PRAKTIKUM PEMROGRAMAN PERANGKAT BERGERAK

MODUL XIV

DATA STORAGE ‘API’

Disusun Oleh:

Aorinka Anendya Chazanah / 2211104013

S1 SE-06-01

Asisten Praktikum:

MuhammadFazaZulian 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. Apa itu REST API

REST API (*Representational State Transfer Application Programming Interface*) adalah sebuah antarmuka yang memungkinkan aplikasi klien berkomunikasi dengan basis data melalui protokol HTTP. Dengan REST API, data dapat dibaca, ditambahkan, diperbarui, atau dihapus tanpa perlu mengakses basis data secara langsung. REST API juga mendukung penggunaan token unik untuk mengidentifikasi setiap perangkat pengguna.

Manfaat REST API

1. Interoperabilitas: Memungkinkan aplikasi web dan mobile untuk mengakses data yang sama.
2. Efisiensi: Menggunakan format data yang ringan seperti JSON atau XML, sehingga data dapat dikirim dan diterima dengan cepat.
3. Keamanan: Mendukung autentikasi dengan token untuk membatasi akses hanya pada pengguna yang berwenang.

B. Apa itu HTTP

HTTP (*Hypertext Transfer Protocol*) adalah protokol utama yang digunakan untuk bertukar data antara klien (seperti browser atau aplikasi) dan server.

Metode Utama dalam HTTP

1. GET: Digunakan untuk mengambil data dari server.
2. POST: Mengirimkan data baru ke server.
3. PUT/PATCH: Memperbarui data yang ada di server.
4. DELETE: Menghapus data dari server.

Komponen HTTP Request

1. URL: Menunjukkan alamat resource tertentu.
2. Method: Operasi yang ingin dilakukan (GET, POST, dll.).
3. Headers: Informasi tambahan seperti format data atau token autentikasi.
4. Body: Data yang dikirimkan ke server (digunakan dalam metode POST/PUT).

Komponen HTTP Response

1. Status Code: Memberikan informasi tentang hasil operasi (misalnya, 200 untuk berhasil, 404 untuk resource tidak ditemukan).
2. Headers: Informasi tambahan yang dikirimkan server.
3. Body: Data yang dikembalikan oleh server, umumnya dalam format JSON.

C. Praktikum

Langkah-langkah implementasi REST API di Flutter

1. Persiapan Projek Flutter

a) Membuat proyek Flutter baru

b) Tambahkan dependency ke pubspec.yaml

```
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
```

2. Membuat Folder Struktur

Kemudian buat folder services untuk file API dan screens untuk file UI pada folder lib.

3. Implementasi HTTP GET

Implementasikan menggunakan API https://jsonplaceholder.typicode.com/

a) Membuat service GET

Buat file api_service.dart dan masukkan kode berikut

```
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 dan masukkan kode berikut

```
import 'package:flutter/material.dart';
import
'package:modul14/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 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;
      });
      _showSnackBar(successMessage);
    } catch (e) {
      _showSnackBar('Error: $e');
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }
```

c) Menampilkan response API

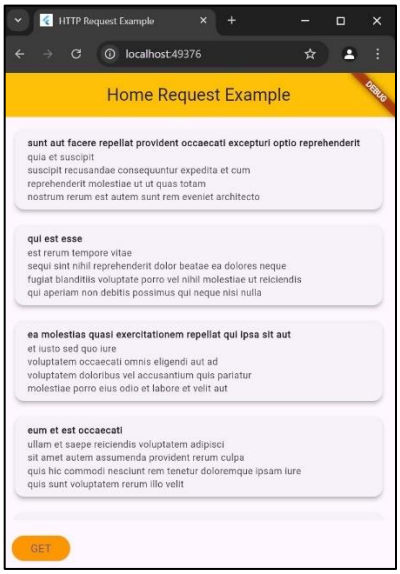
```
Column(
          crossAxisAlignment:
CrossAxisAlignment.start,
          children: [
            _isLoading
                ? const Center(child:
CircularProgressIndicator())
                : _posts.isEmpty
                    ? const Text(
                        "Tekan tombol GET untuk
mengambil 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),
                                    ),
                                  ),
                                ),
                              );
                            },
                          ),
                        ),
            const SizedBox(height: 20),
```

Tambahkan tombol untuk GET di home_screen.dart

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

Output program ketika dijalankan. Klik tombol GET dan data akan muncul.



4. Implementasi HTTP POST

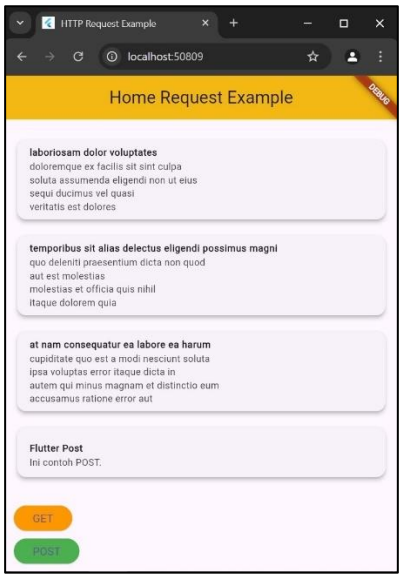
a) Membuat service POST

```
// 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

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

Output program ketika tombol POST di klik.



5. Implementasi HTTP PUT

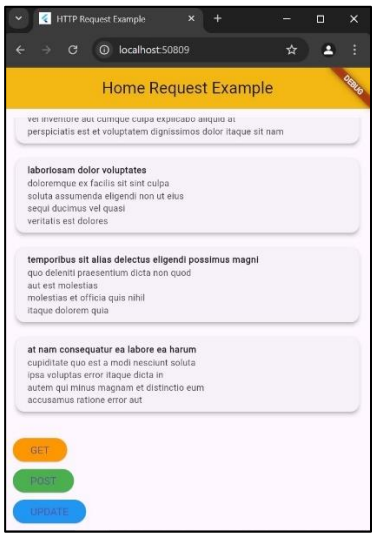
a) Membuat service PUT

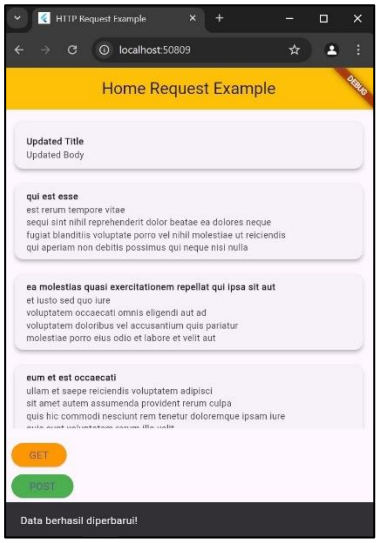
```
// Fungsi untuk UPDATE data
  Future<void> updatePost() async {
    final response = await http.put(
      Uri.parse('$baseUrl/posts/1'),
      body: json.encode({
        'title': 'Updated Title',
        'body': 'Updated Body',
        'userId': 1,
      }),
    );

    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

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

Output program ketika button PUT diklik.





6. Implementasi HTTP DELETE

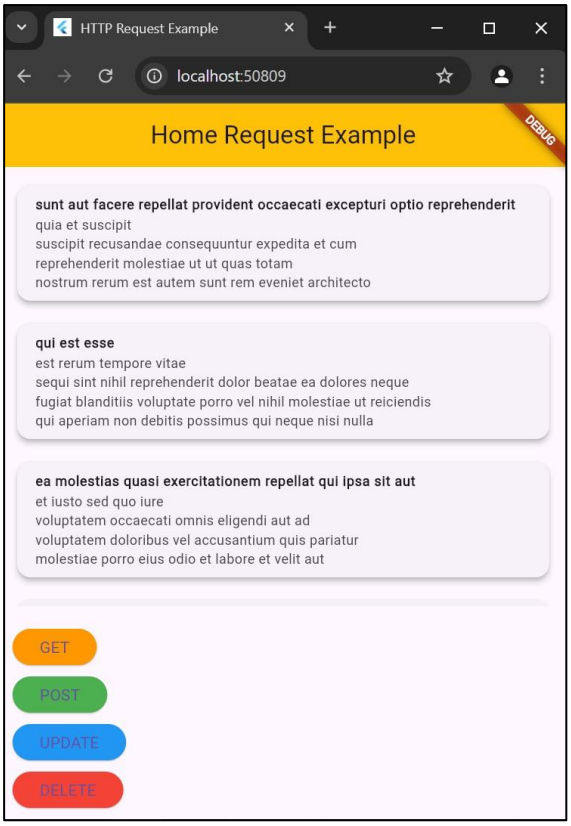
a) Membuat service DELETE

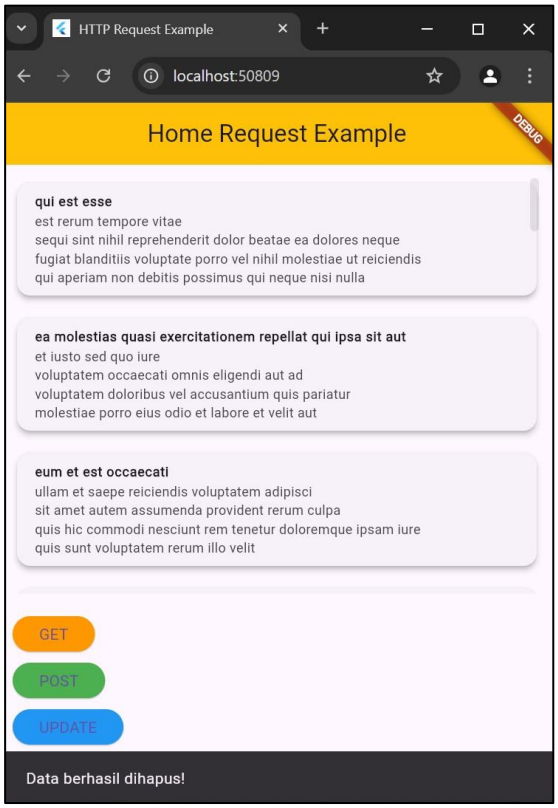
```
// Fungsi untuk DELETE data
  Future<void> deletePost() async {
    final response = await http.delete(
      Uri.parse('$baseUrl/posts/1'),
    );

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

b) Membuat tampilan UI untuk DELETE

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

Output program ketika button DELETE diklik.





UNGUIDED

SOAL

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.

JAWABAN

1. Membuat proyek flutter baru
2. Tambahkan dependency berikut pada pubspec.yaml

```
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
```

3. File post_controllers.dart

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

```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.TOP, // Ubah posisi
snackbar ke atas
      duration: const Duration(seconds: 2),
    );
  }

  void showErrorSnackBar(String message) {
    Get.snackbar(
      'Error',
      message,
      backgroundColor: Colors.red,
      colorText: Colors.white,
      snackPosition: SnackPosition.TOP, // Ubah posisi
snackbar ke atas
      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;
    }
  }
}
```

4. File home_screen.dart

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import
'package:unguided_modul_14/controllers/post_controllers.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 with GetX'),
        centerTitle: true,
        backgroundColor: Colors.amber,
      ),
      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]['ti
tle'],
                                  style: const TextStyle(
                                      fontWeight:
FontWeight.bold,
                                      fontSize: 12),
                                ),
                                subtitle: Text(
                                  controller.posts[index]['bo
dy'],
                                  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'),
            ),
          ],
        ),
      ),
    );
  }
}
```

5. File main.dart

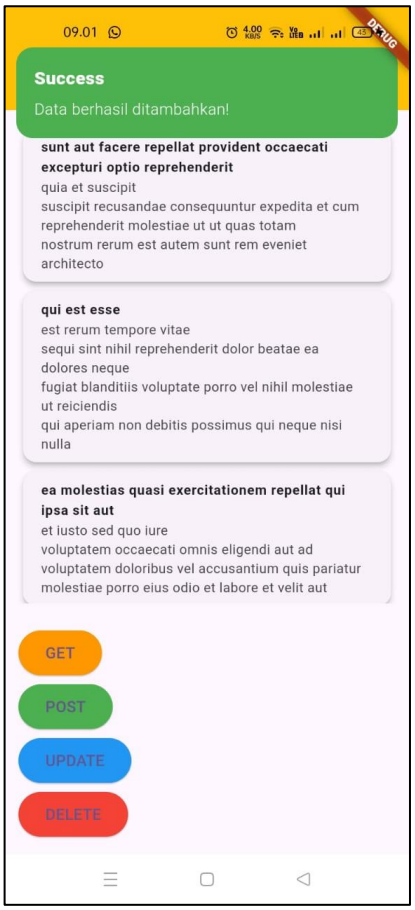
```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import
'package:unguided_modul_14/controllers/post_controllers.dart'
;
```

```dart
import 'package:unguided_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(),
    );
  }
}
```

Output Program

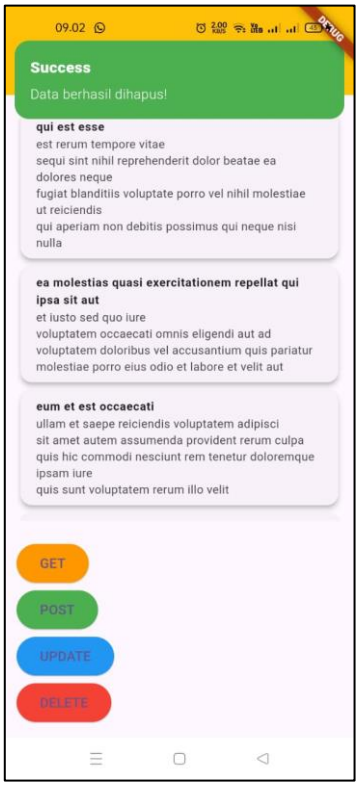
Tampilan ketika tombol GET di klik kemudian muncul data dan Snackbar berwarna hijau “Data berhasil diambil”

Tampilan ketika button POST di klik muncul Snackbar berwarna hijau “Data berhasil ditambahkan”



Tampilan ketika button UPDATE di klik maka muncul snackbar hijau “Data berhasil diperbarui”

Tampilan ketika button DELETE di klik maka akan muncul snackbar “Data berhasil dihapus”



Deskripsi Program

File **main.dart** di dalamnya, aplikasi diatur menggunakan GetMaterialApp, yang memungkinkan integrasi dengan GetX untuk pengelolaan state. File ini mendefinisikan struktur dasar aplikasi, termasuk tema dan judulnya, serta memanfaatkan arsitektur GetX untuk mengelola navigasi dan state management secara efisien.

Pada file **post_controllers.dart**, terdapat logika untuk menangani berbagai operasi HTTP, seperti mengambil data (GET), menambahkan data baru (POST), memperbarui data yang ada (PUT), dan menghapus data (DELETE). Operasi ini dilakukan melalui API eksternal menggunakan library http. Semua data yang diperoleh atau diperbarui disimpan dalam variabel posts yang dikelola secara reaktif menggunakan Rx dari GetX. File ini juga menyertakan fungsi notifikasi menggunakan Snackbar untuk memberikan umpan balik kepada pengguna tentang keberhasilan atau kegagalan setiap operasi. Dengan demikian, file ini menjadi pusat pengelolaan logika aplikasi yang terintegrasi dengan antarmuka pengguna.

File **home_screen.dart** berfungsi sebagai antarmuka utama aplikasi, tempat pengguna dapat melihat dan berinteraksi dengan data yang diambil dari REST API. Data ditampilkan dalam bentuk daftar dinamis yang dirender menggunakan widget ListView.builder. Fungsi Obx digunakan untuk memastikan bahwa tampilan selalu diperbarui secara real-time berdasarkan perubahan state di post_controller.dart. Tombol-tombol di bagian bawah layar memungkinkan pengguna untuk melakukan operasi GET, POST, UPDATE, dan DELETE, masing-masing memanggil fungsi terkait di controller. Selain itu, indikator loading ditampilkan saat aplikasi sedang memproses permintaan, sehingga memberikan pengalaman pengguna yang lebih informatif dan responsif.