LAPORAN PRAKTIKUM
PEMROGRAMAN PERANGKAT BERGERAK

MODUL XIV

DATA STORAGE (API)

Disusun Oleh :

Dimas Cahyo Margono / 2211104060

SE-06-02

Asisten Praktikum :

Muhammad Faza Zulian Gesit Al Barru

Aisyah Hasna Aulia

Dosen Pengampu :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

2024

GUIDED

1. **LOGIC**

   Sebelum kita pergi ke tampilan flutter project, Langkah yang pertama kita lakukan adalah mengatur file-file services untuk Rest API.

   A. PUBSPEC.YAML

   Sourcecode

```yaml
name: pertemuan14
description: "A new Flutter project."
# The following line prevents the package from being accidentally
published to
# pub.dev using `flutter pub publish`. This is preferred for private
packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as
versionCode.
# Read more about Android versioning at
https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-
number is used as CFBundleVersion.
# Read more about iOS versioning at
#
https://developer.apple.com/library/archive/documentation/General/Referenc
e/InfoPlistKeyReference/Articles/CoreFoundationKeys.html
# In Windows, build-name is used as the major, minor, and patch parts
# of the product and file versions while build-number is used as the build
suffix.
version: 1.0.0+1

environment:
  sdk: ^3.5.4

# Dependencies specify other packages that your package needs in order to
work.
# To automatically upgrade your package dependencies to the latest
versions
# consider running `flutter pub upgrade --major-versions`. Alternatively,
# dependencies can be manually updated by changing the version numbers
below to
# the latest version available on pub.dev. To see which dependencies have
newer
# versions available, run `flutter pub outdated`.
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

dev_dependencies:
  flutter_test:
    sdk: flutter

  # The "flutter_lints" package below contains a set of recommended lints
to
  # encourage good coding practices. The lint set provided by the package
is
  # activated in the `analysis_options.yaml` file located at the root of
your
  # package. See that file for information about deactivating specific
lint
  # rules and activating additional ones.
  flutter_lints: ^4.0.0

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  # assets:
  #   - images/a_dot_burr.jpeg
  #   - images/a_dot_ham.jpeg

  # An image asset can refer to one or more resolution-specific
"variants", see
  # https://flutter.dev/to/resolution-aware-images

  # For details regarding adding assets from package dependencies, see
  # https://flutter.dev/to/asset-from-package

  # To add custom fonts to your application, add a fonts section here,
  # in this "flutter" section. Each entry in this list should have a
  # "family" key with the font family name, and a "fonts" key with a
  # list giving the asset and other descriptors for the font. For
  # example:
  # fonts:
  #   - family: Schyler
  #     fonts:
  #       - asset: fonts/Schyler-Regular.ttf
```

```
  #       - asset: fonts/Schyler-Italic.ttf
  #         style: italic
  #   - family: Trajan Pro
  #     fonts:
  #       - asset: fonts/TrajanPro.ttf
  #       - asset: fonts/TrajanPro_Bold.ttf
  #         weight: 700
  #
  # For details regarding fonts from package dependencies,
  # see https://flutter.dev/to/font-from-package
```

Deskripsi Program

Program yang ditampilkan adalah sebuah proyek Flutter dengan nama pertemuan14, yang dirancang untuk membangun aplikasi berbasis Flutter. Pada file konfigurasi pubspec.yaml, proyek ini menggunakan pustaka http sebagai dependency utama untuk memungkinkan aplikasi berkomunikasi dengan server melalui protokol HTTP. Hal ini memungkinkan pengembang untuk mengambil data dari API eksternal, mengirim data ke server, atau melakukan operasi lainnya seperti autentikasi. Selain itu, program mendukung penggunaan ikon gaya iOS melalui cupertino_icons, serta menyertakan flutter_lints untuk memastikan kualitas kode yang baik. Dengan konfigurasi ini, aplikasi siap untuk dikembangkan dengan fitur yang melibatkan komunikasi data secara dinamis.

2. TAMPILAN APLIKASI

Setelah kita melakukan konfigurasi untuk Firebase Notification, tidak lupa kita melakukan coding untuk tampilan aplikasinya

A. SCREEN/HOMEPAGE_SCREEN

Sourcecode

```dart
import 'package:flutter/material.dart';
import '../services/api_services.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'),
                ),
              ],
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

Deskripsi Program

Program di atas adalah implementasi Flutter untuk sebuah halaman bernama `HomepageScreen`, yang dirancang untuk berinteraksi dengan REST API menggunakan layanan `ApiService`. Halaman ini memungkinkan pengguna melakukan operasi CRUD (Create, Read, Update, Delete) pada data API. Saat halaman dimuat, data dari API langsung diambil melalui fungsi `fetchPosts` dan ditampilkan dalam bentuk daftar menggunakan widget `ListView.builder`. Pengguna juga dapat menekan tombol `POST`, `GET`, `UPDATE`, atau `DELETE` untuk melakukan operasi API terkait. Selama operasi berlangsung, indikator loading (`CircularProgressIndicator`) akan ditampilkan untuk memberikan umpan balik visual kepada pengguna. Pesan sukses atau error ditampilkan melalui `SnackBar`, sementara data yang diperbarui langsung disinkronkan ke antarmuka menggunakan `setState`. Desain ini menciptakan pengalaman interaktif untuk memanipulasi data dari API secara dinamis.

B. SERVICES/API_SERVICES.DART

Sourcecode

```
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');
```

```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');
  }
}

// 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');
  }
```

```
  }
}
```

Deskripsi Program

Program di atas adalah implementasi kelas ApiService yang berfungsi untuk melakukan operasi CRUD (Create, Read, Update, Delete) pada data melalui REST API di https://jsonplaceholder.typicode.com. Fungsi fetchPosts digunakan untuk mengambil data postingan, createPost untuk menambahkan data baru ke server, updatePost untuk memperbarui data postingan tertentu, dan deletePost untuk menghapus postingan berdasarkan ID. Setiap operasi memanfaatkan pustaka http untuk mengirimkan permintaan ke server, memproses respons, dan memperbarui data lokal yang disimpan dalam variabel posts. Jika terjadi kesalahan, fungsi-fungsi ini akan melemparkan exception untuk menangani error. Kelas ini menyediakan abstraksi sederhana untuk mengelola komunikasi dengan API, membuatnya mudah digunakan dalam aplikasi Flutter.

C. MAIN.DART

Sourcecode

```dart
import 'package:flutter/material.dart';
import 'package:pertemuan14/screen/homepage_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(),
    );
  }
}
```

Deskripsi Program

Program di atas adalah implementasi utama aplikasi Flutter, di mana kelas `MyApp` merupakan titik masuk utama aplikasi. Fungsi `main()` memanggil `runApp()` untuk menjalankan aplikasi dengan widget `MyApp` sebagai root. Dalam `MyApp`, aplikasi dikonfigurasikan menggunakan `MaterialApp`, yang menetapkan judul

aplikasi, tema dengan skema warna berbasis warna ungu tua (`Colors.deepPurple`), dan mendukung desain Material 3. Halaman awal aplikasi diatur ke widget `HomepageScreen`, yang merupakan layar utama untuk menampilkan dan mengelola data yang diambil dari REST API. Program ini menyediakan kerangka kerja dasar untuk aplikasi Flutter dengan desain modern dan navigasi awal ke halaman utama.

Screenshoot Output

Tampilan Aplikasi

Data yang di POST

Data yang dilakukan UPDATE

Data yang di hapus menggunakan DELETE

Deskripsi Program

Gambar yang Anda lampirkan menunjukkan antarmuka aplikasi Flutter yang dirancang untuk mengelola data menggunakan operasi CRUD (Create, Read, Update, Delete) pada REST API. Aplikasi ini memuat daftar postingan dari API, menampilkan setiap item dalam bentuk kartu dengan judul dan isi. Tombol-tombol di bagian bawah (POST, GET, UPDATE, DELETE) memungkinkan pengguna untuk melakukan operasi API tertentu: menambahkan postingan baru, mengambil data terkini, memperbarui postingan tertentu, atau menghapus postingan. Operasi API memberikan umpan balik secara real-time, seperti pembaruan pada daftar atau notifikasi melalui SnackBar. Indikator loading (CircularProgressIndicator) ditampilkan selama proses berlangsung, memberikan pengalaman pengguna yang interaktif dan responsif. Aplikasi ini mengintegrasikan antarmuka yang intuitif dengan logika backend untuk pengelolaan data dinamis.

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.

A. SCREEN/HOMEPAGE_SCREEN.DART

Sourcecode

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

class ApiController extends GetxController {
  final ApiService _apiService = ApiService();

  // Obx states
  var posts = <dynamic>[].obs;
  var isLoading = false.obs;

  // Fungsi untuk fetch data
  Future<void> fetchPosts() async {
    isLoading(true);
    try {
      await _apiService.fetchPosts();
      posts.assignAll(_apiService.posts);
      Get.snackbar('Success', 'Data berhasil diambil!');
    } catch (e) {
      Get.snackbar('Error', 'Gagal mengambil data: $e');
    } finally {
      isLoading(false);
    }
  }

  // Fungsi untuk create post
  Future<void> createPost() async {
    isLoading(true);
    try {
      await _apiService.createPost();
      posts.assignAll(_apiService.posts);
      Get.snackbar('Success', 'Data berhasil ditambahkan!');
    } catch (e) {
```

```
      Get.snackbar('Error', 'Gagal menambah data: $e');
    } finally {
      isLoading(false);
    }
  }

  // Fungsi untuk update post
  Future<void> updatePost() async {
    isLoading(true);
    try {
      await _apiService.updatePost();
      posts.assignAll(_apiService.posts);
      Get.snackbar('Success', 'Data berhasil diperbarui!');
    } catch (e) {
      Get.snackbar('Error', 'Gagal memperbarui data: $e');
    } finally {
      isLoading(false);
    }
  }

  // Fungsi untuk delete post
  Future<void> deletePost() async {
    isLoading(true);
    try {
      await _apiService.deletePost();
      posts.assignAll(_apiService.posts);
      Get.snackbar('Success', 'Data berhasil dihapus!');
    } catch (e) {
      Get.snackbar('Error', 'Gagal menghapus data: $e');
    } finally {
      isLoading(false);
    }
  }
}

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

  @override
  Widget build(BuildContext context) {
    final ApiController controller = Get.put(ApiController());

    return Scaffold(
      appBar: AppBar(
        title: const Text('REST API with GetX'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(12),
        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 Padding(
                              padding: const EdgeInsets.only(bottom:
12.0),
                              child: 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),
                                  ),
                                ),
                              ),
                            );
                          },
                        ),
                      )),
            Padding(
              padding: const EdgeInsets.all(8.0),
              child: Row(
                children: [
                  ElevatedButton(
                    onPressed: () => controller.createPost(),
                    style:
                        ElevatedButton.styleFrom(backgroundColor:
Colors.green),
                    child: const Text('POST'),
                  ),
                  const SizedBox(height: 10),
                  ElevatedButton(
                    onPressed: () => controller.fetchPosts(),
                    style: ElevatedButton.styleFrom(
                        backgroundColor: Colors.orange),
                    child: const Text('GET'),
                  ),
                  const SizedBox(height: 10),
                  ElevatedButton(
                    onPressed: () => controller.updatePost(),
                    style:
                        ElevatedButton.styleFrom(backgroundColor:
Colors.blue),
```

```
                  child: const Text('UPDATE'),
                ),
                const SizedBox(height: 10),
                ElevatedButton(
                  onPressed: () => controller.deletePost(),
                  style:
                      ElevatedButton.styleFrom(backgroundColor:
Colors.red),
                  child: const Text('DELETE'),
                ),
              ],
            ),
          ),
        ],
      ),
    ),
  );
  }
}
```

B. SERVICES/API_SERVICES.DART

Sourcecode

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

C. MAIN.DART

Sourcecode

```
import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:pertemuan14/screen/homepage_screen.dart';
```

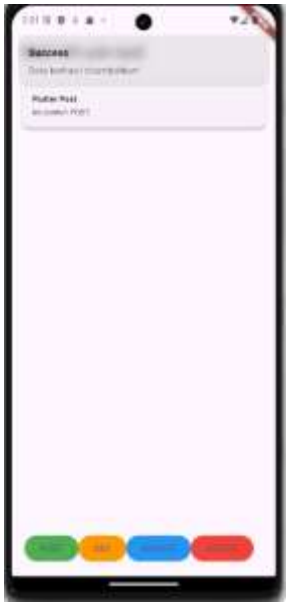
```dart
void main() {
  runApp(const MyApp());
}

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

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

Screenshoot output

Tampilan Aplikasi



Tambah Data dengan POST

Ambil Data dengan GET

Perbarui Data dengan UPDATE

Hapus Data dengan DELETE

Deskripsi Program

Aplikasi yang ditampilkan menggunakan Flutter dengan state management GetX untuk mengelola data dan operasi CRUD (Create, Read, Update, Delete) melalui REST API. Aplikasi ini memuat daftar postingan yang ditampilkan dalam bentuk kartu, dilengkapi dengan tombol-tombol aksi (POST, GET, UPDATE, DELETE) untuk melakukan operasi pada API. Setiap operasi akan menampilkan notifikasi Snackbar menggunakan Get.snackbar, memberikan umpan balik langsung kepada pengguna, seperti "Data berhasil diambil" atau "Data berhasil diperbarui".

Penggunaan Obx dalam widget memastikan tampilan aplikasi diperbarui secara real-time setiap kali data diubah. Antarmuka aplikasi ini dirancang responsif dan user-friendly, membuat interaksi dengan data API menjadi intuitif dan efisien.