# UNGUIDED MODUL 14


Disusun Oleh :

**Nama : Ganes Gemi Putra**

**Kelas : SE-07-02**

**NIM : (2311104075)**

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

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**

**FAKULTAS INFORMATIKA**

**TELKOM UNIVERSITY**

**PURWOKERTO**


**2025**

# BAB I

PENDAHULUAN

## 1.1 Latar Belakang

Perkembangan aplikasi berbasis mobile menuntut adanya sistem penyimpanan data yang efisien, fleksibel, dan dapat diakses oleh berbagai platform. Salah satu solusi yang umum digunakan adalah REST API (Representational State Transfer Application Programming Interface) yang memungkinkan aplikasi klien berkomunikasi dengan server menggunakan protokol HTTP.

Dalam praktikum ini, REST API diimplementasikan pada aplikasi Flutter untuk melakukan pengelolaan data menggunakan metode HTTP GET, POST, PUT, dan DELETE. Selain itu, digunakan state management GetX untuk mempermudah pengelolaan data dan meningkatkan reaktivitas antarmuka pengguna.

## 1.2 Tujuan Praktikum

Tujuan dari praktikum ini adalah:

1. Memahami konsep dasar database web service menggunakan REST API.
2. Mengetahui kegunaan dan penerapan metode HTTP GET, POST, PUT, dan DELETE dalam pengelolaan data.
3. Mengimplementasikan REST API pada aplikasi Flutter.
4. Menggunakan state management GetX untuk mengelola data secara reaktif.
5. Menampilkan notifikasi keberhasilan operasi menggunakan Get.snackbar.

# BAB II

DASAR TEORI

## 2.1 REST API

REST API adalah antarmuka pemrograman aplikasi yang memungkinkan komunikasi antara klien dan server melalui protokol HTTP. REST API digunakan untuk melakukan operasi CRUD (Create, Read, Update, Delete) terhadap data tanpa harus mengakses database secara langsung.

**Kegunaan REST API**

1. **Interoperabilitas**: Dapat digunakan oleh berbagai platform seperti web dan mobile.
2. **Efisiensi**: Menggunakan format data ringan seperti JSON.
3. **Keamanan**: Akses dapat dibatasi menggunakan autentikasi atau token.

## 2.2 HTTP (Hypertext Transfer Protocol)

HTTP merupakan protokol komunikasi utama antara klien dan server.

**Metode HTTP**

- **GET**: Mengambil data dari server
- **POST**: Mengirim data baru ke server
- **PUT**: Memperbarui data yang sudah ada
- **DELETE**: Menghapus data

## 2.3 GetX

GetX adalah library Flutter yang digunakan untuk state management, dependency injection, dan route management. Pada praktikum ini, GetX digunakan untuk mengelola state data menggunakan RxList dan RxBool serta menampilkan notifikasi menggunakan Get.snackbar.

# BAB III

IMPLEMENTASI SISTEM

## 3.1 Alat dan Bahan

- Flutter SDK
- Dart
- Package http
- Package get

- REST API: https://jsonplaceholder.typicode.com/posts

**3.2 Struktur Folder Aplikasi**

Struktur folder aplikasi disusun sebagai berikut:

- services : menangani komunikasi REST API
- controllers : mengelola state menggunakan GetX
- screens : menampilkan antarmuka pengguna

Struktur ini bertujuan untuk memisahkan tanggung jawab agar kode lebih rapi dan mudah dipelihara.

**3.4 Implementasi CRUD REST API**

**a. GET (Mengambil Data)**

Metode GET digunakan untuk mengambil data dari REST API dan menampilkannya dalam bentuk daftar pada aplikasi.

**b. POST (Menambah Data)**

Metode POST digunakan untuk menambahkan data baru ke server. Data dikirim dalam format JSON.

**c. PUT (Memperbarui Data)**

Metode PUT digunakan untuk memperbarui data yang sudah ada pada server berdasarkan ID tertentu.

**d. DELETE (Menghapus Data)**

Metode DELETE digunakan untuk menghapus data berdasarkan ID yang ditentukan.

Setiap operasi CRUD akan menampilkan notifikasi keberhasilan menggunakan Get.snackbar.

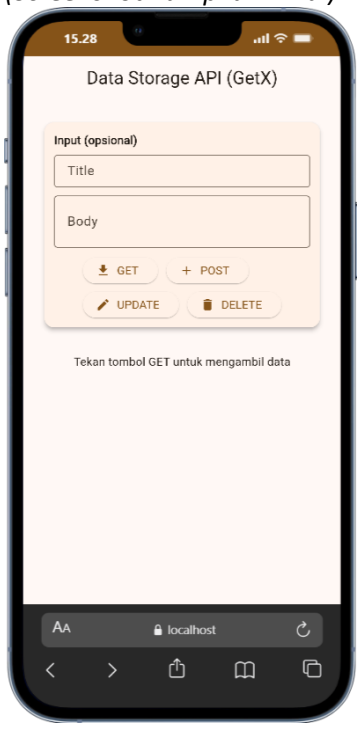
# BAB IV

HASIL DAN PEMBAHASAN

**4.1 Tampilan Awal Aplikasi**

Pada saat aplikasi pertama kali dijalankan, data belum tersedia sehingga aplikasi menampilkan pesan “Tekan tombol GET untuk mengambil data dari API”.

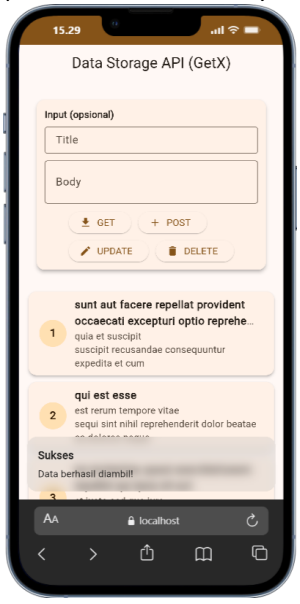
*(Screenshot Tampilan Awal)*



### 4.2 Hasil GET Data

Setelah tombol GET ditekan, aplikasi berhasil mengambil data dari REST API dan menampilkannya dalam bentuk daftar.

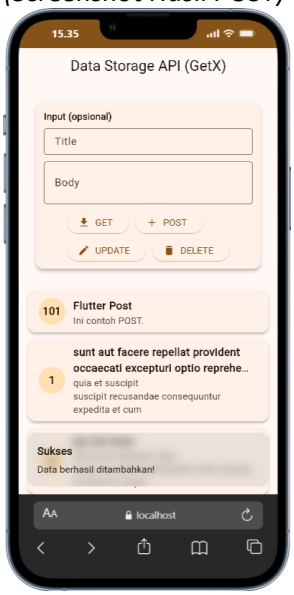
*(Screenshot Hasil GET)*




### 4.3 Hasil POST Data

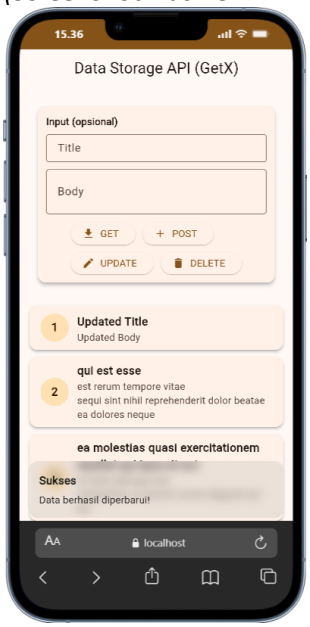
Setelah tombol POST ditekan, data baru berhasil ditambahkan dan muncul pada daftar data.

*(Screenshot Hasil POST)*



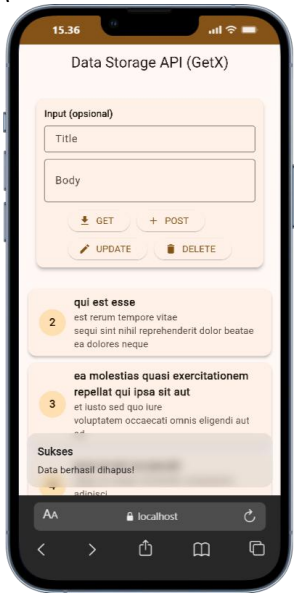
### 4.4 Hasil UPDATE Data

Setelah tombol UPDATE ditekan, data berhasil diperbarui dan perubahan terlihat pada tampilan aplikasi.
*(Screenshot Hasil UPDATE)*




### 4.5 Hasil DELETE Data

Setelah tombol DELETE ditekan, data berhasil dihapus dari daftar.
*(Screenshot Hasil DELETE)*




# BAB V

KESIMPULAN

Berdasarkan praktikum yang telah dilakukan, dapat disimpulkan bahwa REST API dapat digunakan untuk mengelola data menggunakan metode HTTP GET, POST, PUT, dan DELETE. Implementasi REST API pada Flutter dapat berjalan dengan baik menggunakan package http. Penggunaan GetX mempermudah pengelolaan state aplikasi serta

membuat tampilan UI menjadi reaktif melalui widget Obx. Selain itu, Get.snackbar memberikan umpan balik yang jelas kepada pengguna setelah setiap operasi berhasil dilakukan. Seluruh fitur aplikasi berjalan sesuai dengan tujuan praktikum.

# BAB VI

LAMPIRAN

**Lampiran A – Source Code**

Source code aplikasi Flutter disertakan dalam laporan ini dan terdiri dari:

- **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(
      debugShowCheckedModeBanner: false,
      title: 'Data Storage API',
      theme: ThemeData(
        useMaterial3: true,
        colorSchemeSeed: Colors.orange,
      ),
      home: HomeScreen(),
    );
  }
}
```

- **post_controller.dart :**

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

class PostController extends GetxController {
  final ApiService _api = ApiService();

  final posts = <PostModel>[].obs;
  final isLoading = false.obs;

  // untuk input form (opsional, tapi bikin UI terasa "premium")
  final title = ''.obs;
  final body = ''.obs;

  void setTitle(String v) => title.value = v;
  void setBody(String v) => body.value = v;

  Future<void> getPosts() async {
    await _run(() async {
```

```
    final data = await _api.fetchPosts();
    posts.assignAll(data);
    Get.snackbar(
      'Sukses',
      'Data berhasil diambil!',
      snackPosition: SnackPosition.BOTTOM,
      margin: const EdgeInsets.all(12),
    );
  });
}

Future<void> addPost() async {
  await _run(() async {
    final newPost = PostModel(
      title: title.value.isEmpty ? 'Flutter Post' : title.value,
      body: body.value.isEmpty ? 'Ini contoh POST.' : body.value,
      userId: 1,
    );

    final created = await _api.createPost(newPost);

    // JSONPlaceholder tidak benar-benar nyimpen, jadi kita simpan lokal juga
    posts.insert(0, created.copyWith(
      // kalau API balikin null id, bikin id lokal
      id: created.id ?? (posts.isNotEmpty ? (posts.first.id ?? 0) + 1 : 1),
    ));

    Get.snackbar(
      'Sukses',
      'Data berhasil ditambahkan!',
      snackPosition: SnackPosition.BOTTOM,
      margin: const EdgeInsets.all(12),
    );
  });
}

Future<void> updateFirst() async {
  if (posts.isEmpty) {
    Get.snackbar(
      'Info',
      'Ambil data dulu (GET) sebelum UPDATE',
      snackPosition: SnackPosition.BOTTOM,
      margin: const EdgeInsets.all(12),
    );
    return;
  }

  final targetId = posts.first.id ?? 1;

  await _run(() async {
    final updatedReq = PostModel(
      id: targetId,
      title: title.value.isEmpty ? 'Updated Title' : title.value,
      body: body.value.isEmpty ? 'Updated Body' : body.value,
      userId: 1,
    );

    final updated = await _api.updatePost(targetId, updatedReq);

    // update lokal
    posts[0] = posts[0].copyWith(
      title: updated.title,
      body: updated.body,
```

```
    );

    Get.snackbar(
      'Sukses',
      'Data berhasil diperbarui!',
      snackPosition: SnackPosition.BOTTOM,
      margin: const EdgeInsets.all(12),
    );
  });
 }

 Future<void> deleteFirst() async {
  if (posts.isEmpty) {
    Get.snackbar(
      'Info',
      'Tidak ada data untuk dihapus',
      snackPosition: SnackPosition.BOTTOM,
      margin: const EdgeInsets.all(12),
    );
    return;
  }

  final targetId = posts.first.id ?? 1;

  await _run(() async {
    await _api.deletePost(targetId);
    posts.removeAt(0);
    Get.snackbar(
      'Sukses',
      'Data berhasil dihapus!',
      snackPosition: SnackPosition.BOTTOM,
      margin: const EdgeInsets.all(12),
    );
  });
 }

 Future<void> _run(Future<void> Function() operation) async {
  try {
    isLoading.value = true;
    await operation();
  } catch (e) {
    Get.snackbar(
      'Gagal',
      e.toString(),
      snackPosition: SnackPosition.BOTTOM,
      margin: const EdgeInsets.all(12),
    );
  } finally {
    isLoading.value = false;
  }
 }
}
```

**api_service.dart :**

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

```
class ApiService {
  final String baseUrl = "https://jsonplaceholder.typicode.com";

  Future<List<PostModel>> fetchPosts() async {
    final response = await http.get(Uri.parse('$baseUrl/posts'));
    if (response.statusCode == 200) {
      final List data = json.decode(response.body);
      return data.map((e) => PostModel.fromJson(e)).toList();
    }
    throw Exception('Failed to load posts (${response.statusCode})');
  }

  Future<PostModel> createPost(PostModel post) async {
    final response = await http.post(
      Uri.parse('$baseUrl/posts'),
      headers: {'Content-Type': 'application/json'},
      body: json.encode(post.toJson()),
    );

    if (response.statusCode == 201) {
      final Map<String, dynamic> data = json.decode(response.body);
      // JSONPlaceholder biasanya balikin id baru (101)
      return PostModel.fromJson(data);
    }
    throw Exception('Failed to create post (${response.statusCode})');
  }

  Future<PostModel> updatePost(int id, PostModel post) async {
    final response = await http.put(
      Uri.parse('$baseUrl/posts/$id'),
      headers: {'Content-Type': 'application/json'},
      body: json.encode(post.toJson()),
    );

    if (response.statusCode == 200) {
      final Map<String, dynamic> data = json.decode(response.body);
      return PostModel.fromJson(data);
    }
    throw Exception('Failed to update post (${response.statusCode})');
  }

  Future<void> deletePost(int id) async {
    final response = await http.delete(Uri.parse('$baseUrl/posts/$id'));
    if (response.statusCode == 200) return;
    throw Exception('Failed to delete post (${response.statusCode})');
  }
}
```

- **home_screen.dart :**

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

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

  final PostController c = Get.put(PostController());
```

```
@override
Widget build(BuildContext context) {
 return Scaffold(
  appBar: AppBar(
   title: const Text('Data Storage API (GetX)'),
   centerTitle: true,
  ),
  body: SafeArea(
   child: SingleChildScrollView(
    padding: const EdgeInsets.all(14),
    child: Column(
     children: [
      _HeaderCard(controller: c),
      const SizedBox(height: 20),
      Obx(() {
       if (c.isLoading.value) {
        return const Center(child: CircularProgressIndicator());
       }
       if (c.posts.isEmpty) {
        return const Center(
         child: Text(
          'Tekan tombol GET untuk mengambil data',
          style: TextStyle(fontSize: 14, fontWeight: FontWeight.w500),
         ),
        );
       }
       return ListView.builder(
        shrinkWrap: true,
        physics: const NeverScrollableScrollPhysics(),
        itemCount: c.posts.length,
        itemBuilder: (context, index) {
         final p = c.posts[index];
         return Card(
          elevation: 3,
          margin: const EdgeInsets.only(bottom: 10),
          child: ListTile(
           leading: CircleAvatar(
            backgroundColor: Colors.orange.shade100,
            child: Text(
             '${p.id ?? '-'}',
             style: const TextStyle(fontWeight: FontWeight.bold),
            ),
           ),
           title: Text(
            p.title,
            maxLines: 2,
            overflow: TextOverflow.ellipsis,
            style: const TextStyle(fontWeight: FontWeight.w700),
           ),
           subtitle: Text(
            p.body,
            maxLines: 3,
            overflow: TextOverflow.ellipsis,
           ),
          ),
         );
        },
       );
      }),
     ],
    ),
   ),
  ),
```

```
    );
  }
}

class _HeaderCard extends StatelessWidget {
  const _HeaderCard({required this.controller});

  final PostController controller;

  @override
  Widget build(BuildContext context) {
    return Card(
      margin: const EdgeInsets.all(12),
      elevation: 4,
      child: Padding(
        padding: const EdgeInsets.all(12),
        child: Column(
          children: [
            const Align(
              alignment: Alignment.centerLeft,
              child: Text(
                'Input (opsional)',
                style: TextStyle(fontWeight: FontWeight.bold),
              ),
            ),
            const SizedBox(height: 8),
            TextField(
              onChanged: controller.setTitle,
              decoration: const InputDecoration(
                labelText: 'Title',
                border: OutlineInputBorder(),
                isDense: true,
              ),
            ),
            const SizedBox(height: 10),
            TextField(
              onChanged: controller.setBody,
              maxLines: 2,
              decoration: const InputDecoration(
                labelText: 'Body',
                border: OutlineInputBorder(),
                isDense: true,
              ),
            ),
            const SizedBox(height: 12),
            Obx(() => Wrap(
              spacing: 10,
              runSpacing: 10,
              children: [
                ElevatedButton.icon(
                  onPressed: controller.isLoading.value ? null : controller.getPosts,
                  icon: controller.isLoading.value
                      ? const SizedBox(height: 16, width: 16, child: CircularProgressIndicator(strokeWidth: 2))
                      : const Icon(Icons.download),
                  label: controller.isLoading.value
                      ? const SizedBox.shrink()
                      : const Text('GET'),
                ),
                ElevatedButton.icon(
                  onPressed: controller.isLoading.value ? null : controller.addPost,
                  icon: controller.isLoading.value
                      ? const SizedBox(height: 16, width: 16, child: CircularProgressIndicator(strokeWidth: 2))
                      : const Icon(Icons.add),
```

```
              label: controller.isLoading.value
                  ? const SizedBox.shrink()
                  : const Text('POST'),
              ),
              ElevatedButton.icon(
                onPressed: controller.isLoading.value ? null : controller.updateFirst,
                icon: controller.isLoading.value
                  ? const SizedBox(height: 16, width: 16, child: CircularProgressIndicator(strokeWidth: 2))
                  : const Icon(Icons.edit),
                label: controller.isLoading.value
                  ? const SizedBox.shrink()
                  : const Text('UPDATE'),
              ),
              ElevatedButton.icon(
                onPressed: controller.isLoading.value ? null : controller.deleteFirst,
                icon: controller.isLoading.value
                  ? const SizedBox(height: 16, width: 16, child: CircularProgressIndicator(strokeWidth: 2))
                  : const Icon(Icons.delete),
                label: controller.isLoading.value
                  ? const SizedBox.shrink()
                  : const Text('DELETE'),
              ),
            ],
          )),
        ],
      ),
    ),
  );
 }
}
```

**Lampiran B – Screenshot Output**

Berisi screenshot hasil pengujian aplikasi:

1. Tampilan awal aplikasi
2. Hasil GET data
3. Hasil POST data
4. Hasil UPDATE data
5. Hasil DELETE data