Nama : Kintan Kinasih Mahaputri

NIM : H1D022019

Shift : A

Penjelasan code:

Login.bloc

```
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);
  }
}
```

Penjelasan : Kelas LoginBloc bertanggung jawab untuk menangani proses login dengan mengirimkan email dan password ke server menggunakan metode login, yang merupakan fungsi statis dan mengembalikan objek Login di masa depan. Di dalam fungsi tersebut, URL untuk endpoint login diambil dari ApiUrl, dan body permintaan disiapkan dalam bentuk peta (dictionary) yang berisi email dan password. Permintaan POST kemudian dikirim menggunakan kelas Api, dan respons yang diterima diuraikan dari format JSON menjadi objek Dart. Terakhir, data yang telah diuraikan dikonversi menjadi objek Login menggunakan metode fromJson dan dikembalikan ke pemanggil

logout.bloc

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


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

Penjelasan : Kelas LogoutBloc bertanggung jawab untuk menangani proses logout pengguna dalam aplikasi. Di dalam kelas ini, terdapat metode statis logout yang tidak menerima parameter dan mengembalikan objek Future. Metode ini

memanggil metode logout dari kelas UserInfo, yang kemungkinan besar berfungsi untuk menghapus data pengguna dari penyimpanan lokal atau sesi saat ini. Dengan menggunakan struktur ini, kelas LogoutBloc menyediakan cara terpusat untuk melakukan logout, menjaga pemisahan antara logika bisnis dan logika tampilan di dalam aplikasi.

registrasi.bloc

```
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);

  }

}
```

Penjelasan : Kelas `RegistrasiBloc` berfungsi untuk mengelola proses registrasi pengguna dalam aplikasi. Metode statis `registrasi` menerima tiga parameter opsional: `nama`, `email`, dan `password`. Di dalam metode ini, URL untuk API registrasi diambil dari `ApiUrl.registrasi`, dan data pengguna dikemas dalam bentuk map ke dalam variabel `body`. Selanjutnya, metode `post` dari kelas `Api` dipanggil dengan URL dan body tersebut, yang mengirimkan permintaan HTTP ke server. Setelah menerima respons dari server, respons tersebut diuraikan dari format JSON menjadi objek Dart menggunakan `json.decode()`, dan hasilnya kemudian dikonversi menjadi objek `Registrasi` dengan memanggil `Registrasi.fromJson()`. Dengan struktur ini, `RegistrasiBloc` menyediakan antarmuka yang bersih untuk menangani registrasi, memisahkan logika bisnis dari bagian tampilan aplikasi.

```
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 = {
    "id": wisata!.id,
    "Event": wisata.Event,
    "Harga": wisata.Harga,
    "Sheet": wisata.Sheet,
  };
  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 = {
    "id": wisata!.id,
    "Event": wisata.Event,
    "Harga": wisata.Harga,
    "Sheet": wisata.Sheet,
  };
  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;
    }
  }
}
```

Penjelasan : Kelas `WisataBloc` mengelola interaksi antara aplikasi dan API terkait data wisata, menyediakan metode untuk mengambil, menambah, memperbarui, dan menghapus informasi wisata. Metode `getWisata` mengirimkan permintaan HTTP GET ke URL yang ditentukan dalam `ApiUrl.listWisata`, kemudian mendekode respons JSON dan mengonversi setiap item dalam data menjadi objek `Wisata`, mengumpulkan semua objek dalam daftar yang kemudian dikembalikan. Metode `addWisata` dan `updateWisata` masing-masing mengirimkan permintaan POST dan PUT untuk menambah dan memperbarui data wisata, dengan data dikemas dalam format JSON. Dalam `addWisata`, status respons dari server dikembalikan untuk mengecek keberhasilan operasi, sedangkan dalam `updateWisata`, URL spesifik untuk wisata yang diperbarui diambil menggunakan ID wisata. Terakhir, metode `deleteWisata` mengirimkan permintaan DELETE ke server, dan mengembalikan status keberhasilan penghapusan berdasarkan respons yang diterima. Dengan struktur ini, `WisataBloc` memisahkan logika bisnis dari lapisan tampilan, memberikan antarmuka yang jelas untuk mengelola data wisata dalam aplikasi.

Apiulr

```
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';
  }
}
```

Penjelasan: Kelas `ApiUrl` berfungsi untuk menyimpan dan mengelola URL endpoint API yang digunakan dalam aplikasi, menyediakan cara terpusat untuk mengakses berbagai rute tanpa harus menyebarkan string URL secara langsung di seluruh kode. Kelas ini memiliki konstanta `baseUrl` yang menyimpan alamat dasar API. Selanjutnya, terdapat beberapa konstanta statis yang menyimpan URL lengkap untuk registrasi, login, dan daftar wisata, yang dibentuk dengan menggabungkan `baseUrl` dengan endpoint spesifik. Kelas ini juga memiliki metode statis yang mengembalikan URL untuk operasi tertentu berdasarkan ID, seperti `updateWisata`, `showWisata`, dan `deleteWisata`. Metode ini mengonversi ID menjadi string dan menggabungkannya dengan path endpoint yang sesuai. Dengan pendekatan ini, `ApiUrl` memastikan bahwa semua URL API dapat dikelola dengan mudah, meningkatkan konsistensi dan mengurangi kemungkinan kesalahan dalam penulisan URL.

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}');
    }
  }
}
```

Penjelasan: Kelas `Api` bertanggung jawab untuk mengelola komunikasi dengan server melalui HTTP, menyediakan metode untuk melakukan operasi dasar seperti `POST`, `GET`, `PUT`, dan `DELETE`. Setiap metode mengatur token otentikasi yang diambil dari `UserInfo`, memastikan bahwa setiap permintaan dilengkapi dengan header otorisasi yang sesuai. Dalam setiap metode, permintaan dilakukan menggunakan paket `http`, dan hasilnya diproses menggunakan fungsi `_returnResponse`, yang memeriksa kode status HTTP dari respons. Jika kode status adalah 200, respons dikembalikan, tetapi untuk kode kesalahan seperti 400, 401, 403, 422, dan 500, kelas ini melemparkan pengecualian yang sesuai dengan jenis kesalahan yang terjadi, seperti `BadRequestException` atau `FetchDataException`. Jika terjadi kesalahan koneksi, seperti tidak ada internet, kelas ini juga menangani pengecualian tersebut dengan melemparkan `FetchDataException`. Dengan demikian, kelas `Api` berfungsi sebagai lapisan abstraksi yang mengelola komunikasi jaringan dan penanganan kesalahan secara konsisten di seluruh aplikasi.

app

```
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: ");
}
```

Penjelasan : Kumpulan kelas ini mendefinisikan hierarki pengecualian khusus untuk menangani kesalahan yang mungkin terjadi selama komunikasi dengan server dalam aplikasi. Kelas dasar `AppException` mengimplementasikan antarmuka `Exception` dan memiliki dua atribut: `_message` untuk menyimpan pesan kesalahan yang spesifik dan `_prefix` untuk memberikan konteks tambahan pada pesan tersebut. Metode `toString()` di kelas ini digunakan untuk menghasilkan representasi string dari pengecualian, menggabungkan prefix dan pesan.

Kelas-kelas turunan seperti `FetchDataException`, `BadRequestException`, `UnauthorisedException`, `UnprocessableEntityException`, dan `InvalidInputException` masing-masing menangani jenis kesalahan yang berbeda. Misalnya, `FetchDataException` digunakan saat terjadi kesalahan komunikasi, sedangkan

`BadRequestException` merujuk pada permintaan yang tidak valid, dan `UnauthorisedException` menunjukkan bahwa pengguna tidak memiliki izin untuk mengakses sumber daya. Kelas-kelas ini menambahkan prefix yang sesuai ke pesan kesalahan, sehingga saat pengecualian dilemparkan dan ditangani, pengembang dapat dengan mudah memahami jenis kesalahan yang terjadi dan menanganinya dengan tepat. Dengan cara ini, hierarki pengecualian ini meningkatkan keterbacaan dan pengelolaan kesalahan dalam aplikasi.

user_info

```
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();
  }
}
```

Penjelasan : Kelas `UserInfo` digunakan untuk menyimpan dan mengambil informasi pengguna menggunakan `SharedPreferences`, yang memungkinkan penyimpanan data lokal pada perangkat. Kelas ini memiliki beberapa metode asinkron yang berfungsi untuk menangani token otentikasi dan ID pengguna.

1. `setToken(String value)`: Metode ini menyimpan token dalam bentuk string di `SharedPreferences`. Token ini biasanya digunakan untuk autentikasi pengguna saat melakukan permintaan ke server.

2. getToken()` Metode ini mengembalikan token yang disimpan dalam `SharedPreferences`. Jika tidak ada token yang disimpan, metode ini akan mengembalikan `null`.

3. setUserID(int value)`: Metode ini menyimpan ID pengguna dalam bentuk integer di `SharedPreferences`. ID ini dapat digunakan untuk mengidentifikasi pengguna dalam aplikasi.

4. getUserID()`: Metode ini mengembalikan ID pengguna yang disimpan. Seperti pada `getToken()`, jika ID tidak ada, metode ini akan mengembalikan `null`.

5. logout()`: Metode ini menghapus semua data yang tersimpan di `SharedPreferences`, termasuk token dan ID pengguna. Ini biasanya digunakan saat pengguna melakukan logout dari aplikasi untuk memastikan tidak ada informasi sensitif yang tersisa.

login.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'],
```

```
    );
  }
 }
}
```

Penjelasan : Kelas `Login` dirancang untuk merepresentasikan respons dari proses login dalam aplikasi, termasuk informasi penting yang diterima setelah pengguna berhasil masuk. Berikut adalah penjelasan mengenai bagian-bagian dari kelas ini:

1. Atribut:

- `int? code`: Menyimpan kode status dari respons login. Kode ini biasanya menunjukkan apakah login berhasil (misalnya, 200 untuk sukses).

- `bool? status`: Menyimpan status login (true/false) yang menunjukkan apakah proses login berhasil.

- `String? token`: Menyimpan token otentikasi yang diberikan oleh server setelah login berhasil. Token ini digunakan untuk mengautentikasi pengguna di permintaan selanjutnya.

- `int? userID`: Menyimpan ID pengguna yang unik. ID ini dapat digunakan untuk mengidentifikasi pengguna dalam basis data.

- `String? userEmail`: Menyimpan alamat email pengguna yang terdaftar. Ini bisa berguna untuk menampilkan informasi pengguna di antarmuka aplikasi.

2. Konstruktor:

- `Login({this.code, this.status, this.token, this.userID, this.userEmail})`: Konstruktor ini mengizinkan inisialisasi objek `Login` dengan nilai-nilai untuk atribut yang ada. Semua atribut bersifat opsional karena menggunakan tanda tanya (`?`), yang berarti mereka dapat memiliki nilai null.

3. Metode `fromJson:

- `factory Login.fromJson(Map<String, dynamic> obj)`: Metode ini adalah konstruktor pabrik yang menerima objek JSON (yang dipetakan ke dalam `Map`) sebagai argumen dan mengembalikan instance baru dari kelas `Login`.

- Di dalam metode ini, ada pemeriksaan pada `obj['code']` untuk menentukan apakah login berhasil. Jika kode adalah 200, objek `Login` akan diinisialisasi dengan semua informasi yang relevan, termasuk token, ID pengguna, dan alamat email. Jika tidak, objek `Login` hanya akan diisi dengan kode dan status, tanpa informasi tambahan.

registrasi

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

Penjelasan : Kelas `Registrasi` dirancang untuk merepresentasikan respons dari proses pendaftaran pengguna dalam aplikasi. Kelas ini memiliki tiga atribut opsional: `code`, yang menyimpan kode status dari respons pendaftaran (biasanya menunjukkan keberhasilan pendaftaran); `status`, yang merupakan boolean yang menunjukkan apakah pendaftaran berhasil; dan `data`, yang menyimpan informasi tambahan terkait pendaftaran. Konstruktor kelas ini memungkinkan inisialisasi objek `Registrasi` dengan nilai-nilai untuk atribut yang ada. Metode `fromJson` berfungsi sebagai konstruktor pabrik yang menerima objek JSON (dalam bentuk `Map`) dan mengembalikan instance baru dari kelas `Registrasi`. Metode ini mengisi atribut kelas dengan mengambil nilai dari kunci yang sesuai dalam objek JSON yang diterima, sehingga memungkinkan aplikasi untuk menangani hasil dari proses pendaftaran dengan lebih terorganisir dan efisien.

wisata.dart

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

Penjelasan : Kelas `Wisata` digunakan untuk merepresentasikan objek wisata dalam aplikasi, dengan atribut yang mencakup `id`, `Event`, `Harga`, dan `Sheet`. Atribut `id` menyimpan identifikasi unik untuk setiap objek wisata, `Event` berisi nama atau deskripsi dari acara yang ditawarkan, `Harga` menyimpan harga terkait objek wisata dalam bentuk integer, dan `Sheet` berfungsi untuk menyimpan informasi tambahan yang relevan. Kelas ini dilengkapi dengan konstruktor yang memungkinkan inisialisasi objek dengan nilai-nilai untuk atribut yang ada. Metode `fromJson` bertindak sebagai konstruktor pabrik, yang mengambil objek JSON (dalam bentuk `Map`) dan mengembalikan instance baru dari kelas `Wisata`. Metode ini memudahkan pengambilan data dari respons API dengan mengekstrak nilai-nilai dari kunci yang sesuai dalam objek JSON, sehingga memungkinkan aplikasi untuk mengelola informasi tentang objek wisata dengan lebih terstruktur dan efisien.

UI

login_page

```
import 'package:flutter/material.dart';
import '../bloc/login_bloc.dart';
import '../helpers/user_info.dart';
import '../widget/warning_dialog.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'),
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(8.0),
          child: Form(
            key: _formKey,
            child: Column(
              children: [
                _emailTextField(),
                _passwordTextField(),
                _buttonLogin(),
                const SizedBox(
                  height: 30,
                ),
```

```
              _menuRegistrasi()
            ],
          ),
        ),
      ),
    ),
  );
 }

//Membuat Textbox email
 Widget _emailTextField() {
  return TextFormField(
    decoration: const InputDecoration(labelText: "Email"),
    keyboardType: TextInputType.emailAddress,
    controller: _emailTextboxController,
    validator: (value) {
//validasi harus diisi
      if (value!.isEmpty) {
        return 'Email harus diisi';
      }
      return null;
    },
  );
 }

//Membuat Textbox password
 Widget _passwordTextField() {
  return TextFormField(
    decoration: const InputDecoration(labelText: "Password"),
    keyboardType: TextInputType.text,
    obscureText: true,
    controller: _passwordTextboxController,
    validator: (value) {
```

```
//jika karakter yang dimasukkan kurang dari 6 karakter
        if (value!.isEmpty) {
          return "Password harus diisi";
        }
        return null;
      },
    );
  }

  //Membuat Tombol Login
  Widget _buttonLogin() {
    return ElevatedButton(
        child: const Text("Login"),
        onPressed: () {
          var validate = _formKey.currentState!.validate();
          if (validate) {
            if (!_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) {
        print(value.userID);
        await UserInfo().setToken(value.token!);
```

```
        await UserInfo().setUserID(value.userID!);
        Navigator.pushReplacement(context,
            MaterialPageRoute(builder: (context) => const WisataPage()));
      } else {
        showDialog(
            context: context,
            barrierDismissible: false,
            builder: (BuildContext context) => const WarningDialog(
                  description: "Login gagal, silahkan coba lagi",
                ));
      }
    }, onError: (error) {
      print(error);
      showDialog(
          context: context,
          barrierDismissible: false,
          builder: (BuildContext context) => const WarningDialog(
                description: "Login gagal, silahkan coba lagi",
              ));
    });
    setState(() {
      _isLoading = false;
    });
  }

// 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()));
      },
    ),
  );
 }
}
```

Penjelasan: Kelas `LoginPage` adalah halaman login dalam aplikasi Flutter yang memungkinkan pengguna untuk masuk menggunakan email dan kata sandi mereka. Halaman ini menggunakan `StatefulWidget` untuk mengelola keadaan dan memperbarui UI ketika pengguna berinteraksi. Terdapat formulir dengan dua kolom teks: satu untuk email dan satu lagi untuk kata sandi, masing-masing dilengkapi dengan validasi untuk memastikan bahwa kedua kolom tersebut diisi. Tombol "Login" akan memicu proses login dengan memanggil metode `_submit()`, yang memvalidasi formulir dan, jika berhasil, menggunakan `LoginBloc` untuk mengautentikasi pengguna. Jika login berhasil (dengan kode status 200), token dan ID pengguna disimpan menggunakan kelas `UserInfo`, dan pengguna diarahkan ke halaman `WisataPage`. Jika login gagal, dialog peringatan ditampilkan kepada pengguna. Selain itu, ada opsi untuk membuka halaman registrasi, yang diakses dengan mengetuk teks "Registrasi". UI diatur dalam `SingleChildScrollView` untuk memungkinkan pengguna menggulir saat keyboard muncul, menjaga tampilan tetap rapi dan responsif.

registrasi

```
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"),
    ),
    body: SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Form(
          key: _formKey,
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              _namaTextField(),
              _emailTextField(),
              _passwordTextField(),
              _passwordKonfirmasiTextField(),
              _buttonRegistrasi(),
            ],
          ),
        ),
      ),
    ),
  );
}

// Membuat Textbox Nama
Widget _namaTextField() {
  return TextFormField(
    decoration: const InputDecoration(labelText: "Nama"),
```

```
    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"),
    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"),
    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"),
    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(
        child: const Text("Registrasi"),
        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",
            ));
    });
    setState(() {
      _isLoading = false;
    });
  }
}
```

Penjelasan : Kelas `RegistrasiPage` adalah halaman dalam aplikasi Flutter yang memungkinkan pengguna untuk mendaftar dengan mengisi formulir registrasi. Halaman ini menggunakan `StatefulWidget` untuk mengelola status dan tampilan saat pengguna berinteraksi. Di dalamnya terdapat formulir dengan beberapa kolom teks, termasuk untuk nama, email, kata sandi, dan konfirmasi kata sandi. Setiap kolom dilengkapi dengan validasi untuk memastikan input yang valid: nama harus minimal 3 karakter, email harus valid, kata sandi harus minimal 6 karakter, dan konfirmasi kata sandi harus sama dengan kata sandi yang dimasukkan sebelumnya. Tombol "Registrasi" akan memicu proses registrasi yang memanggil metode `_submit()`, yang mengirimkan data ke `RegistrasiBloc` untuk diproses. Jika registrasi berhasil, dialog sukses muncul untuk memberitahukan pengguna bahwa pendaftaran berhasil dan mendorong mereka untuk masuk. Namun, jika terjadi kesalahan selama registrasi, dialog peringatan ditampilkan. UI diatur dalam `SingleChildScrollView` untuk memastikan tampilan tetap responsif saat keyboard muncul, memungkinkan pengguna untuk menggulir halaman dengan mudah.

wisata

```
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, // Warna hijau 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(
                "ID: ${widget.wisata!.id}",
                style: const TextStyle(
                  fontSize: 20.0,
                  color: Colors.black, // Warna teks hitam
                ),
              ),
              const SizedBox(height: 8.0), // Jarak antar elemen
              Text(
                "Event: ${widget.wisata!.Event}", // Akses properti event dari model Wisata
                style: const TextStyle(
                  fontSize: 20.0,
                  color: Colors.black, // Warna teks hitam
                ),
```

```
            ),
            const SizedBox(height: 8.0), // Jarak antar elemen
            Text(
              "Harga: ${widget.wisata!.Harga}", // Perbaikan string interpolasi
              style: const TextStyle(
                fontSize: 18.0,
                color: Colors.black,
              ),
            ),
            const SizedBox(height: 8.0),
            Text(
              "Sheet: ${widget.wisata!.Sheet}", // Akses properti sheet dari model Wisata
              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.green[700], // Warna hijau untuk tombol edit
        ),
        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: () {
          // Pastikan id adalah string
          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);
  }
}
```

Penjelasan: Kelas `WisataDetail` adalah halaman yang menampilkan rincian dari objek `Wisata` dalam aplikasi Flutter. Halaman ini menggunakan `StatefulWidget` untuk mengelola status dan menampilkan data wisata yang dipilih, termasuk ID, nama event, harga, dan sheet. Tampilan halaman terdiri dari beberapa elemen teks yang disusun secara vertikal di tengah layar, dilengkapi dengan padding untuk tata letak yang rapi. Pengguna dapat melakukan dua tindakan utama: mengedit data wisata dengan tombol "EDIT" yang mengarahkan ke halaman `WisataForm`, dan menghapus data dengan tombol "DELETE". Saat tombol hapus ditekan, dialog konfirmasi muncul untuk memastikan bahwa pengguna ingin menghapus data tersebut. Jika pengguna memilih untuk menghapus, maka metode `deleteWisata` dari `WisataBloc` dipanggil untuk menghapus data dari backend. Jika penghapusan berhasil, pengguna akan diarahkan kembali ke halaman `WisataPage`, tetapi jika gagal, dialog peringatan ditampilkan untuk memberi tahu pengguna tentang kesalahan yang terjadi. Halaman ini memberikan interaksi yang jelas dan mudah bagi pengguna untuk mengelola informasi wisata.

wisata_form

```
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 EVENT";
  String tombolSubmit = "SIMPAN";
  final _idTextboxController = TextEditingController();
  final _EventTextboxController = TextEditingController();
  final _HargaTextboxController = TextEditingController();
  final _SheetTextboxController = TextEditingController();

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

  void isUpdate() {
    if (widget.wisata != null) {
      setState(() {
        judul = "UBAH EVENT";
        tombolSubmit = "UBAH";
        _idTextboxController.text = widget.wisata!.id?.toString() ?? '';
        _EventTextboxController.text = widget.wisata!.Event ?? '';
        _HargaTextboxController.text = widget.wisata!.Harga?.toString() ?? '';
```

```
      _SheetTextboxController.text = widget.wisata!.Sheet ?? '';
    });
  } else {
    setState(() {
      judul = "TAMBAH EVENT";
      tombolSubmit = "SIMPAN";
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text(judul),
      backgroundColor: Colors.yellow[700], // Warna kuning untuk AppBar
    ),
    body: Container(
      color: Colors.green[300], // Warna hijau untuk latar belakang
      child: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.all(8.0),
          child: Form(
            key: _formKey,
            child: Column(
              children: [
                _idTextField(),
                _EventTextField(),
                _HargaTextField(),
                _SheetTextField(),
                _buttonSubmit(),
              ],
            ),
```

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

Widget _idTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "ID",
      labelStyle: TextStyle(color: Colors.black),
    ),
    keyboardType: TextInputType.text,
    controller: _idTextboxController,
    validator: (value) {
      if (value!.isEmpty) {
        return "ID harus diisi";
      }
      return null;
    },
  );
}

Widget _EventTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Event",
      labelStyle: TextStyle(color: Colors.black),
    ),
    keyboardType: TextInputType.text,
    controller: _EventTextboxController,
    validator: (value) {
```

```
      if (value!.isEmpty) {
        return "Event harus diisi";
      }
      return null;
    },
  );
}

Widget _HargaTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Harga",
      labelStyle: TextStyle(color: Colors.black),
    ),
    keyboardType: TextInputType.number, // Pastikan ini adalah angka
    controller: _HargaTextboxController,
    validator: (value) {
      if (value!.isEmpty) {
        return "Harga harus diisi";
      }
      return null;
    },
  );
}

Widget _SheetTextField() {
  return TextFormField(
    decoration: const InputDecoration(
      labelText: "Sheet",
      labelStyle: TextStyle(color: Colors.black),
    ),
    keyboardType: TextInputType.text,
    controller: _SheetTextboxController,
```

```
    validator: (value) {
      if (value!.isEmpty) {
        return "Sheet harus diisi";
      }
      return null;
    },
  );
}

Widget _buttonSubmit() {
  return OutlinedButton(
    style: OutlinedButton.styleFrom(
      backgroundColor: Colors.yellow[700], // Warna kuning untuk tombol
    ),
    child: Text(
      tombolSubmit,
      style: const TextStyle(color: Colors.black), // Teks berwarna hitam
    ),
    onPressed: () {
      var validate = _formKey.currentState!.validate();
      if (validate) {
        if (!_isLoading) {
          if (widget.wisata != null) {
            ubah();
          } else {
            simpan();
          }
        }
      }
    },
  );
}
```

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

  // Pastikan Harga diisi dan dikonversi menjadi integer
  int? harga = int.tryParse(_HargaTextboxController.text);
  if (harga == null) {
    // Jika parsing gagal, tampilkan dialog peringatan
    showDialog(
      context: context,
      builder: (BuildContext context) => const WarningDialog(
        description: "Harga harus berupa angka yang valid.",
      ),
    );
    setState(() {
      _isLoading = false; // Reset loading state
    });
    return; // Hentikan eksekusi
  }

  Wisata createWisata = Wisata(
    id: null,
    Event: _EventTextboxController.text,
    Harga: harga,
    Sheet: _SheetTextboxController.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",
      ),
    );
  }).whenComplete(() {
    setState(() {
      _isLoading = false; // Reset loading state
    });
  });
}

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

  // Pastikan Harga diisi dan dikonversi menjadi integer
  int? harga = int.tryParse(_HargaTextboxController.text);
  if (harga == null) {
    // Jika parsing gagal, tampilkan dialog peringatan
    showDialog(
      context: context,
      builder: (BuildContext context) => const WarningDialog(
        description: "Harga harus berupa angka yang valid.",
      ),
    );
    setState(() {
      _isLoading = false; // Reset loading state
    });
```

```
      return; // Hentikan eksekusi
    }

    Wisata updateWisata = Wisata(
      id: widget.wisata!.id,
      Event: _EventTextboxController.text,
      Harga: harga,
      Sheet: _SheetTextboxController.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",
        ),
      );
    }).whenComplete(() {
      setState(() {
        _isLoading = false; // Reset loading state
      });
    });
  }
}
```

Penjelasan :

Kelas `WisataForm` adalah halaman yang digunakan untuk menambah atau mengedit data objek `Wisata` dalam aplikasi Flutter. Halaman ini dirancang dengan menggunakan `StatefulWidget` dan memiliki elemen formulir untuk

mengumpulkan data seperti ID, nama event, harga, dan sheet. Dalam metode `initState`, fungsi `isUpdate` dipanggil untuk menentukan apakah halaman ini digunakan untuk menambah atau mengedit data; jika objek `Wisata` diberikan, judul dan label tombol akan disesuaikan untuk refleksi tindakan pengeditan. Halaman ini memiliki beberapa field teks, masing-masing dengan validasi untuk memastikan input yang valid dari pengguna. Saat pengguna menekan tombol simpan atau ubah, aplikasi akan memvalidasi input dan memastikan bahwa harga dapat dikonversi menjadi angka. Jika validasi berhasil, data wisata baru atau yang diperbarui akan disimpan melalui `WisataBloc`. Jika simpanan atau pengeditan berhasil, pengguna akan diarahkan kembali ke halaman `WisataPage`. Jika terjadi kesalahan, dialog peringatan ditampilkan untuk memberi tahu pengguna bahwa tindakan tersebut gagal. Desain halaman ini memberi pengguna pengalaman interaktif yang intuitif untuk mengelola data wisata.

wisata_page

```
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';


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.green[700], // Warna hijau untuk app bar
        title: const Text(
          'List Event',
          style: TextStyle(
            fontFamily: 'Helvetica',
            fontSize: 24.0,
```

```
      color: Colors.black, // Warna teks hitam untuk kontras
    ),
  ),
  actions: [
    Padding(
      padding: const EdgeInsets.only(right: 20.0),
      child: GestureDetector(
        child: const Icon(Icons.add, size: 26.0, color: Colors.black), // Ikon hitam untuk kontras
        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.green[700], // Latar belakang hijau muda untuk seluruh 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],
        );
      },
    );
  }
}

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.green[50], // Warna hijau muda untuk latar belakang kartu
        elevation: 5.0, // Sedikit elevasi untuk efek bayangan
        child: ListTile(
          title: Text(
            wisata.Event!,
            style: const TextStyle(
              fontFamily: 'Helvetica',
              fontSize: 18.0,
```

```
            color: Colors.black, // Warna teks hitam untuk kontras
          ),
        ),
        subtitle: Text(
          wisata.Harga?.toString() ?? 'Harga tidak tersedia', // Menangani kemungkinan null
          style: const TextStyle(color: Colors.black54),
        ),
        trailing: Text(
          wisata.Sheet!,
          style: const TextStyle(color: Colors.black87), // Hitam lebih gelap untuk teks atraksi
        ),
      ),
    ),
  );
 }
}
```

Penjelasan : Kelas `WisataPage` berfungsi sebagai antarmuka utama untuk menampilkan daftar event wisata dalam aplikasi Flutter. Halaman ini memiliki AppBar dengan judul "List Event" dan ikon untuk menambahkan event baru, serta drawer untuk logout. Dalam body, halaman ini menggunakan `FutureBuilder` untuk mengambil data wisata dari `WisataBloc`, yang memungkinkan aplikasi menampilkan daftar wisata secara asinkron. Jika data berhasil diambil, daftar event ditampilkan menggunakan widget `ListWisata`, yang mengandalkan `ListView.builder` untuk membuat item daftar secara dinamis. Setiap item dalam daftar diwakili oleh kelas `ItemWisata`, yang menampilkan informasi tentang event seperti nama event, harga, dan sheet. Saat pengguna mengetuk salah satu item, aplikasi akan mengarahkan mereka ke halaman detail `WisataDetail`, di mana pengguna dapat melihat informasi lebih lanjut atau mengedit data event. Desain halaman ini mendukung pengalaman pengguna yang interaktif dan intuitif, memudahkan navigasi dan pengelolaan informasi wisata.

widgetsuccess

```
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"),
      ),
```

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

Penjelasan : Kelas `SuccessDialog` adalah widget dialog yang dirancang untuk menampilkan pesan sukses dalam aplikasi Flutter. Dialog ini memiliki tampilan yang bersih dan modern, dengan desain yang menggunakan border-radius untuk sudut yang melengkung dan bayangan untuk memberikan efek kedalaman. Kelas ini mengandalkan konstanta dari kelas `Consts` untuk mengatur padding dan ukuran avatar, sehingga menjaga konsistensi dalam desain. Dalam dialog, terdapat teks yang menampilkan judul "SUKSES" dengan gaya huruf tebal dan berwarna kuning, serta deskripsi yang disediakan oleh pengguna. Di bagian bawah dialog, terdapat tombol "OK" yang, saat ditekan, akan menutup dialog dan memicu fungsi yang ditentukan melalui parameter `okClick`. Dengan pendekatan ini, dialog memberikan pengalaman pengguna yang intuitif dan responsif, memungkinkan pengguna untuk mengonfirmasi dan melanjutkan setelah menerima informasi penting.

warning

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

Penjelasan : Kelas `WarningDialog` adalah widget dialog yang digunakan untuk menampilkan pesan peringatan atau kesalahan dalam aplikasi Flutter. Dengan desain yang mirip dengan dialog sukses, dialog ini menggunakan sudut melengkung dan bayangan untuk memberikan efek kedalaman, menjadikannya tampak modern dan bersih. Judul dialog bertuliskan "GAGAL" dengan font berukuran besar dan tebal, berwarna merah untuk menarik perhatian pengguna, menunjukkan bahwa ada masalah yang perlu diperhatikan. Di bawah judul, terdapat teks deskripsi yang dapat disesuaikan, memberikan informasi lebih lanjut tentang kesalahan yang terjadi. Bagian bawah dialog dilengkapi dengan tombol "OK" yang, ketika ditekan, akan menutup dialog tanpa memanggil fungsi tambahan, sehingga menjaga kesederhanaan dan fokus pengguna pada pesan yang disampaikan. Desain dan fungsi dari `WarningDialog`

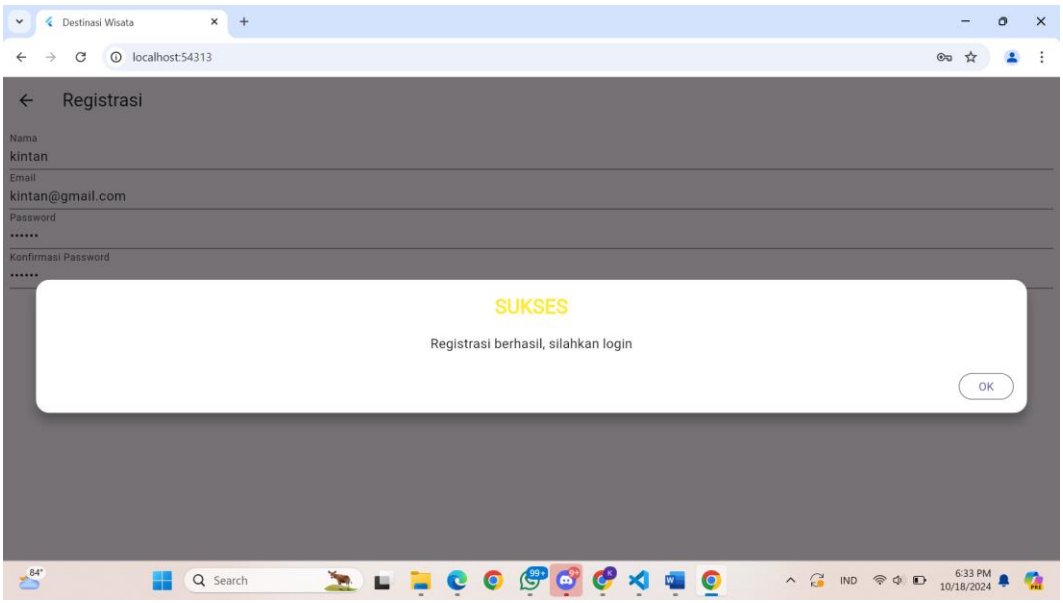
memastikan bahwa pengguna dapat dengan mudah memahami situasi kesalahan yang terjadi dan meresponsnya dengan tepat.

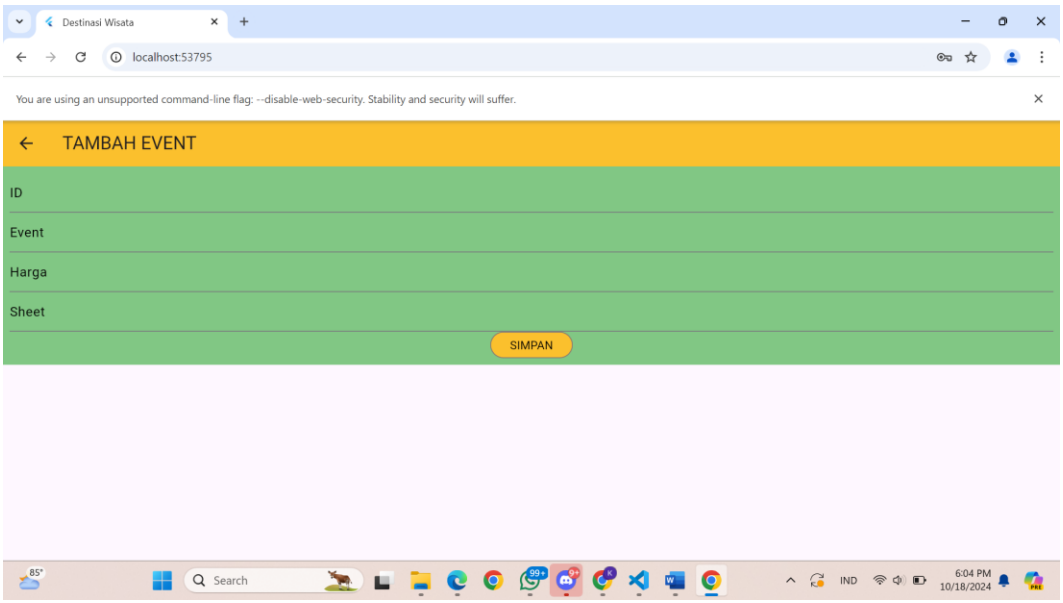
Main.dart

```
import 'package:flutter/material.dart';
import 'package:manajemen_pariwasata/helpers/user_info.dart';
import 'package:manajemen_pariwasata/ui/login_page.dart';
import 'package:manajemen_pariwasata/ui/wisata_page.dart';

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

class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  Widget page = const CircularProgressIndicator();
  @override
  void initState() {
    super.initState();
    isLogin();
  }

  void isLogin() async {
    var token = await UserInfo().getToken();
    if (token != null) {
      setState(() {
        page = const WisataPage();
      });
    } else {
      setState(() {
```

```
        page = const LoginPage();
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Destinasi Wisata',
      debugShowCheckedModeBanner: false,
      home: page,
    );
  }
}
```

Penjelasan : Kelas `MyApp` adalah titik awal dari aplikasi Flutter yang mengelola navigasi berdasarkan status login pengguna. Saat aplikasi dimulai, metode `initState` dipanggil untuk mengecek apakah pengguna telah login atau belum dengan memanggil fungsi `isLogin`. Fungsi ini menggunakan `UserInfo` untuk mendapatkan token yang menandakan status login pengguna. Jika token ditemukan, berarti pengguna sudah login, dan halaman utama aplikasi akan diarahkan ke `WisataPage`, yang menampilkan daftar tempat wisata. Sebaliknya, jika token tidak ada, aplikasi akan menampilkan `LoginPage`, memungkinkan pengguna untuk masuk. Hal ini diimplementasikan dengan mengubah widget yang ditampilkan dalam variabel `page`, yang di-render dalam metode `build`. Dengan demikian, aplikasi memberikan pengalaman pengguna yang mulus dengan langsung menampilkan konten yang relevan sesuai dengan status autentikasi pengguna. Selain itu, pengaturan `debugShowCheckedModeBanner` diatur ke `false` untuk menyembunyikan banner mode debug, memberikan tampilan yang lebih bersih pada aplikasi saat dijalankan.


Destinasi Wisata
localhost:54313
Registrasi
Nama
kintan
Email
kintan@gmail.com
Password
Konfirmasi Password
SUKSES
Registrasi berhasil, silahkan login
OK
Search
IND
6:33 PM
10/18/2024


Destinasi Wisata
localhost:53795
You are using an unsupported command-line flag: --disable-web-security. Stability and security will suffer.
TAMBAH EVENT
ID
Event
Harga
Sheet
SIMPAN
Search
IND
6:04 PM
10/18/2024