LAPORAN PRAKTIKUM
PEMROGRAMAN PERANGKAT BERGERAK

MODUL XIV

DATA STORAGE (API)

Disusun Oleh :

Dewi Atika Muthi / 2211104042

SE-06-02

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

BAB I
PENDAHULUAN

A. DASAR TEORI

REST API (Representational State Transfer Application Programming Interface) adalah arsitektur perangkat lunak yang menerapkan konsep transfer state melalui protokol HTTP. Menurut Fielding (2000) dalam disertasinya "Architectural Styles and the Design of Network-based Software Architectures", REST dirancang untuk memanfaatkan protokol HTTP yang sudah ada untuk komunikasi data.

Komponen utama REST API meliputi:

1. Resources: Setiap informasi diidentifikasi oleh URI (Uniform Resource Identifier)
2. HTTP Methods: Menggunakan metode standar HTTP (GET, POST, PUT, DELETE)
3. Representations: Data dapat direpresentasikan dalam berbagai format (JSON, XML, dll)

Prinsip REST API:

Menurut Massé (2012) dalam bukunya "REST API Design Rulebook", REST API memiliki enam prinsip utama:

- Client-Server Architecture
- Statelessness
- Cacheability
- Layered System
- Code on Demand (optional)
- Uniform Interface

HTTP Protocol

HTTP adalah protokol aplikasi untuk sistem informasi terdistribusi yang kolaboratif. Berdasarkan penelitian Gourley & Totty (2002) dalam "HTTP: The Definitive Guide", HTTP memiliki beberapa karakteristik penting:

Metode HTTP

1. GET: Mengambil resource dari server
   - Idempotent (multiple identical requests should have same effect as single request)
   - Tidak mengubah state server
2. POST: Mengirim data ke server
   - Non-idempotent
   - Dapat membuat resource baru
3. PUT: Memperbarui resource yang ada
   - Idempotent
   - Mengganti seluruh resource target
4. DELETE: Menghapus resource
   - Idempotent
   - Menghapus resource yang ditentukan

HTTP Status Codes

Berdasarkan RFC 7231 (Fielding & Reschke, 2014), status code HTTP terbagi menjadi 5 kategori:

- 1xx: Informational
- 2xx: Success
- 3xx: Redirection
- 4xx: Client Error
- 5xx: Server Error

JSON Format

JSON (JavaScript Object Notation) adalah format pertukaran data ringan yang mudah dibaca dan ditulis oleh manusia serta mudah di-parse dan di-generate oleh mesin (Crockford, 2008). Format ini sering digunakan dalam REST API karena:

- Ringan dan cepat
- Mudah dibaca
- Language independent
- Support untuk nested structures

Contoh format JSON:

```
jsonCopy{
  "id": 1,
  "title": "Sample Post",
  "body": "This is a sample post body",
  "userId": 1
}
```

Flutter dan REST API

Flutter menyediakan package http untuk melakukan HTTP requests. Package ini mendukung semua metode HTTP standar dan dapat menangani berbagai format data termasuk JSON. Untuk menggunakan REST API dalam Flutter, diperlukan beberapa komponen:

- HTTP Client: Untuk melakukan requests
- JSON Serialization: Untuk mengonversi JSON ke objek Dart dan sebaliknya
- Error Handling: Untuk menangani berbagai response status

State Management dalam REST API

Pengelolaan state sangat penting dalam aplikasi yang menggunakan REST API. Beberapa aspek yang perlu diperhatikan:

- Loading State: Menunjukkan proses request sedang berlangsung
- Error State: Menangani dan menampilkan error
- Data State: Menyimpan dan mengelola data dari API

B. MAKSUD DAN TUJUAN

Tujuan praktikum kali ini adalah:

- Memahami konsep dasar database web service menggunakan REST API
- Mengetahui kegunaan dan penerapan metode HTTP (GET, POST, PUT, DELETE) dalam pengelolaan data
- Membuat antarmuka pengguna (UI) yang menampilkan data dari API serta memungkinkan interaksi dengan server

BAB II
IMPLEMENTASI

A. PRAKTIKUM (Guided)

Langkah-Langkah Praktikum:

1. Persiapan Proyek
   - Membuat proyek Flutter baru
   - Menambahkan dependensi http di pubspec.yaml:

```
dependencies:
  flutter:
    sdk: flutter
  http: ^0.15.0
```

2. Implementasi Struktur Proyek
   Membuat dua folder utama dalam lib:
   - services: Untuk file terkait API
   - screens: Untuk file UI

3. Implementasi API Service
   File: **service/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 = []; // 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');
    }
  }

  // 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');
    }
  }

  // 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');
    }
  }

  // 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');
    }
  }
}
```

4. Implementasi UI
   File: **screen/home_screen.dart:**

```
import 'package:flutter/material.dart';
import 'package:md14/service/api_service.dart';

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

  @override
  State<HomepageScreen> createState() => _HomepageScreenState();
}

class _HomepageScreenState extends State<HomepageScreen> {
  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;
      });
    }
  }

  @override
  void initState() {
    super.initState();
    _handleApiOperation(_apiService.fetchPosts(), 'Posts loaded successfully');
```

```
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('REST API'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(12),
        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),
                                  ),
                                ),
                              ),
                            );
                          },
                        ),
                      ),
            Padding(
              padding: const EdgeInsets.all(8.0),
              child: Row(
                children: [
                  ElevatedButton(
                    onPressed: () => _handleApiOperation(
                        _apiService.createPost(), 'Data berhasil ditambahkan!'),
                    style:
                        ElevatedButton.styleFrom(backgroundColor: Colors.green),
                    child: const Text('POST'),
                  ),
                  const SizedBox(height: 10),
                  ElevatedButton(
                    onPressed: () => _handleApiOperation(
                        _apiService.fetchPosts(), 'Data berhasil diambil!'),
                    style: ElevatedButton.styleFrom(
                        backgroundColor: Colors.orange),
                    child: const Text('GET'),
                  ),
                  const SizedBox(height: 10),
                  ElevatedButton(
                    onPressed: () => _handleApiOperation(
                        _apiService.updatePost(), 'Data berhasil diperbarui!'),
                    style:
                        ElevatedButton.styleFrom(backgroundColor: Colors.blue),
                    child: const Text('UPDATE'),
                  ),
                  const SizedBox(height: 10),
                  ElevatedButton(
                    onPressed: () => _handleApiOperation(
                        _apiService.deletePost(), 'Data berhasil dihapus!'),
```

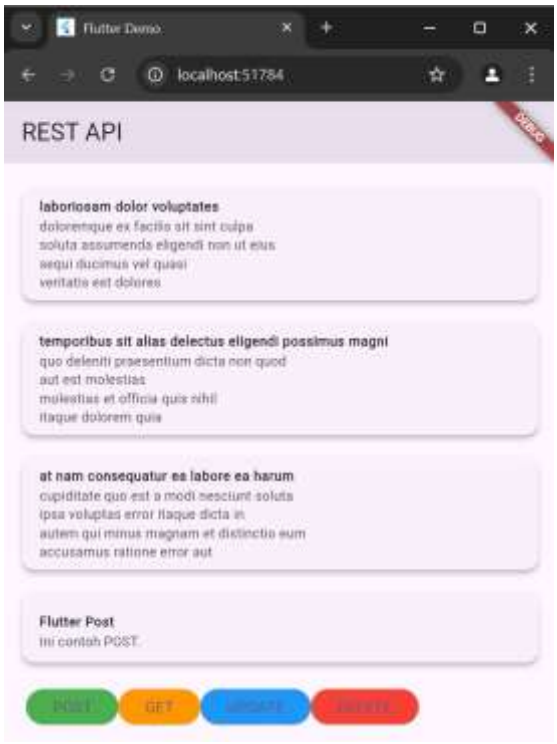
```
                    style:
                        ElevatedButton.styleFrom(backgroundColor: Colors.red),
                    child: const Text('DELETE'),
                  ),
                ],
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

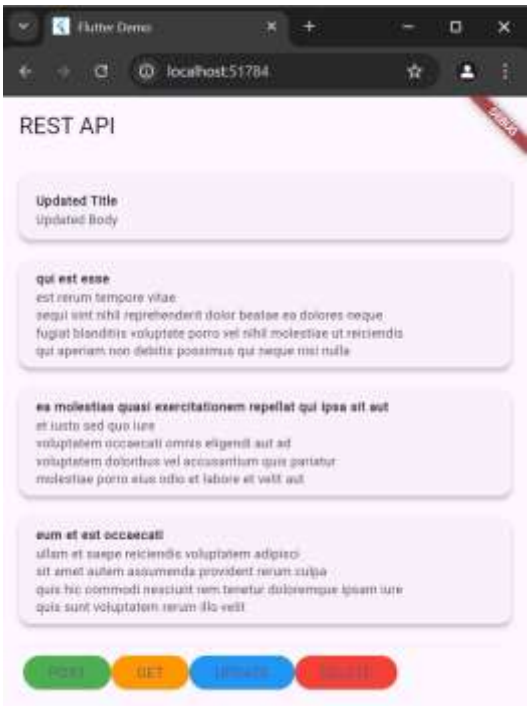
5. Main.dart

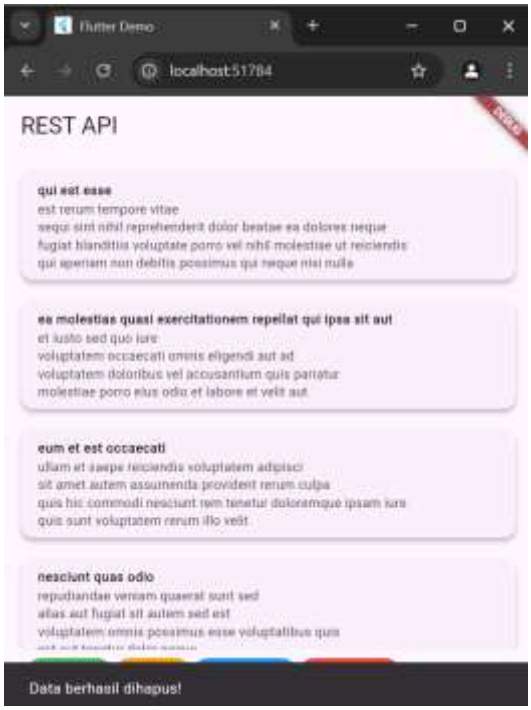
```
import 'package:flutter/material.dart';
import 'package:md14/screen/home_screen.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),
      home: const HomepageScreen(),
    );
  }
}
```

Screenshoot Output:

Post > Get > Update > Delete







Penjelasan program:
Dalam ApiService Class kita menggunakan http package untuk melakukan request ke API, lalu menyimpan data posts dalam List, dan mengimplementasikan metode CRUD (Create, Read, Update, Delete)

Dalam HomepageScreen Widget, program menampilkan daftar posts dalam ListView, menggunakan Card untuk menampilkan setiap post, juga menyediakan tombol untuk operasi CRUD. Lalu menampilkan loading indicator saat operasi berlangsung, dan menggunakan SnackBar untuk notifikasi

B. UNGUIDED (Tugas Mandiri)

Soal Studi Case:

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.

1. Tambahkan GetX dependency ke pubspec.yaml:

```
flutter pub add get
```

2. Membuat post model (lib/models/post_model.dart)

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

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

  factory Post.fromJson(Map<String, dynamic> json) {
    return Post(
      id: json['id'] ?? 0,
      title: json['title'] ?? '',
      body: json['body'] ?? '',
      userId: json['userId'] ?? 0,
    );
  }

  Map<String, dynamic> toJson() {
    return {
      'id': id,
      'title': title,
      'body': body,
      'userId': userId,
    };
  }
}
```

3. Membuat GetX Controller (lib/controllers/post_controller.dart)

```
import 'package:get/get.dart';
import '../models/post_model.dart';
import '../services/api_service.dart';

class PostController extends GetxController {
  final ApiService _apiService = ApiService();
  final RxList<Post> posts = <Post>[].obs;
  final RxBool isLoading = false.obs;

  @override
```

```
  void onInit() {
    super.onInit();
    fetchPosts();
  }

  Future<void> fetchPosts() async {
    try {
      isLoading.value = true;
      final List<Post> fetchedPosts = await _apiService.fetchPosts();
      posts.value = fetchedPosts;
      Get.snackbar(
        'Success',
        'Posts loaded successfully',
        snackPosition: SnackPosition.BOTTOM,
      );
    } catch (e) {
      Get.snackbar(
        'Error',
        'Failed to load posts: $e',
        snackPosition: SnackPosition.BOTTOM,
      );
    } finally {
      isLoading.value = false;
    }
  }

  Future<void> createPost() async {
    try {
      isLoading.value = true;
      final Post newPost = await _apiService.createPost();
      posts.add(newPost);
      Get.snackbar(
        'Success',
        'Post created successfully',
        snackPosition: SnackPosition.BOTTOM,
      );
    } catch (e) {
      Get.snackbar(
        'Error',
        'Failed to create post: $e',
        snackPosition: SnackPosition.BOTTOM,
      );
    } finally {
      isLoading.value = false;
    }
  }

  Future<void> updatePost(int id) async {
    try {
      isLoading.value = true;
      final Post updatedPost = await _apiService.updatePost(id);
      final index = posts.indexWhere((post) => post.id == id);
      if (index != -1) {
        posts[index] = updatedPost;
      }
      Get.snackbar(
        'Success',
        'Post updated successfully',
        snackPosition: SnackPosition.BOTTOM,
      );
    } catch (e) {
      Get.snackbar(
        'Error',
        'Failed to update post: $e',
        snackPosition: SnackPosition.BOTTOM,
      );
    } finally {
      isLoading.value = false;
```

```
    }
  }

  Future<void> deletePost(int id) async {
    try {
      isLoading.value = true;
      await _apiService.deletePost(id);
      posts.removeWhere((post) => post.id == id);
      Get.snackbar(
        'Success',
        'Post deleted successfully',
        snackPosition: SnackPosition.BOTTOM,
      );
    } catch (e) {
      Get.snackbar(
        'Error',
        'Failed to delete post: $e',
        snackPosition: SnackPosition.BOTTOM,
      );
    } finally {
      isLoading.value = false;
    }
  }
}
```

4. Membuat Home Screen (lib/screens/home_screen.dart)

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

class HomeScreen extends StatelessWidget {
  final PostController controller = Get.put(PostController());

  HomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('GetX REST API Demo'),
        backgroundColor: Colors.blue,
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              children: [
                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(1),
                  style: ElevatedButton.styleFrom(backgroundColor:
Colors.blue),
```

```
                  child: const Text('UPDATE'),
                ),
                ElevatedButton(
                  onPressed: () => controller.deletePost(1),
                  style: ElevatedButton.styleFrom(backgroundColor:
Colors.red),
                  child: const Text('DELETE'),
                ),
              ],
            ),
            const SizedBox(height: 16),
            Expanded(
              child: Obx(
                () => controller.isLoading.value
                    ? const Center(child: CircularProgressIndicator())
                    : controller.posts.isEmpty
                        ? const Center(
                            child: Text('No posts available'),
                          )
                        : ListView.builder(
                            itemCount: controller.posts.length,
                            itemBuilder: (context, index) {
                              final post = controller.posts[index];
                              return Card(
                                margin: const EdgeInsets.only(bottom: 16),
                                child: ListTile(
                                  title: Text(
                                    post.title,
                                    style: const TextStyle(
                                      fontWeight: FontWeight.bold,
                                    ),
                                  ),
                                  subtitle: Text(post.body),
                                ),
                              );
                            },
                          ),
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

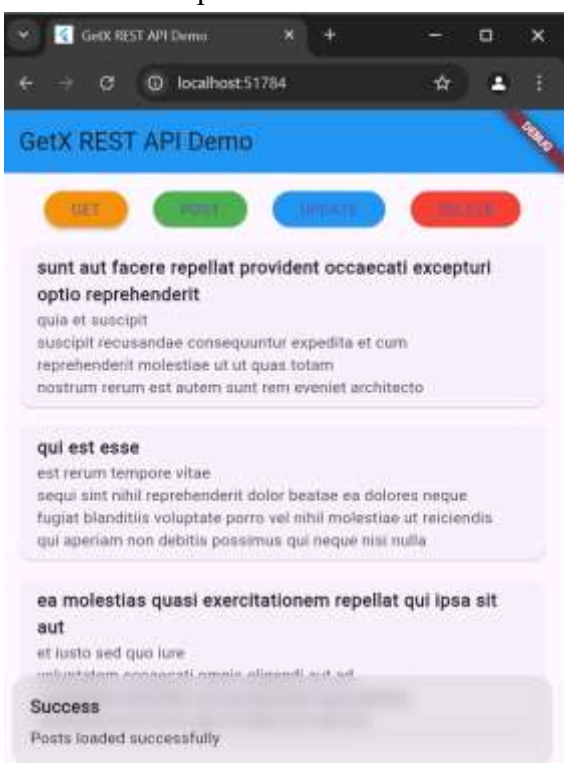
5. Main.dart

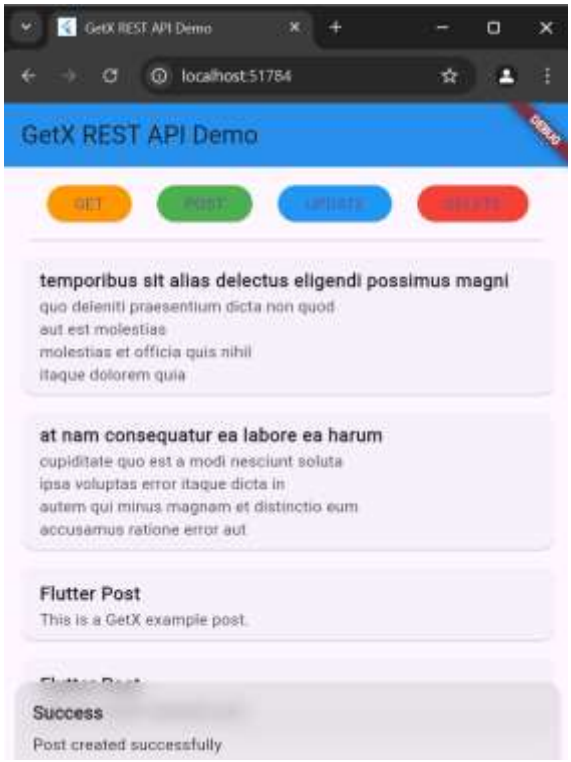
```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'screens/home_screen.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return GetMaterialApp(
      title: 'GetX REST API Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        useMaterial3: true,
```

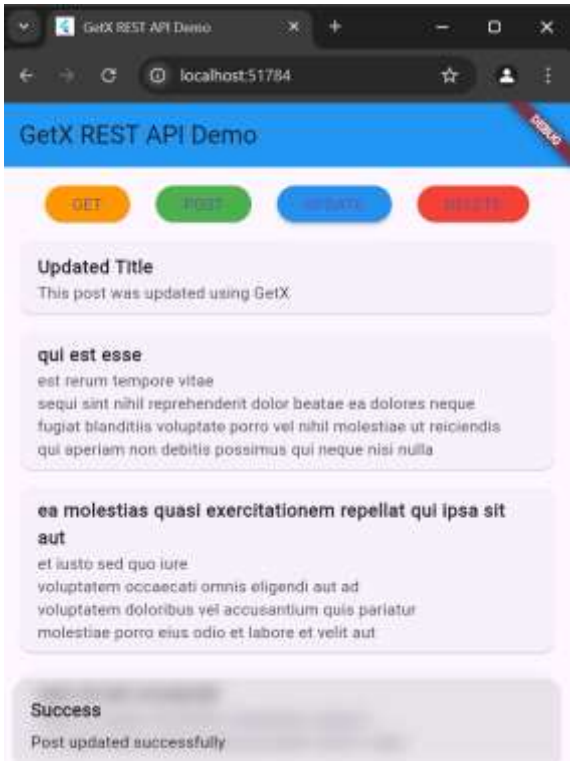
```
            ),
            home: HomeScreen(),
          );
        }
}
```

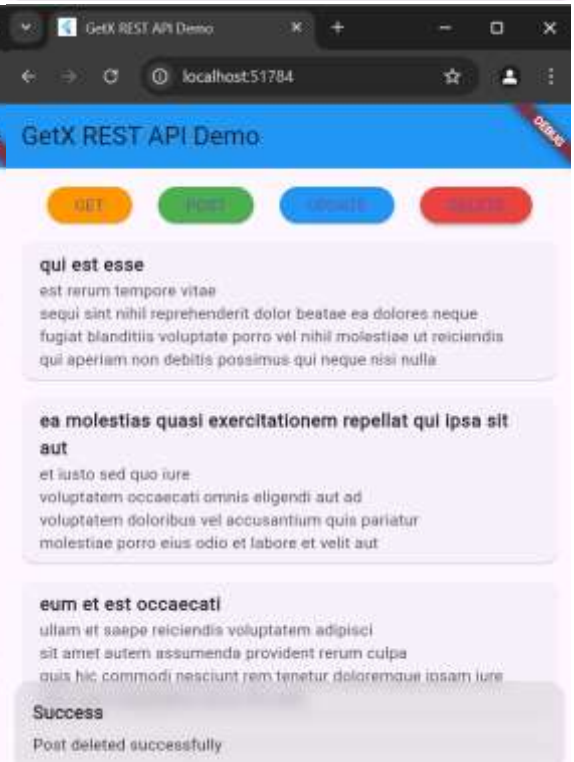
Screenshoot Output:

Get > Post > Update > Delete









Deskripsi Program

Program ini menggunakan GetX untuk membangun aplikasi Flutter dengan struktur folder controllers, models, services, dan screens. Data dikelola secara reaktif menggunakan `RxList` di controller, sementara API service menangani operasi CRUD dengan error handling. UI memanfaatkan `Obx` untuk pembaruan otomatis, lengkap dengan ListView untuk menampilkan data, tombol CRUD, indikator loading, dan snackbar sebagai feedback. Semua komponen terhubung melalui `GetMaterialApp`, menjadikan aplikasi modular, responsif, dan mudah dikembangkan.

BAB III
KESIMPULAN DAN SARAN

A. KESIMPULAN

Pada modul ini, saya mempelajari implementasi Antarmuka Pengguna Lanjutan menggunakan Flutter, termasuk penggunaan CustomScrollView, SliverAppBar, dan SliverList untuk membuat tampilan aplikasi yang dinamis dan interaktif. Saya memahami bagaimana cara mengintegrasikan widget-widget tersebut dalam sebuah aplikasi untuk meningkatkan pengalaman pengguna, terutama dalam skenario seperti aplikasi rekomendasi wisata. Selain itu, modul ini membantu saya mengenali pentingnya struktur yang efisien dalam membangun antarmuka pengguna yang responsif terhadap perubahan data dan ukuran layar. Melalui praktik langsung, saya mampu mengembangkan kemampuan untuk mendesain aplikasi yang tidak hanya menarik secara visual tetapi juga fungsional.

B. REFERENSI

Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine.

Massé, M. (2012). REST API Design Rulebook. O'Reilly Media.

Gourley, D., & Totty, B. (2002). HTTP: The Definitive Guide. O'Reilly Media.

Crockford, D. (2008). JavaScript: The Good Parts. O'Reilly Media.

Flutter Documentation - https://docs.flutter.dev/

HTTP Package - https://pub.dev/packages/http

JSONPlaceholder API - https://jsonplaceholder.typicode.com/

GetX Documentation - https://pub.dev/packages/get