GUIDED&UNGUIDED

PEMROGRAMAN PERANGKAT BERGERAK

MODUL XIV

DATA STORAGE DENGAN REST API

Disusun Oleh :

Nama: Arwin Nabiel Arrofif

NIM: 2211104057

Kelas: SE 06 02

Asisten Praktikum :

Muhammad Faza Zulian Gesit Al Barru

Aisyah Hasna Aulia

Dosen Pengampu :

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

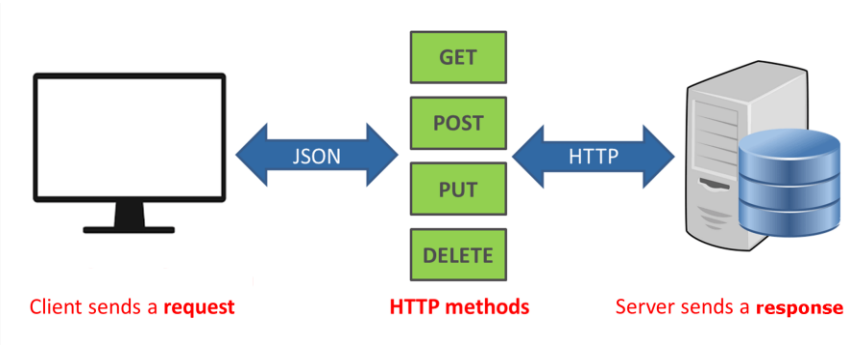
PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

Data Storage dengan Rest API



Gambar 1. Cara Kerja Restful API

A. Apa itu REST API

REST API (Representational State Transfer Application Programming Interface) adalah antarmuka yang memungkinkan aplikasi klien untuk berinteraksi dengan database melalui protokol HTTP. REST API menyediakan cara untuk membaca, menambahkan, memperbarui, dan menghapus data dari database tanpa harus mengakses database langsung.Mendapatkan *token* unik dari setiap perangkat pengguna.

Kegunaan REST API :

1. Interoperabilitas: REST API memungkinkan aplikasi berbasis web dan mobile untuk mengakses data yang sama.
2. Efisiensi: Data yang dikirimkan biasanya ringan (format JSON atau XML), membuatnya cepat untuk dikirim dan diterima.
3. Keamanan: API bisa membatasi akses menggunakan token autentikasi.

B. Apa itu HTTP

HTTP (Hypertext Transfer Protocol) adalah protokol komunikasi utama yang digunakan untuk mengirimkan data antara klien (misalnya browser atau aplikasi) dan server.

Metode HTTP Utama :

1. GET: Mengambil data dari server.
2. POST: Mengirim data baru ke server.
3. PUT/PATCH: Memperbarui data yang ada di server.
4. DELETE: Menghapus data dari server.

Komponen HTTP Request

1. URL: Alamat yang menunjukkan resource tertentu.

2. Method: Operasi yang akan dilakukan (GET, POST, dll.).
3. Headers: Informasi tambahan seperti format data atau token autentikasi.
4. Body: Data yang dikirimkan (digunakan dalam POST/PUT). Komponen HTTP Response

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

C. Praktikum

Langkah-langkah implementasi REST API di Flutter

1. Persiapan Projek Flutter

a. Buat proyek Flutter baru

b. Tambahkan dependensi http ke file pubspec.yaml:

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

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:

```
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

```
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;
        });
      }
  }
```

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),
                                ),
```

```
                              ),
                            ),
                          );
                        },
                      ),
                    ),
        ],
      ),
```

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'),
            ),
```

4. Implementasi HTTP POST

a. Membuat Service POST

Tambahkan metode berikut ke api_service.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:

```
  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');
      }
  }
```

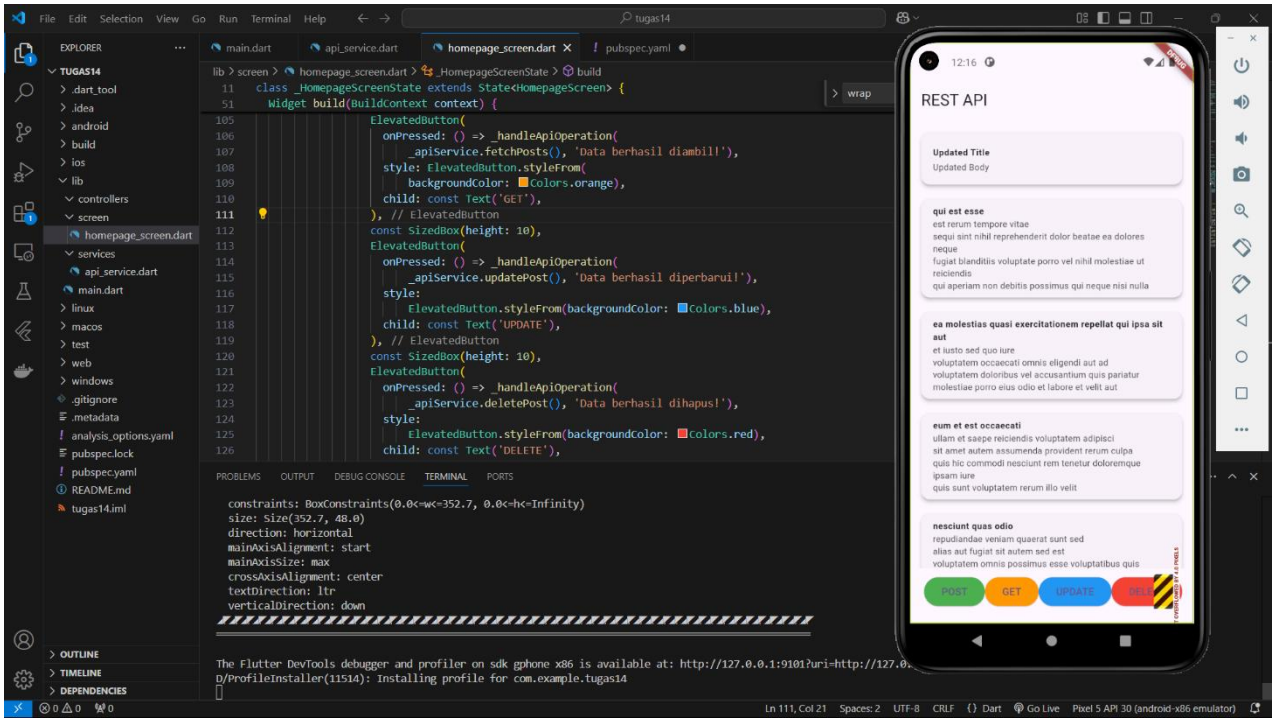
b. Membuat tampilan UI untuk DELETE

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'),
    ),
```

GUIDED

Code hasil praktikum ada di file guided



Tugas Mandiri (Unguided)

Modifikasi tampilan Guided dari praktikum di atas:

a. **Gunakan State Management dengan GetX:**

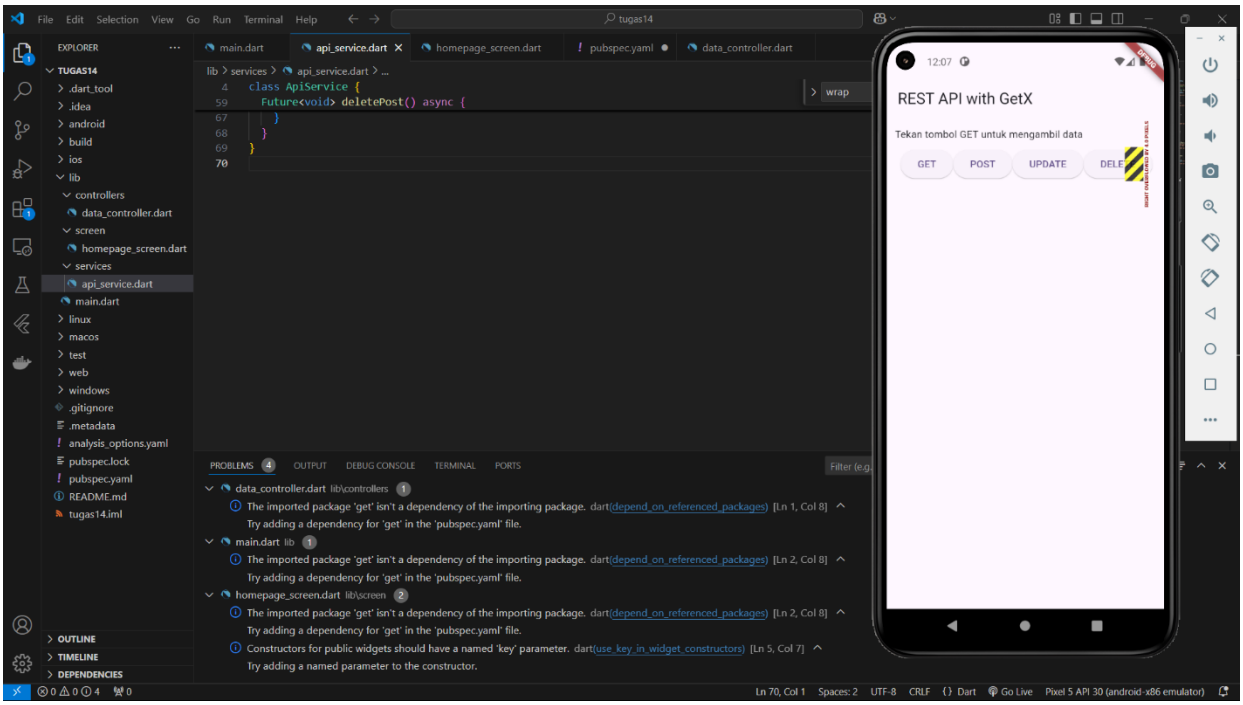
- Atur data menggunakan *state management* GetX agar lebih mudah dikelola.
- Implementasi GetX meliputi pembuatan controller untuk mengelola data dan penggunaan widget Obx untuk menampilkan data secara otomatis setiap kali ada perubahan.

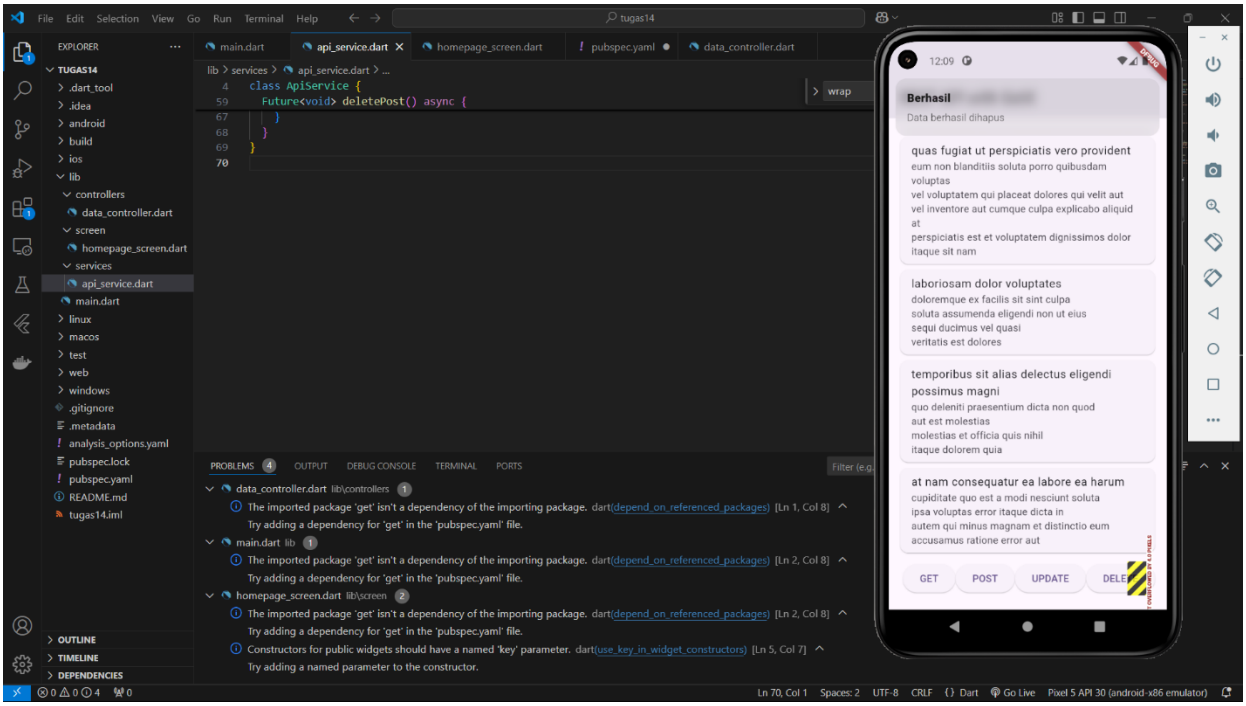
b. **Tambahkan Snackbar untuk Memberikan Respon Berhasil:**

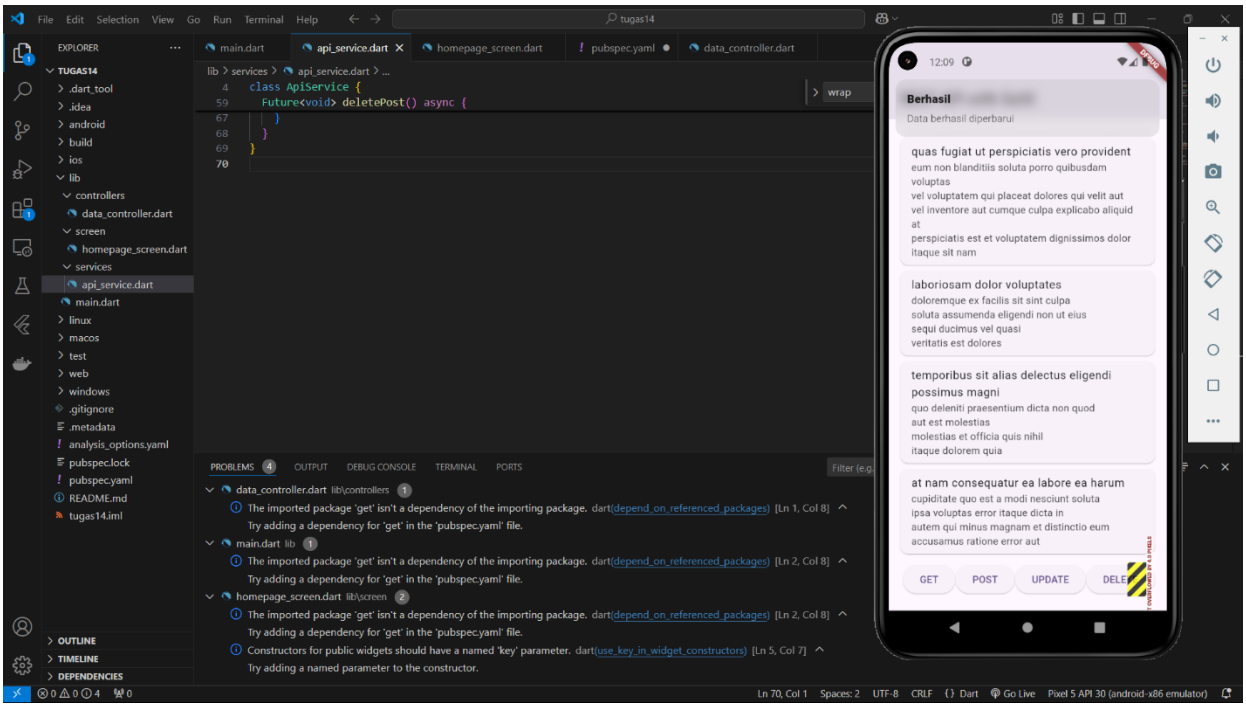
- Tampilkan snackbar setelah setiap operasi berhasil, seperti menambah atau memperbarui data.
- Gunakan Get.snackbar agar pesan sukses muncul di layar dan mudah dipahami oleh pengguna.

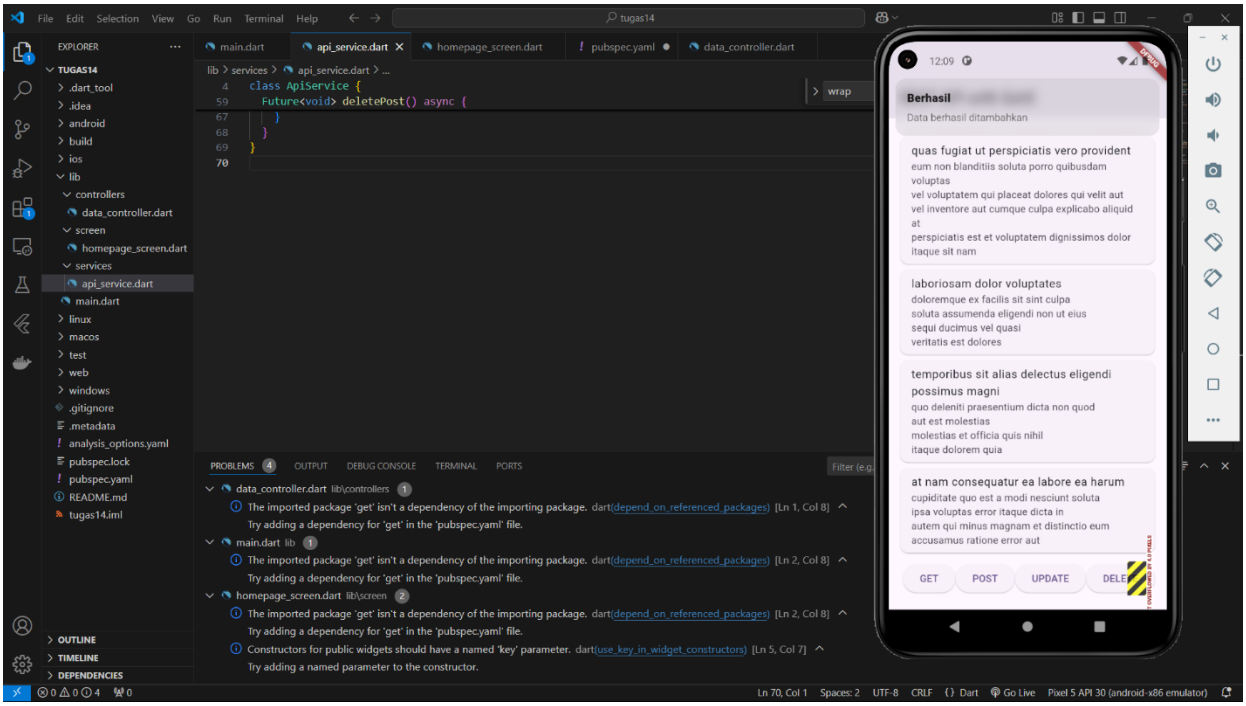
Note: Jangan lupa sertakan source code, screenshoot output, dan deskripsi program. Kreatifitas menjadi nilai tambah.

Jawab:




File Edit Selection View Go Run Terminal Help
tugas14
EXPLORER
TUGAS14
main.dart
api_service.dart
homepage_screen.dart
pubspec.yaml
data_controller.dart
lib > services > api_service.dart > ...
class ApiService {
Future<void> deletePost() async {
wrap
12:09
Berhasil
Data berhasil dihapus
quas fugiat ut perspiciatis vero provident
eum non blanditiis soluta porro quibusdam voluptas
vel voluptatem qui placeat dolores qui velit aut
vel inventore aut cumque culpa explicabo aliquid at
perspiciatis est et voluptatem dignissimos dolor itaque sit nam
laboriosam dolor voluptates
doloremque ex facilis sit sint culpa
soluta assumenda eligendi non ut eius
sequi ducimus vel quasi
veritatis est dolores
temporibus sit alias delectus eligendi possimus magni
quo deleniti praesentium dicta non quod
aut est molestias
molestias et officia quis nihil
itaque dolorem quia
at nam consequatur ea labore ea harum
cupiditate quo est a modi nesciunt soluta
ipsa voluptas error itaque dicta in
autem qui minus magnam et distinctio eum
accusamus ratione error aut
GET
POST
UPDATE
DELE
PROBLEMS 4
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
data_controller.dart lib\controllers 1
The imported package 'get' isn't a dependency of the importing package. dart(depend_on_referenced_packages) [Ln 1, Col 8]
Try adding a dependency for 'get' in the 'pubspec.yaml' file.
main.dart lib 1
The imported package 'get' isn't a dependency of the importing package. dart(depend_on_referenced_packages) [Ln 2, Col 8]
Try adding a dependency for 'get' in the 'pubspec.yaml' file.
homepage_screen.dart lib\screen 2
The imported package 'get' isn't a dependency of the importing package. dart(depend_on_referenced_packages) [Ln 2, Col 8]
Try adding a dependency for 'get' in the 'pubspec.yaml' file.
Constructors for public widgets should have a named 'key' parameter. dart(use_key_in_widget_constructors) [Ln 5, Col 7]
Try adding a named parameter to the constructor.
OUTLINE
TIMELINE
DEPENDENCIES
Ln 70, Col 1
Spaces: 2
UTF-8
CRLF
Dart
Go Live
Pixel 5 API 30 (android-x86 emulator)


File Edit Selection View Go Run Terminal Help
tugas14
EXPLORER
TUGAS14
main.dart
api_service.dart
homepage_screen.dart
pubspec.yaml
data_controller.dart
lib > services > api_service.dart > ...
class ApiService {
Future<void> deletePost() async {
wrap
12:09
Berhasil
Data berhasil diperbarui
quas fugiat ut perspiciatis vero provident
eum non blanditiis soluta porro quibusdam voluptas
vel voluptatem qui placeat dolores qui velit aut
vel inventore aut cumque culpa explicabo aliquid at
perspiciatis est et voluptatem dignissimos dolor itaque sit nam
laboriosam dolor voluptates
doloremque ex facilis sit sint culpa
soluta assumenda eligendi non ut eius
sequi ducimus vel quasi
veritatis est dolores
temporibus sit alias delectus eligendi possimus magni
quo deleniti praesentium dicta non quod
aut est molestias
molestias et officia quis nihil
itaque dolorem quia
at nam consequatur ea labore ea harum
cupiditate quo est a modi nesciunt soluta
ipsa voluptas error itaque dicta in
autem qui minus magnam et distinctio eum
accusamus ratione error aut
GET
POST
UPDATE
DELE
PROBLEMS 4
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
data_controller.dart lib\controllers 1
The imported package 'get' isn't a dependency of the importing package. dart(depend_on_referenced_packages) [Ln 1, Col 8]
Try adding a dependency for 'get' in the 'pubspec.yaml' file.
main.dart lib 1
The imported package 'get' isn't a dependency of the importing package. dart(depend_on_referenced_packages) [Ln 2, Col 8]
Try adding a dependency for 'get' in the 'pubspec.yaml' file.
homepage_screen.dart lib\screen 2
The imported package 'get' isn't a dependency of the importing package. dart(depend_on_referenced_packages) [Ln 2, Col 8]
Try adding a dependency for 'get' in the 'pubspec.yaml' file.
Constructors for public widgets should have a named 'key' parameter. dart(use_key_in_widget_constructors) [Ln 5, Col 7]
Try adding a named parameter to the constructor.
OUTLINE
TIMELINE
DEPENDENCIES
Ln 70, Col 1
Spaces: 2
UTF-8
CRLF
Dart
Go Live
Pixel 5 API 30 (android-x86 emulator)



Untuk code ada di file Unguided

Deskripsi program:

Program ini adalah aplikasi Flutter yang memanfaatkan REST API dan menggunakan GetX sebagai framework untuk manajemen state. Aplikasi ini dirancang untuk melakukan operasi CRUD (Create, Read, Update, Delete) dengan data yang diambil dari API eksternal.

Pada halaman utama, aplikasi menampilkan daftar data berupa judul dan isi dari setiap entri yang diambil dari API. Pengguna dapat menambahkan data baru melalui tombol **POST**, memperbarui data menggunakan tombol **UPDATE**, atau menghapus data dengan tombol **DELETE**. Semua data yang ditampilkan di UI diperbarui secara otomatis menggunakan GetX setiap kali ada perubahan data. Selain itu, aplikasi memberikan feedback interaktif kepada pengguna dengan menampilkan **Snackbar** untuk setiap operasi, baik sukses maupun gagal. Dengan desain UI yang minimalis dan responsif, aplikasi ini memastikan pengalaman pengguna yang intuitif.

Program ini menggunakan API placeholder dari *https://jsonplaceholder.typicode.com* untuk simulasi data, tetapi dapat dengan mudah diadaptasi untuk API lainnya. Teknologi utama yang digunakan adalah Flutter, GetX untuk manajemen state, dan paket HTTP untuk komunikasi dengan API.