Nama : Magdalena Rohmannawati

NIM : H1D022003

Shift : A

PENJELASAN RESPONSI 1

Helpers

1. api_url.dart

```
class ApiUrl {
  static const String baseUrl = 'http://responsi.webwizards.my.id/';
  static const String registrasi = baseUrl + 'api/registrasi';
  static const String login = baseUrl + 'api/login';
  static const String listWisata = baseUrl + 'api/pariwisata/destinasi_wisata';
  static const String createWisata =
      baseUrl + 'api/pariwisata/destinasi_wisata';
  static String updateWisata(int id) {
    return baseUrl +
        'api/pariwisata/destinasi_wisata/' +
        id.toString() +
        '/update';
  }

  static String showWisata(int id) {
    return baseUrl + 'api/pariwisata/destinasi_wisata/' + id.toString();
  }

  static String deleteWisata(int id) {
    return baseUrl +
        'api/pariwisata/destinasi_wisata/' +
        id.toString() +
        '/delete';
  }
}
```

Kode ApiUrl di atas adalah sebuah kelas yang digunakan untuk menyimpan URL-URL endpoint API dalam sebuah aplikasi. Berikut adalah penjelasan untuk tiap bagiannya:

1. baseUrl: URL dasar untuk semua permintaan API, yaitu 'http://responsi.webwizards.my.id/'. Semua endpoint API lainnya dibangun berdasarkan URL ini.
2. registrasi: Endpoint untuk API registrasi pengguna ('api/registrasi').
3. login: Endpoint untuk API login pengguna ('api/login').

4. listWisata: Endpoint untuk mendapatkan daftar destinasi wisata ('api/pariwisata/destinasi_wisata').
5. createWisata: Endpoint untuk membuat data baru destinasi wisata ('api/pariwisata/destinasi_wisata').
6. updateWisata(int id): Fungsi yang mengembalikan URL untuk mengupdate data destinasi wisata dengan ID tertentu. Misalnya, jika ID adalah 5, maka URL yang dihasilkan adalah 'http://responsi.webwizards.my.id/api/pariwisata/destinasi_wisata/5/update'.
7. showWisata(int id): Fungsi yang mengembalikan URL untuk melihat data destinasi wisata dengan ID tertentu. Misalnya, jika ID adalah 3, maka URL yang dihasilkan adalah 'http://responsi.webwizards.my.id/api/pariwisata/destinasi_wisata/3'.
8. deleteWisata(int id): Fungsi yang mengembalikan URL untuk menghapus data destinasi wisata dengan ID tertentu. Misalnya, jika ID adalah 7, maka URL yang dihasilkan adalah 'http://responsi.webwizards.my.id/api/pariwisata/destinasi_wisata/7/delete'.

2. api.dart

```
import 'dart:io';
import 'package:http/http.dart' as http;
import 'package:manajemen_pariwasata/helpers/user_info.dart';
import 'app_exception.dart';

class Api {
Future<dynamic> post(dynamic url, dynamic data) async {
var token = await UserInfo().getToken();
var responseJson;
try {
final response = await http.post(
Uri.parse(url),
body: data,
headers: {
HttpHeaders.authorizationHeader: "Bearer $token",
},
);
responseJson = _returnResponse(response);
} on SocketException {
throw FetchDataException('No Internet connection');
}
return responseJson;
}

Future<dynamic> get(dynamic url) async {
var token = await UserInfo().getToken();
var responseJson;
try {
final response = await http.get(
Uri.parse(url),
```

```
headers: {
HttpHeaders.authorizationHeader: "Bearer $token",
},
);
responseJson = _returnResponse(response);
} on SocketException {
throw FetchDataException('No Internet connection');
}
return responseJson;
}

Future<dynamic> put(dynamic url, dynamic data) async {
var token = await UserInfo().getToken();
var responseJson;
try {
final response = await http.put(
Uri.parse(url),
body: data,
headers: {
HttpHeaders.authorizationHeader: "Bearer $token",
HttpHeaders.contentTypeHeader: "application/json",
},
);
responseJson = _returnResponse(response);
} on SocketException {
throw FetchDataException('No Internet connection');
}
return responseJson;
}

Future<dynamic> delete(dynamic url) async {
var token = await UserInfo().getToken();
var responseJson;
try {
final response = await http.delete(
Uri.parse(url),
headers: {
HttpHeaders.authorizationHeader: "Bearer $token",
},
);
responseJson = _returnResponse(response);
} on SocketException {
throw FetchDataException('No Internet connection');
}
return responseJson;
```

```
}
  dynamic _returnResponse(http.Response response) {
    switch (response.statusCode) {
      case 200:
        return response;
      case 400:
        throw BadRequestException(response.body.toString());
      case 401:
      case 403:
        throw UnauthorisedException(response.body.toString());
      case 422:
        throw InvalidInputException(response.body.toString());
      case 500:
      default:
        throw FetchDataException(
            'Error occurred while communicating with server. Status Code: ${response.statusCode}');
    }
  }
}
```

Kode di atas adalah kelas Api yang digunakan untuk menangani permintaan HTTP (API) dalam aplikasi Flutter. Berikut adalah penjelasan singkatnya:

1. Import:
   - dart:io: Untuk menangani koneksi dan header HTTP.
   - http: Digunakan untuk membuat permintaan HTTP seperti GET, POST, PUT, dan DELETE.
   - UserInfo: Digunakan untuk mengambil token pengguna.
   - app_exception.dart: Berisi definisi exception custom untuk menangani berbagai kesalahan API.
2. Metode HTTP:
   - post(url, data): Mengirim permintaan POST ke url dengan data (body), dan menambahkan token Bearer di header.
   - get(url): Mengirim permintaan GET ke url dengan token di header.
   - put(url, data): Mengirim permintaan PUT ke url dengan data, serta menambahkan token dan header JSON.
   - delete(url): Mengirim permintaan DELETE ke url dengan token.
3. Exception Handling:
   - Semua metode (POST, GET, PUT, DELETE) menggunakan blok try-catch untuk menangani potensi kegagalan jaringan (SocketException) dengan melempar FetchDataException.
4. Token Pengguna:
   - Sebelum mengirim permintaan, aplikasi mengambil token autentikasi dari UserInfo().getToken(), lalu memasukkannya dalam header sebagai Bearer token.
5. Handling Response:
   - _returnResponse(response): Fungsi ini menangani respon API. Berdasarkan status kode HTTP, ia akan:
     - 200: Mengembalikan respon jika sukses.
     - 400, 401, 403, 422, 500: Melempar exception custom (BadRequestException, UnauthorisedException, InvalidInputException, dll.) jika terjadi kesalahan.

3. app_exception.dart

```dart
class AppException implements Exception {
  final String? _message;
  final String _prefix;

  AppException([this._message, this._prefix = ""]);

  @override
  String toString() {
    return "$_prefix$_message";
  }
}

class FetchDataException extends AppException {
  FetchDataException([String? message])
      : super(message, "Error During Communication: ");
}

class BadRequestException extends AppException {
  BadRequestException([String? message]) : super(message, "Invalid Request: ");
}

class UnauthorisedException extends AppException {
  UnauthorisedException([String? message]) : super(message, "Unauthorised: ");
}

class UnprocessableEntityException extends AppException {
  UnprocessableEntityException([String? message])
      : super(message, "Unprocessable Entity: ");
}

class InvalidInputException extends AppException {
  InvalidInputException([String? message]) : super(message, "Invalid Input: ");
}
```

Kode di atas mendefinisikan kelas-kelas custom exception yang mewarisi dari kelas Exception di Dart. Berikut adalah penjelasan singkat tentang masing-masing kelas:

1. AppException:
    - Kelas dasar untuk semua exception khusus dalam aplikasi ini.
    - Memiliki dua properti: _message (pesan kesalahan) dan _prefix (prefix deskriptif sebelum pesan kesalahan).
    - Metode toString() digunakan untuk menghasilkan string deskriptif dari kesalahan yang terdiri dari _prefix diikuti oleh _message.
2. FetchDataException:
    - Exception ini digunakan ketika terjadi kesalahan saat berkomunikasi dengan server (misalnya, jaringan putus atau server tidak merespons).

- Memiliki prefix "Error During Communication: ".

3. BadRequestException:
   - Digunakan untuk kesalahan permintaan yang tidak valid (status HTTP 400).
   - Prefix yang digunakan adalah "Invalid Request: ".
4. UnauthorisedException:
   - Digunakan ketika terjadi kesalahan autentikasi atau otorisasi (status HTTP 401 atau 403).
   - Prefix yang digunakan adalah "Unauthorised: ".
5. UnprocessableEntityException:
   - Digunakan untuk kesalahan saat server tidak bisa memproses entitas yang dikirim (status HTTP 422).
   - Prefix yang digunakan adalah "Unprocessable Entity: ".
6. InvalidInputException:
   - Digunakan ketika input yang diberikan tidak valid (biasanya untuk kesalahan input dari pengguna).
   - Prefix yang digunakan adalah "Invalid Input: ".

4. user_info.dart

```dart
import 'package:shared_preferences/shared_preferences.dart';

class UserInfo {
  Future<void> setToken(String value) async {
    SharedPreferences pref = await SharedPreferences.getInstance();
    await pref.setString("token", value);
  }

  Future<String?> getToken() async {
    SharedPreferences pref = await SharedPreferences.getInstance();
    return pref.getString("token");
  }

  Future<void> setUserID(int value) async {
    SharedPreferences pref = await SharedPreferences.getInstance();
    await pref.setInt("userID", value);
  }

  Future<int?> getUserID() async {
    SharedPreferences pref = await SharedPreferences.getInstance();
    return pref.getInt("userID");
  }

  Future<void> logout() async {
    SharedPreferences pref = await SharedPreferences.getInstance();
    await pref.clear();
  }
}
```

Kode di atas adalah kelas UserInfo yang berfungsi untuk menyimpan, mengambil, dan menghapus data pengguna menggunakan SharedPreferences di Flutter. Berikut penjelasan fungsi-fungsi utamanya:

1. setToken(String value):
    - Menyimpan token otentikasi (biasanya token JWT) ke dalam penyimpanan lokal menggunakan SharedPreferences.
    - Fungsi ini dipanggil saat pengguna berhasil login dan token disimpan untuk digunakan dalam permintaan API berikutnya.
2. getToken():
    - Mengambil token otentikasi yang disimpan di penyimpanan lokal.
    - Fungsi ini digunakan untuk mengambil token saat aplikasi butuh otorisasi untuk mengakses API.
3. setUserID(int value):
    - Menyimpan userID (ID pengguna) ke dalam SharedPreferences.
    - Berguna untuk menyimpan informasi identitas pengguna setelah login.
4. getUserID():
    - Mengambil userID dari penyimpanan lokal.
    - Ini memungkinkan aplikasi untuk mendapatkan ID pengguna yang sedang aktif tanpa perlu melakukan login ulang.
5. logout():
    - Menghapus semua data yang disimpan (token dan userID) dari SharedPreferences.
    - Fungsi ini dipanggil saat pengguna logout, memastikan bahwa data pengguna sebelumnya dihapus.

Model

1. login.dart

```dart
class Login {
  int? code;
  bool? status;
  String? token;
  int? userID;
  String? userEmail;
  Login({this.code, this.status, this.token, this.userID, this.userEmail});
  factory Login.fromJson(Map<String, dynamic> obj) {
    if (obj['code'] == 200) {
      return Login(
          code: obj['code'],
          status: obj['status'],
          token: obj['data']['token'],
          userID: obj['data']['user']['id'],
```

```
          userEmail: obj['data']['user']['email']);
    } else {
      return Login(
        code: obj['code'],
        status: obj['status'],
      );
    }
  }
}
```

Kelas Login di atas digunakan untuk memodelkan data hasil respons dari API ketika melakukan proses login. Berikut penjelasan singkat setiap bagian:

Properti:

1. int? code:
    - Kode status HTTP atau kode respons yang diterima dari API (misalnya, 200 untuk berhasil).
2. bool? status:
    - Status login, biasanya true jika login berhasil, dan false jika gagal.
3. String? token:
    - Token otentikasi (misalnya, JWT) yang diberikan jika login berhasil. Token ini diperlukan untuk otorisasi permintaan API berikutnya.
4. int? userID:
    - ID pengguna yang login. Disimpan jika login berhasil.
5. String? userEmail:
    - Email pengguna yang login. Disimpan untuk referensi lebih lanjut atau digunakan dalam aplikasi.

Konstruktor:

- Login({this.code, this.status, this.token, this.userID, this.userEmail}):
    - Konstruktor untuk membuat instance dari kelas Login, dengan semua parameter opsional (nullable).

factory Login.fromJson(Map<String, dynamic> obj):

- factory memungkinkan penggunaan logika untuk menginisialisasi objek dari data JSON. Pada dasarnya, ini adalah konstruktor khusus yang dipakai untuk mengubah data respons API (berformat JSON) menjadi objek Login.
- Logika di dalam fungsi ini:
    1. Jika kode respons obj['code'] adalah 200 (berhasil), maka diinisialisasi semua properti termasuk token, userID, dan userEmail, dengan data yang berasal dari obj['data']['user'].
    2. Jika tidak, hanya code dan status yang akan diambil (karena login gagal atau ada kesalahan).

2. registrasi.dart

```
class Registrasi {
  int? code;
  bool? status;
  String? data;
  Registrasi({this.code, this.status, this.data});
  factory Registrasi.fromJson(Map<String, dynamic> obj) {
    return Registrasi(
        code: obj['code'], status: obj['status'], data: obj['data']);
  }
}
```

Kelas Registrasi digunakan untuk memodelkan data hasil respons dari API saat proses registrasi pengguna. Berikut adalah penjelasan singkat tentang setiap bagian:
Properti:

1. int? code:
    - Menyimpan kode status HTTP atau kode respons dari API (misalnya, 200 untuk berhasil).
2. bool? status:
    - Menyimpan status registrasi, apakah berhasil (true) atau gagal (false).
3. String? data:
    - Menyimpan pesan atau data lain yang diterima dari API setelah registrasi (misalnya, pesan sukses atau error).

Konstruktor:

- Registrasi({this.code, this.status, this.data}):
    - Konstruktor ini digunakan untuk menginisialisasi objek Registrasi dengan semua properti bersifat opsional (nullable).

factory Registrasi.fromJson(Map<String, dynamic> obj):

- Konstruktor factory ini digunakan untuk membuat objek Registrasi dari respons API yang berformat JSON.
- Logika di dalam fungsi ini:
    - Data yang diterima dari API (format JSON) akan dipetakan ke properti code, status, dan data di dalam objek Registrasi.

3. wisata.dart

```
class Wisata {
  int? id;
  String? destination;
  String? location;
  String? attraction;
  Wisata({this.id, this.destination, this.location, this.attraction});
  factory Wisata.fromJson(Map<String, dynamic> obj) {
    return Wisata(
        id: obj['id'],
        destination: obj['destination'],
        location: obj['location'],
        attraction: obj['attraction']);
  }
}
```

Kelas Wisata digunakan untuk memodelkan objek wisata, yang terdiri dari berbagai atribut seperti id, destination (tujuan wisata), location (lokasi), dan attraction (daya tarik wisata). Berikut penjelasan dari setiap bagian:

Properti:

1. int? id:
    - Menyimpan ID dari destinasi wisata, bersifat opsional (nullable).
2. String? destination:
    - Menyimpan nama atau tujuan destinasi wisata, juga bersifat opsional.
3. String? location:
    - Menyimpan lokasi dari destinasi wisata (misalnya nama kota atau alamat), bersifat opsional.
4. String? attraction:
    - Menyimpan informasi tentang atraksi utama atau daya tarik wisata di lokasi tersebut, juga bersifat opsional.

Konstruktor:

- Wisata({this.id, this.destination, this.location, this.attraction}):
    - Konstruktor ini digunakan untuk menginisialisasi objek Wisata dengan semua properti yang bersifat opsional (nullable).

factory Wisata.fromJson(Map<String, dynamic> obj):

- Konstruktor factory ini digunakan untuk membuat objek Wisata dari respons JSON yang diperoleh dari API atau data sumber lainnya.
- Logika di dalam fungsi ini:
    - Data yang diterima dalam bentuk JSON dipetakan ke properti id, destination, location, dan attraction di objek Wisata.

Bloc

1. login_bloc.dart

```dart
import 'dart:convert';
import 'package:manajemen_pariwasata/helpers/api.dart';
import 'package:manajemen_pariwasata/helpers/api_url.dart';
import 'package:manajemen_pariwasata/model/login.dart';

class LoginBloc {
  static Future<Login> login({String? email, String? password}) async {
    String apiUrl = ApiUrl.login;
    var body = {"email": email, "password": password};
    var response = await Api().post(apiUrl, body);
    var jsonObj = json.decode(response.body);
    return Login.fromJson(jsonObj);
  }
}
```

Kode LoginBloc ini digunakan untuk menangani proses login dengan mengirimkan data email dan password ke API, serta mengelola responsnya. Berikut penjelasan singkat setiap bagian dari kode:

Kelas LoginBloc:

- Kelas ini bertindak sebagai Business Logic Component (BLoC) yang mengelola proses login.

Metode login:

- Tipe: Future<Login> menandakan metode ini mengembalikan Future yang akan berisi objek Login setelah respons dari server diterima dan diproses.
- Parameter:
    - String? email: Email pengguna yang akan digunakan untuk login.
    - String? password: Password yang dimasukkan untuk login.
- Proses yang dilakukan dalam metode ini:

  1. String apiUrl = ApiUrl.login: Mendapatkan URL endpoint API untuk login dari kelas ApiUrl.
  2. var body = {"email": email, "password": password};: Membuat objek body yang berisi email dan password dalam bentuk map.
  3. var response = await Api().post(apiUrl, body);: Mengirimkan request POST ke API dengan data login, menggunakan metode post dari kelas Api.
  4. var jsonObj = json.decode(response.body);: Menguraikan respons dari API yang berupa JSON ke dalam format Map.
  5. return Login.fromJson(jsonObj);: Mengubah objek Map hasil dari JSON tersebut menjadi objek Login yang kemudian dikembalikan sebagai hasil dari proses login.

  Alur Kerja:
- Saat metode login dipanggil, data login dikirim ke server melalui API, kemudian respons diterima, diubah dari format JSON menjadi objek Login, dan hasilnya dikembalikan.

2. registrasi_bloc.dart

```dart
import 'dart:convert';
import '/helpers/api.dart';
import '/helpers/api_url.dart';
import '/model/registrasi.dart';

class RegistrasiBloc {
  static Future<Registrasi> registrasi(
      {String? nama, String? email, String? password}) async {
    String apiUrl = ApiUrl.registrasi;

    var body = {"nama": nama, "email": email, "password": password};

    var response = await Api().post(apiUrl, body);
    var jsonObj = json.decode(response.body);
    return Registrasi.fromJson(jsonObj);
  }
}
```

Kode RegistrasiBloc ini bertanggung jawab untuk menangani proses registrasi pengguna baru. Berikut adalah penjelasan singkat setiap bagian dari kode:

Kelas RegistrasiBloc:

- Kelas ini mengelola logika bisnis untuk proses registrasi dengan mengirimkan data nama, email, dan password ke API.

Metode registrasi:

- Tipe: Future<Registrasi> berarti metode ini mengembalikan Future yang berisi objek Registrasi setelah respons dari server diproses.
- Parameter:
    - String? nama: Nama pengguna yang didaftarkan.
    - String? email: Email yang akan didaftarkan.
    - String? password: Password yang akan digunakan oleh pengguna untuk login.
- Langkah-langkah dalam metode ini:

  1. String apiUrl = ApiUrl.registrasi;: Mendapatkan URL endpoint API untuk registrasi dari kelas ApiUrl.
  2. var body = {"nama": nama, "email": email, "password": password};: Membuat body yang berisi data registrasi pengguna (nama, email, password).
  3. var response = await Api().post(apiUrl, body);: Mengirimkan request POST ke server dengan data registrasi menggunakan metode post dari kelas Api.
  4. var jsonObj = json.decode(response.body);: Mengubah respons yang berupa JSON menjadi objek Map.
  5. return Registrasi.fromJson(jsonObj);: Mengonversi Map JSON tersebut menjadi objek Registrasi yang akan dikembalikan sebagai hasil dari proses registrasi.

  Alur Kerja:
- Ketika metode registrasi dipanggil, data registrasi (nama, email, password) dikirim ke server API.
- Server mengembalikan respons dalam bentuk JSON.
- JSON respons diubah menjadi objek Registrasi, yang kemudian diteruskan ke komponen aplikasi yang memanggil fungsi ini.

3. wisata_bloc.dart

```dart
import 'dart:convert';
import 'package:manajemen_pariwasata/helpers/api.dart';
import 'package:manajemen_pariwasata/helpers/api_url.dart';
import 'package:manajemen_pariwasata/model/wisata.dart';
class WisataBloc {
  static Future<List<Wisata>> getWisata() async {
    String apiUrl = ApiUrl.listWisata;
    var response = await Api().get(apiUrl);
    var jsonObj = json.decode(response.body);
    List<dynamic> listWisata = (jsonObj as Map<String, dynamic>)['data'];
    List<Wisata> wisata = [];
    for (int i = 0; i < listWisata.length; i++) {
      wisata.add(Wisata.fromJson(listWisata[i]));
    }
    return wisata;
  }

  static Future addWisata({Wisata? wisata}) async {
    String apiUrl = ApiUrl.createWisata;
    var body = {
      "destination": wisata!.destination,
```

```
      "location": wisata.location,
      "attraction": wisata.attraction,
    };
    var response = await Api().post(apiUrl, body);
    var jsonObj = json.decode(response.body);
    return jsonObj['status'];
  }

  static Future updateWisata({required Wisata wisata}) async {
    String apiUrl = ApiUrl.updateWisata(wisata.id!);
    print(apiUrl);
    var body = {
      "destination": wisata.destination,
      "location": wisata.location,
      "attraction": wisata.attraction,
    };
    print("Body : $body");
    var response = await Api().put(apiUrl, jsonEncode(body));
    var jsonObj = json.decode(response.body);
    return jsonObj['status'];
  }

  static Future<bool> deleteWisata({int? id}) async {
    String apiUrl = ApiUrl.deleteWisata(id!);
    var response = await Api().delete(apiUrl);

    if (response != null && response.body.isNotEmpty) {
      var jsonObj = json.decode(response.body);
      return jsonObj['status'] == true;
    } else {
      return false;
    }
  }
}
```

Berikut penjelasan mengenai kode WisataBloc yang bertanggung jawab untuk menangani logika terkait pengelolaan data objek Wisata dari aplikasi:

Kelas WisataBloc:

Kelas ini digunakan untuk mengelola komunikasi antara aplikasi dan server API untuk operasi CRUD (Create, Read, Update, Delete) pada data wisata.

Metode dalam WisataBloc:

1. getWisata():
    - Tipe: Future<List<Wisata>> mengembalikan daftar objek wisata dalam bentuk list.

- Langkah-langkah:
    - Mengambil URL dari ApiUrl.listWisata yang berisi endpoint untuk mengambil daftar wisata.
    - Mengirim request GET menggunakan metode Api().get().
    - Mengubah respons JSON menjadi objek Map.
    - Memproses data dalam jsonObj['data'], yang berisi daftar wisata, menjadi list Wisata.
- Hasil: Mengembalikan list objek wisata.

2. addWisata():
    - Tipe: Future yang akan menambahkan data wisata baru ke API.
    - Parameter: Objek Wisata yang berisi informasi tujuan, lokasi, dan atraksi wisata.
    - Langkah-langkah:
        - Membuat body dari objek Wisata yang berisi data destination, location, dan attraction.
        - Mengirim request POST ke API menggunakan Api().post().
        - Mengubah respons JSON menjadi objek Map dan mengembalikan status hasil dari operasi tersebut.
    - Hasil: Mengembalikan status operasi apakah berhasil atau tidak.
3. updateWisata():
    - Tipe: Future yang akan mengupdate data wisata berdasarkan ID yang diberikan.
    - Parameter: Objek Wisata yang akan di-update.
    - Langkah-langkah:
        - Membuat URL dari ApiUrl.updateWisata(wisata.id!) yang berisi endpoint update sesuai ID.
        - Membuat body yang berisi data yang akan di-update, seperti destination, location, dan attraction.
        - Mengirim request PUT ke API menggunakan Api().put().
        - Mengubah respons JSON menjadi objek Map dan mengembalikan status operasi.
    - Hasil: Mengembalikan status operasi apakah berhasil atau tidak.
4. deleteWisata():
    - Tipe: Future<bool> mengembalikan boolean untuk mengetahui apakah operasi penghapusan berhasil atau gagal.
    - Parameter: ID wisata yang akan dihapus.
    - Langkah-langkah:
        - Membuat URL dari ApiUrl.deleteWisata(id!) sesuai dengan ID yang diberikan.
        - Mengirim request DELETE ke API menggunakan Api().delete().

        - Mengubah respons JSON menjadi objek Map dan mengembalikan status operasi.
    - Hasil: Mengembalikan nilai true jika berhasil dihapus, false jika gagal.

Alur Kerja:

- getWisata() mengambil daftar wisata dari server.
- addWisata() menambahkan wisata baru dengan mengirim data ke server.
- updateWisata() memperbarui data wisata yang ada berdasarkan ID-nya.
- deleteWisata() menghapus wisata berdasarkan ID-nya.

4. Logout_bloc.dart

```dart
import 'package:manajemen_pariwasata/helpers/user_info.dart';

class LogoutBloc {
  static Future logout() async {
    await UserInfo().logout();
  }
}
```

Berikut adalah penjelasan singkat mengenai kelas LogoutBloc yang digunakan untuk mengelola logika logout pengguna dalam aplikasi:

Kelas LogoutBloc:

Kelas ini bertanggung jawab untuk menangani proses logout pengguna dengan menghapus token dan informasi pengguna yang tersimpan di penyimpanan lokal.

Metode dalam LogoutBloc:

1. logout():
    - Tipe: Future yang akan melakukan proses logout.
    - Langkah-langkah:
        - Memanggil metode logout() dari kelas UserInfo untuk membersihkan semua data pengguna yang tersimpan, termasuk token autentikasi.
    - Hasil: Proses logout tidak mengembalikan nilai, tetapi efeknya adalah menghapus informasi pengguna dari penyimpanan lokal.

Alur Kerja:

- Saat metode logout() dipanggil, ia akan menghapus semua data pengguna yang tersimpan di SharedPreferences, sehingga pengguna akan dianggap tidak terautentikasi dan harus melakukan login kembali untuk mengakses aplikasi.

UI

1. Registrasi_page.dart

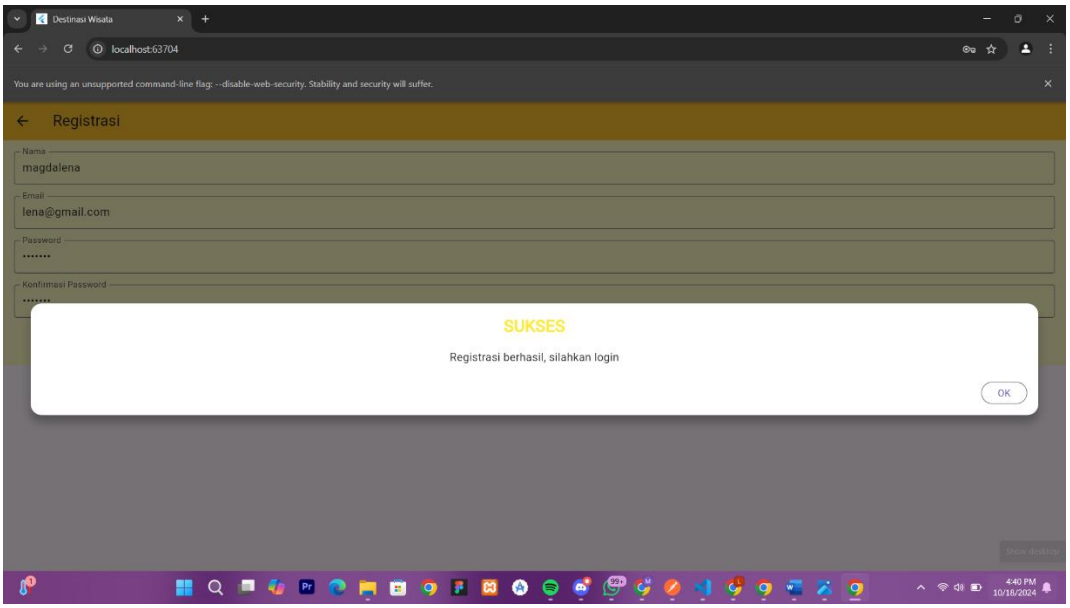
```dart
import 'package:flutter/material.dart';
import '../bloc/registrasi_bloc.dart';
import '../widget/success_dialog.dart';
import '../widget/warning_dialog.dart';

class RegistrasiPage extends StatefulWidget {
  const RegistrasiPage({Key? key}) : super(key: key);

  @override
  _RegistrasiPageState createState() => _RegistrasiPageState();
}

class _RegistrasiPageState extends State<RegistrasiPage> {
  final _formKey = GlobalKey<FormState>();
  bool _isLoading = false;
  final _namaTextboxController = TextEditingController();
  final _emailTextboxController = TextEditingController();
  final _passwordTextboxController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Registrasi"),
        backgroundColor: Colors.yellow[700], // Warna kuning terang untuk AppBar
      ),
      body: Container(
        color: Colors.yellow[100], // Latar belakang kuning terang
        padding: const EdgeInsets.all(16.0),
        child: SingleChildScrollView(
          child: Form(
            key: _formKey,
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                _namaTextField(),
                const SizedBox(height: 16.0),
                _emailTextField(),
                const SizedBox(height: 16.0),
                _passwordTextField(),
                const SizedBox(height: 16.0),
                _passwordKonfirmasiTextField(),
                const SizedBox(height: 20.0),
                _buttonRegistrasi(),
              ],
```

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

 // Membuat Textbox Nama
 Widget _namaTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Nama",
      border: OutlineInputBorder(), // Border sederhana
    ),
    keyboardType: TextInputType.text,
    controller: _namaTextboxController,
    validator: (value) {
      if (value!.length < 3) {
        return "Nama harus diisi minimal 3 karakter";
      }
      return null;
    },
  );
 }

 // Membuat Textbox email
 Widget _emailTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Email",
      border: OutlineInputBorder(), // Border sederhana
    ),
    keyboardType: TextInputType.emailAddress,
    controller: _emailTextboxController,
    validator: (value) {
      // Validasi harus diisi
      if (value!.isEmpty) {
        return 'Email harus diisi';
      }
      // Validasi email
      Pattern pattern =
          r'^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-
9]+\.)+[a-zA-Z]{2,}))$';
      RegExp regex = RegExp(pattern.toString());
      if (!regex.hasMatch(value)) {
```

```
        return "Email tidak valid";
      }
      return null;
    },
  );
}

// Membuat Textbox password
Widget _passwordTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Password",
      border: OutlineInputBorder(), // Border sederhana
    ),
    keyboardType: TextInputType.text,
    obscureText: true,
    controller: _passwordTextboxController,
    validator: (value) {
      // Jika karakter yang dimasukkan kurang dari 6 karakter
      if (value!.length < 6) {
        return "Password harus diisi minimal 6 karakter";
      }
      return null;
    },
  );
}

// Membuat textbox Konfirmasi Password
Widget _passwordKonfirmasiTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Konfirmasi Password",
      border: OutlineInputBorder(), // Border sederhana
    ),
    keyboardType: TextInputType.text,
    obscureText: true,
    validator: (value) {
      // Jika inputan tidak sama dengan password
      if (value != _passwordTextboxController.text) {
        return "Konfirmasi Password tidak sama";
      }
      return null;
    },
  );
}
```

```
// Membuat Tombol Registrasi
Widget _buttonRegistrasi() {
  return ElevatedButton(
    style: ElevatedButton.styleFrom(
      backgroundColor: Colors.yellow[700], // Warna tombol kuning terang
    ),
    child: const Text("Registrasi", style: TextStyle(color: Colors.black)), // Teks hitam
    onPressed: () {
      var validate = _formKey.currentState!.validate();
      if (validate) {
        if (!_isLoading) _submit();
      }
    },
  );
}

void _submit() {
  _formKey.currentState!.save();
  setState(() {
    _isLoading = true;
  });
  RegistrasiBloc.registrasi(
    nama: _namaTextboxController.text,
    email: _emailTextboxController.text,
    password: _passwordTextboxController.text,
  ).then((value) {
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) => SuccessDialog(
        description: "Registrasi berhasil, silahkan login",
        okClick: () {
          Navigator.pop(context);
        },
      ),
    );
  }, onError: (error) {
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (BuildContext context) => const WarningDialog(
        description: "Registrasi gagal, silahkan coba lagi",
      ),
    );
```

```
  }).whenComplete(() {
    setState(() {
      _isLoading = false;
    });
  });
 }
}
```



Berikut adalah penjelasan singkat tentang kode RegistrasiPage yang merupakan bagian dari aplikasi Flutter untuk mengelola proses pendaftaran pengguna.

Kelas RegistrasiPage

RegistrasiPage adalah widget berbasis StatefulWidget yang menyediakan antarmuka pengguna untuk mendaftar (registrasi) pengguna baru.

Struktur Kelas:

1. State Management:
    - Menggunakan GlobalKey<FormState> untuk mengelola status form.
    - Menggunakan TextEditingController untuk mengontrol input dari pengguna (nama, email, password).
2. UI:
    - Terdapat Scaffold dengan AppBar dan Container yang mengatur tampilan latar belakang dan padding.
    - Menggunakan SingleChildScrollView untuk mendukung scroll jika konten melebihi layar.
    - Mengandung beberapa TextFormField untuk input nama, email, password, dan konfirmasi password.

3. Validasi Input:
    - Setiap TextFormField dilengkapi dengan validator untuk memastikan input valid sebelum proses pendaftaran.
4. Tombol Registrasi:
    - Terdapat tombol untuk memicu proses pendaftaran. Jika semua input valid, akan memanggil metode _submit().

Metode Kelas:

1. _namaTextField():
    - Mengembalikan widget TextFormField untuk input nama dengan validasi minimal 3 karakter.
2. _emailTextField():
    - Mengembalikan widget TextFormField untuk input email dengan validasi format email.
3. _passwordTextField():
    - Mengembalikan widget TextFormField untuk input password dengan validasi minimal 6 karakter.
4. _passwordKonfirmasiTextField():
    - Mengembalikan widget TextFormField untuk konfirmasi password dan memastikan kecocokan dengan input password.
5. _buttonRegistrasi():
    - Mengembalikan widget ElevatedButton untuk memicu proses pendaftaran.
6. _submit():
    - Mengelola pengiriman data registrasi.
    - Menggunakan RegistrasiBloc untuk melakukan pendaftaran dengan memanggil API.
    - Menampilkan dialog sukses atau gagal tergantung pada hasil pendaftaran.
    - Mengelola status loading menggunakan setState().

Penanganan Respons:

- Jika pendaftaran berhasil, akan menampilkan dialog sukses dengan pesan bahwa registrasi berhasil.
- Jika gagal, akan menampilkan dialog peringatan dengan pesan bahwa registrasi gagal.

Tampilan:

- Antarmuka didominasi dengan warna kuning terang, memberikan kesan ceria dan menarik.

2. Login_page.dart

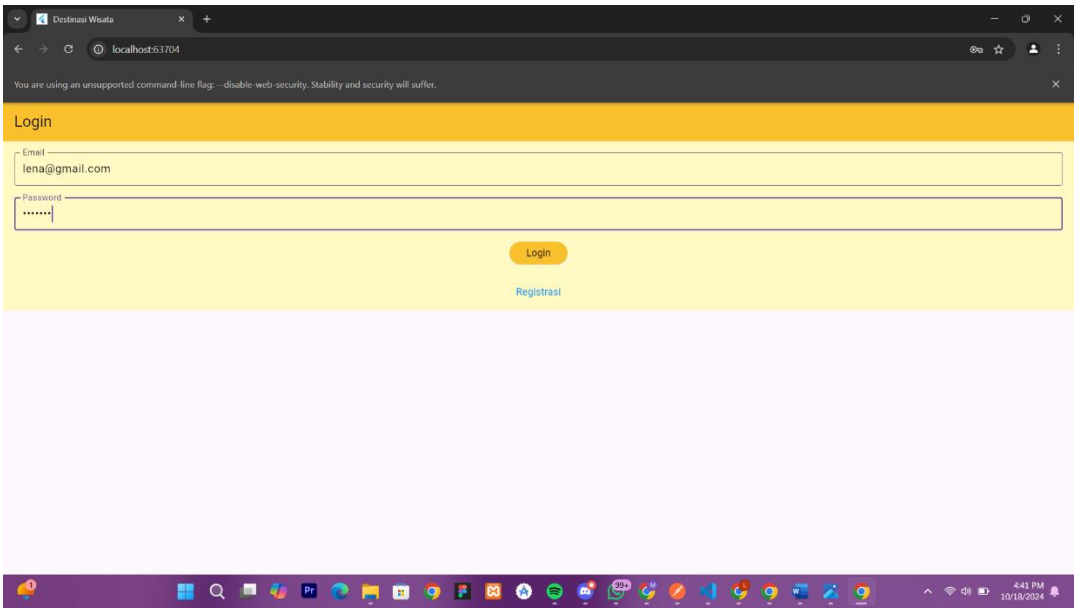
```
import 'package:flutter/material.dart';
import '../bloc/login_bloc.dart';
import '../helpers/user_info.dart';
import '../widget/warning_dialog.dart';
```

```dart
import '/ui/registrasi_page.dart';
import 'wisata_page.dart';

class LoginPage extends StatefulWidget {
  const LoginPage({Key? key}) : super(key: key);

  @override
  _LoginPageState createState() => _LoginPageState();
}

class _LoginPageState extends State<LoginPage> {
  final _formKey = GlobalKey<FormState>();
  bool _isLoading = false;
  final _emailTextboxController = TextEditingController();
  final _passwordTextboxController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Login'),
        backgroundColor: Colors.yellow[700], // Warna kuning terang untuk AppBar
      ),
      body: Container(
        color: Colors.yellow[100], // Latar belakang kuning terang
        padding: const EdgeInsets.all(16.0),
        child: SingleChildScrollView(
          child: Form(
            key: _formKey,
            child: Column(
              children: [
                _emailTextField(),
                const SizedBox(height: 16.0),
                _passwordTextField(),
                const SizedBox(height: 16.0),
                _buttonLogin(),
                const SizedBox(height: 30),
                _menuRegistrasi(),
              ],
            ),
          ),
        ),
      ),
    );
  }
```

```
// Membuat Textbox email
Widget _emailTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Email",
      border: OutlineInputBorder(), // Border sederhana
    ),
    keyboardType: TextInputType.emailAddress,
    controller: _emailTextboxController,
    validator: (value) {
      // Validasi harus diisi
      if (value == null || value.isEmpty) {
        return 'Email harus diisi';
      }
      return null;
    },
  );
}

// Membuat Textbox password
Widget _passwordTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Password",
      border: OutlineInputBorder(), // Border sederhana
    ),
    keyboardType: TextInputType.text,
    obscureText: true,
    controller: _passwordTextboxController,
    validator: (value) {
      // Validasi harus diisi
      if (value == null || value.isEmpty) {
        return "Password harus diisi";
      }
      return null;
    },
  );
}

// Membuat Tombol Login
Widget _buttonLogin() {
  return ElevatedButton(
    style: ElevatedButton.styleFrom(
      backgroundColor: Colors.yellow[700], // Warna tombol kuning terang
```

```
    ),
    child: const Text("Login", style: TextStyle(color: Colors.black)), // Teks hitam
    onPressed: () {
      var validate = _formKey.currentState?.validate();
      if (validate == true && !_isLoading) {
        _submit();
      }
    },
  );
}

void _submit() {
  _formKey.currentState!.save();
  setState(() {
    _isLoading = true;
  });

  LoginBloc.login(
    email: _emailTextboxController.text,
    password: _passwordTextboxController.text,
  ).then((value) async {
    if (value.code == 200) {
      await UserInfo().setToken(value.token!);
      await UserInfo().setUserID(value.userID!);
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => const WisataPage()),
      );
    } else {
      _showErrorDialog("Login gagal, silahkan coba lagi");
    }
  }).catchError((error) {
    _showErrorDialog("Login gagal, silahkan coba lagi");
  }).whenComplete(() {
    setState(() {
      _isLoading = false;
    });
  });
}

void _showErrorDialog(String message) {
  showDialog(
    context: context,
    barrierDismissible: false,
    builder: (BuildContext context) => WarningDialog(
```

```
        description: message,
      ),
    );
  }

  // Membuat menu untuk membuka halaman registrasi
  Widget _menuRegistrasi() {
    return Center(
      child: InkWell(
        child: const Text(
          "Registrasi",
          style: TextStyle(color: Colors.blue),
        ),
        onTap: () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => const RegistrasiPage()),
          );
        },
      ),
    );
  }
}
```



Berikut adalah penjelasan singkat mengenai kode LoginPage, yang merupakan bagian dari aplikasi Flutter untuk mengelola proses login pengguna.

Kelas LoginPage

LoginPage adalah widget berbasis StatefulWidget yang menyediakan antarmuka pengguna untuk login ke dalam aplikasi.

Struktur Kelas:

1. State Management:
    - Menggunakan GlobalKey<FormState> untuk mengelola status form.
    - Menggunakan TextEditingController untuk mengontrol input dari pengguna (email dan password).
2. UI:
    - Terdapat Scaffold dengan AppBar dan Container yang mengatur tampilan latar belakang dan padding.
    - Menggunakan SingleChildScrollView untuk mendukung scroll jika konten melebihi layar.
    - Mengandung dua TextFormField untuk input email dan password, serta tombol login dan menu untuk registrasi.
3. Validasi Input:
    - Setiap TextFormField dilengkapi dengan validator untuk memastikan input valid sebelum proses login.
4. Tombol Login:
    - Terdapat tombol yang akan memicu proses login. Jika semua input valid, akan memanggil metode _submit().

Metode Kelas:

1. _emailTextField():
    - Mengembalikan widget TextFormField untuk input email dengan validasi bahwa email harus diisi.
2. _passwordTextField():
    - Mengembalikan widget TextFormField untuk input password dengan validasi bahwa password harus diisi.
3. _buttonLogin():
    - Mengembalikan widget ElevatedButton untuk memicu proses login.
4. _submit():
    - Mengelola pengiriman data login.
    - Menggunakan LoginBloc untuk melakukan login dengan memanggil API.
    - Jika login berhasil (kode 200), menyimpan token dan user ID menggunakan UserInfo dan mengarahkan pengguna ke halaman WisataPage.
    - Jika gagal, memanggil metode _showErrorDialog() untuk menampilkan dialog peringatan.
5. _showErrorDialog(String message):
    - Menampilkan dialog peringatan jika login gagal.
6. _menuRegistrasi():
    - Mengembalikan widget InkWell yang akan mengarahkan pengguna ke halaman registrasi saat diklik.

Penanganan Respons:

- Jika login berhasil, pengguna diarahkan ke halaman WisataPage.
- Jika login gagal, akan menampilkan dialog peringatan dengan pesan bahwa login gagal.

Tampilan:

- Antarmuka didominasi dengan warna kuning terang, memberikan kesan ceria dan menarik.

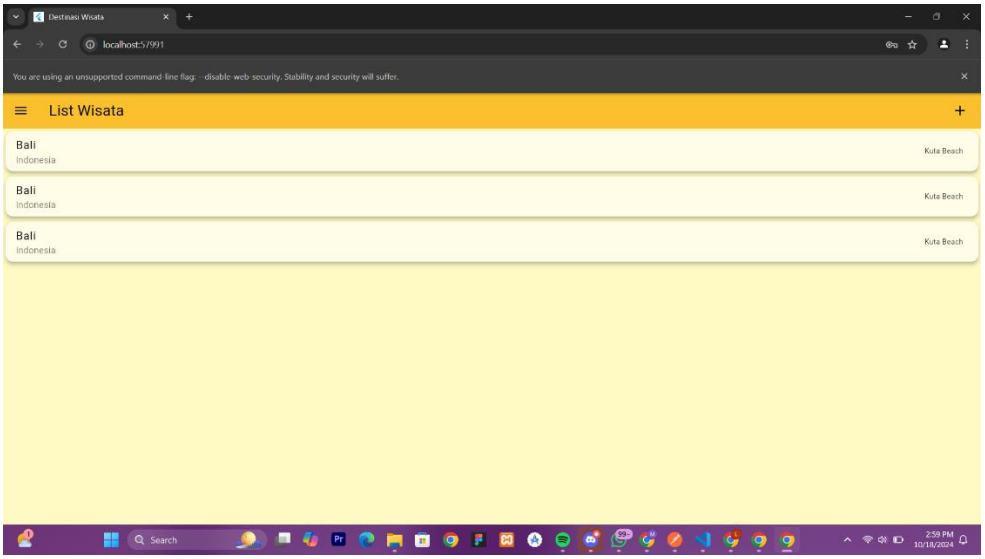
3. Wisata_page.dart

```dart
import 'package:flutter/material.dart';
import '../bloc/logout_bloc.dart';
import '../bloc/wisata_bloc.dart';
import '/model/wisata.dart';
import '/ui/wisata_detail.dart';
import '/ui/wisata_form.dart';
import 'login_page.dart';
```

```dart
class WisataPage extends StatefulWidget {
  const WisataPage({Key? key}) : super(key: key);

  @override
  _WisataPageState createState() => _WisataPageState();
}

class _WisataPageState extends State<WisataPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.yellow[700], // Bright yellow for the app bar
        title: const Text(
          'List Wisata',
          style: TextStyle(
            fontFamily: 'Helvetica',
            fontSize: 24.0,
            color: Colors.black, // Text color black for contrast
          ),
        ),
        actions: [
          Padding(
            padding: const EdgeInsets.only(right: 20.0),
            child: GestureDetector(
              child: const Icon(Icons.add, size: 26.0, color: Colors.black), // Black icon for
contrast
              onTap: () async {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => WisataForm()),
                );
              },
            ),
          ),
        ],
      ),
      drawer: Drawer(
        child: ListView(
          children: [
            ListTile(
              title: const Text('Logout'),
              trailing: const Icon(Icons.logout),
              onTap: () async {
                await LogoutBloc.logout().then((value) => {
```

```
                    Navigator.of(context).pushAndRemoveUntil(
                      MaterialPageRoute(builder: (context) => LoginPage()),
                      (route) => false,
                    );
                  });
            },
          ),
        ],
      ),
    ),
    body: Container(
      color: Colors.yellow[100], // Light yellow background for the entire body
      child: FutureBuilder<List>(
        future: WisataBloc.getWisata(),
        builder: (context, snapshot) {
          if (snapshot.hasError) print(snapshot.error);
          return snapshot.hasData
              ? ListWisata(
                  list: snapshot.data,
                )
              : const Center(
                  child: CircularProgressIndicator(),
                );
        },
      ),
    ),
  );
 }
}

class ListWisata extends StatelessWidget {
  final List? list;

  const ListWisata({Key? key, this.list}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return ListView.builder(
      itemCount: list == null ? 0 : list!.length,
      itemBuilder: (context, i) {
        return ItemWisata(
          wisata: list![i],
        );
      },
    );
```

```dart
  }
}

class ItemWisata extends StatelessWidget {
  final Wisata wisata;

  const ItemWisata({Key? key, required this.wisata}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => WisataDetail(wisata: wisata),
          ),
        );
      },
      child: Card(
        color: Colors.yellow[50], // Light yellow card background
        elevation: 5.0, // Slight elevation for shadow effect
        child: ListTile(
          title: Text(
            wisata.destination!,
            style: const TextStyle(
              fontFamily: 'Helvetica',
              fontSize: 18.0,
              color: Colors.black, // Black text color for contrast
            ),
          ),
          subtitle: Text(
            wisata.location!,
            style: const TextStyle(color: Colors.black54), // Subtle black for subtitle
          ),
          trailing: Text(
            wisata.attraction!,
            style: const TextStyle(color: Colors.black87), // Darker black for attraction text
          ),
        ),
      ),
    );
  }
}
```



Berikut adalah penjelasan singkat mengenai kode WisataPage, yang merupakan bagian dari aplikasi Flutter untuk mengelola daftar wisata.

Kelas WisataPage

WisataPage adalah widget berbasis StatefulWidget yang menyediakan antarmuka pengguna untuk menampilkan daftar wisata.

Struktur Kelas:

1. State Management:
    - Menggunakan State untuk mengelola status halaman.
2. UI:
    - Terdapat Scaffold dengan AppBar, Drawer, dan Container untuk tampilan.
    - AppBar berisi judul dan tombol untuk menambahkan wisata baru.
    - Drawer berisi opsi untuk logout.
3. Daftar Wisata:
    - Menggunakan FutureBuilder untuk mengambil data wisata dari API dengan memanggil WisataBloc.getWisata().
    - Jika data tersedia, akan menampilkan daftar wisata menggunakan widget ListWisata.
    - Jika ada kesalahan saat mengambil data, kesalahan tersebut dicetak ke konsol.
    - Menampilkan CircularProgressIndicator saat data masih dimuat.

Metode Kelas:

1. build(BuildContext context):
    - Mengatur tampilan umum halaman, termasuk AppBar, Drawer, dan konten utama.
    - Menampilkan daftar wisata menggunakan FutureBuilder.
2. ListWisata:
    - Widget ini menerima daftar wisata dan membuat ListView menggunakan ListView.builder.
    - Untuk setiap item dalam daftar, widget ItemWisata dibuat.
3. ItemWisata:
    - Widget ini mewakili satu item wisata dalam daftar.
    - Menggunakan GestureDetector untuk menangani klik pada item, yang akan membuka detail wisata di halaman WisataDetail.
    - Menggunakan Card untuk memberikan tampilan yang menarik dengan sedikit bayangan.
    - ListTile menampilkan nama tujuan, lokasi, dan atraksi wisata.

Penanganan Respons:

- Ketika pengguna menekan item daftar wisata, aplikasi akan mengarahkan pengguna ke halaman detail wisata menggunakan Navigator.push.

Desain UI:

- Menggunakan warna kuning cerah untuk latar belakang dan elemen UI untuk memberikan tampilan yang ceria dan menarik.
- Memastikan kontras yang baik antara teks dan latar belakang agar mudah dibaca.

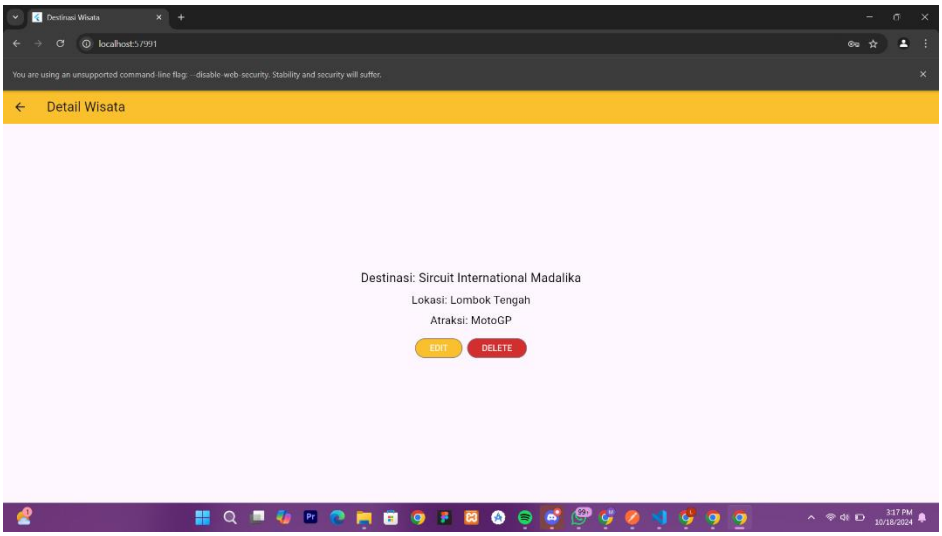
4. Wisata_detail.dart

```dart
import 'package:flutter/material.dart';
import '../bloc/wisata_bloc.dart';
import '../widget/warning_dialog.dart';
import '/model/wisata.dart';
import 'wisata_form.dart';
import 'wisata_page.dart';

// ignore: must_be_immutable
class WisataDetail extends StatefulWidget {
  Wisata? wisata;

  WisataDetail({Key? key, this.wisata}) : super(key: key);

  @override
  _WisataDetailState createState() => _WisataDetailState();
}

class _WisataDetailState extends State<WisataDetail> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Detail Wisata'),
        backgroundColor: Colors.yellow[700], // Warna kuning terang untuk AppBar
      ),
      body: Center(
        child: Padding(
          padding: const EdgeInsets.all(
              16.0), // Padding untuk tata letak yang lebih rapi
          child: Column(
            mainAxisAlignment:
                MainAxisAlignment.center, // Posisi di tengah layar
            children: [
              Text(
                "Destinasi: ${widget.wisata!.destination}",
                style: const TextStyle(
                  fontSize: 20.0,
                  color: Colors.black, // Warna teks hitam
                ),
```

```
            ),
            const SizedBox(height: 8.0), // Jarak antar elemen
            Text(
              "Lokasi: ${widget.wisata!.location}",
              style: const TextStyle(
                fontSize: 18.0,
                color: Colors.black,
              ),
            ),
            const SizedBox(height: 8.0),
            Text(
              "Atraksi: ${widget.wisata!.attraction}",
              style: const TextStyle(
                fontSize: 18.0,
                color: Colors.black,
              ),
            ),
            const SizedBox(height: 16.0), // Jarak lebih besar untuk tombol
            _tombolHapusEdit(),
          ],
        ),
      ),
    ),
  );
}

Widget _tombolHapusEdit() {
  return Row(
    mainAxisSize: MainAxisSize.min,
    children: [
      // Tombol Edit
      OutlinedButton(
        style: OutlinedButton.styleFrom(
          backgroundColor: Colors.yellow[700], // Warna kuning terang
        ),
        child: const Text(
          "EDIT",
          style: TextStyle(color: Colors.white), // Teks berwarna putih
        ),
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => WisataForm(
                wisata: widget.wisata!,
```

```
              ),
            ),
          );
        },
      ),
      const SizedBox(width: 8.0), // Jarak antara tombol
      // Tombol Hapus
      OutlinedButton(
        style: OutlinedButton.styleFrom(
          backgroundColor: Colors.red[700], // Warna merah untuk tombol hapus
        ),
        child: const Text(
          "DELETE",
          style: TextStyle(color: Colors.white), // Teks berwarna putih
        ),
        onPressed: () => confirmHapus(),
      ),
    ],
  );
}

void confirmHapus() {
  AlertDialog alertDialog = AlertDialog(
    content: const Text("Yakin ingin menghapus data ini?"),
    actions: [
      // tombol hapus
      OutlinedButton(
        style: OutlinedButton.styleFrom(
          backgroundColor:
              Colors.red[700], // Warna merah untuk konfirmasi hapus
        ),
        child: const Text(
          "Ya",
          style: TextStyle(color: Colors.white), // Teks berwarna putih
        ),
        onPressed: () {
          WisataBloc.deleteWisata(id: widget.wisata!.id!).then(
              (value) => {
                    Navigator.of(context).push(
                      MaterialPageRoute(
                          builder: (context) => const WisataPage()),
                    )
                  }, onError: (error) {
            showDialog(
                context: context,
```

```
                builder: (BuildContext context) => const WarningDialog(
                      description: "Hapus gagal, silahkan coba lagi",
                    ));
            });
          },
        ),
        // tombol batal
        OutlinedButton(
          style: OutlinedButton.styleFrom(
            backgroundColor: Colors.grey, // Warna abu-abu untuk batal
          ),
          child: const Text(
            "Batal",
            style: TextStyle(color: Colors.white), // Teks berwarna putih
          ),
          onPressed: () => Navigator.pop(context),
        )
      ],
    );

    showDialog(builder: (context) => alertDialog, context: context);
  }
}
```



Berikut adalah penjelasan mengenai kelas WisataDetail dalam aplikasi Flutter yang digunakan untuk menampilkan detail dari sebuah objek wisata. Kelas ini juga menyediakan opsi untuk mengedit dan menghapus data wisata.

Kelas WisataDetail

WisataDetail adalah widget berbasis StatefulWidget yang menampilkan informasi detail tentang sebuah wisata.

Struktur Kelas:

1. State Management:

- Menggunakan State untuk mengelola status halaman.

2. UI:
   - Terdapat Scaffold dengan AppBar dan Body untuk tampilan.
   - AppBar menampilkan judul "Detail Wisata".
   - Body berisi informasi detail wisata dan tombol untuk mengedit atau menghapus data.

Metode Kelas:

1. build(BuildContext context):
   - Mengatur tampilan umum halaman, termasuk menampilkan nama destinasi, lokasi, dan atraksi.
   - Menggunakan Padding untuk memberikan jarak yang rapi di sekitar elemen.
   - Menampilkan tombol edit dan delete menggunakan metode _tombolHapusEdit().
2. _tombolHapusEdit():
   - Mengembalikan Row yang berisi dua tombol: Edit dan Delete.
   - Tombol Edit: Mengarahkan pengguna ke halaman WisataForm untuk mengedit informasi wisata.
   - Tombol Delete: Memanggil metode confirmHapus() untuk mengonfirmasi tindakan penghapusan.
3. confirmHapus():
   - Menampilkan AlertDialog untuk meminta konfirmasi pengguna sebelum menghapus data wisata.
   - Jika pengguna mengonfirmasi, maka WisataBloc.deleteWisata() dipanggil untuk menghapus wisata dari database.
   - Menangani respons dengan menavigasi kembali ke halaman WisataPage jika berhasil, atau menampilkan WarningDialog jika terjadi kesalahan.

Desain UI:

- Menggunakan warna kuning cerah untuk tombol edit, dan merah untuk tombol hapus, untuk memberikan tampilan yang jelas dan intuitif.
- Menggunakan teks putih untuk kontras yang baik dengan latar belakang tombol.

5. Wisata_form.dart

```dart
import 'package:flutter/material.dart';
import '../bloc/wisata_bloc.dart';
import '../widget/warning_dialog.dart';
import '/model/wisata.dart';
import 'wisata_page.dart';

class WisataForm extends StatefulWidget {
  final Wisata? wisata;

  const WisataForm({Key? key, this.wisata}) : super(key: key);

  @override
  _WisataFormState createState() => _WisataFormState();
}

class _WisataFormState extends State<WisataForm> {
  final _formKey = GlobalKey<FormState>();
  bool _isLoading = false;
  String judul = "TAMBAH WISATA";
  String tombolSubmit = "SIMPAN";
  final _destinationTextboxController = TextEditingController();
  final _locationTextboxController = TextEditingController();
```

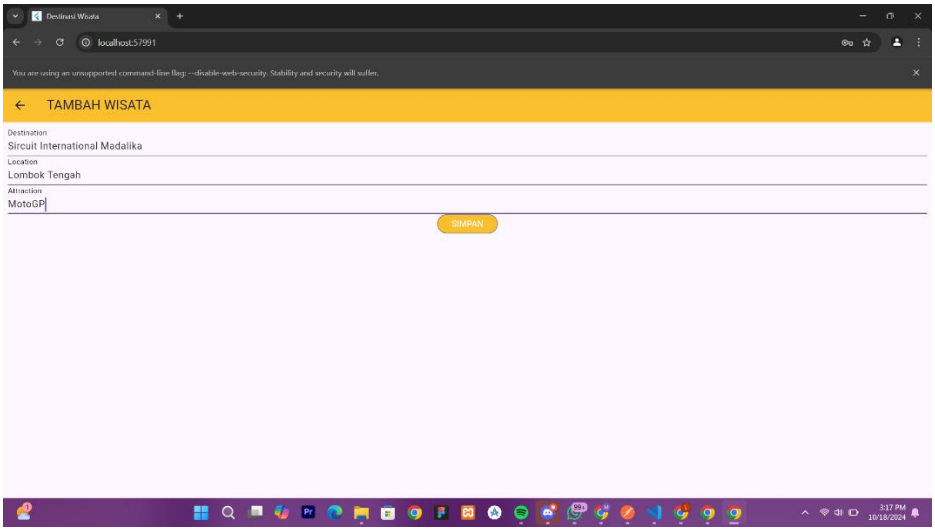
```dart
final _attractionTextboxController = TextEditingController();

@override
void initState() {
  super.initState();
  isUpdate();
}

void isUpdate() {
  if (widget.wisata != null) {
    setState(() {
      judul = "UBAH WISATA";
      tombolSubmit = "UBAH";
      _destinationTextboxController.text = widget.wisata!.destination ?? '';
      _locationTextboxController.text = widget.wisata!.location ?? '';
      _attractionTextboxController.text = widget.wisata!.attraction ?? '';
    });
  } else {
    judul = "TAMBAH WISATA";
    tombolSubmit = "SIMPAN";
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(judul),
      backgroundColor: Colors.yellow[700], // Warna kuning terang
    ),
    body: SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              _destinationTextField(),
              _locationTextField(),
              _attractionTextField(),
              _buttonSubmit(),
            ],
          ),
        ),
      ),
```

```
    ),
  );
}

// Membuat Textbox untuk Destination
Widget _destinationTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Destination",
      labelStyle: TextStyle(color: Colors.black), // Teks warna hitam
    ),
    keyboardType: TextInputType.text,
    controller: _destinationTextboxController,
    validator: (value) {
      if (value!.isEmpty) {
        return "Destination harus diisi";
      }
      return null;
    },
  );
}

// Membuat Textbox untuk Location
Widget _locationTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Location",
      labelStyle: TextStyle(color: Colors.black), // Teks warna hitam
    ),
    keyboardType: TextInputType.text,
    controller: _locationTextboxController,
    validator: (value) {
      if (value!.isEmpty) {
        return "Location harus diisi";
      }
      return null;
    },
  );
}

// Membuat Textbox untuk Attraction
Widget _attractionTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Attraction",
```

```
      labelStyle: TextStyle(color: Colors.black), // Teks warna hitam
    ),
    keyboardType: TextInputType.text,
    controller: _attractionTextboxController,
    validator: (value) {
      if (value!.isEmpty) {
        return "Attraction harus diisi";
      }
      return null;
    },
  );
}

// Membuat Tombol Simpan/Ubah
Widget _buttonSubmit() {
  return OutlinedButton(
    style: OutlinedButton.styleFrom(
      backgroundColor: Colors.yellow[700], // Warna kuning terang
    ),
    child: Text(
      tombolSubmit,
      style: const TextStyle(color: Colors.white),
    ),
    onPressed: () {
      var validate = _formKey.currentState!.validate();
      if (validate) {
        if (!_isLoading) {
          if (widget.wisata != null) {
            // Kondisi update wisata
            ubah();
          } else {
            // Kondisi tambah wisata
            simpan();
          }
        }
      }
    },
  );
}

void simpan() {
  setState(() {
    _isLoading = true;
  });
```

```
    Wisata createWisata = Wisata(
      id: null,
      destination: _destinationTextboxController.text,
      location: _locationTextboxController.text,
      attraction: _attractionTextboxController.text,
    );

    WisataBloc.addWisata(wisata: createWisata).then((value) {
      Navigator.of(context).push(
        MaterialPageRoute(
          builder: (BuildContext context) => const WisataPage(),
        ),
      );
    }, onError: (error) {
      showDialog(
        context: context,
        builder: (BuildContext context) => const WarningDialog(
          description: "Simpan gagal, silahkan coba lagi",
        ),
      );
    });

    setState(() {
      _isLoading = false;
    });
  }

  void ubah() {
    setState(() {
      _isLoading = true;
    });

    Wisata updateWisata = Wisata(
      id: widget.wisata!.id!,
      destination: _destinationTextboxController.text,
      location: _locationTextboxController.text,
      attraction: _attractionTextboxController.text,
    );

    WisataBloc.updateWisata(wisata: updateWisata).then((value) {
      Navigator.of(context).push(
        MaterialPageRoute(
          builder: (BuildContext context) => const WisataPage(),
        ),
      );
```

```
    }, onError: (error) {
      showDialog(
        context: context,
        builder: (BuildContext context) => const WarningDialog(
          description: "Permintaan ubah data gagal, silahkan coba lagi",
        ),
      );
    });

    setState(() {
      _isLoading = false;
    });
  }
}
```



Berikut adalah penjelasan tentang kelas WisataForm, yang digunakan dalam aplikasi Flutter untuk mengelola form penambahan atau pengeditan data wisata. Kelas ini memungkinkan pengguna untuk memasukkan informasi mengenai destinasi, lokasi, dan atraksi dari wisata tertentu.

Kelas WisataForm

WisataForm adalah widget berbasis StatefulWidget yang menangani form untuk menambah atau mengedit data wisata.

Struktur Kelas:

1. State Management:
    - Menggunakan State untuk mengelola status halaman dan kontrol form.
2. UI:
    - Memiliki Scaffold dengan AppBar yang menampilkan judul, dan body yang berisi form input.

- Form ini memungkinkan pengguna untuk mengisi tiga input: Destination, Location, dan Attraction.

Metode Kelas:

1. initState():
    - Dipanggil saat objek state di-inisialisasi. Menggunakan metode isUpdate() untuk memeriksa apakah form ini digunakan untuk mengedit data yang sudah ada.
2. isUpdate():
    - Memeriksa apakah ada objek wisata yang diberikan (untuk mengedit). Jika ya, mengubah judul dan tombol submit serta mengisi kontrol dengan nilai yang ada.
3. build(BuildContext context):
    - Mengatur tampilan form dengan beberapa TextFormField untuk input.
    - Menggunakan SingleChildScrollView untuk menghindari overflow saat keyboard muncul.
4. Text Fields:
    - _destinationTextField(): Input untuk destinasi wisata dengan validasi.
    - _locationTextField(): Input untuk lokasi wisata dengan validasi.
    - _attractionTextField(): Input untuk atraksi wisata dengan validasi.
5. _buttonSubmit():
    - Mengembalikan tombol submit yang akan memanggil metode simpan() atau ubah() tergantung pada apakah form ini dalam mode tambah atau edit.
6. simpan():
    - Menyimpan data wisata baru. Menggunakan WisataBloc.addWisata() untuk menambahkan data ke database.
    - Jika berhasil, pengguna diarahkan kembali ke WisataPage. Jika gagal, menampilkan dialog peringatan.
7. ubah():
    - Memperbarui data wisata yang ada. Menggunakan WisataBloc.updateWisata() untuk memperbarui data di database.
    - Memiliki penanganan yang sama seperti metode simpan() untuk navigasi dan penanganan error.

Desain UI:

- Menggunakan warna kuning cerah untuk AppBar dan tombol submit, memberikan tampilan yang konsisten dengan bagian lain aplikasi.
- Setiap TextFormField memiliki label yang jelas dan validasi untuk memastikan bahwa semua informasi yang diperlukan diisi sebelum pengiriman.

Widget

1. Success_dialog.dart

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

class Consts {
  Consts._();
  static const double padding = 16.0;
  static const double avatarRadius = 66.0;
}

class SuccessDialog extends StatelessWidget {
  final String? description;
  final VoidCallback? okClick;

  const SuccessDialog({Key? key, this.description, this.okClick})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Dialog(
      shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(Consts.padding)),
      elevation: 0.0,
      backgroundColor: Colors.transparent,
      child: dialogContent(context),
    );
  }

  dialogContent(BuildContext context) {
    return Container(
      padding: const EdgeInsets.only(
        top: Consts.padding,
        bottom: Consts.padding,
        left: Consts.padding,
        right: Consts.padding,
      ),
      margin: const EdgeInsets.only(top: Consts.avatarRadius),
      decoration: BoxDecoration(
        color: Colors.white,
        shape: BoxShape.rectangle,
        borderRadius: BorderRadius.circular(Consts.padding),
        boxShadow: const [
          BoxShadow(
            color: Colors.black26,
```

```
            blurRadius: 10.0,
            offset: Offset(0.0, 10.0),
          ),
        ],
      ),
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          const Text(
            "SUKSES",
            style: TextStyle(
                fontSize: 24.0,
                fontWeight: FontWeight.w700,
                color: Colors.yellow),
          ),
          const SizedBox(height: 16.0),
          Text(
            description!,
            textAlign: TextAlign.center,
            style: const TextStyle(
              fontSize: 16.0,
            ),
          ),
          const SizedBox(height: 24.0),
          Align(
            alignment: Alignment.bottomRight,
            child: OutlinedButton(
              onPressed: () {
                Navigator.of(context).pop(); // To close the dialog
                okClick!();
              },
              child: const Text("OK"),
            ),
          )
        ],
      ),
    );
  }
}
```

Berikut adalah penjelasan tentang kelas SuccessDialog yang digunakan untuk menampilkan dialog sukses dalam aplikasi Flutter. Kelas ini dirancang untuk memberikan umpan balik kepada pengguna setelah suatu tindakan berhasil, seperti menyimpan atau memperbarui data.

Kelas SuccessDialog

SuccessDialog adalah widget berbasis StatelessWidget yang menyajikan dialog dengan pesan sukses dan tombol OK untuk menutup dialog.

Struktur Kelas:

1. Konstanta:
    - Menggunakan kelas Consts untuk menyimpan nilai konstan seperti padding dan radius avatar. Hal ini membantu menjaga konsistensi dalam desain UI.
2. Parameter Kelas:
    - description: Menerima teks deskripsi yang akan ditampilkan dalam dialog.
    - okClick: Menerima callback yang akan dipanggil saat pengguna menekan tombol OK.

Metode Kelas:

1. build(BuildContext context):
    - Mengembalikan widget Dialog dengan gaya tertentu dan memanggil metode dialogContent() untuk menyusun konten dialog.
2. dialogContent(BuildContext context):
    - Membangun konten dialog menggunakan Container dengan padding, margin, dan dekorasi yang sesuai.
    - Mengandung:
        - Text untuk menampilkan judul "SUKSES".
        - Text untuk menampilkan deskripsi yang diberikan kepada dialog.
        - OutlinedButton untuk tombol OK, yang memanggil metode okClick saat ditekan dan menutup dialog.

Desain UI:

- Dialog menggunakan latar belakang putih dengan sudut melengkung dan bayangan halus, memberikan tampilan modern.
- Menggunakan warna kuning untuk judul untuk menekankan kesuksesan, sedangkan teks deskripsi menggunakan warna hitam default agar mudah dibaca.
- Menggunakan SizedBox untuk menambahkan ruang antar elemen, menjaga tata letak yang rapi.

2. Warning_dialog.dart

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

class Consts {
  Consts._();
  static const double padding = 16.0;
  static const double avatarRadius = 66.0;
}

class WarningDialog extends StatelessWidget {
  final String? description;
  final VoidCallback? okClick;

  const WarningDialog({Key? key, this.description, this.okClick})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Dialog(
      shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(Consts.padding)),
      elevation: 0.0,
```

```
    backgroundColor: Colors.transparent,
    child: dialogContent(context),
  );
}

dialogContent(BuildContext context) {
  return Container(
    padding: const EdgeInsets.only(
      top: Consts.padding,
      bottom: Consts.padding,
      left: Consts.padding,
      right: Consts.padding,
    ),
    margin: const EdgeInsets.only(top: Consts.avatarRadius),
    decoration: BoxDecoration(
      color: Colors.white,
      shape: BoxShape.rectangle,
      borderRadius: BorderRadius.circular(Consts.padding),
      boxShadow: const [
        BoxShadow(
          color: Colors.black26,
          blurRadius: 10.0,
          offset: Offset(0.0, 10.0),
        ),
      ],
    ),
    child: Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        const Text(
          "GAGAL",
          style: TextStyle(
              fontSize: 24.0, fontWeight: FontWeight.w700, color: Colors.red),
        ),
        const SizedBox(height: 16.0),
        Text(
          description!,
          textAlign: TextAlign.center,
          style: const TextStyle(
            fontSize: 16.0,
          ),
        ),
        const SizedBox(height: 24.0),
        Align(
          alignment: Alignment.bottomRight,
```

```
            child: ElevatedButton(
              onPressed: () {
                Navigator.of(context).pop(); // To close the dialog
              },
              child: const Text("OK"),
            ),
          )
        ],
      ),
    );
  }
}
```

Berikut adalah penjelasan mengenai kelas **WarningDialog** dalam aplikasi Flutter. Kelas ini dirancang untuk menampilkan dialog peringatan kepada pengguna, biasanya digunakan untuk menunjukkan kegagalan atau kesalahan dalam suatu tindakan.

Kelas WarningDialog

WarningDialog adalah widget yang memperluas **StatelessWidget** dan menampilkan dialog peringatan dengan pesan dan tombol OK untuk menutup dialog.

Struktur Kelas:

1. **Konstanta**:
    - Menggunakan kelas **Consts** untuk menyimpan nilai konstan seperti padding dan radius avatar, yang menjaga konsistensi dalam desain UI.
2. **Parameter Kelas**:
    - **description**: Menyimpan teks deskripsi yang akan ditampilkan dalam dialog.
    - **okClick**: Menerima callback, meskipun tidak digunakan dalam implementasi ini, yang bisa digunakan untuk menangani aksi saat tombol OK ditekan.

Metode Kelas:

1. **build(BuildContext context)**:
    - Mengembalikan widget **Dialog** dengan desain tertentu dan memanggil metode **dialogContent()** untuk membangun konten dialog.
2. **dialogContent(BuildContext context)**:
    - Membangun konten dialog dengan menggunakan **Container** yang memiliki padding, margin, dan dekorasi untuk penampilan yang rapi.
    - Konten terdiri dari:
        - **Text** untuk menampilkan judul "GAGAL" dengan warna merah untuk menekankan bahwa ada masalah.
        - **Text** untuk menampilkan deskripsi yang diberikan kepada dialog.
        - **ElevatedButton** untuk tombol OK yang menutup dialog saat ditekan.

Desain UI:

- Dialog memiliki latar belakang putih dengan sudut melengkung dan bayangan halus, memberikan tampilan yang bersih dan profesional.
- Judul menggunakan warna merah untuk menarik perhatian dan menunjukkan bahwa ada kesalahan.

- Teks deskripsi ditampilkan dengan warna default agar mudah dibaca.
- Menggunakan **SizedBox** untuk menambahkan ruang antar elemen, menjaga tata letak yang rapi.