”KALORIKU” PLATFORM INFORMASI SEHAT UNTUK GAYA HIDUP LEBIH MUDAH

LAPORAN TUGAS BESAR PEMROGRAMAN IV

Diajukan untuk Memenuhi Kelulusan Matakuliah Pemrograman IV
pada Program Studi DIV Teknik Informatika

Universitas Logistik & Bisnis Internasional

DOSEN PENGAMPU :

Indra Riksa Herlambang, S.Tr.Kom., M.Kom., SFPC.

DISUSUN OLEH :

Nida Sakina Aulia (714220040)

Ruth Diana Purnamasari Sagala (714220042)

PROGRAM STUDI DIV TEKNIK INFORMATIKA

UNIVERSITAS LOGISTIK & BISNIS INTERNASIONAL

BANDUNG

2024

BAB I
DESKRIPSI APLIKASI

1. Latar Belakang

Seiring meningkatnya kesadaran masyarakat tentang pentingnya gaya hidup sehat, kebutuhan akan makanan yang bergizi dan terukur pun meningkat secara signifikan. Terutama di kawasan perkotaan, banyak orang yang mulai memahami manfaat pola makan yang seimbang untuk menjaga kesehatan serta mencapai bentuk tubuh yang ideal. Namun, tantangan utama yang dihadapi adalah kesibukan sehari-hari, yang sering membuat mereka kesulitan menyiapkan makanan sehat sesuai kebutuhan kalori dan nutrisi.

Kaloriku hadir sebagai solusi praktis untuk menjawab kebutuhan dan tantangan ini, sebuah aplikasi website inovatif yang dirancang untuk memudahkan pengguna menjalani gaya hidup sehat tanpa harus menghitung kalori secara manual. Terinspirasi dari keberhasilan layanan katering sehat seperti YellowFit, Kaloriku memberikan solusi praktis bagi mereka yang ingin tetap menjaga pola makan sehat. Melalui aplikasi ini, pengguna dapat memilih rekomendasi makanan berdasarkan target, kebutuhan, dan preferensi.

Tidak hanya sebagai platform rekomendasi makanan sehat, KaloriKu juga dilengkapi dengan berbagai fitur pendukung, seperti edukasi berupa artikel tips kesehatan harian yang membantu pengguna menjalani gaya hidup sehat secara berkelanjutan. Dengan fitur-fitur ini, KaloriKu tidak hanya menjadi alat untuk mencapai tujuan diet, tetapi juga mitra yang menginspirasi pengguna untuk tetap konsisten dalam perjalanan menuju gaya hidup yang lebih sehat.

Pada aplikasi ini terdapat beberapa page, diantaranya:

1. **Home Page**

   Halaman Home pada aplikasi KaloriKu berfungsi sebagai tampilan awal saat pengguna membuka aplikasi. Halaman ini menyajikan gambaran singkat tentang aplikasi, seperti slogan atau ajakan untuk hidup sehat dengan katering bernutrisi. Dari halaman ini, pengguna akan diarahkan ke Login Page untuk mengakses fitur utama aplikasi.

2. **Login Page**

   Login page berfungsi sebagai gerbang utama bagi admin untuk mengakses fitur aplikasi KaloriKu. Admin dapat masuk menggunakan alamat email dan kata sandi yang telah terdaftar.

3. **Profile Page**

   Halaman Profil pada aplikasi KaloriKu menampilkan informasi lengkap tentang pembuat aplikasi. Di halaman ini, pengguna dapat melihat informasi seperti nama pembuat, NPM, program studi.

4. **Description Page**

   Halaman Deskripsi pada aplikasi KaloriKu berisi penjelasan singkat mengenai tujuan dan fungsi aplikasi. Di halaman ini, pengguna dapat memahami bahwa KaloriKu adalah aplikasi yang menyediakan informasi makanan sehat yang membantu menjaga pola makan seimbang melalui berbagai pilihan menu bernutrisi sesuai kebutuhan kalori harian. Selain fitur informasi, aplikasi ini juga menyediakan artikel kesehatan dan berbagai tips untuk mendukung gaya hidup sehat.

5. **Menu Page**

   Halaman Menu pada aplikasi KaloriKu adalah area khusus untuk admin dalam mengelola data menu katering. Halaman ini memiliki fitur CRUD (Create, Read, Update, Delete) yang memungkinkan admin menambahkan menu baru, melihat daftar menu yang tersedia, memperbarui informasi menu (seperti nama, deskripsi, komposisi, informasi kalori dan kategori), serta menghapus menu yang sudah tidak tersedia.

BAB II

PEMBUATAN APLIKASI

2.1 Main

Berikut ini code pada file main.dart:

```dart
import 'package:flutter/material.dart';
import 'package:kaloriku/view/screen/bottom_navbar.dart';
import 'package:kaloriku/services/auth_manager.dart';
import 'package:kaloriku/view/screen/home_page.dart';
import 'package:kaloriku/view/screen/login_page.dart';

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

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

  @override
  State<MainApp> createState() => _MainAppState();
}

class _MainAppState extends State<MainApp> {
  @override
  void initState() {
    super.initState();
    checkAuthStatus();
  }

  /// Fungsi untuk mengecek status login setelah HomePage
  Future<void> checkAuthStatus() async {
    await Future.delayed(const Duration(seconds: 2));
    bool isAuthenticated = await AuthManager.isLoggedIn();

    if (!isAuthenticated && mounted) {
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => const LoginPage()),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'KaloriKu',
      home: HomePage(),
    );
  }
}
```

Gambar 1 Main.dart

Kode ini merupakan aplikasi **KaloriKu** yang memeriksa status login pengguna. Aplikasi dimulai dari main() yang menjalankan MainApp. Di dalam initState(), fungsi asinkron checkAuthStatus() menunda proses selama 2 detik lalu memeriksa autentikasi melalui AuthManager.isLoggedIn(). Jika belum login, pengguna diarahkan ke LoginPage() menggunakan Navigator.pushReplacement(), jika sudah login, ditampilkan HomePage(). Aplikasi menggunakan MaterialApp dengan judul "KaloriKu" dan menyembunyikan banner debug.

2.2 Model Data

2.2.1 Login_model.dart

```dart
class LoginInput {
  final String username;
  final String password;

  LoginInput({
    required this.username,
    required this.password,
  });

  Map<String, dynamic> toJson() => {
        "username": username,
        "password": password,
      };
}

class LoginResponse {
  final String? token;
  final String message;
  final int status;
  final String role;
  final String fullname;
  final String phone;

  LoginResponse({
    this.token,
    required this.message,
    required this.status,
    required this.role,
    required this.fullname,
    required this.phone,
  });

  factory LoginResponse.fromJson(Map<String, dynamic> json) =>
LoginResponse(
        token: json["token"] ?? '',
        message: json["message"] ?? '',
        status: json["status"] ?? 0,
        role: json["role"] ?? '',
        fullname: json["fullname"] ?? '',
        phone: json["phone"] ?? '',
      );

  Map<String, dynamic> toJson() => {
        "token": token,
        "message": message,
        "status": status,
        "role": role,
        "fullname": fullname,
        "phone": phone,
      };
}
```

Gambar 2 Login_mode.dart

Penjelasan:

- Class **LoginInput** digunakan pada saat input data saat user melakukan login. Disini user diminta untuk menginputkan username dan password. Keduanya didefinisikan dengan type string dan menggunakan parameter required, sehingga username dan password wajib diisi saat login. Disini juga terdapat metode toJson() yang mengonversi objek LoginInput menjadi format Map<String, dynamic>, yang berguna untuk mengirim data sebagai JSON dalam permintaan API.
- Class **LoginResponse** digunakan untuk memproses reponse dari server setelah login. Data yang disimpan meliputi token sebagai tanda bahwa pengguna sudah login, serta message (pesan dari server), status (kode status respons), role (peran pengguna), fullname (nama lengkap), dan phone (nomor telepon), yang semuanya wajib diisi. Kelas ini memiliki metode fromJson() yang berfungsi untuk membuat objek LoginResponse dari data JSON yang diterima dari server. Jika ada data yang kosong atau null, maka akan diisi dengan nilai default menggunakan operator ??. Selain itu, ada metode toJson() yang digunakan untuk mengubah objek ini menjadi format JSON agar mudah dikirim kembali ke server atau digunakan di bagian lain aplikasi.

2.2.2 Food_model.dart

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

class FoodsModel {
  final String id;
  final String name;
  final String ingredients;
  final String description;
  final int calories;
  final String category;

  FoodsModel({
    required this.id,
    required this.name,
    required this.ingredients,
    required this.description,
    required this.calories,
    required this.category,
  });

  factory FoodsModel.fromJson(Map<String, dynamic> json) {
    return FoodsModel(
      id: json['_id'] ?? '',
      name: json['name'] ?? '',
      ingredients: json['ingredients'] ?? '',
      description: json['description'] ?? '',
      calories: _parseCalories(json['calories']),
      category: json['category'] ?? '',
    );
  }

  static int _parseCalories(dynamic value) {
    if (value is int) return value;
    if (value is String) return int.tryParse(value) ?? 0;
    if (value is double) return value.toInt();
    return 0;
  }

  Map<String, dynamic> toJson() => {
        "_id": id,
        "name": name,
        "ingredients": ingredients,
        "description": description,
        "calories": calories,
        "category": category,
      };
  String get formattedCalories => '${calories.toStringAsFixed(1)} kcal';
}


//  Model untuk form input makanan
class FoodInput {
  final String name;
  final String ingredients;
  final String description;
  final double calories;
  final String category;

  FoodInput({
    required this.name,
    required this.ingredients,
    required this.description,
    required this.calories,
    required this.category,
  });

  Map<String, dynamic> toJson() => {
        "name": name,
        "ingredients": ingredients,
        "description": description,
        "calories": calories,
        "category": category,
      };
}

//  Model untuk response dari API
class FoodResponse {
  final String? insertedId;
  final String message;
  final int status;

  FoodResponse({
    this.insertedId,
    required this.message,
    required this.status,
  });

  factory FoodResponse.fromJson(Map<String, dynamic> json) =>
FoodResponse(insertedId: json["inserted_id"],
        message: json["message"],
        status: json["status"],
      );
}
```

Gambar 3 Food_model.dart

Penjelasan:

Kode ini digunakan untuk memproses data makanan, mulai dari input, mengonversi data ke JSON untuk dikirim keserver, hingga menerima dan meproses response dari API. Berikut penjelasan setiap functionnya;

- Class **FoodModel**, digunakan untuk merepresentasikan data makanan di aplikasi. Propertynya mencakup informasi; id, nama makanan, komposisi, deskiripsi, jumlah kalori, dan kategori makanan.
  - **fromJson()**: Mengonversi data JSON menjadi objek FoodModel. Ini digunakan saat mengambil data dari API.
  - **_parseCalories()**: Fungsi privat (hanya bisa diakses dalam kelas ini) yang memastikan nilai kalori diubah ke tipe int, baik inputnya berupa angka, string, atau double.
  - **toJson()**: Mengubah objek FoodModel menjadi format JSON agar bisa dikirim kembali ke server atau disimpan.
  - **formattedCalories**: Getter yang memformat kalori menjadi string dengan satu desimal dan menambahkan label "kcal" di belakangnya (contoh: "120.0 kcal").
- Class **FoodInput**, berfungsi sebagai model untuk menangkap input makanan dari pengguna, biasanya digunakan saat pengguna mengisi formulir. Properti di dalamnya mencakup name (nama makanan), ingredients (bahan-bahan), description (deskripsi makanan), calories (jumlah kalori dalam tipe double), dan category (kategori makanan). toJson(): Mengonversi data input menjadi format JSON agar mudah dikirim ke server melalui API.
- Class **FoodResponse**, digunakan untuk menangkap respons dari server setelah mengirim data makanan. Properti di dalamnya mencakup insertedId (ID makanan yang baru disimpan di database), message (pesan dari server, misalnya "Data berhasil disimpan"), dan status (kode status respons, misalnya 200 untuk sukses). **fromJson()**: Factory constructor yang mengubah respons JSON dari API menjadi objek FoodResponse. Ini memudahkan pengolahan hasil dari server di aplikasi.

2.3 Services

2.3.1 Api_service.dart

```dart
import 'package:dio/dio.dart';
import 'package:kaloriku/model/food_model.dart';
import 'package:kaloriku/model/login_model.dart';
import 'package:flutter/material.dart';

class ApiServices {
  final Dio dio = Dio();
  final String _baseUrl = 'https://ws-kaloriku-4cf736fcbaf0.herokuapp.com';

  Future<Iterable<FoodsModel>?> getAllMenuItem() async {
    try {
      var response = await dio.get('$_baseUrl/menu');

      debugPrint('Response API: ${response.data}');

      if (response.statusCode == 200) {
        if (response.data is List) {
          // ✅ Jika response berupa List, lakukan mapping
          final foodList = (response.data as List)
              .map((food) => FoodsModel.fromJson(food))
              .toList();
          return foodList;
        } else if (response.data is Map<String, dynamic>) {
          // ✅ Jika hanya ada satu objek, langsung ubah ke List
          return [FoodsModel.fromJson(response.data)];
        }
      }
      return null;
    } on DioException catch (e) {
      if (e.response != null && e.response!.statusCode != 200) {
        debugPrint('❌ Client error - the request cannot be fulfilled');
        return null;
      }
      rethrow;
    } catch (e) {
      rethrow;
    }
  }

  Future<FoodResponse?> postMenu(FoodInput food) async {
    try {
      final response = await dio.post(
        '$_baseUrl/insertMenu',
        data: food.toJson(),
        options: Options(headers: {'Authorization': 'Bearer $token'}),
      );

      if (response.statusCode == 200) {
        return FoodResponse.fromJson(response.data);
      }
    } catch (e) {
      debugPrint('❌ Error posting menu: $e');
    }
    return null;
  }

  Future<FoodsModel?> getSingleFood(String id) async {
    try {
      var response = await dio.get('$_baseUrl/menu/$id');

      print('📥 Response dari API: ${response.data}');

      if (response.statusCode == 200) {
        final data = response.data;
        return FoodsModel.fromJson(data);
      }
      return null;
    } on DioException catch (e) {
      if (e.response != null && e.response!.statusCode != 200) {
        debugPrint('Client error - the request cannot be fulfilled');
        return null;
      }
      rethrow;
    } catch (e) {
      rethrow;
    }
  }

  Future<FoodResponse?> putMenu(String id, FoodInput food) async {
    try {
      final response = await dio.put(
        '$_baseUrl/updateMenu/$id',
        data: food.toJson(),
        options: Options(headers: {'Authorization': 'Bearer $token'}),
      );

      if (response.statusCode == 200) {
        return FoodResponse.fromJson(response.data);
      }
    } catch (e) {
      debugPrint('❌ Error updating menu: $e');
    }
    return null;
  }

  Future<void> deleteFood(String id) async {
    try {
      final response = await dio.delete(
        '$_baseUrl/deleteMenu/$id',
        options: Options(headers: {'Authorization': 'Bearer $token'}),
      );

      if (response.statusCode == 200) {
        debugPrint('✅ Menu berhasil dihapus');
      }
    } catch (e) {
      debugPrint('❌ Error deleting menu: $e');
    }
  }

  Future<LoginResponse?> login(LoginInput login) async {
    try {
      final String url = '$_baseUrl/login';

      debugPrint('🔍 Request ke: $url');

      final response = await dio.post(
        url,
        data: login.toJson(),
        options: Options(
          headers: {
            'Content-Type': 'application/json',
            'Accept': 'application/json',
          },
        ),
      );

      debugPrint('✅ Response: ${response.data}');

      if (response.statusCode == 200) {
        return LoginResponse.fromJson(response.data);
      }
      return null;
    } on DioException catch (e) {
      if (e.response != null) {
        debugPrint(
            '🔥 Login error: ${e.response!.statusCode} -
${e.response!.data}');
        if (e.response!.statusCode == 404) {
          debugPrint('⚠️ Endpoint /login tidak ditemukan! Cek API backend.');
        } else if (e.response!.statusCode == 401) {
          debugPrint('⚠️ Unauthorized! Pastikan username & password benar.');
        }

        return null;
      } else {
        debugPrint('🔥 Request error: ${e.message}');
        return null;
      }
    } catch (e) {
      debugPrint('🔥 Unexpected error: $e');
      return null;
    }
  }
}
```

Gambar 4 Food_model.dart

Penjelasan:

- Function **getAllMenuItem**, digunakan untuk mengambil data menu dari API. Fungsi ini menggunakan metode GET ke endpoint /menu dan menangkap respons dari server. Jika respons berhasil dengan kode status 200, maka fungsi akan memeriksa apakah data yang diterima berupa **List** atau **Map**. Jika berupa List, data tersebut diubah menjadi daftar objek FoodsModel menggunakan metode .map() dan dikembalikan sebagai hasil. Jika hanya terdapat satu objek Map, maka fungsi akan membungkusnya dalam sebuah list sebelum dikembalikan. Apabila terjadi kesalahan atau respons yang diterima bukan kode 200, fungsi ini akan mengembalikan null. Untuk menangkap kesalahan yang terjadi selama proses, digunakan penanganan error dengan DioException yang mencetak log di konsol jika terjadi kesalahan.
- Function **postMenu**, digunakan untuk menambahkan menu baru ke dalam database melalui API. Fungsi ini mengirimkan permintaan POST ke endpoint /insertMenu dengan data makanan yang dikirim dalam format JSON menggunakan metode toJson() dari objek FoodInput. Header permintaan juga menyertakan token otorisasi berbentuk Bearer token untuk memastikan akses yang sah. Jika permintaan berhasil dan mendapat respons dengan kode status 200, fungsi akan mengembalikan hasil dalam bentuk objek FoodResponse. Jika terjadi kesalahan saat mengirim permintaan atau respons yang diterima tidak sesuai, fungsi akan menangkap error, mencetak log kesalahan, dan mengembalikan null. Token autentikasi harus disiapkan dan diisi dengan benar agar fungsi ini dapat berjalan.
- Function **getSingleFood**, digunakan untuk mengambil informasi detail dari satu menu berdasarkan **ID** yang diberikan. Fungsi ini mengirimkan permintaan GET ke endpoint /menu/{id}. Jika respons berhasil dengan kode status 200, data yang diterima dari API dikonversi menjadi objek FoodsModel menggunakan metode fromJson() dan dikembalikan sebagai hasil. Jika terjadi kesalahan seperti **ID tidak ditemukan** atau **kesalahan jaringan**, fungsi akan mencetak pesan error dan mengembalikan null. Fungsi ini berguna ketika pengguna ingin melihat informasi lengkap tentang menu tertentu berdasarkan ID uniknya.

- Function **putMenu**, digunakan untuk memperbarui data menu berdasarkan ID yang diberikan. Permintaan PUT dikirimkan ke endpoint /updateMenu/{id} dengan menyertakan data makanan dalam bentuk JSON. Sama seperti fungsi postMenu(), fungsi ini juga memerlukan token autentikasi yang disertakan dalam header dengan format Bearer token. Jika permintaan berhasil dan server memberikan respons dengan kode status 200, fungsi akan mengembalikan hasil sebagai objek FoodResponse. Jika terjadi kesalahan, seperti ID tidak valid atau masalah autentikasi, fungsi akan menangkap error dan mencetak pesan kesalahan di konsol. Fungsi ini sangat berguna ketika pengguna ingin memperbarui informasi pada menu yang sudah ada.
- Function **deleteMenu**, digunakan untuk menghapus menu berdasarkan **ID** yang diberikan. Fungsi mengirimkan permintaan DELETE ke endpoint /deleteMenu/{id} dan menyertakan token autentikasi dalam header. Jika permintaan berhasil dan server merespons dengan kode status 200, fungsi akan mencetak pesan keberhasilan di log konsol. Jika terjadi kesalahan seperti **ID yang tidak ditemukan** atau **kesalahan autentikasi**, fungsi akan menangkap error dan mencetak pesan kesalahan. Fungsi ini berguna untuk menghapus data makanan secara permanen dari sistem melalui API.
- Function **login**, digunakan untuk melakukan autentikasi pengguna melalui proses login. Fungsi ini mengirimkan permintaan POST ke endpoint /login dengan menyertakan data username dan password dalam format JSON menggunakan metode toJson() dari objek LoginInput. Header permintaan menyertakan Content-Type dan Accept sebagai application/json untuk memastikan format data sesuai dengan kebutuhan API. Jika autentikasi berhasil (kode status 200), respons dari server dikonversi menjadi objek LoginResponse yang berisi token autentikasi, informasi pengguna, dan status login. Jika terjadi kesalahan, seperti **endpoint tidak ditemukan (404)** atau **kredensial salah (401)**, fungsi akan menangkap dan mencetak log kesalahan di konsol. Fungsi ini penting untuk mengelola proses login pengguna di aplikasi, memungkinkan akses ke fitur yang memerlukan autentikasi.

2.3.2 Auth_manager.dart

```dart
// import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:jwt_decoder/jwt_decoder.dart';
// import 'dart:convert';

class AuthManager {
  static const String loginStatusKey = 'loginStatusKey';
  static const String loginTimeKey = 'loginTimeKey';
  static const String usernameKey = 'username';
  static const String tokenKey = 'token';

  /// Memeriksa apakah pengguna sedang login.
  static Future<bool> isLoggedIn() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    bool isLoggedIn = prefs.getBool(loginStatusKey) ?? false;

    if (!isLoggedIn) return false;
    if (!await isTokenValid()) {
      await logout();
      return false;
    }

    return true;
  }

  /// Memeriksa apakah token masih valid.
  static Future<bool> isTokenValid() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    String? token = prefs.getString(tokenKey);
    if (token == null) return false;
    return !JwtDecoder.isExpired(token);
  }

  /// Login pengguna dan menyimpan status login.
  static Future<void> login(String username, String token) async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    await prefs.setBool(loginStatusKey, true);
    await prefs.setString(loginTimeKey, DateTime.now().toString());
    await prefs.setString(usernameKey, username);
    await prefs.setString(tokenKey, token);
  }

  /// Mengambil token pengguna yang sedang login.
  static Future<String?> getToken() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    return prefs.getString(tokenKey);
  }

  /// Mengambil username pengguna yang sedang login.
  static Future<String?> getUsername() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    return prefs.getString(usernameKey);
  }

  /// Logout pengguna dan menghapus data terkait login.
  static Future<void> logout() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    await prefs.setBool(loginStatusKey, false);
    await prefs.remove(loginTimeKey);
    await prefs.remove(usernameKey);
    await prefs.remove(tokenKey);
  }

  /// Menghapus semua data di SharedPreferences.
  static Future<void> clearAllData() async {
    SharedPreferences prefs = await
SharedPreferences.getInstance();
    await prefs.clear();
  }
}
```

Gambar 5 auth_manager.dart

Penjelasan:

- Function **isLoggedIn(),** digunakan untuk memeriksa apakah pengguna sedang dalam keadaan login. Pertama, fungsi memeriksa status login pengguna dengan mengambil nilai dari SharedPreferences menggunakan kunci loginStatusKey. Jika status login tidak ditemukan atau bernilai false, maka fungsi mengembalikan false. Jika status login true, fungsi kemudian memeriksa apakah token autentikasi yang tersimpan masih valid dengan memanggil fungsi isTokenValid(). Jika token tidak valid, maka fungsi akan memanggil logout() untuk menghapus status login dan mengembalikan false. Jika status login valid dan token masih berlaku, maka fungsi mengembalikan true, menunjukkan bahwa pengguna sedang login.
- Function **isTokenValid()**, digunakan untuk memeriksa apakah token yang tersimpan masih valid. Fungsi mengambil token yang disimpan di SharedPreferences menggunakan kunci tokenKey. Jika token tidak ditemukan (null), maka fungsi mengembalikan false. Jika token ada, maka fungsi akan menggunakan pustaka jwt_decoder untuk memeriksa apakah token tersebut sudah kedaluwarsa. Fungsi ini mengembalikan true jika token masih valid dan false jika token sudah kedaluwarsa.
- Function **login**, digunakan untuk menyimpan status login pengguna dan data terkait (username dan token) setelah login berhasil. Fungsi ini menerima dua parameter, yaitu username dan token, yang akan disimpan di SharedPreferences. Status login diset menjadi true dengan menyimpan nilai true di kunci loginStatusKey, dan waktu login disimpan menggunakan DateTime.now(). Selain itu, username dan token juga disimpan di dalam SharedPreferences dengan kunci yang sesuai, yaitu usernameKey dan tokenKey. Fungsi ini memastikan bahwa data login pengguna tersimpan untuk memudahkan akses di masa depan.
- Function **getToken(),** digunakan untuk mengambil token pengguna yang sedang login dari SharedPreferences. Fungsi ini mengembalikan token yang disimpan dengan kunci tokenKey, atau null jika token tidak ditemukan. Token ini biasanya digunakan untuk autentikasi dalam melakukan permintaan ke server atau API.
- Function **getUsername,** digunakan untuk mengambil username pengguna yang sedang login dari SharedPreferences. Fungsi ini mengembalikan username yang disimpan dengan kunci usernameKey, atau null jika username tidak ditemukan. Username ini berguna untuk menampilkan nama pengguna di antarmuka atau untuk autentikasi lebih lanjut.
- Function **logout(),** digunakan untuk keluar dari sesi login pengguna dan menghapus semua data yang terkait dengan login. Fungsi ini akan menghapus data login yang disimpan di SharedPreferences dengan menghapus kunci loginStatusKey, loginTimeKey, usernameKey, dan tokenKey. Setelah dipanggil, pengguna dianggap sudah logout dan data login tidak akan tersedia lagi.
- Function **clearAllData(),** digunakan untuk menghapus semua data yang tersimpan di SharedPreferences. Fungsi ini memanggil metode clear() pada SharedPreferences yang akan menghapus seluruh data, termasuk status

login, token, username, dan data lainnya. Fungsi ini berguna untuk melakukan pembersihan data secara menyeluruh, misalnya saat melakukan reset aplikasi atau penghapusan data secara permanen.

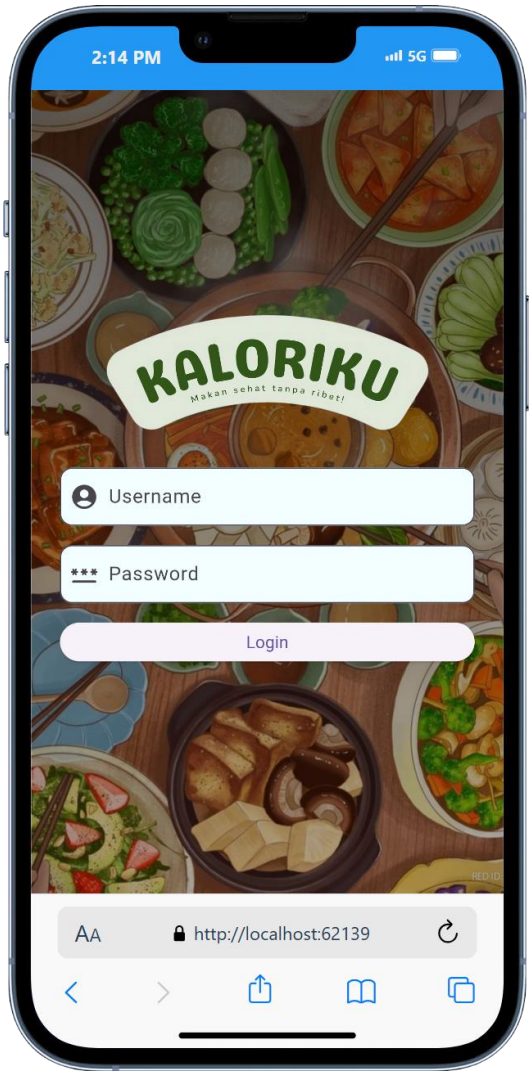
2.4 User Interface

2.4.1 Home_page

Gambar 6 UI home page / landing page

Halaman Kaloriku ini adalah tampilan utama aplikasinya, yang bertujuan untuk memperkenalkan Kaloriku sebagai solusi catering sehat. Di bagian atas, terdapat logo dan tagline yang langsung menyampaikan pesan utama aplikasi ini. Kemudian, ada informasi singkat tentang Kaloriku. Fitur unggulannya adalah solusi praktis untuk hidup lebih sehat dan teratur. Lalu, ada tombol "Login" untuk memudahkan pengguna masuk ke aplikasi.

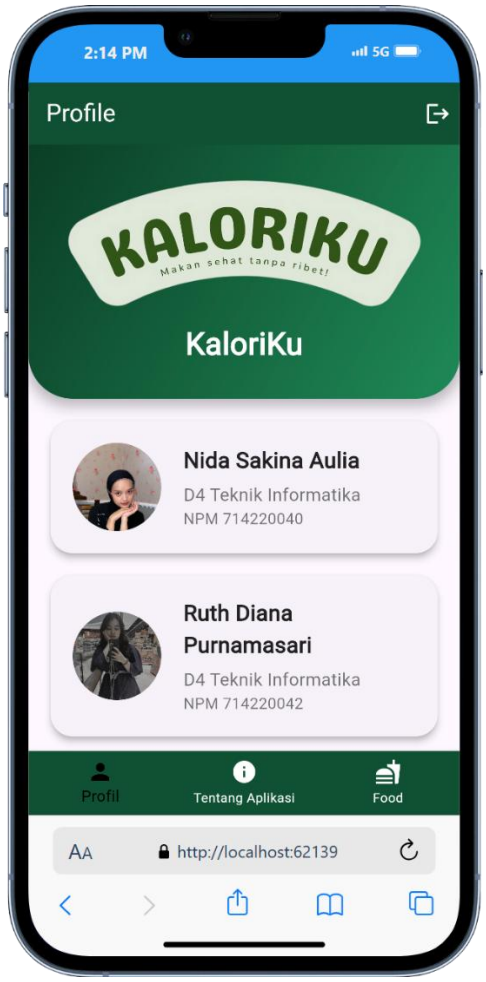
2.4.2 Login_page



Gambar 7 Login_page.dart

Halaman login Kaloriku adalah gerbang utama bagi pengguna untuk mengakses aplikasi. Pengguna akan melihat kolom username dan password untuk memasukkan informasi akun mereka, serta tombol "Login" untuk masuk ke aplikasi untuk memudahkan pengguna masuk dan mengakses fitur-fitur Kaloriku.

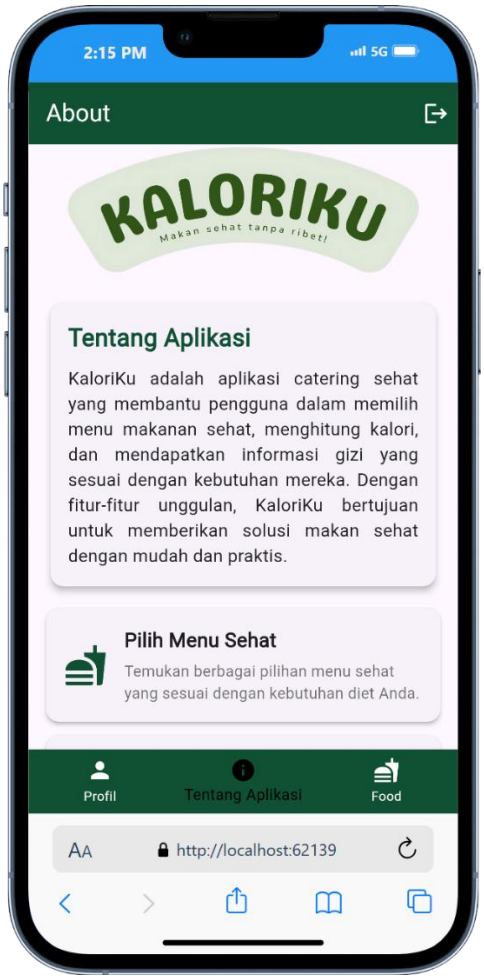
2.4.3 Profile page



Gambar 8 profile_page

Halaman profil ini menampilkan informasi detail mengenai Kaloriku dan tim yang terlibat di baliknya. Di bagian atas terdapat logo Kaloriku dengan tagline "Makan sehat tanpa ribet!" yang menekankan nilai utama aplikasi ini. Kemudian, terdapat informasi tentang Kaloriku. Selanjutnya, profil singkat dari kedua anggota tim pengembang, Di bagian bawah halaman terdapat navigation bar yang memungkinkan pengguna untuk mengakses halaman "Profil", "Tentang Aplikasi", dan "Food". Halaman ini memberikan gambaran jelas tentang siapa yang mengembangkan Kaloriku dan apa tujuan dari aplikasi ini.

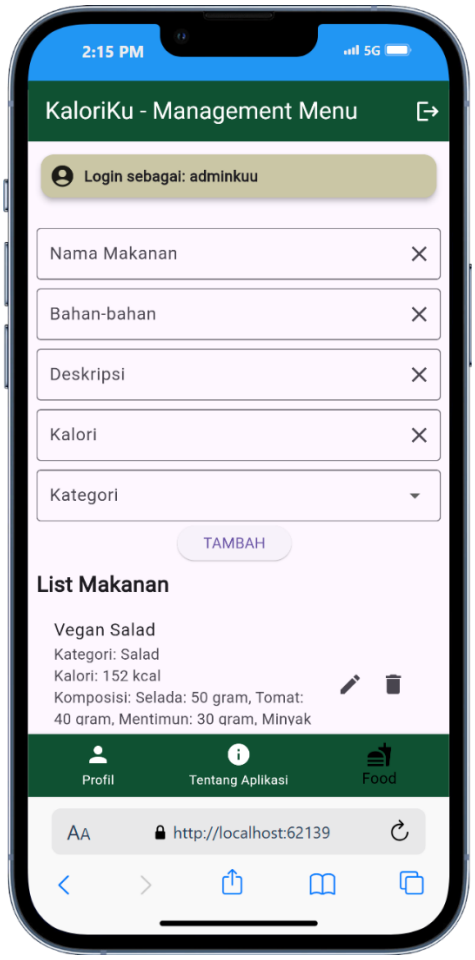
2.4.4 Description page



Gambar 9 description page

Halaman "About" ini memberikan informasi detail mengenai aplikasi Kaloriku. Fitur-fitur unggulan Kaloriku bertujuan untuk memberikan solusi makan sehat yang mudah dan praktis bagi penggunanya. Selain itu, terdapat juga informasi tambahan yang menekankan kemudahan dalam memilih menu sehat melalui aplikasi ini. Secara keseluruhan, halaman "About" ini memberikan pemahaman tentang apa itu Kaloriku, bagaimana cara kerjanya, dan manfaat apa yang ditawarkannya bagi pengguna yang peduli akan kesehatan dan gizi.

2.4.5 Menu page



Gambar 10 menu page

Halaman "Food" atau "Menu" ini berfungsi sebagai pusat pengelolaan dan tampilan daftar makanan yang tersedia di aplikasi Kaloriku. Di bagian atas, admin dapat melihat informasi akun mereka, yaitu "Login sebagai: adminkuu". Terdapat juga beberapa kolom yang dapat diisi untuk menambahkan menu baru, meliputi Nama Makanan, Bahan-bahan, Deskripsi, Kalori, dan Kategori. Setelah data diisi, admin dapat menekan tombol "TAMBAH" untuk menambahkan menu baru ke dalam daftar. Di bagian bawah halaman, ditampilkan daftar makanan yang sudah ada, lengkap dengan informasi seperti kategori, kalori, dan komposisi bahan makanan. Setiap item makanan juga dilengkapi dengan ikon pensil untuk mengedit dan ikon tempat sampah untuk menghapus menu tersebut. Halaman ini memungkinkan admin untuk mengatur dan memelihara informasi menu makanan yang akan ditampilkan kepada pengguna aplikasi Kaloriku.

BAB III
KESIMPULAN

3.1 Kesimpulan Aplikasi

Berdasarkan pada tujuan penelitian mengenai aplikasi KaloriKu, berikut merupakan kesimpulan yang dapat diambil sesuai dengan tujuan yang telah ditetapkan:

1. Aplikasi KaloriKu telah berhasil dikembangkan sebagai platform yang menyediakan informasi terkait pola makan sehat dengan rekomendasi yang dikategorikan berdasarkan preferensi menu yang diinginkan masing-masing pengguna. Menu yang ditampilkan disertai dengan kandungan kalori yang terkandung di masing-masing menu yang ada.
2. Aplikasi ini menawarkan panduan pola makan sehat yang sistematis dan mudah diterapkan, sehingga dapat mendukung pengguna yang memiliki kesibukan atau aktivitas harian yang padat
3. Konten edukatif dalam bentuk artikel yang tersedia di dalam aplikasi membantu meningkatkan pemahaman pengguna mengenai pentingnya pola makan sehat

Dengan fitur-fitur yang tersedia, KaloriKu diharapkan dapat menjadi solusi yang efektif dalam membantu masyarakat menerapkan gaya hidup sehat secara lebih mudah dan terencana.