Snackbar Notifikasi

- Menggunakan Get.snackbar untuk memberikan notifikasi instan.
- Snackbar akan muncul saat operasi berhasil (warna hijau) atau gagal (warna merah).

Cara Kerja Aplikasi

1. State Management dengan GetX

- PostController:

- Mengelola data dari API menggunakan ApiService (GET, POST, PUT, DELETE).
- Menyimpan data di dalam state RxList (posts) agar perubahan data langsung mempengaruhi tampilan.
- State dikelola menggunakan Obx, sehingga tampilan diperbarui otomatis

2. Operasi CRUD

GET:

- Memanggil ApiService.fetchPosts() untuk mengambil data dari API.
- Data ditampilkan dalam ListView.

POST:

- Memanggil ApiService.createPost() untuk menambahkan data baru.
- Data akan langsung ditambahkan ke dalam state posts dan ditampilkan.

PUT:

- Memanggil ApiService.updatePost() untuk memperbarui data berdasarkan ID.
- State diperbarui agar tampilan otomatis merefleksikan perubahan.

DELETE:

- Memanggil ApiService.deletePost() untuk menghapus data berdasarkan ID.
- Data dihapus dari state posts dan ListView diperbarui.

3. snackbar Notifikasi

Setelah setiap operasi (GET, POST, PUT, DELETE), Snackbar akan ditampilkan menggunakan Get.snackbar:

- Sukses: Menampilkan pesan positif.
- Error: Menampilkan pesan dengan background merah.

4. interaksi Pengguna:

Pengguna menekan tombol GET, POST, PUT, atau DELETE pada UI.

Tombol memanggil fungsi di PostController, yang berinteraksi dengan API melalui ApiService.

Data akan ditampilkan atau diperbarui di layar berkat GetX State Management.

Respon sukses atau error ditampilkan melalui Snackbar untuk memberikan feedback langsung kepada pengguna.