RESPONSI I

PRAKTIKUM PEMROGRAMAN MOBILE

Nama : Yulio Putra Wardana

NIM : H1D022085

Shift Awal : A

Shift Baru : F

Penjelasan Responsi 1

1. **Registrasi Code**
   a. **ui/registrasi_page.dart**

```
import 'package:flutter/material.dart';
import 'package:responsi1/bloc/registrasi_bloc.dart';
import 'package:responsi1/widget/success_dialog.dart';
import 'package:responsi1/widget/warning_dialog.dart';

class RegistrasiPage extends StatefulWidget {
  const RegistrasiPage({super.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: const Color.fromARGB(255, 255, 255, 255),
      ),
```

```
    body: SingleChildScrollView(
      child: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 24.0, vertical:
16.0),
        child: Form(
          key: _formKey,
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: [
              const SizedBox(height: 20),
              const Text(
                "Daftar Akun",
                style: TextStyle(
                  fontSize: 24,
                  fontWeight: FontWeight.bold,
                  color: Color.fromARGB(255, 0, 0, 0),
                  fontFamily: 'Georgia',
                ),
                textAlign: TextAlign.center,
              ),
              const SizedBox(height: 40),
              _namaTextField(),
              const SizedBox(height: 16),
              _emailTextField(),
              const SizedBox(height: 16),
              _passwordTextField(),
              const SizedBox(height: 16),
              _passwordKonfirmasiTextField(),
              const SizedBox(height: 30),
              _buttonRegistrasi(),
              const SizedBox(height: 20),
            ],
          ),
        ),
      ),
    );
  }

  Widget _namaTextField() {
    return TextFormField(
      decoration: InputDecoration(
        labelText: "Nama",
```

```
      labelStyle: const TextStyle(color: Colors.black),
      border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10.0)),
      focusedBorder: OutlineInputBorder(
       borderSide: const BorderSide(color: Colors.black),
       borderRadius: BorderRadius.circular(10.0),
      ),
     ),
     controller: _namaTextboxController,
     validator: (value) {
      if (value!.length < 3) {
       return "Nama harus diisi minimal 3 karakter";
      }
      return null;
     },
    );
  }

  Widget _emailTextField() {
   return TextFormField(
    decoration: InputDecoration(
      labelText: "Email",
      labelStyle: const TextStyle(color: Colors.black),
      border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10.0)),
      focusedBorder: OutlineInputBorder(
       borderSide: const BorderSide(color: Colors.black),
       borderRadius: BorderRadius.circular(10.0),
      ),
     ),
     keyboardType: TextInputType.emailAddress,
     controller: _emailTextboxController,
     validator: (value) {
      if (value!.isEmpty) {
       return 'Email harus diisi';
      }
      Pattern pattern =
          r'^[\w-\.]+@([\w-]+\.)+[\w]{2,4}$';
      RegExp regex = RegExp(pattern.toString());
      if (!regex.hasMatch(value)) {
       return "Email tidak valid";
      }
      return null;
```

```
    },
  );
 }

 Widget _passwordTextField() {
  return TextFormField(
   decoration: InputDecoration(
    labelText: "Password",
    labelStyle: const TextStyle(color: Colors.black),
    border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10.0)),
    focusedBorder: OutlineInputBorder(
     borderSide: const BorderSide(color: Colors.black),
     borderRadius: BorderRadius.circular(10.0),
    ),
   ),
   obscureText: true,
   controller: _passwordTextboxController,
   validator: (value) {
    if (value!.length < 6) {
     return "Password harus diisi minimal 6 karakter";
    }
    return null;
   },
  );
 }

 Widget _passwordKonfirmasiTextField() {
  return TextFormField(
   decoration: InputDecoration(
    labelText: "Konfirmasi Password",
    labelStyle: const TextStyle(color: Colors.black),
    border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10.0)),
    focusedBorder: OutlineInputBorder(
     borderSide: const BorderSide(color: Colors.black),
     borderRadius: BorderRadius.circular(10.0),
    ),
   ),
   obscureText: true,
   validator: (value) {
    if (value != _passwordTextboxController.text) {
     return "Konfirmasi Password tidak sama";
```

```
    }
    return null;
   },
  );
}

Widget _buttonRegistrasi() {
  return ElevatedButton(
    style: ElevatedButton.styleFrom(
      padding: const EdgeInsets.symmetric(vertical: 16.0),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(10.0),
      ),
      backgroundColor: Colors.black,
    ),
    child: _isLoading
        ? const CircularProgressIndicator(
            valueColor: AlwaysStoppedAnimation<Color>(Colors.white),
          )
        : const Text(
            "Registrasi",
            style: TextStyle(fontSize: 18, color: Colors.white),
          ),
    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;
    });
  }
}
```

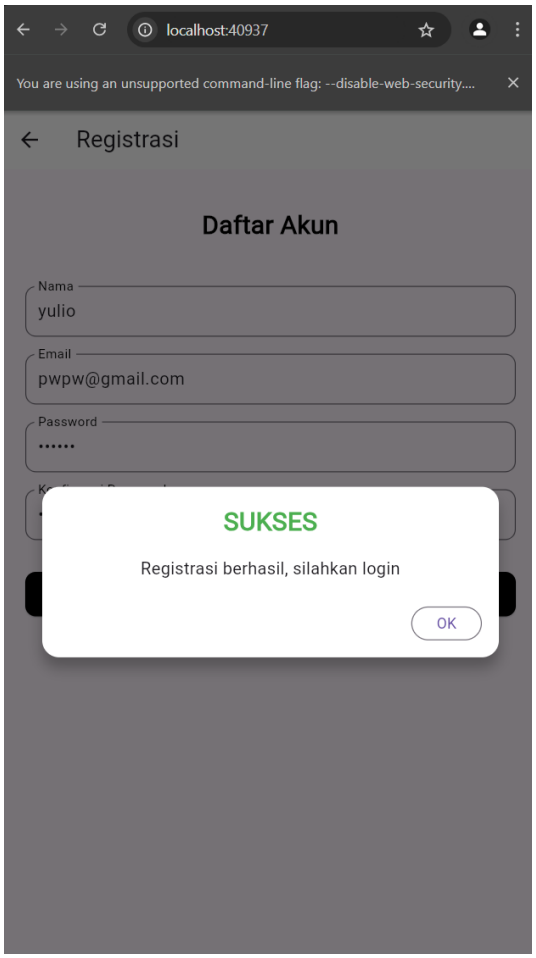
Penjelasan

1. Struktur Umum:
    - Kode ini membuat halaman registrasi dengan menggunakan Flutter dan melibatkan form input untuk pengguna mendaftar akun.
    - Form ini memiliki beberapa kolom input seperti nama, email, password, dan konfirmasi password.
2. Controller:
    - Terdapat tiga TextEditingController untuk menangani input teks pada form: nama, email, dan password.
3. Form Validation: Setiap input field memiliki validator untuk memeriksa validitas data yang dimasukkan. Misalnya, email harus valid dan password harus lebih dari 6 karakter. Konfirmasi password harus sesuai dengan input di kolom password.
4. Submit Button:

- Tombol registrasi menggunakan ElevatedButton. Saat ditekan, tombol memeriksa validasi form, dan jika valid, memanggil fungsi _submit().

5. Fungsi _submit():
   - Fungsi ini mengirim data registrasi ke server melalui RegistrasiBloc.registrasi(). Jika berhasil, dialog sukses muncul, dan jika gagal, dialog peringatan ditampilkan.

ScreenShoot:



Registrasi_page.dart

Berhasil masuk dan tercantum dalam API

b. model/registrasi.dart

1. Code

```
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'],
    );
  }
}

```

18. Penjelasan

- Kelas Registrasi:

Ini adalah model data yang merepresentasikan respons dari API untuk proses registrasi. Terdiri dari tiga properti: code: kode status (int), yang mungkin menunjukkan status HTTP atau hasil dari proses registrasi.

status: status boolean (true/false) yang menunjukkan apakah registrasi berhasil atau tidak. data: pesan atau informasi tambahan dari respons API (String).

- Constructor:

Constructor ini memungkinkan pembuatan objek Registrasi dengan nilai properti yang bisa diinisialisasi saat objek dibuat.

- Factory Constructor fromJson():

Ini adalah metode untuk mengonversi respons JSON (berbentuk Map<String, dynamic>) menjadi objek Registrasi.

Mengambil nilai dari kunci JSON (code, status, dan data) dan menggunakannya untuk mengisi properti kelas Registrasi.

c. bloc/registrasi_bloc.dart

1. Code

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

import 'package:responsi1/helpers/api_url.dart';

class RegistrasiBloc {
  static Future<void> registrasi({
    required String nama,
    required String email,
    required String password,
  }) async {
    try {
      final response = await http.post(
        Uri.parse(ApiUrl.registrasi),
        headers: {
          'Content-Type': 'application/json',
        },
        body: json.encode({
          'nama': nama,
          'email': email,
          'password': password,
        }),
      );

      print('Response Status: ${response.statusCode}');
      print('Response Body: ${response.body}');

      if (response.statusCode == 200) {
        print('Registrasi berhasil');
        return;
      } else {
        print('Error: ${response.body}');
        throw Exception('Registrasi gagal');
      }
    } catch (e) {
      print('Error terjadi: $e');
      throw Exception('Gagal melakukan request');
    }
  }
}

```

a. Penjelasan

- Tujuan: Kode ini bertanggung jawab untuk menangani proses registrasi melalui API dengan melakukan HTTP POST request.
- Fungsi registrasi(): Fungsi ini menerima tiga parameter wajib: nama, email, dan password. Data ini dikirimkan ke endpoint API menggunakan metode POST.
- HTTP Request: Menggunakan package http untuk mengirim request ke URL API (ApiUrl.registrasi). Data yang dikirim dikonversi ke format JSON (json.encode), dan dikirim bersama header Content-Type yang mengindikasikan bahwa data dalam bentuk JSON.
- Response Handling: Jika status respons dari API adalah 200 (berhasil), pesan "Registrasi berhasil" akan dicetak. Jika terjadi kesalahan (status selain 200), pesan error akan dicetak dan Exception akan dilempar dengan pesan "Registrasi gagal".
- Error Handling: Jika terjadi kesalahan selama request (misalnya masalah jaringan), fungsi menangkap kesalahan dan melempar Exception dengan pesan "Gagal melakukan request".

3. Login

a. Ui/login_page.dart

1. Code

```dart
import 'package:flutter/material.dart';
import 'package:responsi1/helpers/user_info.dart';
import 'package:responsi1/ui/medicine_screen.dart';
import 'package:responsi1/ui/registrasi_page.dart';
import 'package:responsi1/widget/warning_dialog.dart';
import 'package:responsi1/bloc/login_bloc.dart';

class LoginPage extends StatefulWidget {
  const LoginPage({super.key});

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

class _LoginPageState extends State<LoginPage> {
```

```dart
  final _formKey = GlobalKey<FormState>();
  bool _isLoading = false;
  final _emailTextboxController = TextEditingController();
  final _passwordTextboxController = TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.black,
      appBar: AppBar(
        title: const Text('Login', style: TextStyle(fontFamily:
'Georgia')),
        backgroundColor: const Color.fromARGB(255, 255, 255, 255),
      ),
      body: SingleChildScrollView(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 24.0,
vertical: 40.0),
          child: Form(
            key: _formKey,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: [
                _buildLogo(),
                const SizedBox(height: 40),
                _emailTextField(),
                const SizedBox(height: 20),
                _passwordTextField(),
                const SizedBox(height: 40),
                _buttonLogin(),
                const SizedBox(height: 30),
                _menuRegistrasi(),
              ],
            ),
          ),
        ),
      ),
    );
  }

  Widget _buildLogo() {
    return const Column(
      children: [
        Icon(
          Icons.login_rounded,
```

```
          size: 100,
          color: Colors.white,
        ),
        SizedBox(height: 20),
        Text(
          'Welcome Back!',
          style: TextStyle(
            fontSize: 24,
            fontWeight: FontWeight.bold,
            color: Colors.white,
            fontFamily: 'Georgia',
          ),
        ),
      ],
    );
  }

  Widget _emailTextField() {
    return TextFormField(
      decoration: InputDecoration(
        labelText: "Email",
        labelStyle: const TextStyle(color: Colors.white),
        filled: true,
        fillColor: Colors.grey[800],
        border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10.0)),
        focusedBorder: OutlineInputBorder(
          borderSide: const BorderSide(color: Colors.white),
          borderRadius: BorderRadius.circular(10.0),
        ),
      ),
      keyboardType: TextInputType.emailAddress,
      controller: _emailTextboxController,
      validator: (value) {
        if (value!.isEmpty) {
          return 'Email harus diisi';
        }
        return null;
      },
    );
  }

  Widget _passwordTextField() {
    return TextFormField(
      decoration: InputDecoration(
```

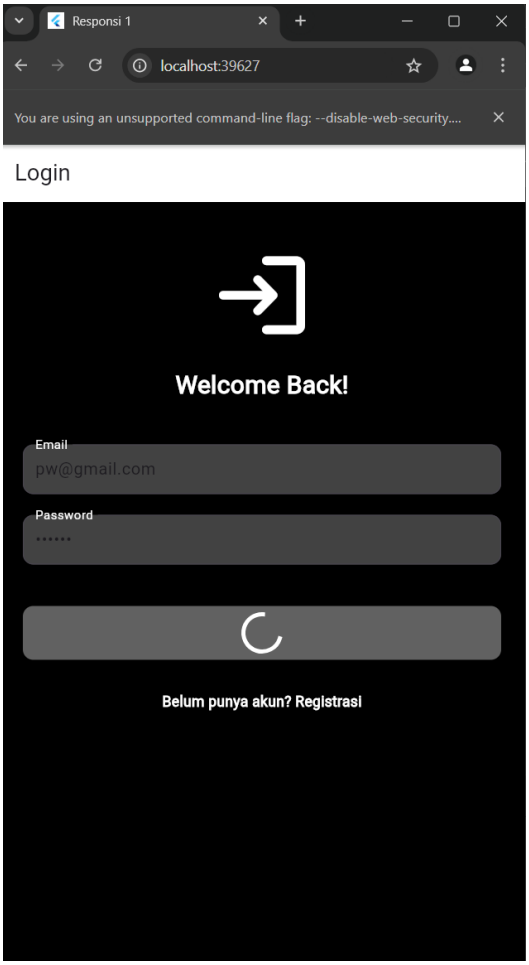
```
        labelText: "Password",
        labelStyle: const TextStyle(color: Colors.white),
        filled: true,
        fillColor: Colors.grey[800],
        border: OutlineInputBorder(borderRadius:
BorderRadius.circular(10.0)),
        focusedBorder: OutlineInputBorder(
          borderSide: const BorderSide(color: Colors.white),
          borderRadius: BorderRadius.circular(10.0),
        ),
      ),
      keyboardType: TextInputType.text,
      obscureText: true,
      controller: _passwordTextboxController,
      validator: (value) {
        if (value!.isEmpty) {
          return "Password harus diisi";
        }
        return null;
      },
    );
  }

  Widget _buttonLogin() {
    return ElevatedButton(
      style: ElevatedButton.styleFrom(
        padding: const EdgeInsets.symmetric(vertical: 16.0),
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(10.0),
        ),
        backgroundColor: Colors.grey[700],
      ),
      child: _isLoading
          ? const CircularProgressIndicator(
              valueColor:
AlwaysStoppedAnimation<Color>(Colors.white),
            )
          : const Text(
              "Login",
              style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold, color: Colors.white),
            ),
      onPressed: () {
        var validate = _formKey.currentState!.validate();
        if (validate && !_isLoading) {
```

```
          _submit();
        }
      },
    );
  }

  void _submit() async {
    if (!_formKey.currentState!.validate()) return;

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

    try {
      var result = await LoginBloc.login(
        email: _emailTextboxController.text,
        password: _passwordTextboxController.text,
      );

      await UserInfo().setToken(result.token!);
      await UserInfo().setUserID(result.userID!);

      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => MedicineScreen()),
      );
    } catch (e) {
      showDialog(
        context: context,
        barrierDismissible: false,
        builder: (BuildContext context) => const WarningDialog(
          description: "Login gagal, silahkan coba lagi",
        ),
      );
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }

  Widget _menuRegistrasi() {
    return Center(
      child: InkWell(
        child: const Text(
```

```
          "Belum punya akun? Registrasi",
          style: TextStyle(
            color: Colors.white,
            fontWeight: FontWeight.w600,
            fontFamily: 'Georgia',
          ),
        ),
        onTap: () {
          Navigator.push(context,
              MaterialPageRoute(builder: (context) => const
RegistrasiPage()));
        },
      ),
    );
  }
}
```

2. **Penjelasan**
   - **Struktur Umum:** Kode ini membuat halaman login menggunakan Flutter dengan form input untuk email dan password. Terdapat fitur validasi form serta interaksi dengan API untuk autentikasi pengguna.
   - **Controller:** Dua TextEditingController digunakan untuk menangani input teks dari email dan password.
   - **Form Validation:** Setiap input memiliki validator: email tidak boleh kosong dan password harus diisi.
   - **Submit Button:** Tombol login mengirim data setelah validasi form dan menampilkan indikator loading selama proses berlangsung. Saat login berhasil, data token dan userID disimpan, kemudian pengguna dialihkan ke halaman MedicineScreen. Jika gagal, dialog peringatan ditampilkan.
   - **Navigasi Registrasi:** Terdapat tautan untuk pengguna yang belum memiliki akun, mengarahkan mereka ke halaman `RegistrasiPage`.

3. **Screenshoot**



Login_page.dart

Berhasil masuk dan langsung mengarah halaman berikutnya

b. Model/login.dart

1. Code

```
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 && obj['status'] == true) {
      var data = obj['data'];
      var user = data != null ? data['user'] : null;

      return Login(
        code: obj['code'],
        status: obj['status'],
        token: data != null ? data['token'] : null,
        userID: user != null && user['id'] != null ?
int.tryParse(user['id'].toString()) : null,
        userEmail: user != null ? user['email'] : null,
      );
    } else {
      return Login(
        code: obj['code'],
        status: obj['status'],
      );
    }
  }
}
```

2. **Penjelasan**

- Kelas Login: Model ini digunakan untuk merepresentasikan respons dari API saat login. Terdapat beberapa properti: code: kode status HTTP dari API. status: menunjukkan keberhasilan atau kegagalan login (boolean). token: token autentikasi pengguna. userID: ID pengguna yang berhasil login. userEmail: email pengguna yang login.
- Constructor: Constructor untuk menginisialisasi objek Login dengan nilai-nilai yang didapat dari respons API.
- Factory Constructor fromJson(): Metode ini mengubah respons JSON menjadi objek Login. Jika kode status adalah 200 dan login berhasil (status true), data seperti token, userID, dan userEmail

diambil dari JSON. Jika login gagal, hanya properti code dan status yang diambil.

c. Bloc/login_bloc.dart

1. Code

```dart
import 'dart:convert';
import 'package:http/http.dart' as http;
import 'package:responsi1/helpers/api_url.dart';
import 'package:responsi1/model/login.dart';

class LoginBloc {
  static Future<Login> login({
    required String email,
    required String password,
  }) async {
    var body = {
      'email': email,
      'password': password,
    };

    try {
      final response = await http.post(
        Uri.parse(ApiUrl.login),
        headers: {
          'Content-Type': 'application/json',
        },
        body: json.encode(body),
      );

      if (response.statusCode == 200) {
        var jsonResponse = json.decode(response.body);
        return Login.fromJson(jsonResponse);
      } else {
        throw Exception('Login gagal: ${response.body}');
      }
    } catch (e) {
      throw Exception('Terjadi error saat login: $e');
    }
  }
}
```

2. **Penjelasan**

- **Fungsi login():** Fungsi ini digunakan untuk melakukan proses login dengan mengirimkan email dan password ke API. Parameter email dan password diterima sebagai input yang diperlukan.
- **HTTP Request:** Menggunakan metode **POST** untuk mengirim data email dan password ke API login (ApiUrl.login). Data dikonversi ke format **JSON** dengan json.encode() sebelum dikirim.
- **Response Handling:** Jika status kode respons API adalah 200, respons JSON di-decode dan dikonversi menjadi objek Login menggunakan Login.fromJson(). Jika respons gagal, Exception akan dilempar dengan pesan error.
- **Error Handling:** Jika terjadi error selama proses request (misalnya jaringan), exception ditangani dengan pesan error yang relevan.

4. Medicine

a. Ui/medicine_screen.dart

1. Code

```
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:responsi1/bloc/medicine_bloc.dart';
import 'package:responsi1/model/medicine.dart';

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

  final MedicineBloc _medicineBloc = MedicineBloc();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[900],
      appBar: AppBar(
```

```
        title: const Text(
          'Pengingat Obat CRUD',
          style: TextStyle(
            fontFamily: 'Georgia',
          ),
        ),
        backgroundColor: const Color.fromARGB(255, 255, 255, 255),
        actions: [
          IconButton(
            icon: const Icon(Icons.add, color: Colors.white),
            onPressed: () {
              _showMedicineForm(context);
            },
          ),
        ],
      ),
      body: BlocProvider(
        create: (context) => _medicineBloc..add(LoadMedicines()),
        child: BlocBuilder<MedicineBloc, MedicineState>(
          builder: (context, state) {
            if (state.medicines.isEmpty) {
              return const Center(
                child: Text(
                  "No Medicines",
                  style: TextStyle(
                    fontFamily: 'Georgia',
                    fontSize: 18,
                    color: Colors.white,
                  ),
                ),
              );
            }
            return ListView.builder(
              itemCount: state.medicines.length,
              itemBuilder: (context, index) {
                final medicine = state.medicines[index];
                return Card(
                  color: Colors.grey[850],
                  margin: const EdgeInsets.symmetric(vertical: 10,
horizontal: 15),
                  child: ListTile(
                    title: Text(
                      medicine.medicineName,
                      style: const TextStyle(
                        fontFamily: 'Georgia',
```

```
                        color: Colors.white,
                      ),
                    ),
                    subtitle: Text(
                      'Dosage: ${medicine.dosageMg} mg, Times per
day: ${medicine.timesPerDay}',
                      style: const TextStyle(
                        fontFamily: 'Georgia',
                        color: Colors.grey,
                      ),
                    ),
                    trailing: Row(
                      mainAxisSize: MainAxisSize.min,
                      children: [
                        IconButton(
                          icon: const Icon(Icons.edit, color:
Colors.white),
                          onPressed: () {
                            _showMedicineForm(context, medicine:
medicine);
                          },
                        ),
                        IconButton(
                          icon: const Icon(Icons.delete, color:
Colors.red),
                          onPressed: () {
                            context.read<MedicineBloc>().add(DeleteM
edicine(medicine.id));
                          },
                        ),
                      ],
                    ),
                  ),
                );
              },
            );
          },
        ),
      ),
    );
  }

  void _showMedicineForm(BuildContext context, {Medicine? medicine})
{
    final nameController = TextEditingController(
```

```
        text: medicine != null ? medicine.medicineName : '');
    final dosageController = TextEditingController(
        text: medicine != null ? medicine.dosageMg.toString() : '');
    final timesController = TextEditingController(
        text: medicine != null ? medicine.timesPerDay.toString() :
'');

    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          backgroundColor: Colors.grey[800],
          title: Text(
            medicine != null ? 'Edit Medicine' : 'Add Medicine',
            style: const TextStyle(
              fontFamily: 'Georgia',
              color: Colors.white,
            ),
          ),
          content: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              TextField(
                controller: nameController,
                style: const TextStyle(color: Colors.white),
                decoration: const InputDecoration(
                  labelText: 'Medicine Name',
                  labelStyle: TextStyle(color: Colors.grey),
                  enabledBorder: UnderlineInputBorder(
                    borderSide: BorderSide(color: Colors.white),
                  ),
                  focusedBorder: UnderlineInputBorder(
                    borderSide: BorderSide(color: Colors.white),
                  ),
                ),
              ),
              TextField(
                controller: dosageController,
                keyboardType: TextInputType.number,
                style: const TextStyle(color: Colors.white),
                decoration: const InputDecoration(
                  labelText: 'Dosage (mg)',
                  labelStyle: TextStyle(color: Colors.grey),
                  enabledBorder: UnderlineInputBorder(
                    borderSide: BorderSide(color: Colors.white),
```

```
                ),
                focusedBorder: UnderlineInputBorder(
                  borderSide: BorderSide(color: Colors.white),
                ),
              ),
            ),
            TextField(
              controller: timesController,
              keyboardType: TextInputType.number,
              style: const TextStyle(color: Colors.white),
              decoration: const InputDecoration(
                labelText: 'Times per Day',
                labelStyle: TextStyle(color: Colors.grey),
                enabledBorder: UnderlineInputBorder(
                  borderSide: BorderSide(color: Colors.white),
                ),
                focusedBorder: UnderlineInputBorder(
                  borderSide: BorderSide(color: Colors.white),
                ),
              ),
            ),
          ],
        ),
        actions: [
          ElevatedButton(
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.grey[700],
            ),
            onPressed: () {
              if (medicine != null) {
                context.read<MedicineBloc>().add(
                      UpdateMedicine(Medicine(
                        id: medicine.id,
                        medicineName: nameController.text,
                        dosageMg:
int.parse(dosageController.text),
                        timesPerDay:
int.parse(timesController.text),
                      )),
                    );
              } else {
                context.read<MedicineBloc>().add(
                      AddMedicine(Medicine(
                        id: DateTime.now().millisecondsSinceEpoch,
                        medicineName: nameController.text,
```

```
                          dosageMg:
int.parse(dosageController.text),
                          timesPerDay:
int.parse(timesController.text),
                        )),
                      );
                }
                Navigator.pop(context);
              },
              child: const Text('Save', style: TextStyle(fontFamily:
'Georgia')),
            ),
            TextButton(
              onPressed: () => Navigator.pop(context),
              child: const Text('Cancel', style:
TextStyle(fontFamily: 'Georgia', color: Colors.grey)),
            ),
          ],
        );
      },
    );
  }
}
```

2. **Penjelasan**

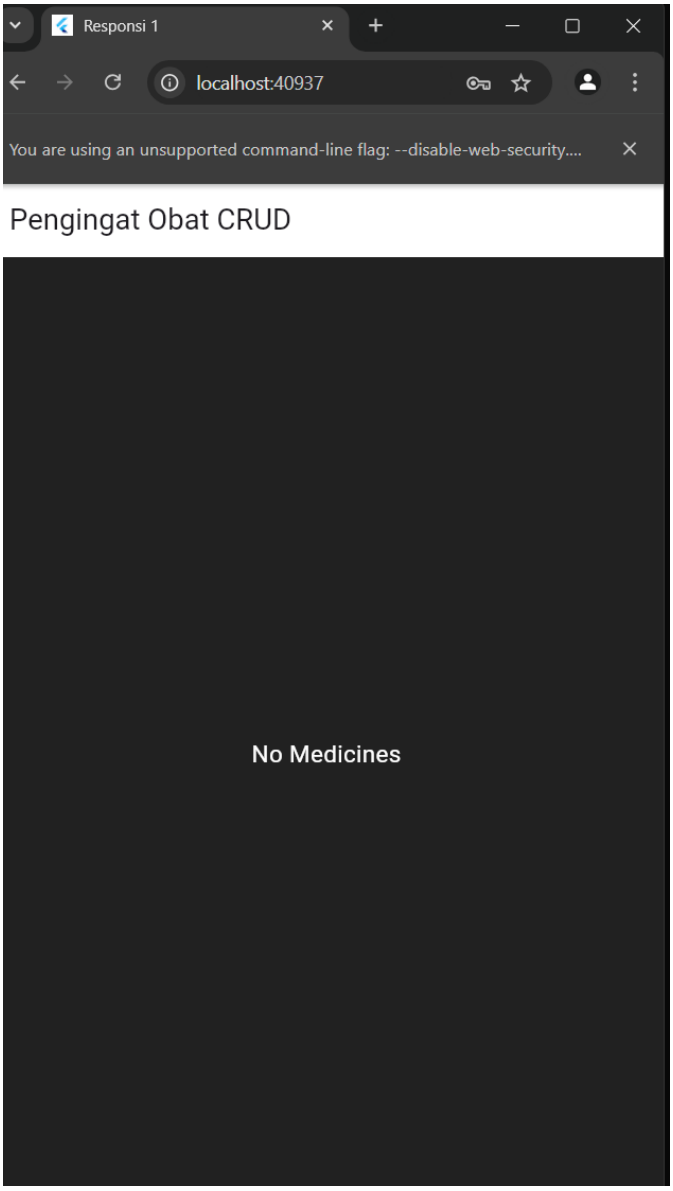
- MedicineScreen: Ini adalah tampilan utama untuk CRUD pengingat obat. Menggunakan Bloc untuk memuat, menambah, mengedit, dan menghapus data obat.Terdapat tombol Add di app bar yang memunculkan form input untuk menambah atau mengedit data obat.
- BlocProvider dan BlocBuilder: BlocProvider menyediakan instance MedicineBloc untuk mengelola state obat. BlocBuilder merender tampilan sesuai state yang diterima dari MedicineBloc.
- UI List Obat: Jika tidak ada data obat, ditampilkan teks "No Medicines". Jika ada data obat, ditampilkan dalam bentuk ListTile di dalam ListView, yang menampilkan nama obat, dosis, dan jumlah

konsumsi per hari. Ada dua tombol untuk setiap item: Edit: Mengedit data obat. Delete: Menghapus data obat.

- Form Input Obat: Digunakan untuk menambah atau mengedit obat. Menampilkan AlertDialog dengan input nama obat, dosis, dan jumlah konsumsi per hari. Tombol Save akan memanggil AddMedicine atau UpdateMedicine tergantung apakah sedang menambah atau mengedit.

3. Screenshot

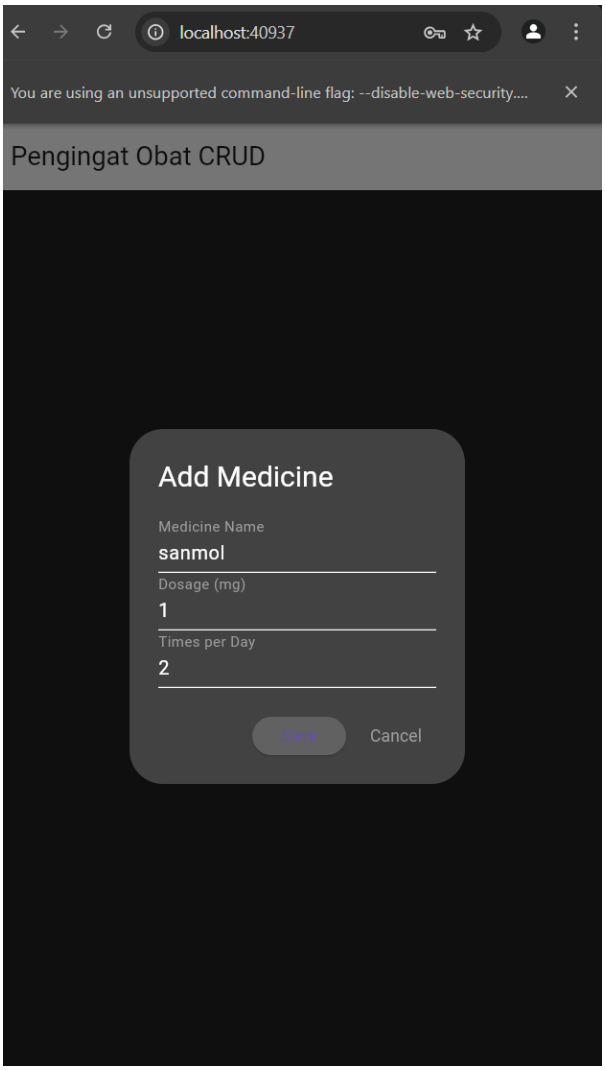
a. View



View medicine_screen.dart

(maaf soal warna nyaru hectic gakeburu mas server down buat revisi)

b. Add



Add medicine_screen.dart

(maaf soal warna nyaru hectic gakeburu mas server down buat revisi)

c. Update

Belum tuntas (maaf mas gakeburu server sering down)

d. Delete

Belum Tuntas (maaf mas gakeburu karna server sering down)

b. Model/medicine.dart

1. Code

```
class Medicine {
```

```
  final int id;
  final String medicineName;
  final int dosageMg;
  final int timesPerDay;

  Medicine({
    required this.id,
    required this.medicineName,
    required this.dosageMg,
    required this.timesPerDay,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'medicine_name': medicineName,
      'dosage_mg': dosageMg,
      'times_per_day': timesPerDay,
    };
  }

  factory Medicine.fromMap(Map<String, dynamic> map) {
    return Medicine(
      id: map['id'],
      medicineName: map['medicine_name'],
      dosageMg: map['dosage_mg'],
      timesPerDay: map['times_per_day'],
    );
  }
}
```

2. **Penjelasan**

- Kelas Medicine: Model yang merepresentasikan data obat, terdiri dari: id: ID unik untuk setiap obat. medicineName: Nama obat. dosageMg: Dosis obat dalam miligram (mg). timesPerDay: Jumlah konsumsi obat per hari.
- Metode toMap(): Mengonversi objek Medicine menjadi Map<String, dynamic>, yang biasanya digunakan untuk

menyimpan data dalam database atau mengirimkan data melalui jaringan (seperti API).

- Fabrik fromMap(): Mengonversi data dari Map<String, dynamic> menjadi objek Medicine. Memetakan nilai dari map ke atribut kelas Medicine.

c. Bloc/medicine_bloc.dart

1. Code

```
import 'dart:convert';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:http/http.dart' as http;
import 'package:responsi1/helpers/api_url.dart';
import 'package:responsi1/model/medicine.dart';


class MedicineState {
  final List<Medicine> medicines;


  MedicineState({required this.medicines});
}

abstract class MedicineEvent {}

class LoadMedicines extends MedicineEvent {}

class AddMedicine extends MedicineEvent {
  final Medicine medicine;

  AddMedicine(this.medicine);
}

class UpdateMedicine extends MedicineEvent {
  final Medicine medicine;

  UpdateMedicine(this.medicine);
}

class DeleteMedicine extends MedicineEvent {
  final int id;

  DeleteMedicine(this.id);
```

```dart
}

class MedicineBloc extends Bloc<MedicineEvent, MedicineState> {
  MedicineBloc() : super(MedicineState(medicines: [])) {
    on<LoadMedicines>(_onLoadMedicines);
    on<AddMedicine>(_onAddMedicine);
    on<UpdateMedicine>(_onUpdateMedicine);
    on<DeleteMedicine>(_onDeleteMedicine);
  }


  Future<void> _onLoadMedicines(
      LoadMedicines event, Emitter<MedicineState> emit) async {
    try {
      final response = await
http.get(Uri.parse(ApiUrl.getMedicines));

      if (response.statusCode == 200) {
        List<dynamic> data = json.decode(response.body);
        List<Medicine> medicines = data.map((item) {
          return Medicine(
            id: item['id'],
            medicineName: item['medicine_name'],
            dosageMg: item['dosage_mg'],
            timesPerDay: item['times_per_day'],
          );
        }).toList();
        emit(MedicineState(medicines: medicines));
      } else {
        throw Exception('Failed to load medicines');
      }
    } catch (error) {
      print('Error loading medicines: $error');
    }
  }

  Future<void> _onAddMedicine(
      AddMedicine event, Emitter<MedicineState> emit) async {
    try {
      final response = await http.post(
        Uri.parse(ApiUrl.addMedicine),
        headers: {'Content-Type': 'application/json'},
        body: json.encode({
          'medicine_name': event.medicine.medicineName,
          'dosage_mg': event.medicine.dosageMg,
```

```
          'times_per_day': event.medicine.timesPerDay,
        }),
      );

      if (response.statusCode == 201) {
        add(LoadMedicines());
      } else {
        throw Exception('Failed to add medicine');
      }
    } catch (error) {
      print('Error adding medicine: $error');
    }
  }

  Future<void> _onUpdateMedicine(
      UpdateMedicine event, Emitter<MedicineState> emit) async {
    try {
      final response = await http.put(
        Uri.parse('${ApiUrl.updateMedicine}/${event.medicine.id}'),
        headers: {'Content-Type': 'application/json'},
        body: json.encode({
          'medicine_name': event.medicine.medicineName,
          'dosage_mg': event.medicine.dosageMg,
          'times_per_day': event.medicine.timesPerDay,
        }),
      );

      if (response.statusCode == 200) {
        add(LoadMedicines());
      } else {
        throw Exception('Failed to update medicine');
      }
    } catch (error) {
      print('Error updating medicine: $error');
    }
  }

  Future<void> _onDeleteMedicine(
      DeleteMedicine event, Emitter<MedicineState> emit) async {
    try {
      final response = await http.delete(
        Uri.parse('${ApiUrl.deleteMedicine}/${event.id}'),
      );

      if (response.statusCode == 200) {
```

```
        add(LoadMedicines());
      } else {
        throw Exception('Failed to delete medicine');
      }
    } catch (error) {
      print('Error deleting medicine: $error');
    }
  }
}
```

2. **Penjelasan**
    - MedicineState: Menyimpan data obat dalam bentuk list medicines. Digunakan untuk menyimpan dan mengelola state aplikasi terkait data obat.
    - MedicineEvent (abstract class): Menyediakan event dasar untuk memanipulasi data obat. Event yang ada:LoadMedicines: Memuat daftar obat. AddMedicine: Menambahkan obat baru. UpdateMedicine: Memperbarui data obat. DeleteMedicine: Menghapus obat.
    - MedicineBloc: Menggunakan BLoC (Business Logic Component) untuk mengelola state aplikasi dengan mendengarkan event dan mengubah state berdasarkan event tersebut. Empat fungsi utama untuk menangani event: _onLoadMedicines: Mengambil data obat dari API dan mengupdate state. _onAddMedicine: Menambahkan obat baru melalui API dan memperbarui state dengan memuat ulang daftar obat. _onUpdateMedicine: Memperbarui data obat melalui API dan memperbarui state. _onDeleteMedicine: Menghapus obat dari API dan memperbarui state.

d. Helpers

1. **Api_url.dart**
    a. **Code**

```
class ApiUrl {
```

```
  static const String baseUrl =
'http://103.196.155.42/api/kesehatan/pengingat_obat';
  static const String registrasi =
'http://103.196.155.42/api/registrasi';
  static const String login = 'http://103.196.155.42/api/login';
  static const String getMedicines =
'http://103.196.155.42/api/kesehatan/pengingat_obat/1'; // Untuk
mendapatkan daftar obat
  static const String addMedicine =
'http://103.196.155.42/api/kesehatan/pengingat_obat'; // Untuk
menambahkan obat
  static const String updateMedicine =
'http://103.196.155.42/api/kesehatan/pengingat_obat/1/update';
  static const String deleteMedicine =
'http://103.196.155.42/api/kesehatan/pengingat_obat/1/delete';
}
//htt
```

b. Penjelasan

baseUrl: URL dasar untuk API.

registrasi: URL untuk registrasi pengguna.

login: URL untuk login pengguna.

getMedicines: URL untuk mendapatkan daftar obat (mengambil data obat dengan ID 1).

addMedicine: URL untuk menambahkan obat baru.

updateMedicine: URL untuk memperbarui informasi obat (obat dengan ID 1).

deleteMedicine: URL untuk menghapus obat (obat dengan ID 1).

2. Api.dart

a. Code

```
import 'dart:io';
import 'package:http/http.dart' as http;
import 'package:responsi1/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"});
      print(response);
      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 occured while Communication with Server with
StatusCode : ${response.statusCode}');
    }
  }
}
```

b. Penjelasan

post: Mengirim permintaan POST dengan data dan header autentikasi. Mengembalikan respons setelah memverifikasi statusnya.

get: Mengirim permintaan GET dengan header autentikasi dan mengembalikan respons.

put: Mengirim permintaan PUT dengan data dan header autentikasi untuk memperbarui data di server.

delete: Mengirim permintaan DELETE dengan header autentikasi untuk menghapus data di server.

3. Sisanya di repositoryyy