PRAKTIKUM PEMROGRAMAN PERANGKAT BERGERAK
TUGAS GUIDED & UNGUIDED

MODUL XIV
DATA STORAGE (API)

Disusun Oleh :

Rizky Hanifa Afania / 2211104017

SE-06-01

Asisten Praktikum :

Muhammad Faza Zulian Gesit Al Barru

Aisyah Hasna Aulia

Dosen Pengampu :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

GUIDED

A. REST API

REST API (Representational State Transfer Application Programming Interface) adalah sebuah antarmuka yang memungkinkan aplikasi klien untuk berkomunikasi dengan database melalui protokol HTTP. REST API memberikan cara untuk mengakses data, seperti membaca, menambah, memperbarui, dan menghapus informasi dari database, tanpa perlu langsung mengakses database itu sendiri. Selain itu, REST API juga memungkinkan pengambilan token unik dari setiap perangkat pengguna.

Manfaat REST API:

1. **Interoperabilitas**: REST API memungkinkan aplikasi web dan mobile untuk mengakses data yang serupa.
2. **Efisiensi**: Data yang dikirim umumnya berukuran kecil (dalam format JSON atau XML), sehingga proses pengiriman dan penerimaannya lebih cepat.
3. **Keamanan**: API dapat membatasi akses dengan menggunakan token autentikasi.

B. HTTP

HTTP (Hypertext Transfer Protocol) adalah protokol komunikasi utama yang digunakan untuk mentransfer data antara klien (seperti browser atau aplikasi) dan server.
Metode HTTP Utama:

1. **GET**: Untuk mengambil data dari server.
2. **POST**: Untuk mengirim data baru ke server.
3. **PUT/PATCH**: Untuk memperbarui data yang ada di server.
4. **DELETE**: Untuk menghapus data dari server.

Komponen HTTP Request:

1. **URL**: Alamat yang menunjukkan sumber daya tertentu.
2. **Method**: Jenis operasi yang dilakukan (seperti GET, POST, dll.).
3. **Headers**: Informasi tambahan, seperti format data atau token autentikasi.
4. **Body**: Data yang dikirim (digunakan pada POST/PUT).

Komponen HTTP Response:

1. **Status Code**: Menunjukkan hasil dari operasi (misalnya, 200 untuk berhasil, 404 untuk sumber daya tidak ditemukan).
2. **Headers**: Informasi tambahan yang dikirim oleh server.
3. **Body**: Data yang dikembalikan oleh server (biasanya dalam format JSON).

C. Praktikum

1. Persiapan Projek Flutter

a. Buat proyek Flutter baru

b. Tambahkan dependensi http ke file pubspec.yaml:

```
http: ^1.2.2
```

Jalankan perintah flutter pub get untuk menginstal dependensi:

2. Membuat Folder Struktur

Buat folder services untuk file API dan screens untuk file UI di dalam folder lib.

3. Implementasi HTTP GET

Kita akan menggunakan API dari https://jsonplaceholder.typicode.com/

a. Membuat Service GET

Buat file api_service.dart di dalam folder services:

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

class ApiService {
  final String baseUrl = "https://jsonplaceholder.typicode.com";
  List<dynamic> posts = []; // Menyimpan data post yang diterima
  // Fungsi untuk GET data
  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');
    }
  }
```

b. Membuat tampilan UI untuk GET

Buat file home_screen.dart di dalam folder screens:

Fungsi untuk memanggil file api service

```
import 'package:flutter/material.dart';
import 'package:prak_api/services/api_service.dart';

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

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  List<dynamic> _posts = []; // Menyimpan list posts
  bool _isLoading = false; // Untuk indikator loading
    final  ApiService  _apiService  =  ApiService();  //  Instance
ApiService

  // Fungsi untuk menampilkan SnackBar
  void _showSnackBar(String message) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text(message)),
    );
  }

  // Fungsi untuk mengambil data posts
  Future<void> _fetchPosts() async {
    await _handleApiOperation(
       _apiService.fetchPosts(), // Ganti dengan fungsi fetchPosts
milik Anda
      'Data berhasil diambil!',
    );
  }

  // Fungsi untuk memanggil API dan menangani operasi
  Future<void> _handleApiOperation(
      Future<void> operation, String successMessage) async {
    setState(() {
      _isLoading = true;
    });

    try {
      await operation; // Menjalankan operasi API
      setState(() {
            _posts  =  _apiService.posts;  //  Ambil  data  posts  dari
```

```
service
      });
      _showSnackBar(successMessage);
    } catch (e) {
      _showSnackBar('Error: $e');
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }
```

Menampilkan response API

```
child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              _isLoading
                  ? const Center(child: CircularProgressIndicator())
                  : _posts.isEmpty
                      ? const Text(
                          "Tekan tombol GET untuk mengambil data",
                          style: TextStyle(fontSize: 12),
                        )
                      : Expanded(
                          child: ListView.builder(
                            itemCount: _posts.length,
                            itemBuilder: (context, index) {
                              return Padding(
                                padding: const EdgeInsets.only(bottom: 12.0),
                                child: Card(
                                  elevation: 4,
                                  child: ListTile(
                                    title: Text(
                                      _posts[index]['title'],
                                      style: const TextStyle(
                                          fontWeight: FontWeight.bold,
                                          fontSize: 12),
                                    ),
                                    subtitle: Text(
                                      _posts[index]['body'],
                                      style: const TextStyle(fontSize: 12),
                                    ),
                                  ),
                                ),
                              );
                            },
                          ),
                        ),
```

Tambahkan tombol untuk GET di home_screen.dart:

```dart
ElevatedButton(
                onPressed: () => _handleApiOperation(
                    _apiService.fetchPosts(), 'Data berhasil diambil!'),
                style: ElevatedButton.styleFrom(backgroundColor: Colors.orange),
                child: const Text('GET'),
              ),
```

4. Implementasi HTTP POST

a. Membuat Service POST

Tambahkan metode berikut ke api_service.dart:

```dart
// Fungsi untuk POST data
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': 'Ini contoh POST.',
      'userId': 1,
    }),
  );
  if (response.statusCode == 201) {
    posts.add({
      'title': 'Flutter Post',
      'body': 'Ini contoh POST.',
      'id': posts.length + 1,
    });
  } else {
    throw Exception('Failed to create post');
  }
 }
}
```

b. Membuat tampilan UI untuk POST

Tambahkan tombol untuk POST di home_screen.dart:

```dart
ElevatedButton(
                onPressed: () => _handleApiOperation(
                    _apiService.createPost(), 'Data berhasil ditambahkan!'),
                style: ElevatedButton.styleFrom(backgroundColor: Colors.green),
                child: const Text('POST'),
              ),
```

5. Implementasi HTTP PUT

a. Membuat Service PUT

Tambahkan metode berikut ke api_service.dart:

```
// Fungsi untuk UPDATE data
  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');
    }
  }
```

b. Membuat tampilan UI untuk PUT

Tambahkan logika untuk memperbarui postingan di home_screen.dart:

```
ElevatedButton(
                onPressed: () => _handleApiOperation(
                    _apiService.updatePost(), 'Data berhasil diperbarui!'),
                style: ElevatedButton.styleFrom(backgroundColor: Colors.blue),
                child: const Text('UPDATE'),
              ),
```

6. Implementasi HTTP DELETE

a. Membuat Service DELETE

Tambahkan metode berikut ke api_service.dart:

```
  // Fungsi untuk DELETE data
  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');
    }
  }
```

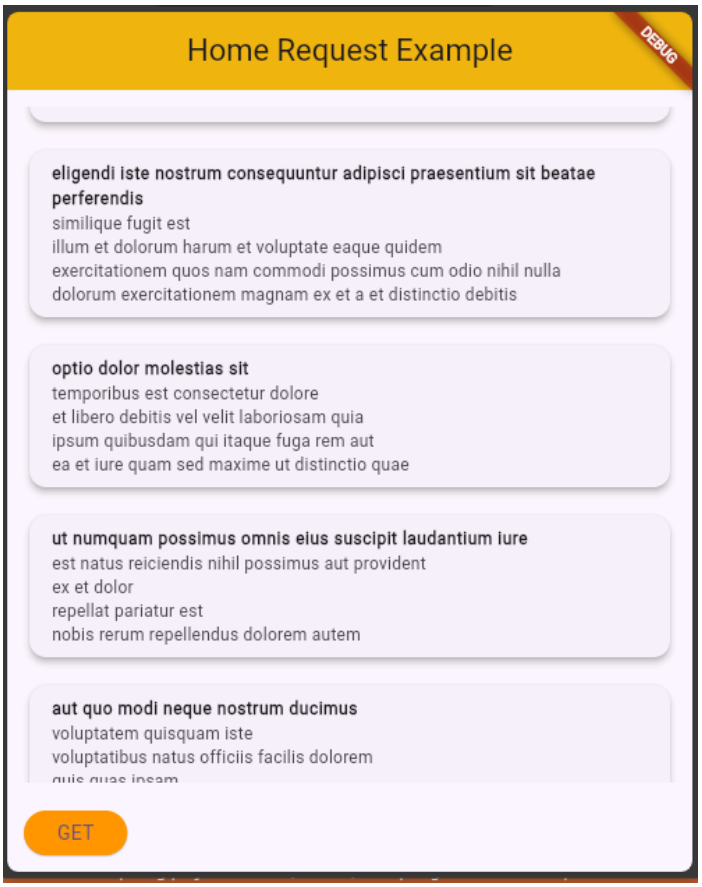
b. Membuat tampilan UI untuk DELETE

Tambahkan tombol untuk DELETE di home_screen.dart:

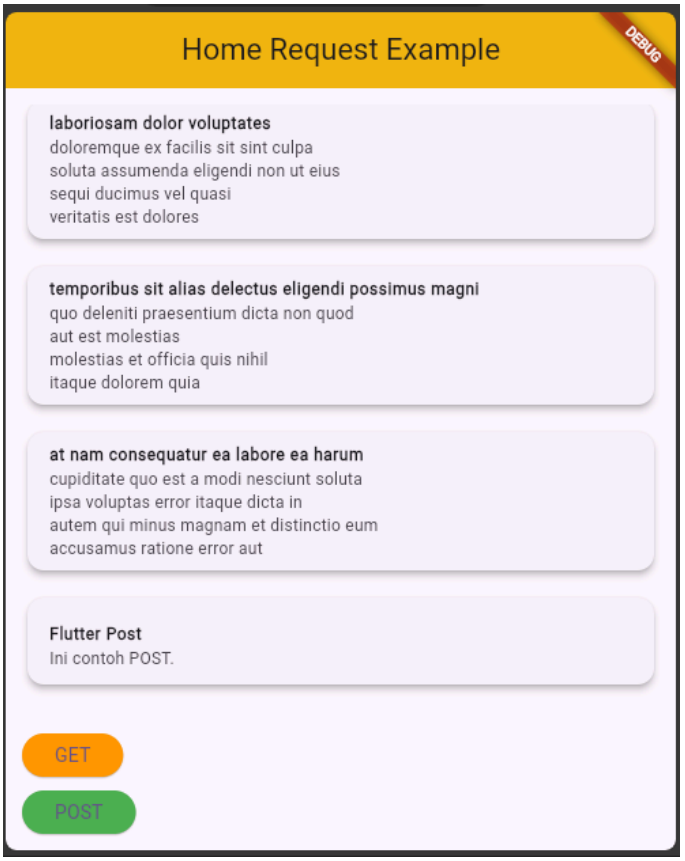
```
ElevatedButton(
                onPressed: () => _handleApiOperation(
                    _apiService.deletePost(), 'Data berhasil dihapus!'),
                style: ElevatedButton.styleFrom(backgroundColor: Colors.red),
                child: const Text('DELETE'),
              ),
```

D. Hasil Praktikum

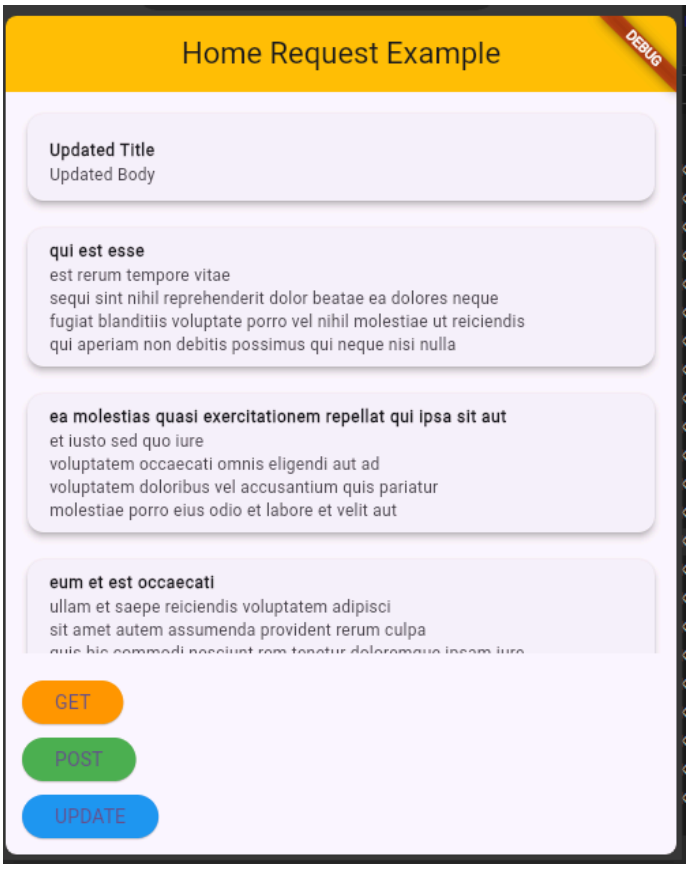
1. Tampilan setelah Get data



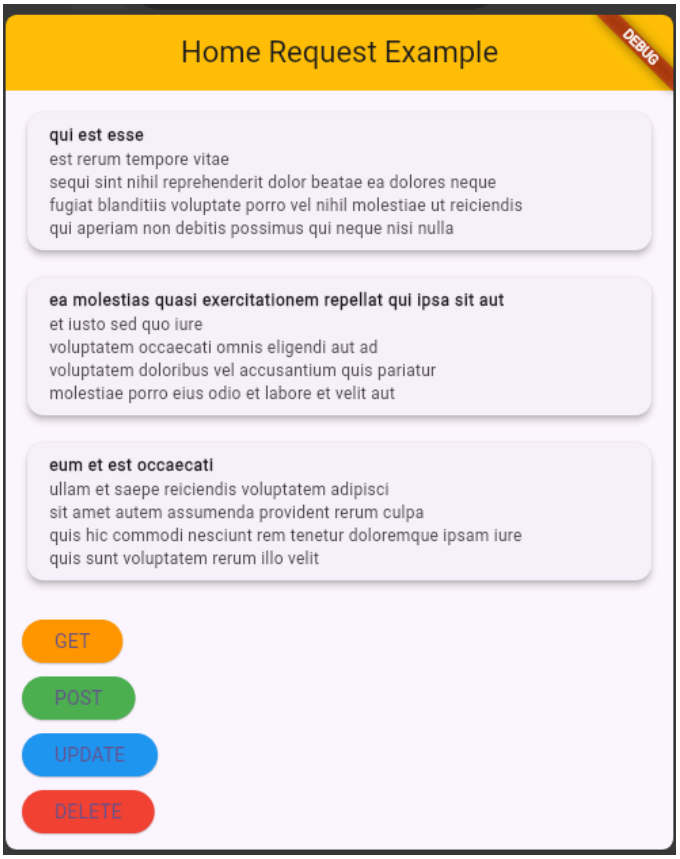
2. Tampilan setelah Post data



3. Tampilan setelah Update data



4. Tampilan setelah Delete data



UNGUIDED

Modifikasi tampilan Guided dari praktikum di atas:

a. Gunakan State Management dengan GetX:
   - Atur data menggunakan state management GetX agar lebih mudah dikelola.
   - Implementasi GetX meliputi pembuatan controller untuk mengelola data dan penggunaan widget Obx untuk menampilkan data secara otomatis setiap kali ada perubahan.

b. Tambahkan Snackbar untuk Memberikan Respon Berhasil:
   - Tampilkan snackbar setelah setiap operasi berhasil, seperti menambah atau memperbarui data.
   - Gunakan Get.snackbar agar pesan sukses muncul di layar dan mudah dipahami oleh pengguna.

Note: Jangan lupa sertakan source code, screenshoot output, dan deskripsi program.

Kreatifitas menjadi nilai tambah

Source Code

File main.dart

```dart
import 'package:flutter/material.dart';
import 'package:get/get_navigation/src/root/get_material_app.dart';
import 'package:prak_api/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: 'Unguided Data Storage',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: HomeScreen(),
```

```
    );
  }
}
```

File home_screen.dart

```dart
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:prak_api/view_model/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('Data Storage API'),
        centerTitle: true,
        backgroundColor: const Color.fromARGB(255, 206, 137, 247),
        foregroundColor: Colors.white,
      ),
      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),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                ElevatedButton(
                  onPressed: controller.fetchPosts,
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.orange,
                    foregroundColor: Colors.white,
                  ),
                  child: const Text('GET'),
                ),
                ElevatedButton(
                  onPressed: controller.createPost,
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.green,
                    foregroundColor: Colors.white,
                  ),
                  child: const Text('POST'),
                ),
                ElevatedButton(
                  onPressed: controller.updatePost,
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.blue,
                    foregroundColor: Colors.white,
                  ),
                  child: const Text('UPDATE'),
                ),
```

```
                ElevatedButton(
                  onPressed: controller.deletePost,
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.red,
                    foregroundColor: Colors.white,
                  ),
                  child: const Text('DELETE'),
                ),
              ],
            ),
          ],
        ),
      ),
    );
  }
}
```

File controller.dart

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'dart:convert';
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,
      duration: const Duration(seconds: 5),
    );
  }

  void showErrorSnackBar(String message) {
    Get.snackbar(
```

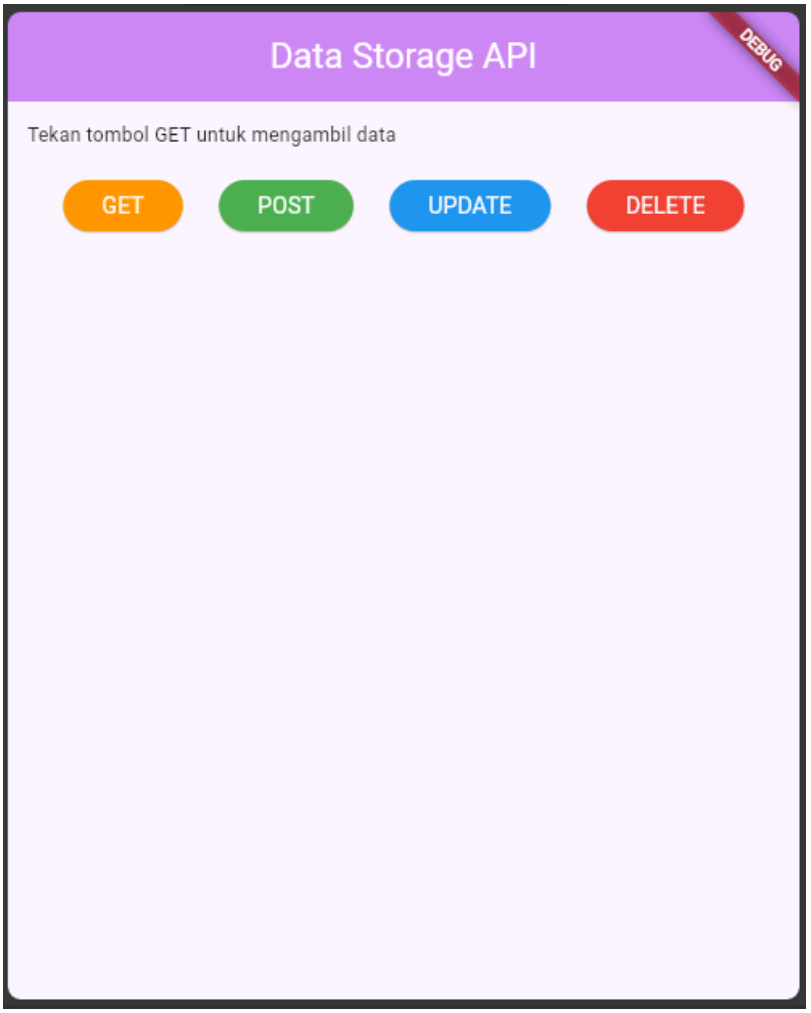
```
      'Error',
      message,
      backgroundColor: Colors.red,
      colorText: Colors.white,
      duration: const Duration(seconds: 5),
    );
  }

  // 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;
```

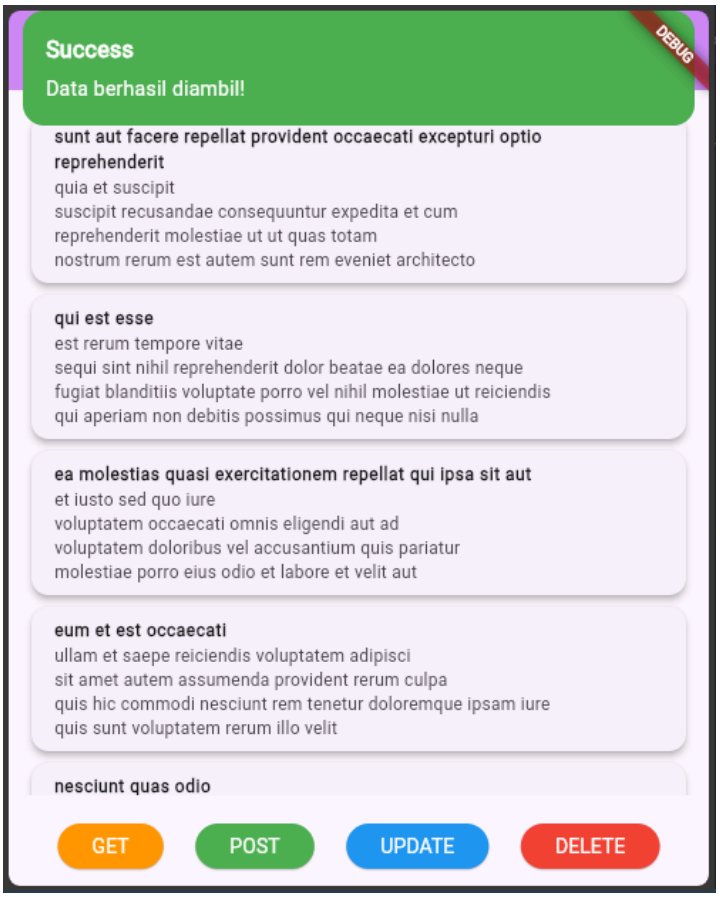
```
    try {
      final response = await http.delete(Uri.parse('$baseUrl/posts/1'));
      if (response.statusCode == 200) {
        posts.removeWhere((post) => post['id'] == 1);
        showSuccessSnackBar('Data berhasil dihapus!');
        const Duration(seconds: 5);
      } else {
        throw Exception('Failed to delete post');
      }
    } catch (e) {
      showErrorSnackBar('Error: $e');
    } finally {
      isLoading.value = false;
    }
  }
}
```

Screenshot Output

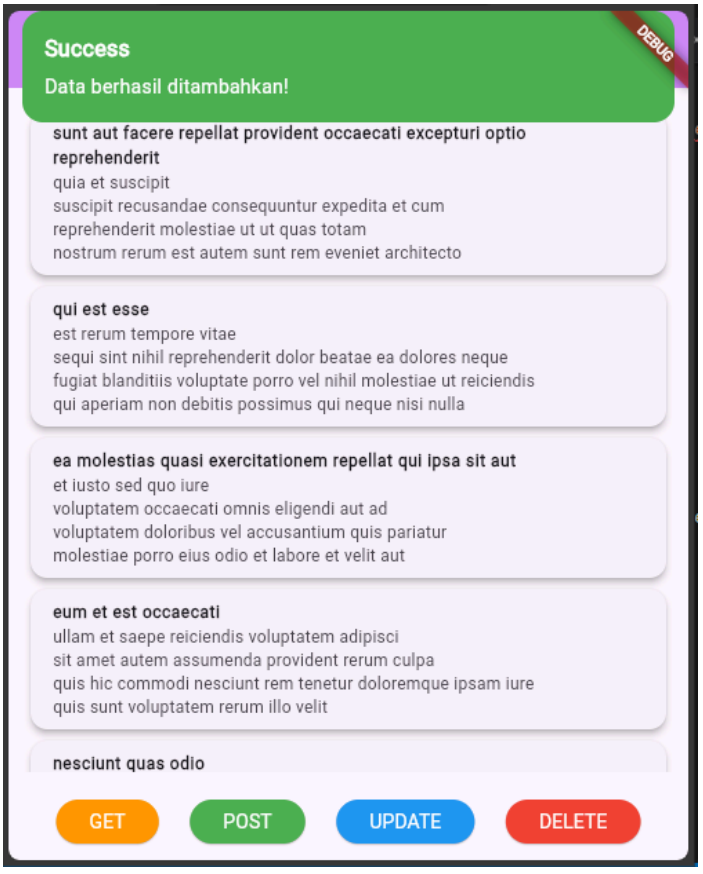
Tampilan awal program



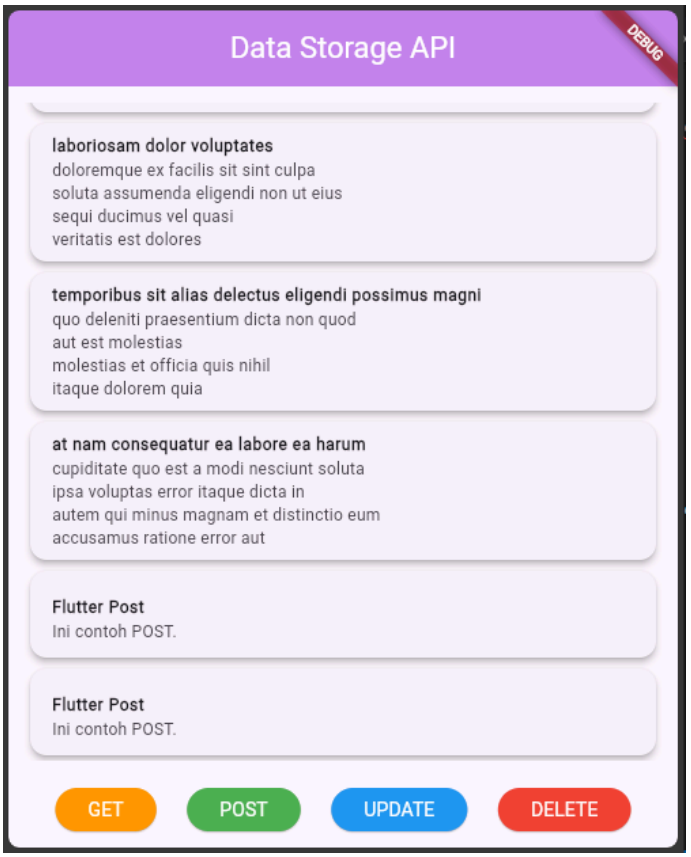
Tampilan setelah Get data



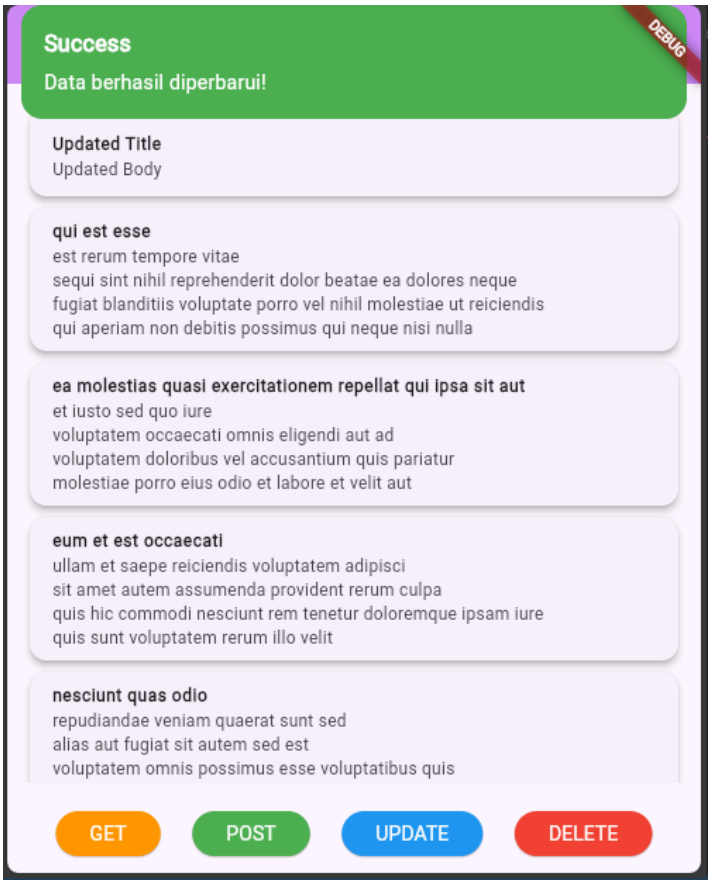
Tampilan setelah Post data



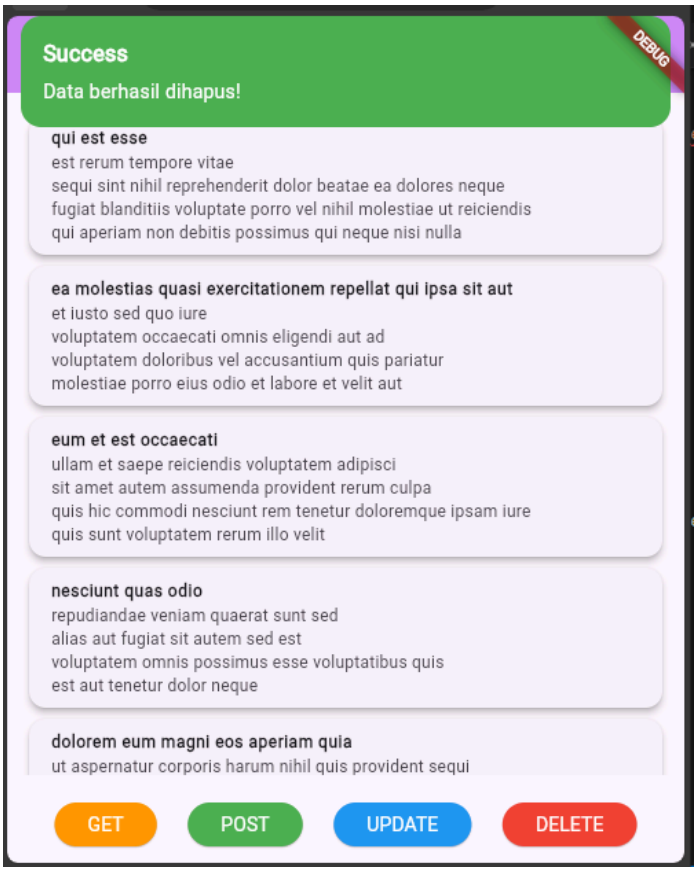
Data yang berhasil ditambahkan melalui “POST” akan berada di bawah sendiri



Tampilan setelah Update data



Tampilan setelah Delete data



Deskripsi Program:

Program diatas mengimplementasikan untuk memanajemen state, snackbar, dan berkomunikasi dengan API eksternal menggunakan HTTP. Program ini mempunyai tiga file yaitu, home_screen.dart untuk antarmuka pengguna, controller.dart untuk logika dan pengelolaan data, dan main.dart untuk konfigurasi utama. Ketika pengguna menekan tombol GET maka akan memanggil fungsi fetchPosts di controller.dart untuk mengambil data dari API, dan isLoading diubah untuk menampilkan indikator pemuatan. Setelah data diterima, daftar data diperbarui secara otomatis di UI melalui widget reaktif Obx.

Ketika pengguna menekan salah satu tombol, yaitu get, post, update, maupun deleta maka program menampilkan notifikasi kepada pengguna menggunakan GetX Snackbar. Fungsi showSuccessSnackBar dan showErrorSnackBar memberikan feedback dengan warna hijau atau merah, sehingga pengguna tahu apakah operasi berhasil atau gagal. Dengan program ini, pengguna dapat melihat data dari API, menambah data baru, memperbarui, dan menghapus data secara langsung dari aplikasi.