Bahria University Islamabad

Department of Computer Science

Bachelor in Computer Science

TasteAI: Smart Recipe Recommender App

Supervisor
Respected Sir, Mohsin Javed Butt

Course Field
Mobile Application Development-I

Submitted By:
Hamza Afzal

Enrollment:
01-134222-060

Class/Section: 6-B

Academic Year 2022-2026

Declaration

This is to certify that the thesis titled **TasteAI: Smart Recipe Recommender App** has been authored by project team. It presents the research conducted under the supervision of **Respected Sir, Qazi Mohsin Ijaz** at Institute of Bahria University Islamabad.

To the best of our knowledge, it is an original work, both in terms of research content and narrative, and has not been submitted elsewhere, in part or in full, for a degree. Notions taken over directly or indirectly from other sources have been identified as such.This paper has not previously been presented in an identical or similar form to any other Pakistan or foreign Source.

Hamza Afzal
Enrollment No. 01-134222-060
Bahria University Islamabad

Acknowledgements

All gratitude and thanks to Almighty ALLAH, who gave us courage and wisdom to undertake this task, and it is with the Grace of ALLAH, the Almighty, that the project developer is able to accomplish this task. The project team owe special thanks to our supervisor Sir, Qazi Mohsin Ijaz for his invaluable guidance and insightful ideas, which played a key role in the successful completion of our project.

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
DBMS	Database Management System
UI	User Interface
UX	User Experience
ML	Machine Learning
DOC	Document File Format
TF	TensorFlow
DL	Deep Learning
JWT	JSON Web Token
REST	Representational State Transfer
CRUD	Create, Read, Update, Delete
UI/UX	User Interface/User Experience

Chapter 1

Project Overview

1.1 Functional Requirements

Table 1.1: Github Report Link

S. No.	Github Report Link	Type	Status
1	https://github.com/Hamza2-2/MAD-Lab-Semester-Project	Core	Complete

1.2 Team Members

This project is conducted by three group member's with the below details

Name	Enrollment ID	Section
Hamza Afzal	01-134222-060	6-B
Khawaja Fahad Zia	01-134221-031	6-C

1.3 Introduction

TasteAI is a recipe recommendation app that suggests meals based on a user's preferences and past cooking history. Users can create accounts, save recipes, and recover lost credentials.

The app is built using **Django** for the backend and **Flutter** for the front end. It follows a **Django API architecture**, ensuring smooth communication between the client and server. TasteAI analyzes user input and past recipes to provide relevant meal suggestions.

This mobile app works on both **Android and iOS** with an easy-to-use interface. Users can log their recipes, get AI-based recommendations, and download recipe PDFs for convenience.

With its smart recommendation system and simple design, TasteAI makes it easier for users to find and enjoy new recipes.

1.4 Functional Requirements

Table 1.2: Functional Requirements

S. No.	Functional Requirement	Type	Status
1	Users must be able to create an account by providing details such as name, email, and password for personalized recipe management.	Core	Complete
2	Users must be able to log in securely using their registered email and password to access their saved recipes and preferences.	Core	Complete
3	Users must be able to recover their account via a "Forgot Password" option, using email verification or OTP-based authentication.	Core	Complete
4	Users must be able to input past recipes they have made, including ingredients, name etc. for better recommendation accuracy.	Core	Complete
5	Users may request personalized recipe recommendations based on past ingredients, dietary preferences, and available ingredients at home etc. depending on the features implemented	Core	Complete
6	The app must be fully functional on both Android and iOS devices, ensuring a seamless cross-platform user experience.	Core	Complete

1.5 Scope of your project

Our project team is developing a smart recipe recommendation app called **TasteAI: Smart Recipe Recommender App**. As people look for easier ways to plan meals and try new recipes, our app uses AI to suggest dishes based on ingredients users have, their past cooking history, and preferences.

The app will have a user-friendly interface where users can create accounts, save recipes, and get personalized meal suggestions. It will also allow users to input their own recipes, store them, and generate recipe documents in PDF or DOC format for easy sharing.

With AI-powered recommendations, cross-platform support (Android iOS), and a responsive design, our app will make cooking more convenient and enjoyable.

Initially, it will focus on home cooks, food enthusiasts, and people looking for meal planning ways.

Table 1.3: Project Scope

S. No.	Scope Description
1	**Cross-Platform Mobile Application:** The application will be developed using Flutter, ensuring compatibility on both Android and iOS platforms. It will provide a seamless and responsive experience across different screen sizes and resolutions.
2	**Django-Based Backend with User Management:** The backend will be developed using Django, providing a secure and scalable system to manage user accounts, authentication, and recipe-related data. Features such as user registration, login, and account recovery will be implemented.
3	**AI-Powered Recipe Recommendation System:** An intelligent recommendation system will suggest recipes based on user preferences, past cooking history, and available ingredients. The AI model will continuously improve its recommendations based on user feedback and interactions.
4	**Dedicated Screens for Each Functionality:** The app will have a well-structured and intuitive user interface, with dedicated screens for recipe input, recommendations, saved recipes, and user profile management. The design will follow best UI/UX practices.
5	**User Recipe Input and Storage:** Users will be able to input and store their recipes, including ingredients, cooking steps, and images. The system will allow users to edit, delete, and organize their saved recipes efficiently.

1.6 Tools and Technologies

Table 1.4: Technologies and Tools Used in the Project

Technology/Tool	**Purpose**	**Why?**
Python	Backend programming language	Used for developing the AI-based recipe recommendation system and backend logic.
Django	Web framework	Provides a robust backend structure for handling user authentication, requests, and database management.
TensorFlow/Keras	Machine Learning frameworks	Used for training and deploying the AI model that recommends recipes based on user input.
Flutter	Cross-platform app development	Enables app deployment on both Android and iOS with a single codebase.
REST API	Backend communication	Connects the Flutter frontend with the Django backend for seamless data exchange.
NumPy & Pandas	Data handling	Used for efficient ingredient data processing, storage, and manipulation.
Matplotlib/Seaborn	Data visualization	Helps in displaying recipe trends and user preferences graphically.
Figma	UI/UX Design	Used for designing and prototyping the app interface before implementation.

Continued on next page

Continued from previous page

Technology/Tool	**Purpose**	**Why?**
GitHub	Version control system	Used for project collaboration, tracking changes, and maintaining different versions.

Chapter 2

Front-End Design and Implementation

2.1 Version Control-Access to Project Files

The complete codebase for this recipe recommendation system is organized in a shared Google Drive folder, containing both the Django backend and Flutter frontend components. The backend includes the REST API, database models for recipes, and the recommendation engine that filters recipes based on user-input criteria (ingredients and nutritional constraints). The Flutter app provides the user interface for selecting preferences, viewing recommendations, and saving favorite recipes. Additional documentation covers setup instructions, API usage, and testing guidelines. All project files—source code, assets, and documentation—are available via the below links:

Project File: Click the hyperlink to access the Project File
Project Video: Click the hyperlink to access the Project Video

2.2 Flutter Application Prototype

The system's user interface was developed as a fully functional Flutter application, implementing all navigation flows, interactive components, and platform-specific behaviors. This production-ready prototype validates the design through actual user interactions on both iOS and Android devices, demonstrating responsive layouts, state management, and real-time performance that static mockups cannot replicate. The implementation confirms the usability and technical feasibility of the proposed solution.

2.3 Flutter Application Screen 1 - Login

Table 2.1: Use Case 1 - Login Screen

Use Case ID	1001
Reference	Figure 2.1
Use Case Name	User Login
Actors	User Flutter Application
Purpose	The Login Screen allows registered users to authenticate themselves using their email and password to access the application's features.
Input Elements	1. Email Field: • Text input for user's email address 2. Password Field: • Secure text input for user's password 3. Continue Button: • Submits the login credentials 4. Forgot Password Link: • Navigates to password recovery 5. Sign Up Link: • Navigates to registration screen
Output Elements	1. Success Notification: • User is authenticated and directed to main screen 2. Error Messages: • Invalid email format • Incorrect password • Account not found

Navigation Flow	1. Application Launch: The Flutter app opens, displaying the Login Screen. 2. User enters email and password. 3. User clicks "Continue" to submit credentials. 4. Alternative Flows: • If "Forgot Password" is clicked, navigate to password recovery. • If "Sign Up" is clicked, navigate to registration. 5. Error Handling: • Invalid inputs display appropriate error messages.

Code Implementation

```
import 'package:flutter/material.dart';
import 'package:flutter/gestures.dart';
import 'package:flutter_django_recipes_frontend/forgot_password.dart';
import 'package:flutter_django_recipes_frontend/app.dart';
import 'package:flutter_django_recipes_frontend/services/auth_service.dart';

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

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

class LoginPageState extends State<LoginPage> {
  bool _isLoginView = true;
  bool _isLoading = false;
  bool _isPasswordVisible = false;

  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
```

```
final TextEditingController _firstNameController =
 TextEditingController();
final TextEditingController _lastNameController =
 TextEditingController();

@override
void dispose() {
  _emailController.dispose();
  _passwordController.dispose();
  _firstNameController.dispose();
  _lastNameController.dispose();
  super.dispose();
}

bool _validateEmail() {
  final email = _emailController.text.trim();
  return RegExp(
    r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
  ).hasMatch(email);
}

Future<void> _handleLogin(BuildContext context) async {
  if (!_validateEmail()) {
    _showErrorDialog(context, 'Please enter a valid email');
    return;
  }

  setState(() => _isLoading = true);

  try {
    final result = await AuthService.login(
      email: _emailController.text,
      password: _passwordController.text,
    );

    if (result['success'] == true) {
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (context) => const MainPage()),
      );
```

```
      } else {
        _showErrorDialog(context, result['message'] ?? 'Login failed');
      }
    } catch (e) {
      _showErrorDialog(context, 'An error occurred during login');
    } finally {
      setState(() => _isLoading = false);
    }
  }

  Future<void> _handleRegister(BuildContext context) async {
    if (!_validateEmail()) {
      _showErrorDialog(context, 'Please enter a valid email');
      return;
    }

    if (_firstNameController.text.isEmpty || _lastNameController.text.isEmpty) {
      _showErrorDialog(context, 'First and last name are required');
      return;
    }

    if (_passwordController.text.length < 8) {
      _showErrorDialog(context, 'Password must be at least 8 characters');
      return;
    }

    setState(() => _isLoading = true);

    try {
      final result = await AuthService.register(
        email: _emailController.text,
        password: _passwordController.text,
        firstName: _firstNameController.text,
        lastName: _lastNameController.text,
      );

      if (result['success'] == true) {
```

```
        Navigator.pushReplacement(
          context,
          MaterialPageRoute(builder: (context) => const MainPage)),
        );
      } else {
        _showErrorDialog(context, result['message'] ?? 'Registration failed');
      }
    } catch (e) {
      _showErrorDialog(context, 'An error occurred during registration');
    } finally {
      setState(() => _isLoading = false);
    }
  }

  void _showErrorDialog(BuildContext context, String message) {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        title: const Text(
          "Error",
          style: TextStyle(
            color: Colors.red,
            fontWeight: FontWeight.bold,
          ),
        ),
        content: Text(message),
        actions: [
          TextButton(
            onPressed: () => Navigator.of(context).pop(),
            child: const Text("OK"),
          ),
        ],
      ),
    );
  }

  void _switchView() {
    setState(() {
```

```
      _isLoginView = !_isLoginView;
      _emailController.clear();
      _passwordController.clear();
      if (!_isLoginView) {
        _firstNameController.clear();
        _lastNameController.clear();
      }
    });
  }

  Widget _buildLoginForm(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 20.0,
 vertical: 40.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const Text(
            "Email",
            style: TextStyle(fontWeight: FontWeight.bold, fontSize:
 16.0),
          ),
          const SizedBox(height: 8),
          TextField(
            controller: _emailController,
            keyboardType: TextInputType.emailAddress,
            decoration: _inputDecoration(),
          ),
          const SizedBox(height: 16),
          const Text(
            "Password",
            style: TextStyle(fontWeight: FontWeight.bold, fontSize:
 16.0),
          ),
          const SizedBox(height: 8),
          TextField(
            controller: _passwordController,
            obscureText: !_isPasswordVisible,
            decoration: _inputDecoration(
              suffixIcon: IconButton(
```

```
                icon: Icon(
                  _isPasswordVisible ? Icons.visibility : Icons.visibility_off,
                ),
                onPressed: () => setState(() => _isPasswordVisible = !_isPasswordVisible),
              ),
            ),
          ),
          const SizedBox(height: 8),
          Align(
            alignment: Alignment.centerRight,
            child: TextButton(
              onPressed: () => Navigator.push(
                context,
                MaterialPageRoute(builder: (context) => const ForgotPassword()),
              ),
              child: Text(
                "Forgot Password?",
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  color: Colors.blue.shade600,
                ),
              ),
            ),
          ),
          const SizedBox(height: 24),
          SizedBox(
            width: double.infinity,
            height: 50,
            child: ElevatedButton(
              onPressed: _isLoading ? null : () => _handleLogin(context),
              style: ElevatedButton.styleFrom(
                backgroundColor: Colors.blue,
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(12.0),
                ),
              ),
              child: _isLoading
```

```
                  ? const CircularProgressIndicator(color: Colors.
    white)
                  : const Text(
                      "Log In",
                      style: TextStyle(
                        color: Colors.white,
                        fontWeight: FontWeight.bold,
                        fontSize: 16.0,
                      ),
                    ),
            ),
          ),
          const SizedBox(height: 24),
          Center(
            child: RichText(
              text: TextSpan(
                text: "Don't have an account? ",
                style: TextStyle(
                  color: Colors.grey.shade600,
                  fontWeight: FontWeight.bold,
                ),
                children: [
                  TextSpan(
                    text: "Sign Up",
                    style: TextStyle(
                      color: Colors.blue.shade600,
                      fontWeight: FontWeight.bold,
                    ),
                    recognizer: TapGestureRecognizer()..onTap =
    _switchView,
                  ),
                ],
              ),
            ),
          ),
        ],
      ),
    );
  }

  Widget _buildRegisterForm(BuildContext context) {
```

```
    return Padding(
      padding: const EdgeInsets.symmetric(horizontal: 20.0,
   vertical: 20.0),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const Text(
            "First Name",
            style: TextStyle(fontWeight: FontWeight.bold, fontSize:
    16.0),
          ),
          const SizedBox(height: 8),
          TextField(
            controller: _firstNameController,
            decoration: _inputDecoration(),
          ),
          const SizedBox(height: 16),
          const Text(
            "Last Name",
            style: TextStyle(fontWeight: FontWeight.bold, fontSize:
    16.0),
          ),
          const SizedBox(height: 8),
          TextField(
            controller: _lastNameController,
            decoration: _inputDecoration(),
          ),
          const SizedBox(height: 16),
          const Text(
            "Email",
            style: TextStyle(fontWeight: FontWeight.bold, fontSize:
    16.0),
          ),
          const SizedBox(height: 8),
          TextField(
            controller: _emailController,
            keyboardType: TextInputType.emailAddress,
            decoration: _inputDecoration(),
          ),
          const SizedBox(height: 16),
```

```
        const Text(
          "Password",
          style: TextStyle(fontWeight: FontWeight.bold, fontSize: 16.0),
        ),
        const SizedBox(height: 8),
        TextField(
          controller: _passwordController,
          obscureText: !_isPasswordVisible,
          decoration: _inputDecoration(
            suffixIcon: IconButton(
              icon: Icon(
                _isPasswordVisible ? Icons.visibility : Icons.visibility_off,
              ),
              onPressed: () => setState(() => _isPasswordVisible = !_isPasswordVisible),
            ),
          ),
        ),
        const SizedBox(height: 24),
        SizedBox(
          width: double.infinity,
          height: 50,
          child: ElevatedButton(
            onPressed: _isLoading ? null : () => _handleRegister(context),
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.blue,
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(12.0),
              ),
            ),
            child: _isLoading
                ? const CircularProgressIndicator(color: Colors.white)
                : const Text(
                    "Register",
                    style: TextStyle(
                      color: Colors.white,
                      fontWeight: FontWeight.bold,
```

```
                        fontSize: 16.0,
                      ),
                    ),
            ),
          ),
          const SizedBox(height: 24),
          Center(
            child: RichText(
              text: TextSpan(
                text: "Already have an account? ",
                style: TextStyle(
                  color: Colors.grey.shade600,
                  fontWeight: FontWeight.bold,
                ),
                children: [
                  TextSpan(
                    text: "Log In",
                    style: TextStyle(
                      color: Colors.blue.shade600,
                      fontWeight: FontWeight.bold,
                    ),
                    recognizer: TapGestureRecognizer()..onTap = _switchView,
                  ),
                ],
              ),
            ),
          ),
        ],
      ),
    );
  }

  InputDecoration _inputDecoration({Widget? suffixIcon}) {
    return InputDecoration(
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12.0),
        borderSide: const BorderSide(color: Colors.black54, width: 1.0),
      ),
      focusedBorder: OutlineInputBorder(
```

```
        borderRadius: BorderRadius.circular(12.0),
        borderSide: const BorderSide(color: Colors.blue, width: 2.0),
      ),
      enabledBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12.0),
        borderSide: BorderSide(
          color: Colors.grey.shade400,
          width: 1.0,
        ),
      ),
      contentPadding: const EdgeInsets.symmetric(horizontal: 16, vertical: 14),
      suffixIcon: suffixIcon,
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            GestureDetector(
              onTap: _isLoginView ? null : () => setState(() => _isLoginView = true),
              child: Container(
                padding: const EdgeInsets.symmetric(vertical: 8.0, horizontal: 24.0),
                decoration: _isLoginView
                    ? const BoxDecoration(
                        border: Border(
                          bottom: BorderSide(width: 2.0, color: Colors.blue),
                        ),
                      )
                    : null,
                child: Text(
                  "Log In",
                  style: TextStyle(
```

```
                    color: _isLoginView ? Colors.blue : Colors.black54,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
            ),
            GestureDetector(
              onTap: _isLoginView ? () => setState(() => _isLoginView = false) : null,
              child: Container(
                padding: const EdgeInsets.symmetric(vertical: 8.0, horizontal: 24.0),
                decoration: !_isLoginView
                    ? const BoxDecoration(
                        border: Border(
                          bottom: BorderSide(width: 2.0, color: Colors.blue),
                        ),
                      )
                    : null,
                child: Text(
                  "Sign Up",
                  style: TextStyle(
                    color: !_isLoginView ? Colors.blue : Colors.black54,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
      body: SingleChildScrollView(
        child: _isLoginView ? _buildLoginForm(context) : _buildRegisterForm(context),),);}}
```

Listing 2.1: Login Code

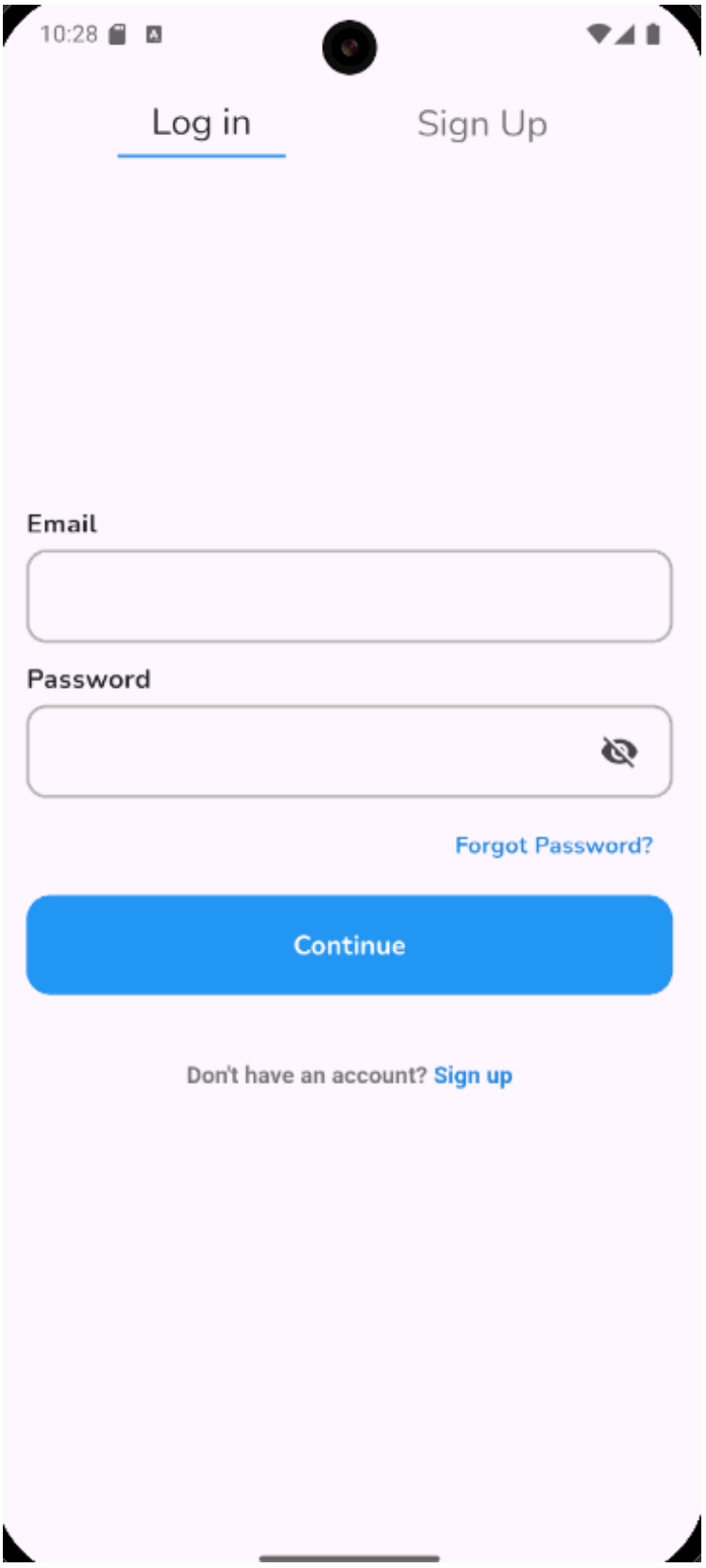

FIGURE 2.1: Login Screen

2.4 Flutter Application Screen 2 - Sign Up

Table 2.2: Use Case 2 - User Registration

Use Case ID	1002
Reference	Figure 6.2
Use Case Name	User Registration
Actors	New User Flutter Application
Purpose	This screen allows new users to create an account by providing their personal information and credentials to access the application's features.
Input Elements	1. First Name Field: • Text input for user's first name 2. Last Name Field: • Text input for user's last name 3. Email Field: • Text input for user's email address 4. Password Field: • Secure text input for account password 5. Continue Button: • Submits the registration form 6. Log In Link: • Navigates to login screen for existing users
Output Elements	1. Success Notification: • Account created confirmation • Automatic login and redirection to main screen 2. Error Messages: • Missing required fields • Invalid email format • Password requirements not met • Email already registered

Navigation Flow	1. User accesses the Sign Up screen from Login. 2. User enters first name, last name, email, and password. 3. User clicks "Continue" to submit registration. 4. Alternative Flow: • If "Log In" is clicked, navigate to login screen. 5. Error Handling: • Invalid inputs display appropriate error messages. 6. Success Flow: • After successful registration, user is authenticated and directed to application's main screen.

Code Implementation

```
import "package:flutter/material.dart";
import 'package:google_fonts/google_fonts.dart';
import 'package:flutter_django_recipes_frontend/app.dart';
import 'package:flutter_django_recipes_frontend/services/
    auth_service.dart';

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

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

class SignUpPageState extends State<SignUpPage> {
  bool loggingInRn = false; // Loading indicator during
    registration
  bool _isPassword = true;  // Toggle password visibility
  TextEditingController emailController = TextEditingController();
  TextEditingController passwordController = TextEditingController
    ();
  TextEditingController firstNameController = TextEditingController
    ();
  TextEditingController lastNameController = TextEditingController
    ();
```

```

  // Validates email format
  bool emailValidation() {
    String email = emailController.text.trim();
    if (email.length < 5 || email.length > 50) {
      return false;
    }
    return RegExp(
      r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
    ).hasMatch(email);
  }

  // Handles registration submission
  Future<void> submitRegisterInformation(BuildContext context) async {
    if (!emailValidation()) {
      showErrorDialog(context, 'Invalid email format');
      return;
    }
    if (firstNameController.text.isEmpty || lastNameController.text.isEmpty) {
      showErrorDialog(context, 'First name and last name cannot be empty');
      return;
    }
    if (passwordController.text.length < 8) {
      showErrorDialog(context, 'Password must be at least 8 characters long');
      return;
    }

    final result = await AuthService.register(
      email: emailController.text,
      password: passwordController.text,
      firstName: firstNameController.text,
      lastName: lastNameController.text,
    );

    if (result['success']) {
      Navigator.pushReplacement(
        context,
```

```
        MaterialPageRoute(builder: (context) => MainPage()),
      );
    } else {
      showErrorDialog(context, result['message']);
    }
  }

  // Shows error dialog
  void showErrorDialog(BuildContext context, String message) {
    showDialog(
      context: context,
      builder: (context) {
        return AlertDialog(
          title: Text(
            "Error",
            style: TextStyle(
              color: Colors.red,
              fontWeight: FontWeight.bold,
              fontSize: 20,
            ),
          ),
          content: Text(message),
          actions: [
            TextButton(
              onPressed: () => Navigator.of(context).pop(),
              child: Text("OK"),
            ),
          ],
        );
      },
    );
  }

  @override
  void dispose() {
    emailController.dispose();
    passwordController.dispose();
    firstNameController.dispose();
    lastNameController.dispose();
    super.dispose();
  }
```

```

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Sign Up", style: GoogleFonts.nunito()),
      ),
      body: Padding(
        padding: EdgeInsets.symmetric(horizontal: 20.0, vertical: 20),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            // First Name Input
            Text(
              "First Name",
              style: GoogleFonts.nunito(
                fontWeight: FontWeight.bold,
                fontSize: 16.0,
              ),
            ),
            Padding(padding: EdgeInsets.all(2.0)),
            TextField(
              controller: firstNameController,
              decoration: InputDecoration(
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                focusedBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                enabledBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                  borderSide: BorderSide(
                    width: 2.0,
                    color: const Color.fromARGB(255, 184, 183, 183),
                  ),
                ),
              ),
            ),
            Padding(padding: EdgeInsets.all(5.0)),
```

```

            // Last Name Input
            Text(
              "Last Name",
              style: GoogleFonts.nunito(
                fontWeight: FontWeight.bold,
                fontSize: 16.0,
              ),
            ),
            Padding(padding: EdgeInsets.all(2.0)),
            TextField(
              controller: lastNameController,
              decoration: InputDecoration(
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                ),
                focusedBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                ),
                enabledBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                  borderSide: BorderSide(
                    width: 2.0,
                    color: const Color.fromARGB(255, 184, 183, 183)
,
                  ),
                ),
              ),
            ),
            Padding(padding: EdgeInsets.all(5.0)),

            // Email Input
            Text(
              "Email",
              style: GoogleFonts.nunito(
                fontWeight: FontWeight.bold,
                fontSize: 16.0,
              ),
            ),
            Padding(padding: EdgeInsets.all(2.0)),
            TextField(
```

```
              controller: emailController,
              decoration: InputDecoration(
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                ),
                focusedBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                ),
                enabledBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                  borderSide: BorderSide(
                    width: 2.0,
                    color: const Color.fromARGB(255, 184, 183, 183),
                  ),
                ),
              ),
            ),
            Padding(padding: EdgeInsets.all(5.0)),

            // Password Input
            Text(
              "Password",
              style: GoogleFonts.nunito(
                fontWeight: FontWeight.bold,
                fontSize: 16.0,
              ),
            ),
            Padding(padding: EdgeInsets.all(2.0)),
            TextField(
              controller: passwordController,
              obscureText: _isPassword,
              decoration: InputDecoration(
                suffixIcon: Padding(
                  padding: const EdgeInsets.symmetric(horizontal: 8.0),
                  child: IconButton(
                    icon: Icon(
                      _isPassword ? Icons.visibility_off : Icons.visibility,
                    ),
```

```
                    onPressed: () => setState(() => _isPassword = !_isPassword),
                  ),
                ),
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                ),
                enabledBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                  borderSide: BorderSide(
                    width: 2.0,
                    color: const Color.fromARGB(255, 184, 183, 183),
                ),
                focusedBorder: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12.0),
                  borderSide: BorderSide(width: 2.0, color: Colors.black54),
                ),
              ),
            ),
            Padding(padding: EdgeInsets.symmetric(vertical: 10.0)),

            // Register Button
            SizedBox(height: 10),
            OutlinedButton(
              style: OutlinedButton.styleFrom(
                backgroundColor: Colors.blue,
                minimumSize: Size(double.infinity, 60.0),
                side: BorderSide(color: Colors.blue),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(15.0),
                ),
              ),
              onPressed: () async {
                setState(() => loggingInRn = true);
                await submitRegisterInformation(context);
                setState(() => loggingInRn = false);
              },
              child: loggingInRn
                  ? CircularProgressIndicator()
```

```
                  : Text(
                      "Continue",
                      style: GoogleFonts.nunito(
                        color: Colors.white,
                        fontWeight: FontWeight.bold,
                        fontSize: 16.0,
                      ),
                    ),
            ),

            // Login Redirect
            Padding(padding: EdgeInsets.all(20.0)),
            Center(
              child: RichText(
                text: TextSpan(
                  text: "Already have an account? ",
                  style: TextStyle(
                    color: Colors.grey.shade600,
                    fontWeight: FontWeight.bold,
                  ),
                  children: [
                    TextSpan(
                      text: "Log in",
                      style: TextStyle(
                        color: Colors.blue.shade600,
                        fontWeight: FontWeight.bold,
                      ),
                      recognizer: TapGestureRecognizer()
                        ..onTap = () {
                          Navigator.pushReplacement(
                            context,
                            MaterialPageRoute(builder: (context) =>
    SignInPage()),
                          );
                        },
                    ),
                  ],
                ),
              ),),],),),);}}
```

Listing 2.2: Sign Up Code

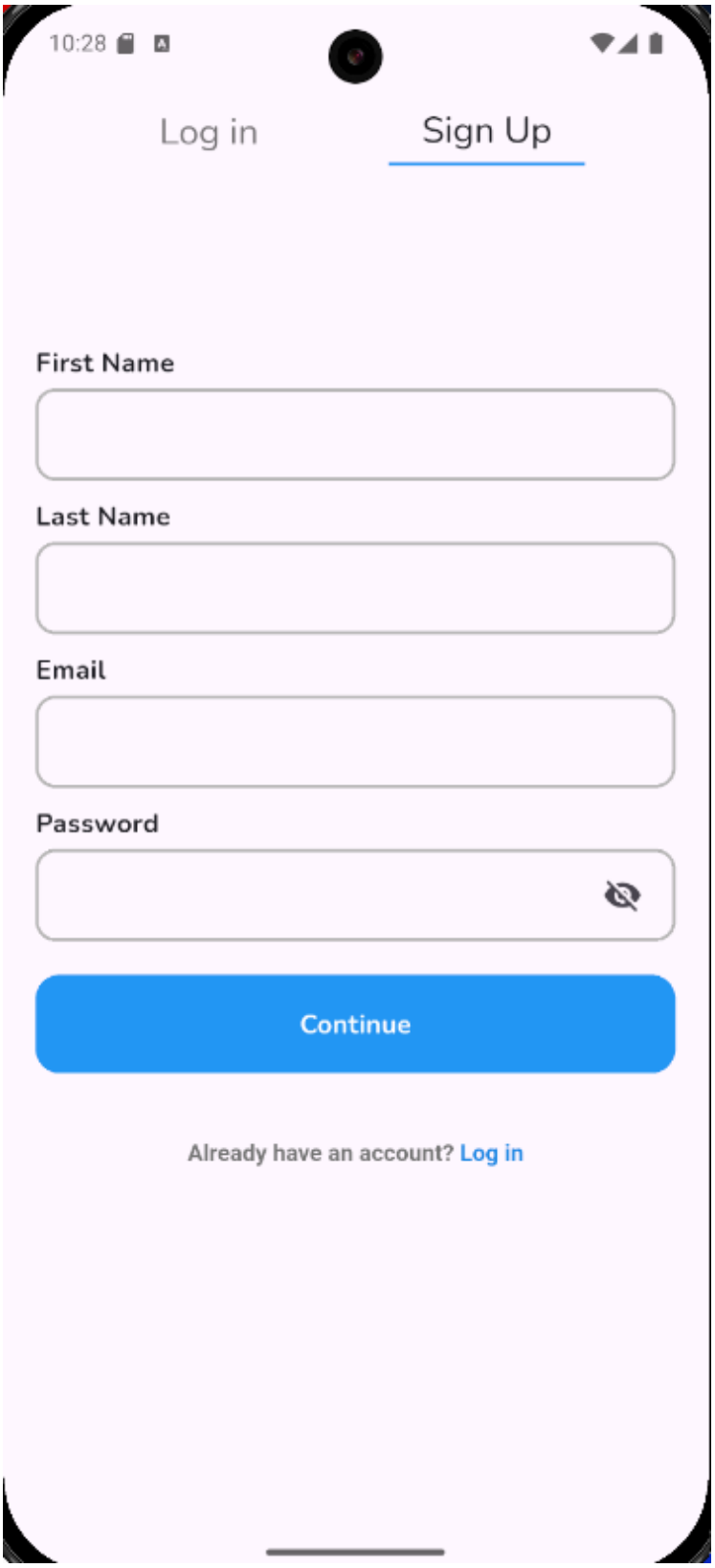

FIGURE 2.2: Sign Up Screen

2.5 Flutter Application Screen 3 - Forgot Password

Table 2.3: Use Case 3 - Password Recovery

Use Case ID	1003
Reference	Figure 6.3
Use Case Name	Password Recovery
Actors	User Flutter Application
Purpose	This screen allows users who have forgotten their password to initiate a password reset process by providing their registered email address.
Preconditions	1. User must have a registered account. 2. User must have access to the registered email.
Input Elements	1. Email Field: • Text input for user's registered email address 2. Reset Password Button: • Submits the password reset request 3. Back Button (Clicked): • Returns to login screen
Output Elements	1. Success Notification: • Confirmation email sent message 2. Error Messages: • Email not registered • Invalid email format • Network connection issues

Navigation Flow	1. User accesses the Forgot Password screen from Login. 2. User enters their registered email address. 3. User clicks "Reset Password" to submit request. 4. System sends password reset link to the email. 5. Alternative Flows: • If email isn't registered, show error message • If invalid email format, show validation error 6. Success Flow: • User receives email with password reset instructions • After reset, user can return to login screen

Code Implementation

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

import 'package:flutter_django_recipes_frontend/otp.dart';
import 'package:flutter_django_recipes_frontend/services/
    auth_service.dart';

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

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

class ForgotPasswordState extends State<ForgotPassword> {
  @override
  initState() {
    super.initState();
  }

  final TextEditingController _emailController =
    TextEditingController();

  Future<void> _sendResetPasswordEmail(BuildContext context) async{
    if(_emailController.text.isEmpty){
      showErrorDialog(context, "Please enter your email");
```

```
    }
    final String email = _emailController.text.trim();
    final bool? result = await AuthService.sendResetPasswordEmail(email);
    if (result == null){
      showErrorDialog(context, "Failed to send reset password email");
    } else if (result) {
      Navigator.push(context, MaterialPageRoute(builder: (context) => EmailVerificationScreen(email: email)));
    } else {
      showErrorDialog(context, "User does not exist");
    }
  }

  void showErrorDialog(BuildContext context, String message) {
    setState(() {
      showDialog(
        context: context,
        builder: (context) {
          return AlertDialog(
            title: Text(
              "Error",
              style: TextStyle(
                color: Colors.red,
                fontWeight: FontWeight.bold,
                fontSize: 20,
              ),
            ),
            content: Text(message),
            actions: [
              TextButton(
                onPressed: () {
                  Navigator.of(context).pop();
                },
                child: Text("OK"),
              ),
            ],
          );
        },
      );
```

```
    });
  }

  @override
  void dispose() {
    _emailController.dispose();
    super.dispose();
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: ListView(
        physics: NeverScrollableScrollPhysics(),
        children: [
          Padding(
            padding: EdgeInsets.all(20.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  "Forgot Password",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    fontSize: 24.0,
                  ),
                ),
                Padding(padding: EdgeInsets.all(3.0)),
                Text(
                  "Please enter your email to reset the password",
                  style: TextStyle(
                    fontSize: 16.0,
                  ),
                ),
                Padding(padding: EdgeInsets.all(8.0)),
                Text(
                  "Email",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    fontSize: 16.0,
```

```
                  ),
                ),
                Padding(padding: EdgeInsets.all(2.0)),
                TextField(
                  controller: _emailController,
                  decoration: InputDecoration(
                    border: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(12.0),
                      borderSide: BorderSide(color: Colors.black54,
    width: 2.0),
                    ),
                    focusedBorder: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(12.0),
                      borderSide: BorderSide(color: Colors.black54,
    width: 2.0),
                    ),
                    enabledBorder: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(12.0),
                      borderSide: BorderSide(
                        width: 2.0,
                        color: const Color.fromARGB(255, 184, 183,
   183),
                      ),
                    ),
                  ),
                ),
                Padding(padding: EdgeInsets.all(10.0)),
                OutlinedButton(
                  style: OutlinedButton.styleFrom(
                    backgroundColor: Colors.blue,
                    minimumSize: Size(double.infinity, 60.0),
                    side: BorderSide(color: Colors.blue),
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(15.0),
                    ),
                  ),
                  onPressed: () async => await
   _sendResetPasswordEmail(context),
                  child: Text(
                    "Reset Password",
                    style: TextStyle(
```

```
                        color: Colors.white,
                        fontWeight: FontWeight.bold,
                        fontSize: 16.0,
                      ),
                    ),
                  ),
                ],
              ),
            ),
          ],
        ),
      );
    }
}
```

Listing 2.3: Forgot Password Code

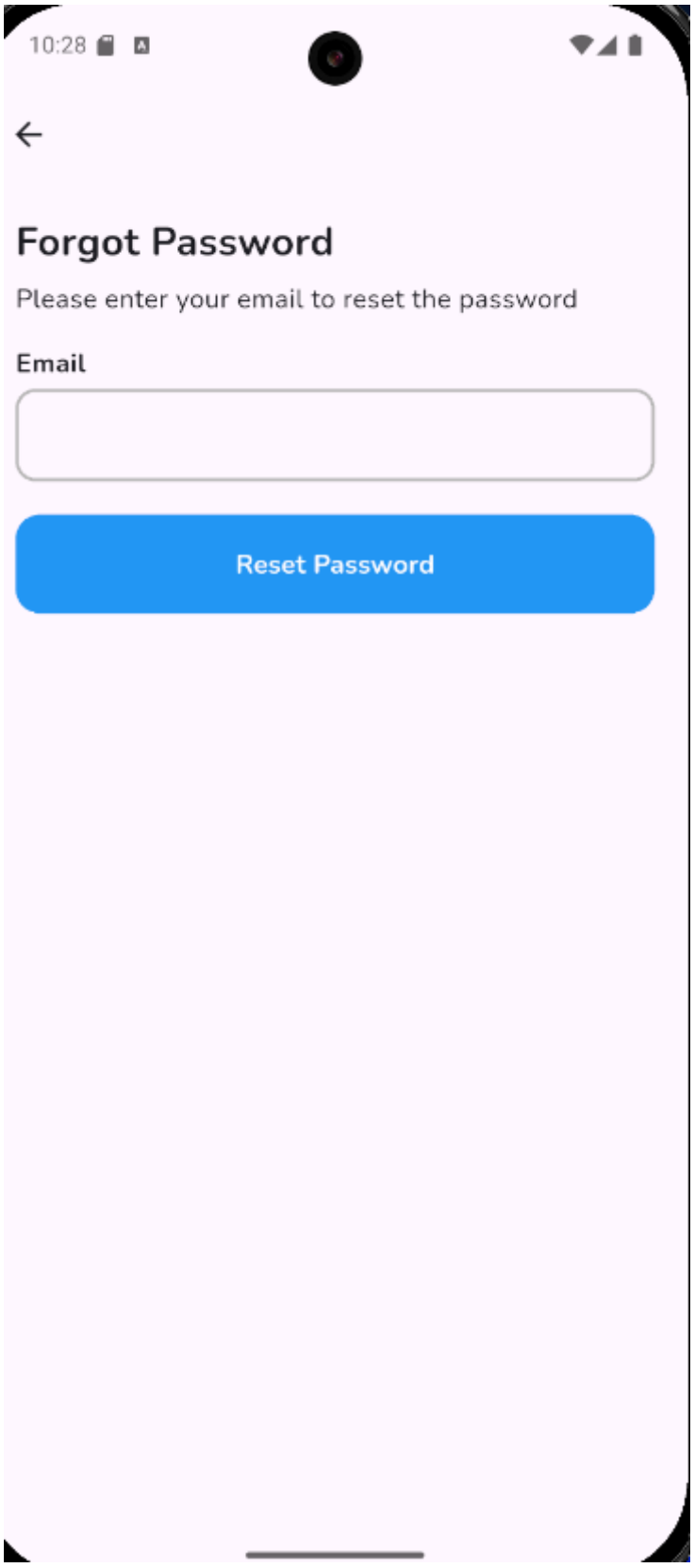

FIGURE 2.3: Forgot Password Screen

2.6 Flutter Application Screen 4 - Password Reset Verification

Table 2.4: Use Case 4 - Password Reset Verification

Use Case ID	1004
Reference	Figure 6.4
Use Case Name	Password Reset Verification
Actors	User Flutter Application Email Service
Purpose	This screen verifies the user's identity through a 5-digit code sent to their email, completing the password reset process.
Preconditions	1. User must have requested password reset. 2. User must have received the verification email. 3. The 5-digit code must be valid and unexpired.
Input Elements	1. Verification Code Field: • Input for 5-digit verification code from email 2. Verify Code Button: • Submits the code for validation 3. Resend Email Link: • Re-sends verification code if not received 4. Back Button (Clicked): • Returns to previous screen
Output Elements	1. Success Notification: • Code verified, proceed to password reset screen 2. Error Messages: • Invalid code • Expired code • Too many attempts 3. Resend Confirmation: • New code sent notification

Navigation Flow	1. User arrives from password reset request screen. 2. User checks email for 5-digit verification code. 3. User enters the code in the verification field. 4. User clicks "Verify Code" to submit. 5. Alternative Flows: • If code is invalid/expired, show error message • If "Resend email" clicked, send new code 6. Success Flow: • After successful verification, navigate to password reset screen 7. Failure Flow: • After multiple failures, may lock account temporarily

Code Implementation

```
import 'package:flutter/material.dart';
import 'package:flutter_django_recipes_frontend/set_new_password.dart';
import 'package:flutter_django_recipes_frontend/services/auth_service.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

class EmailVerificationScreen extends StatefulWidget {
  const EmailVerificationScreen({super.key, required this.email});
  final String email;

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

class _EmailVerificationScreenState extends State<EmailVerificationScreen> {
  final List<TextEditingController> _controllers = List.generate(
    5,
    (_) => TextEditingController(),
```

```
  );
  bool _isButtonEnabled = false;
  bool _isLoading = false;
  String? _errorMessage;

  @override
  void initState() {
    super.initState();
    for (var controller in _controllers) {
      controller.addListener(_checkIfAllFieldsFilled);
    }
  }

  void _checkIfAllFieldsFilled() {
    setState(() {
      _isButtonEnabled = _controllers.every(
        (controller) => controller.text.isNotEmpty,
      );
    });
  }

  Future<void> _verifyOTP() async {
    setState(() {
      _isLoading = true;
      _errorMessage = null;
    });

    try {
      final otp = _controllers.map((c) => c.text).join();
      final csrfData = await AuthService.getCsrfToken();

      if (csrfData == null) {
        setState(() {
          _errorMessage = 'Failed to get CSRF token';
          _isLoading = false;
        });
        return;
      }

      final client = http.Client();
      final response = await client.post(
```

```
        Uri.parse('${AuthService.baseUrl}/verifyOTP/'),
        headers: {
          'Content-Type': 'application/json',
          'X-CSRFToken': csrfData['csrfToken'],
          'Cookie': csrfData['csrfCookie'],
        },
        body: json.encode({
          'email': widget.email,
          'otp': otp,
        }),
      );

      final responseData = json.decode(response.body);
      client.close();

      if (response.statusCode == 200) {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => SetNewPassword(email: widget.email),
          ),
        );
      } else {
        setState(() {
          _errorMessage = responseData['message'] ?? 'Verification failed';
        });
      }
    } catch (e) {
      setState(() {
        _errorMessage = 'An error occurred. Please try again.';
      });
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }

  Future<void> _resendOTP() async {
```

```
    setState(() {
      _isLoading = true;
      _errorMessage = null;
    });

    try {
      final success = await AuthService.sendResetPasswordEmail(
  widget.email);
      if (success != true) {
        setState(() {
          _errorMessage = 'Failed to resend OTP';
        });
      }
    } catch (e) {
      setState(() {
        _errorMessage = 'An error occurred. Please try again.';
      });
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }

  @override
  void dispose() {
    for (var controller in _controllers) {
      controller.dispose();
    }
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(backgroundColor: Colors.transparent),
      body: Padding(
        padding: const EdgeInsets.symmetric(horizontal: 24.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
```

```
            Text(
              'Check your email',
              style: TextStyle(fontSize: 24, fontWeight: FontWeight
    .bold),
            ),
            SizedBox(height: 8),
            Text(
              'We sent a reset link to ${widget.email}\nEnter 5
    digit code that is mentioned in the email',
              style: TextStyle(fontSize: 16, color: Colors.grey),
            ),
            SizedBox(height: 32),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: List.generate(5, (index) {
                return SizedBox(
                  width: 50,
                  height: 50,
                  child: TextField(
                    controller: _controllers[index],
                    maxLength: 1,
                    keyboardType: TextInputType.number,
                    textAlign: TextAlign.center,
                    decoration: InputDecoration(
                      counterText: '',
                      border: OutlineInputBorder(
                        borderRadius: BorderRadius.circular(8),
                      ),
                    ),
                    onChanged: (value) {
                      if (value.length == 1 && index < 4) {
                        FocusScope.of(context).nextFocus();
                      }
                    },
                  ),
                );
              }),
            ),
            if (_errorMessage != null)
              Padding(
                padding: const EdgeInsets.only(top: 16.0),
```

```
                child: Text(
                  _errorMessage!,
                  style: TextStyle(color: Colors.red),
                ),
              ),
            SizedBox(height: 32),
            SizedBox(
              width: double.infinity,
              child: ElevatedButton(
                onPressed: _isButtonEnabled && !_isLoading
                    ? _verifyOTP
                    : null,
                style: ElevatedButton.styleFrom(
                  backgroundColor: _isButtonEnabled && !_isLoading
                      ? Colors.blue
                      : Colors.grey[300],
                  padding: EdgeInsets.symmetric(vertical: 16),
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(8),
                  ),
                ),
                child: _isLoading
                    ? CircularProgressIndicator(color: Colors.white
  )
                    : Text(
                        'Verify Code',
                        style: TextStyle(
                          fontSize: 16,
                          color: _isButtonEnabled && !_isLoading
                              ? Colors.white
                              : Colors.grey,
                        ),
                      ),
              ),
            ),
            SizedBox(height: 16),
            Center(
              child: TextButton(
                onPressed: _isLoading ? null : _resendOTP,
                child: Text(
                  "Haven't got the email yet? Resend email",
```

```
                    style: TextStyle(color: Colors.blue),
                  ),
                ),
              ),
            ],
          ),
        ),
      );
    }
}
```

Listing 2.4: Password Reset Verification Code

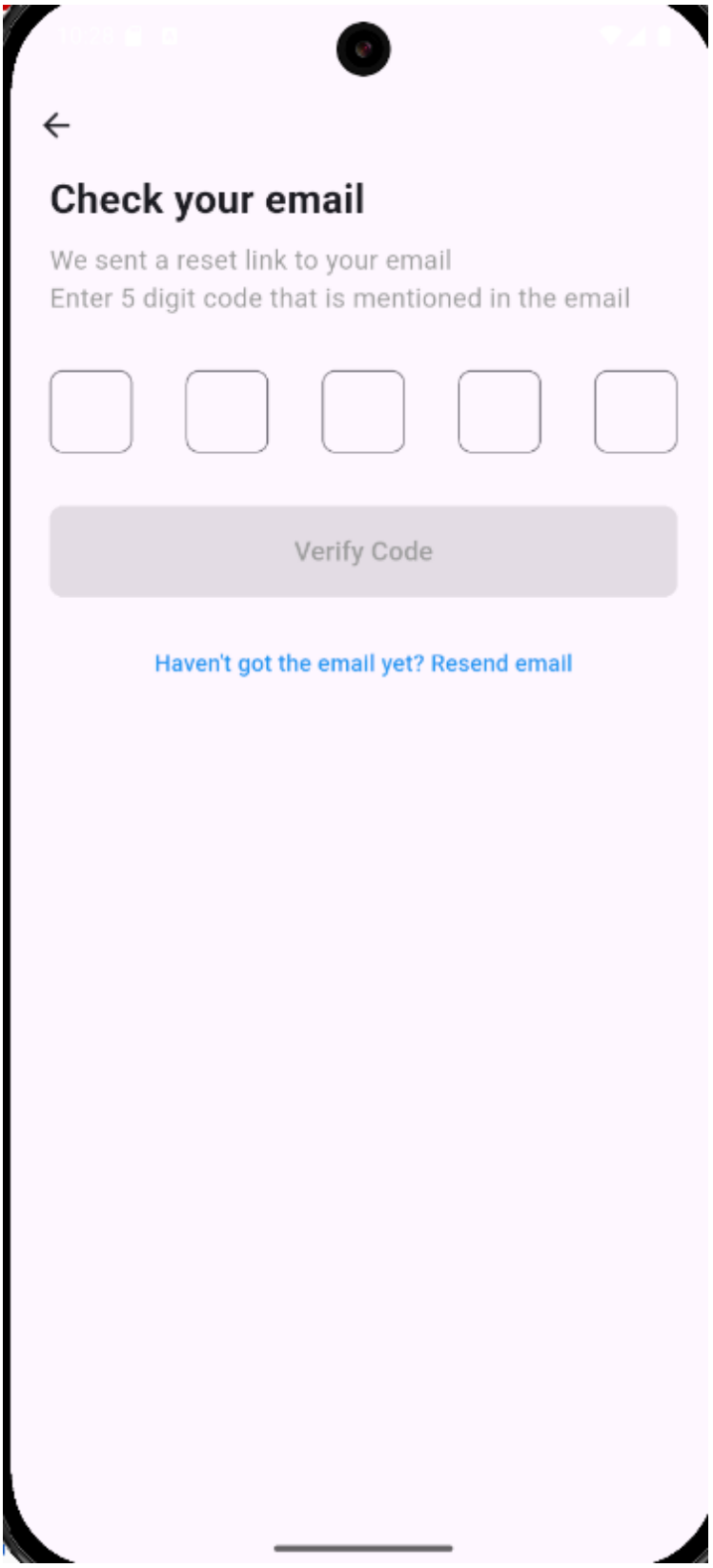

FIGURE 2.4: Password Reset Verification Screen

2.7 Flutter Application Screen 5 - Set New Password

Table 2.5: Use Case 5 - Password Update

Use Case ID	1005
Reference	Figure 6.5
Use Case Name	Password Update
Actors	User Flutter Application Authentication Service
Purpose	This screen allows users to set a new password after successful verification, ensuring account security by requiring a password different from previous ones.
Preconditions	1. User must have completed identity verification. 2. User must have valid access to this password reset flow.
Input Elements	1. Password Field: • Secure input for new password 2. Confirm Password Field: • Re-entry of new password for verification 3. Update Password Button: • Submits the new password 4. Password Strength Indicator (Clicked): • Visual feedback on password complexity
Output Elements	1. Success Notification: • Password updated confirmation • Automatic redirection to login screen 2. Error Messages: • Passwords don't match • Password same as previous • Invalid password format

Navigation Flow	1. User arrives from successful verification screen. 2. User enters new password in both fields. 3. User clicks "Update Password" to submit. 4. System validates new password requirements: • Checks password strength • Verifies password match • Compares with previous passwords 5. Success Flow: • Password updated • User redirected to login screen 6. Failure Flow: • Error messages shown for invalid inputs 7. Security Consideration: • Invalid attempts may trigger security timeout

Code Implementation

```
import 'package:flutter/material.dart';
import 'package:flutter_django_recipes_frontend/services/
    auth_service.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

class SetNewPassword extends StatefulWidget {
  const SetNewPassword({super.key, required this.email});
  final String email;

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

class SetNewPasswordState extends State<SetNewPassword> {
  final _passwordController = TextEditingController();
  final _confirmPasswordController = TextEditingController();
  bool _isPassword = true;
  bool _isPassword2 = true;
  bool _isLoading = false;
  String? _errorMessage;
```

```
21
22  Future<void> _updatePassword() async {
23    if (_passwordController.text.isEmpty ||
24        _confirmPasswordController.text.isEmpty) {
25      setState(() {
26        _errorMessage = 'Please fill in all fields';
27      });
28      return;
29    }
30
31    if (_passwordController.text != _confirmPasswordController.text
    ) {
32      setState(() {
33        _errorMessage = 'Passwords do not match';
34      });
35      return;
36    }
37
38    setState(() {
39      _isLoading = true;
40      _errorMessage = null;
41    });
42
43    try {
44      final csrfData = await AuthService.getCsrfToken();
45      if (csrfData == null) {
46        setState(() {
47          _errorMessage = 'Failed to get CSRF token';
48          _isLoading = false;
49        });
50        return;
51      }
52
53      final client = http.Client();
54      final response = await client.post(
55        Uri.parse('${AuthService.baseUrl}/updatePassword/'),
56        headers: {
57          'Content-Type': 'application/json',
58          'X-CSRFToken': csrfData['csrfToken'],
59          'Cookie': csrfData['csrfCookie'],
60        },
```

```
        body: json.encode({
          'email': widget.email,
          'new_password': _passwordController.text,
        }),
      );

      final responseData = json.decode(response.body);
      client.close();

      if (response.statusCode == 200) {
        Navigator.popUntil(context, (route) => route.isFirst);
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content: Text('Password updated successfully!'),
            backgroundColor: Colors.green,
          ),
        );
      } else {
        setState(() {
          _errorMessage = responseData['message'] ?? 'Failed to
  update password';
        });
      }
    } catch (e) {
      setState(() {
        _errorMessage = 'An error occurred. Please try again.';
      });
    } finally {
      setState(() {
        _isLoading = false;
      });
    }
  }

  @override
  void dispose() {
    _passwordController.dispose();
    _confirmPasswordController.dispose();
    super.dispose();
  }

```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(),
      body: ListView(
        physics: NeverScrollableScrollPhysics(),
        children: [
          Padding(
            padding: EdgeInsets.all(20.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.start,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  "Set a new password",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    fontSize: 24.0,
                  ),
                ),
                Padding(padding: EdgeInsets.all(3.0)),
                Text(
                  "Create a new password. Ensure it differs from
 previous ones for security",
                  style: TextStyle(fontSize: 16.0),
                ),
                Padding(padding: EdgeInsets.all(8.0)),

                if (_errorMessage != null)
                  Padding(
                    padding: const EdgeInsets.only(bottom: 8.0),
                    child: Text(
                      _errorMessage!,
                      style: TextStyle(
                        color: Colors.red,
                        fontSize: 14.0,
                      ),
                    ),
                  ),

                Text(
```

```
                    "Password",
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      fontSize: 16.0,
                    ),
                  ),
                  Padding(padding: EdgeInsets.all(2.0)),
                  TextField(
                    controller: _passwordController,
                    obscureText: _isPassword,
                    decoration: InputDecoration(
                      suffixIcon: Padding(
                        padding: const EdgeInsets.symmetric(
    horizontal: 8.0),
                        child: IconButton(
                          icon: Icon(
                            _isPassword ? Icons.visibility_off :
    Icons.visibility,
                          ),
                          onPressed: () => setState(() => _isPassword
     = !_isPassword),
                        ),
                      ),
                      border: OutlineInputBorder(
                        borderRadius: BorderRadius.circular(12.0),
                        borderSide: BorderSide(
                          width: 2.0,
                          color: const Color.fromARGB(255, 184, 183,
    183),
                        ),
                      ),
                      enabledBorder: OutlineInputBorder(
                        borderRadius: BorderRadius.circular(12.0),
                        borderSide: BorderSide(
                          width: 2.0,
                          color: const Color.fromARGB(255, 184, 183,
    183),
                        ),
                      ),
                      focusedBorder: OutlineInputBorder(
                        borderRadius: BorderRadius.circular(12.0),
```

```
                      borderSide: BorderSide(width: 2.0, color: Colors.black54),
                    ),
                  ),
                ),
                Padding(padding: EdgeInsets.all(5.0)),

                Text(
                  "Confirm Password",
                  style: TextStyle(
                    fontWeight: FontWeight.bold,
                    fontSize: 16.0,
                  ),
                ),
                Padding(padding: EdgeInsets.all(2.0)),
                TextField(
                  controller: _confirmPasswordController,
                  obscureText: _isPassword2,
                  decoration: InputDecoration(
                    suffixIcon: Padding(
                      padding: const EdgeInsets.symmetric(horizontal: 8.0),
                      child: IconButton(
                        icon: Icon(
                          _isPassword2 ? Icons.visibility_off : Icons.visibility,
                        ),
                        onPressed: () => setState(() => _isPassword2 = !_isPassword2),
                      ),
                    ),
                    border: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(12.0),
                      borderSide: BorderSide(
                        width: 2.0,
                        color: const Color.fromARGB(255, 184, 183, 183),
                    ),),
                    enabledBorder: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(12.0),
                      borderSide: BorderSide(
```

```
                        width: 2.0,
                        color: const Color.fromARGB(255, 184, 183, 183),
                      ),
                    ),
                    focusedBorder: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(12.0),
                      borderSide: BorderSide(width: 2.0, color: Colors.black54),
                    ),
                  ),
                ),
                Padding(padding: EdgeInsets.all(10.0)),
                OutlinedButton(
                  style: OutlinedButton.styleFrom(
                    backgroundColor: _isLoading ? Colors.blue[200] : Colors.blue,
                    minimumSize: Size(double.infinity, 60.0),
                    side: BorderSide(color: Colors.blue),
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(15.0),
                    ),
                  ),
                  onPressed: _isLoading ? null : _updatePassword,
                  child: _isLoading
                      ? CircularProgressIndicator(color: Colors.white)
                      : Text(
                          "Update Password",
                          style: TextStyle(
                            color: Colors.white,
                            fontWeight: FontWeight.bold,
                            fontSize: 16.0,
                          ),
                        ),
                ),
              ],
            ),
          ),],
      ),);}}
```

Listing 2.5: Set New Password Code

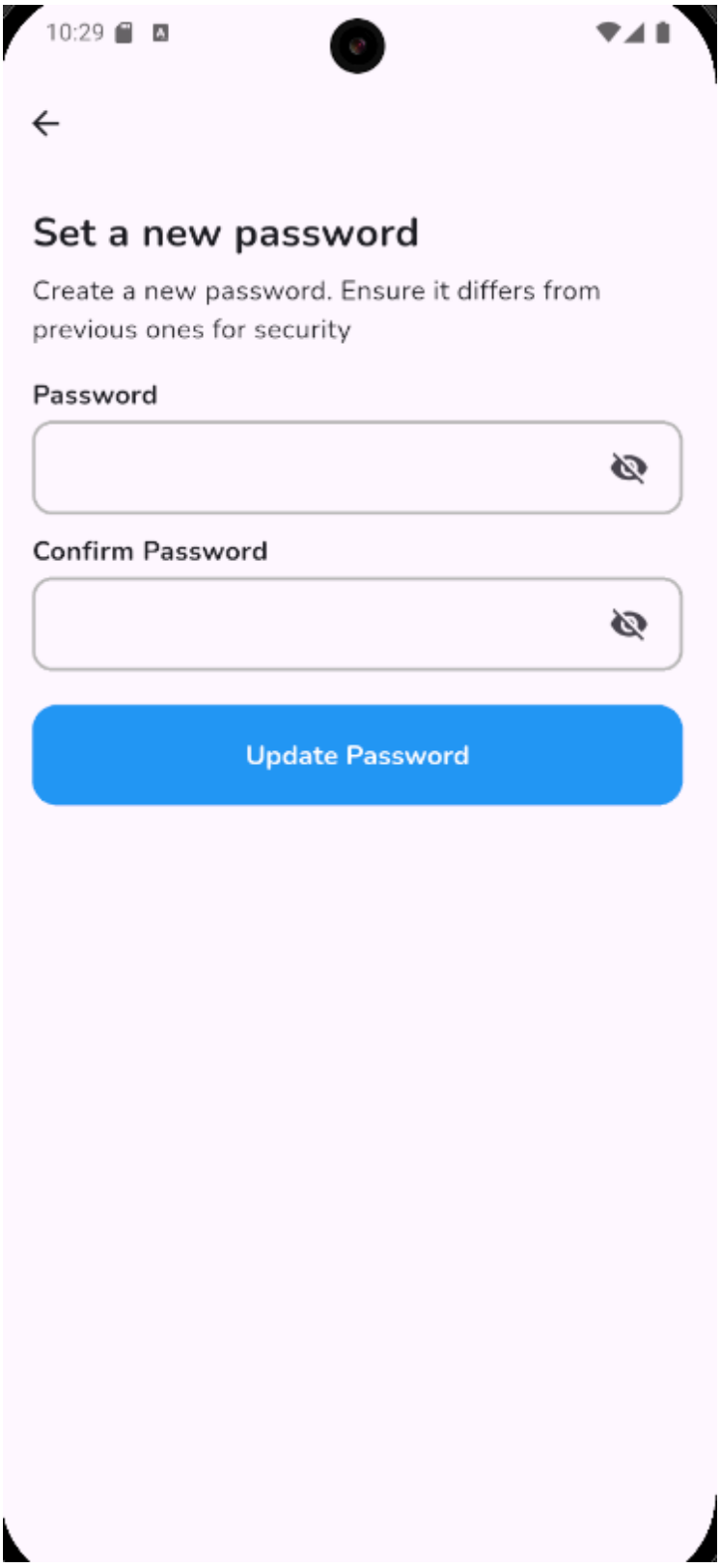

FIGURE 2.5: Set New Password Screen

2.8 Flutter Application Screen 6 - Home Page

Table 2.6: Use Case 6 - Home Page

Use Case ID	1006
Reference	Figure 6.6
Use Case Name	Home Page
Actors	User Flutter Application Recipe Database
Purpose	This screen displays the user's saved recipes and provides options to generate new recipes or recipes with specific constraints.
Preconditions	1. User must be logged in. 2. Application must have network connection. 3. Recipe database must be accessible.
Input Elements	1. Home Button: • Returns to main menu 2. Generate Button: • Creates a new random recipe 3. Generate with Constraints Button: • Opens dialog for constrained recipe generation 4. Recipe Cards (tappable): • Displays recipe previews • Opens detailed recipe view when tapped
Output Elements	1. Recipe List: • Displays saved recipes with names and descriptions 2. New Recipe Dialog: • Appears when generating new recipes 3. Constraint Selection Interface: • Appears when generating with constraints

Navigation Flow	1. User accesses the recipe management screen from main menu. 2. System displays saved recipes in card format. 3. User can: • Tap a recipe to view details • Press "Generate" for random recipe • Press "Generate with Constraints" for specific needs 4. For generation options: • System processes request • Displays new recipe • Offers save option 5. Alternative Flows: • No recipes: Shows empty state with prompt to generate • Network issues: Shows error message 6. Home button returns to main menu.

Code Implementation

```
import 'package:flutter/material.dart';
import 'package:flutter_django_recipes_frontend/add_edit_recipe_page.dart';
import 'package:flutter_django_recipes_frontend/login_page.dart';
import 'package:flutter_django_recipes_frontend/recipe_detail_page.dart';
import 'package:flutter_django_recipes_frontend/services/recipe_service.dart';
import 'package:flutter_django_recipes_frontend/ingredients_list.dart';

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

  @override
  State<HomePage> createState() => _HomePageState();
}

```

```
class _HomePageState extends State<HomePage> {
  List<dynamic> recipes = [];
  bool isLoading = true;
  bool hasError = false;
  String errorMessage = '';

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

  Future<void> _loadRecipes() async {
    setState(() {
      isLoading = true;
      hasError = false;
    });

    try {
      final loadedRecipes = await RecipeService.getRecipes();
      setState(() {
        recipes = loadedRecipes;
        isLoading = false;
      });
    } catch (e) {
      setState(() {
        isLoading = false;
        hasError = true;
        errorMessage = e.toString();
      });
      _showErrorSnackbar('Failed to load recipes: $e');
    }
  }

  Future<void> _handleAddRecipe() async {
    final result = await Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => const AddEditRecipePage(
          availableIngredients: availableIngredients,
        ),
```

```
      ),
    );

    if (result != null) {
      try {
        setState(() => isLoading = true);
        final newRecipe = await RecipeService.addRecipe(
          name: result['name'],
          description: result['description'],
          ingredients: result['ingredients'],
        );
        setState(() {
          recipes.add(newRecipe);
          isLoading = false;
        });
      } catch (e) {
        setState(() => isLoading = false);
        _showErrorSnackbar('Failed to add recipe: $e');
      }
    }
  }

  Future<void> _handleEditRecipe(Map<String, dynamic> recipe, int
   index) async {
    final result = await Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => AddEditRecipePage(
          recipe: recipe,
          availableIngredients: availableIngredients,
        ),
      ),
    );

    if (result != null) {
      try {
        setState(() => isLoading = true);
        final updatedRecipe = await RecipeService.updateRecipe(
          id: recipe['id'],
          name: result['name'],
          description: result['description'],
```

```
          ingredients: result['ingredients'],
        );
        setState(() {
          recipes[index] = updatedRecipe;
          isLoading = false;
        });
      } catch (e) {
        setState(() => isLoading = false);
        _showErrorSnackbar('Failed to update recipe: $e');
      }
    }
  }

  Future<void> _handleDeleteRecipe(int index) async {
    final recipe = recipes[index];
    try {
      setState(() => isLoading = true);
      await RecipeService.deleteRecipe(recipe['id']);
      setState(() {
        recipes.removeAt(index);
        isLoading = false;
      });
    } catch (e) {
      setState(() => isLoading = false);
      _showErrorSnackbar('Failed to delete recipe: $e');
    }
  }

  void _showErrorSnackbar(String message) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(message),
        backgroundColor: Colors.red,
      ),
    );
  }

  void _viewRecipeDetails(Map<String, dynamic> recipe) {
    Navigator.push(
      context,
      MaterialPageRoute(
```

```
        builder: (context) => RecipeDetailPage(recipe: recipe),
      ),
    );
  }

  void _logout() {
    Navigator.of(context).pushReplacement(
      MaterialPageRoute(builder: (context) => const LoginPage()),
    );
  }

  Widget _buildRecipeList() {
    if (hasError) {
      return Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Icon(Icons.error_outline, size: 48, color: Colors.red),
            const SizedBox(height: 16),
            Text(errorMessage, style: const TextStyle(color: Colors.red)),
            const SizedBox(height: 16),
            ElevatedButton(
              onPressed: _loadRecipes,
              child: const Text('Retry'),
            ),
          ],
        ),
      );
    }

    if (recipes.isEmpty) {
      return const Center(
        child: Text('No recipes found. Add your first recipe!'),
      );
    }

    return ListView.builder(
      itemCount: recipes.length,
      itemBuilder: (context, index) {
```

```
      final recipe = recipes[index];
      final ingredients = (recipe['ingredients'] as List<dynamic>)
          .map<String>((i) => i['name'] as String)
          .join(', ');

      return Card(
        margin: const EdgeInsets.all(8),
        elevation: 2,
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(12),
        ),
        child: ListTile(
          title: Text(
            recipe['name'],
            style: const TextStyle(fontWeight: FontWeight.bold),
          ),
          subtitle: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Text(recipe['description']),
              const SizedBox(height: 4),
              Text(
                'Ingredients: $ingredients',
                style: const TextStyle(fontSize: 12, color: Colors.grey),
              ),
            ],
          ),
          onTap: () => _viewRecipeDetails(recipe),
          trailing: PopupMenuButton<String>(
            onSelected: (value) {
              if (value == 'edit') _handleEditRecipe(recipe, index);
              if (value == 'delete') _handleDeleteRecipe(index);
            },
            itemBuilder: (context) => [
              const PopupMenuItem(
                value: 'edit',
                child: Text('Edit'),
              ),
```

```
                const PopupMenuItem(
                  value: 'delete',
                  child: Text('Delete'),
                ),
              ],
            ),
          ),
        );
      },
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Your Recipes'),
        actions: [
          IconButton(
            icon: const Icon(Icons.logout),
            onPressed: _logout,
            tooltip: 'Logout',
          ),
          IconButton(
            icon: const Icon(Icons.add),
            onPressed: _handleAddRecipe,
            tooltip: 'Add Recipe',
          ),
        ],
      ),
      body: isLoading
          ? const Center(child: CircularProgressIndicator())
          : RefreshIndicator(
              onRefresh: _loadRecipes,
              child: _buildRecipeList(),
            ),
      floatingActionButton: FloatingActionButton(
        onPressed: _handleAddRecipe,
        child: const Icon(Icons.add),), );}}
```

Listing 2.6: Home Page Code

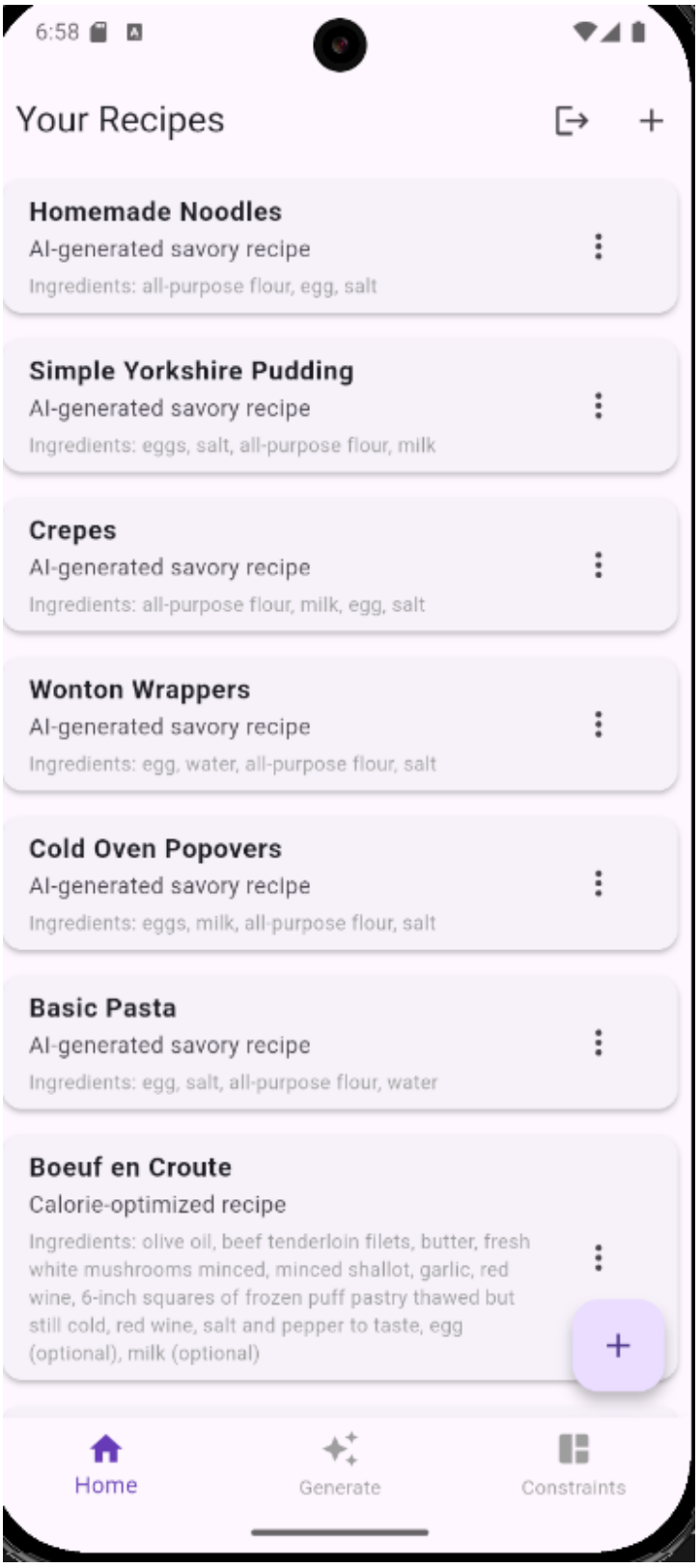

FIGURE 2.6: Home Screen

2.9 Flutter Application Screen 7 - Recipe Creation

Table 2.7: Use Case 7 - Recipe Creation

Use Case ID	1007
Reference	Figure 6.7
Use Case Name	Recipe Creation
Actors	User Flutter Application Recipe Database
Purpose	This screen allows users to create new recipes by entering recipe details and selecting from available ingredients, with support for saving custom recipes.
Preconditions	1. User must be logged in. 2. Application must have network connection. 3. Ingredient database must be accessible.
Input Elements	1. Recipe Name Field: • Text input for recipe title 2. Description Field: • Multi-line input for recipe instructions/details 3. Ingredient Selection: • Checkbox/tag system for selecting ingredients • Supports up to 30 ingredients 4. Add Recipe Button: • Submits the completed recipe 5. Back Button (Clicked): • Returns to previous screen

Output Elements	1. Success Notification: • Recipe saved confirmation • Automatic return to recipe list 2. Error Messages: • Missing required fields • Too many ingredients selected • Duplicate recipe name 3. Ingredient Badges: • Visual display of selected ingredients 4. Character Counters: • For name and description fields
Navigation Flow	1. User accesses the recipe creation screen from recipe list. 2. User enters recipe name and description. 3. User selects ingredients from available options. 4. User clicks "Add Recipe" to submit. 5. System validates inputs: • Checks for required fields • Verifies ingredient count • Checks for duplicates 6. Success Flow: • Recipe saved to database • User returned to recipe list with new recipe visible 7. Alternative Flows: • Validation errors show specific messages • Network issues show retry option 8. Back button cancels creation and returns to list.

Code Implementation

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

class AddEditRecipePage extends StatefulWidget {
  final Map? recipe; // If null, were adding a new recipe. If not
    null, were editing.
```

```
  final List<String> availableIngredients; // Ingredients to choose from (e.g., loaded via Django backend)

  const AddEditRecipePage({super.key, this.recipe, required this.availableIngredients});

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

class _AddEditRecipePageState extends State<AddEditRecipePage> {
  final _formKey = GlobalKey<FormState>();

  // Controllers for input fields
  late TextEditingController _nameController;
  late TextEditingController _descriptionController;

  // Stores selected ingredients from chips
  List<String> selectedIngredients = [];

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

    // Pre-fill data if editing
    _nameController = TextEditingController(text: widget.recipe?['name'] ?? '');
    _descriptionController = TextEditingController(
      text: widget.recipe?['description'] ?? '',
    );

    // Initialize selected ingredients (if editing)
    selectedIngredients = widget.recipe?['ingredientsList']?.cast<String>() ?? [];
  }

  // Called when user presses "Add" or "Save Changes"
  void _saveRecipe() {
    if (_formKey.currentState!.validate()) {
      final newRecipe = {
```

```
        "name": _nameController.text,
        "description": _descriptionController.text,
        "ingredients": selectedIngredients.join(', '), // For display
        "ingredientsList": selectedIngredients,         // For internal use or backend
      };
      // Return recipe to previous screen
      Navigator.pop(context, newRecipe);
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.recipe == null ? 'Add Recipe' : 'Edit Recipe'),
      ),
      body: Padding(
        padding: EdgeInsets.all(16),
        child: Form(
          key: _formKey,
          child: ListView(
            children: [
              // Recipe Name Input
              TextFormField(
                controller: _nameController,
                decoration: InputDecoration(labelText: 'Recipe Name'),
                validator: (value) => value!.isEmpty ? 'Enter recipe name' : null,
              ),
              SizedBox(height: 16),

              // Description Input
              TextFormField(
                controller: _descriptionController,
                decoration: InputDecoration(labelText: 'Description'),
```

```
                validator: (value) => value!.isEmpty ? 'Enter description' : null,
              ),
              SizedBox(height: 16),

              // Ingredient Selection Section
              Text(
                'Select Ingredients (up to 30)',
                style: TextStyle(fontWeight: FontWeight.bold),
              ),
              Wrap(
                spacing: 8,
                children: widget.availableIngredients.map((ingredient) {
                  final isSelected = selectedIngredients.contains(ingredient);
                  return FilterChip(
                    label: Text(ingredient),
                    selected: isSelected,
                    onSelected: (selected) {
                      setState(() {
                        if (selected) {
                          // Add if not more than 30
                          if (selectedIngredients.length < 30) {
                            selectedIngredients.add(ingredient);
                          }
                        } else {
                          // Remove if deselected
                          selectedIngredients.remove(ingredient);
                        }
                      });
                    },);});).toList(),
                    ),
              SizedBox(height: 24),

              // Submit Button
              ElevatedButton(
                onPressed: _saveRecipe,
                child: Text(widget.recipe == null ? 'Add Recipe' : 'Save Changes'),),],),),,   ),    );  }}
```

Listing 2.7: Recipe Creation Code

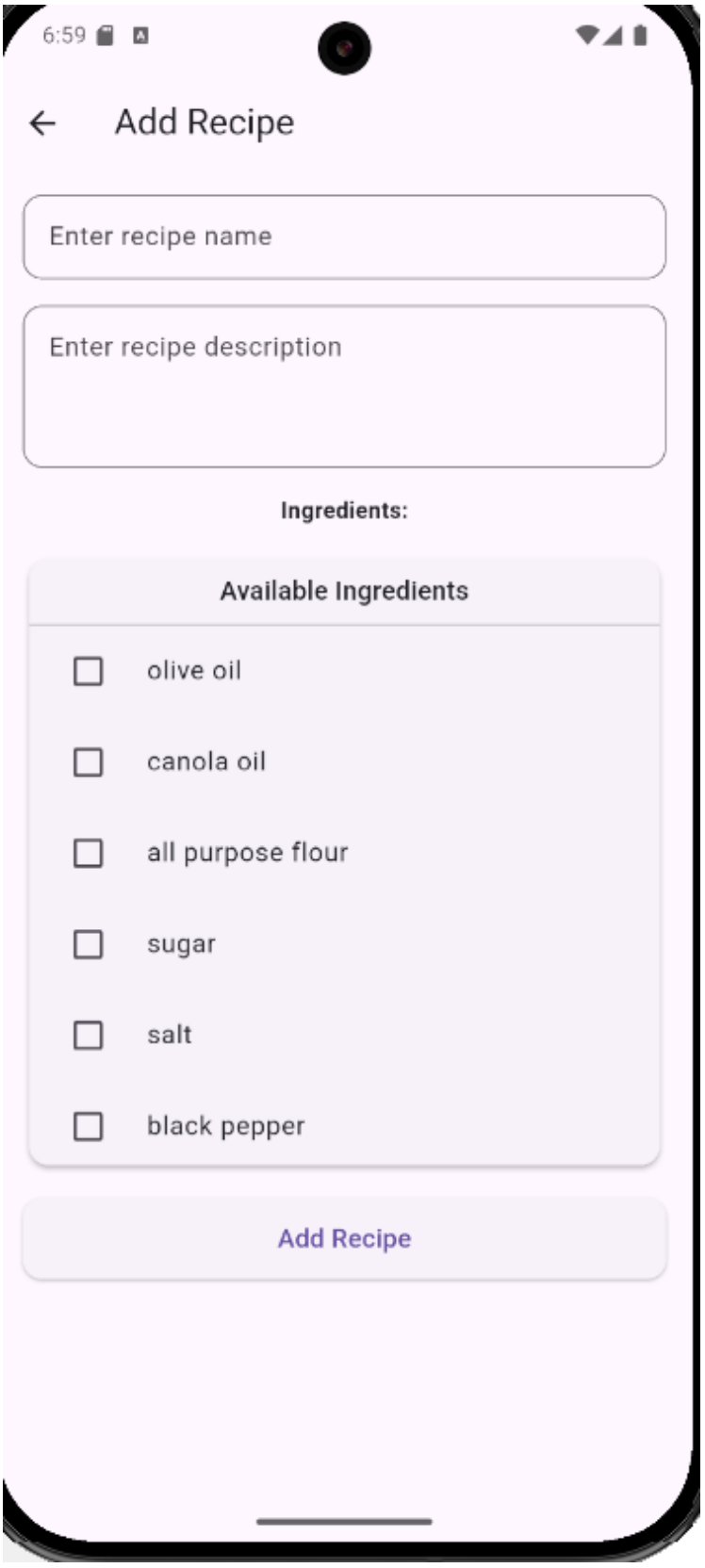

FIGURE 2.7: Recipe Creation Screen

2.10 Flutter Application Screen 8 - Recipe Editing

Table 2.8: Use Case 8 - Recipe Editing

Use Case ID	1008
Reference	Figure 6.8
Use Case Name	Recipe Editing
Actors	User Flutter Application Recipe Database
Purpose	This screen allows users to modify existing recipes by updating recipe details, ingredients, and saving the changes to their recipe collection.
Preconditions	1. User must be logged in. 2. Recipe must exist in user's collection. 3. Application must have network connection.
Input Elements	1. Recipe Name Field: • Editable text field for recipe title 2. Description Field: • Editable multi-line input for recipe details 3. Ingredient Selection: • Checkbox/tag system for modifying ingredients • Supports up to 30 ingredients 4. Save Changes Button: • Submits the updated recipe 5. Back Button (Clicked): • Returns to previous screen

Output Elements	1. Success Notification: • Changes saved confirmation • Automatic return to recipe list 2. Error Messages: • Missing required fields • Too many ingredients selected 3. Original Recipe Display: • Shows pre-edit values for reference 4. Ingredient Badges: • Visual display of currently selected ingredients
Navigation Flow	1. User accesses the edit screen from recipe details. 2. System pre-populates fields with current values. 3. User modifies name, description, or ingredients. 4. User clicks "Save Changes" to submit updates. 5. System validates inputs: • Checks for required fields • Verifies ingredient count 6. Success Flow: • Updated recipe saved to database • User returned to recipe list with changes visible 7. Alternative Flows: • Validation errors show specific messages • Cancelation returns without saving 8. Back button cancels editing if changes not saved.

Code Implementation

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

class AddEditRecipePage extends StatefulWidget {
  final Map? recipe;
  final List<String> availableIngredients;

  const AddEditRecipePage({
    super.key,
    this.recipe,
```

```
    required this.availableIngredients,
  });

  @override
  State<AddEditRecipePage> createState() => _AddEditRecipePageState
    ();
}

class _AddEditRecipePageState extends State<AddEditRecipePage> {
  late final TextEditingController _nameController;
  late final TextEditingController _descController;
  List<String> _selectedIngredients = [];
  String? _nameError;
  String? _descError;

  @override
  void initState() {
    super.initState();
    _nameController = TextEditingController(text: widget.recipe?['
    name'] ?? '');
    _descController = TextEditingController(
      text: widget.recipe?['description'] ?? '',
    );

    if (widget.recipe != null && widget.recipe![’ingredients’] !=
    null) {
      final ingredientsData = widget.recipe![’ingredients’];
      if (ingredientsData is List) {
        _selectedIngredients =
            ingredientsData.map<String>((e) => e[’name’] as String)
    .toList();
      }
    }
  }

  @override
  void dispose() {
    _nameController.dispose();
    _descController.dispose();
    super.dispose();
  }
```

```

  bool _validateFields() {
    setState(() {
      _nameError = _nameController.text.isEmpty ? 'Recipe name is required' : null;
      _descError = _descController.text.isEmpty ? 'Description is required' : null;
    });
    return _nameError == null && _descError == null;
  }

  void _saveRecipe() {
    if (_validateFields()) {
      Navigator.pop(context, {
        'name': _nameController.text,
        'description': _descController.text,
        'ingredients': _selectedIngredients,
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    final screenHeight = MediaQuery.of(context).size.height;

    return Scaffold(
      appBar: AppBar(
        title: Text(widget.recipe == null ? 'Add Recipe' : 'Edit Recipe'),
      ),
      body: SafeArea(
        child: ListView(

          padding: const EdgeInsets.all(16),
          children: [
            Column(
              children: [
                TextField(
                  controller: _nameController,
                  decoration: InputDecoration(
                    hintText: 'Enter recipe name',
```

```
                    errorText: _nameError,
                    border: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(12),
                    ),
                    contentPadding: const EdgeInsets.symmetric(
                      horizontal: 16,
                      vertical: 14,
                    ),
                  ),
                  onChanged: (_) {
                    if (_nameError != null) {
                      setState(() => _nameError = null);
                    }
                  },
                ),
                const SizedBox(height: 16),

                TextField(
                  controller: _descController,
                  maxLines: 3,
                  decoration: InputDecoration(
                    hintText: 'Enter recipe description',
                    errorText: _descError,
                    border: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(12),
                    ),
                    contentPadding: const EdgeInsets.symmetric(
                      horizontal: 16,
                      vertical: 14,
                    ),
                  ),
                  onChanged: (_) {
                    if (_descError != null) {
                      setState(() => _descError = null);
                    }
                  },
                ),
                const SizedBox(height: 16),

                const Text(
                  'Ingredients:',
```

```
                  style: TextStyle(fontWeight: FontWeight.bold),
                ),
                const SizedBox(height: 8),

                Wrap(
                  spacing: 8,
                  runSpacing: 8,
                  children: _selectedIngredients.map((ingredient) {
                    return Chip(
                      label: Text(ingredient),
                      onDeleted: () => setState(() {
                        _selectedIngredients.remove(ingredient);
                      }),
                      deleteIcon: const Icon(Icons.close, size: 18)
,
                    );
                  }).toList(),
                ),
                const SizedBox(height: 8),

                SizedBox(
                  height: screenHeight * 0.4,
                  child: Card(
                    elevation: 2,
                    child: Column(
                      children: [
                        Padding(
                          padding: const EdgeInsets.all(8.0),
                          child: Text(
                            'Available Ingredients',
                            style: Theme.of(context).textTheme.
titleMedium,
                          ),
                        ),
                        const Divider(height: 1),
                        Expanded(
                          child: ListView.builder(
                            itemCount: widget.availableIngredients.
length,
                            itemBuilder: (context, index) {
```

```
                              final ingredient = widget.availableIngredients[index];
                              final isSelected = _selectedIngredients.contains(ingredient);
                              return CheckboxListTile(
                                title: Text(ingredient),
                                value: isSelected,
                                onChanged: (selected) {
                                  setState(() {
                                    if (selected == true) {
                                      _selectedIngredients.add(ingredient);
                                    } else {
                                      _selectedIngredients.remove(ingredient);
                                    }
                                  });
                                },
                                controlAffinity: ListTileControlAffinity.leading,
                              );
                            },
                          ),
                        ),
                      ],
                    ),
                  ),
                ),

                const SizedBox(height: 16),
                ElevatedButton(
                  onPressed: _saveRecipe,
                  style: ElevatedButton.styleFrom(
                    minimumSize: const Size(double.infinity, 50),
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(12),
                    ),
                  ),
                  child: Text(
                    widget.recipe == null ? 'Add Recipe' : 'Save Changes',
```

```
                      style: const TextStyle(fontSize: 16),
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      );
    }
}
```

Listing 2.8: Recipe Editing Code

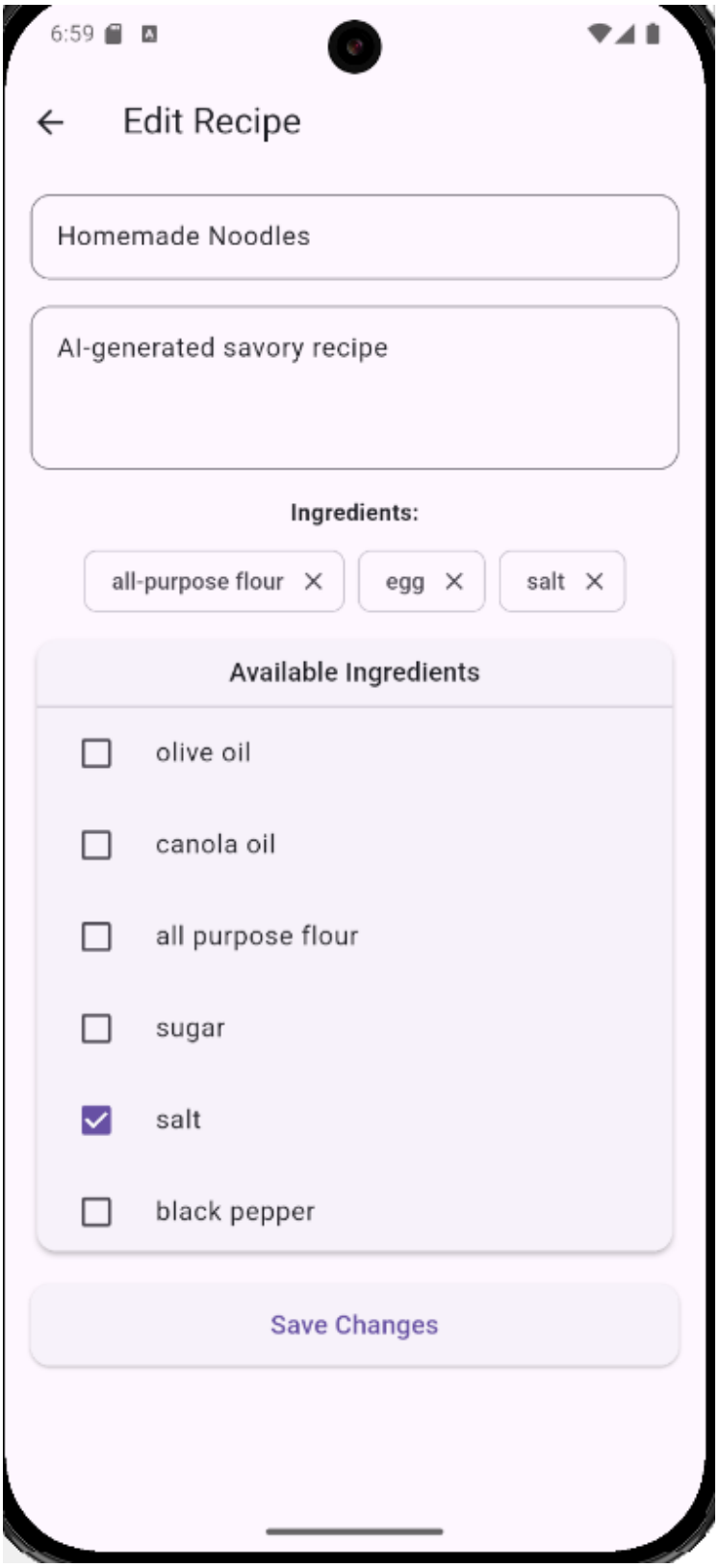

FIGURE 2.8: Recipe Editing Screen

2.11 Flutter Application Screen 9 - AI Recipe Generation

Table 2.9: Use Case 9 - AI Recipe Generation

Use Case ID	1009
Reference	Figure 6.9
Use Case Name	AI Recipe Generation
Actors	User Flutter Application AI Recipe Model Recipe Database
Purpose	This screen presents AI-generated recipes to users and provides options to accept or discard the suggestions, with functionality to generate new variations.
Preconditions	1. User must be logged in. 2. AI service must be available. 3. Application must have network connection.
Input Elements	1. Accept Button: • Saves the AI-generated recipe 2. Discard Button: • Rejects the current suggestion 3. Home Button: • Returns to main menu 4. Generate Button: • Creates a new random recipe 5. Generate with Constraints Button: • Opens dialog for constrained generation
Output Elements	1. AI-Generated Recipe Display: • Shows recipe name and description 2. Success Notification: • Recipe saved confirmation 3. New Recipe Generation: • Displays alternate recipe when generated

Navigation Flow	1. System displays AI-generated recipe. 2. User can: • Accept to save to their collection • Discard to remove the suggestion • Generate new random recipe • Generate with specific constraints 3. For accepted recipes: • Saved to user's collection • Confirmation message displayed 4. Alternative Flows: • Network issues show error and retry options • Empty results show appropriate message 5. Home button returns to main menu.

Code Implementation

```
import 'package:flutter/material.dart';
import 'package:flutter_django_recipes_frontend/recipe_detail_page.
    dart';
import 'package:flutter_django_recipes_frontend/services/
    recipe_service.dart';
import 'package:flutter_django_recipes_frontend/services/
    auth_service.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

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

  @override
  State<GenerateRecipePage> createState() =>
    _GenerateRecipePageState();
}

class _GenerateRecipePageState extends State<GenerateRecipePage> {
  final List<String> paletteList = [
    'savory',
    'spicy',
```

```
    'sweet',
    'sour',
    'herby',
    'umami',
    'earthy',
    'fruity',
    'smoky',
    'neutral',
    'bitter',
  ];

  String? selectedPalette;
  List<Map<String, dynamic>> recommendedRecipes = [];
  bool isLoading = false;
  bool isGenerating = false;
  String? errorMessage;

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

  Future<void> _loadUserIngredients() async {
    setState(() {
      isLoading = true;
      errorMessage = null;
    });

    try {
      final ingredients = await _getUserTopIngredients();
      if (ingredients.isEmpty) {
        setState(() {
          errorMessage =
              'No ingredients found in your recipes. Add some recipes first.';
          isLoading = false;
        });
      } else {
        setState(() => isLoading = false);
      }
```

```
    } catch (e) {
      setState(() {
        errorMessage = 'Failed to load your ingredients: $e';
        isLoading = false;
      });
      debugPrint('Error loading ingredients: $e');
    }
  }

  Future<List<String>> _getUserTopIngredients() async {
    final client = http.Client();
    try {
      final headers = await AuthService.getAuthHeaders();

      final response = await client.get(
        Uri.parse('${RecipeService.baseUrl}/recipes/'),
        headers: headers,
      );

      if (response.statusCode == 200) {
        final recipes = jsonDecode(response.body) as List;

        if (recipes.isEmpty) return [];

        final ingredientCounts = <String, int>{};

        for (final recipe in recipes) {
          final ingredients = recipe['ingredients'] as List;
          for (final ing in ingredients) {
            final name = ing['name'].toString().toLowerCase().trim
    ();
            ingredientCounts[name] = (ingredientCounts[name] ?? 0)
    + 1;
          }
        }

        final sortedIngredients =
            ingredientCounts.entries.toList()
              ..sort((a, b) => b.value.compareTo(a.value));

        final topIngredients =
```

```
            sortedIngredients.take(4).map((e) => e.key).toList();
        return topIngredients;
      } else {
        throw Exception('Failed to load recipes: ${response.
    statusCode}');
      }
    } finally {
      client.close();
    }
  }

  Future<void> _generateRecipes() async {
    if (selectedPalette == null) {
      setState(() => errorMessage = 'Please select a palette');
      return;
    }

    setState(() {
      isGenerating = true;
      errorMessage = null;
      recommendedRecipes = [];
    });

    try {
      final ingredients = await _getUserTopIngredients();
      if (ingredients.isEmpty) {
        throw Exception('No ingredients found in your recipes');
      }

      final headers = await AuthService.getAuthHeaders();
      final response = await http.post(
        Uri.parse('${RecipeService.baseUrl}/recipes/recommend/'),
        headers: headers,
        body: jsonEncode({
          'ingredients': ingredients,
          'palette': [selectedPalette],
        }),
      );

      if (response.statusCode == 200) {
        final data = jsonDecode(response.body) as List;
```

```
        setState(() {
          recommendedRecipes = List<Map<String, dynamic>>.from(data);
          isGenerating = false;
        });
      } else {
        throw Exception(
          'Failed to generate recipes: ${response.statusCode} - ${response.body}',
        );
      }
    } catch (e) {
      setState(() {
        errorMessage = 'Failed to generate recipes: $e';
        isGenerating = false;
      });
    }
  }

  Future<void> _saveRecipe(Map<String, dynamic> recipe) async {
  try {
    final recipeName = recipe['recipe_name'] ?? recipe['name'] ?? 'Unnamed Recipe';

    List<String> ingredients = [];

    if (recipe['ingredients_list'] != null) {
      if (recipe['ingredients_list'] is String) {
        String ingredientsStr = recipe['ingredients_list'] as String;
        ingredientsStr = ingredientsStr
            .replaceAll('[', '')
            .replaceAll(']', '')
            .replaceAll("'", "")
            .trim();

        if (ingredientsStr.isNotEmpty) {
          ingredients = ingredientsStr.split(',').map((e) => e.trim()).toList();
        }
      }
```

```

      else if (recipe['ingredients_list'] is List) {
        ingredients = (recipe['ingredients_list'] as List)
            .map((e) => e.toString().trim())
            .toList();
      }
    }
    else if (recipe['ingredients'] != null) {
      if (recipe['ingredients'] is String) {
        ingredients = (recipe['ingredients'] as String)
            .split(',')
            .map((e) => e.trim())
            .toList();
      } else if (recipe['ingredients'] is List) {
        ingredients = (recipe['ingredients'] as List)
            .map((e) => e.toString().trim())
            .toList();
      }
    }


    final savedRecipe = await RecipeService.addRecipe(
      name: recipeName,
      description: 'AI-generated $selectedPalette recipe',
      ingredients: ingredients,
    );

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Saved ${savedRecipe['name']} to your recipes
    !'),
        backgroundColor: Colors.green,
      ),
    );

    setState(() {
      recommendedRecipes.removeWhere((r) =>
        (r['recipe_name'] ?? r['name']) == recipeName);
    });
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
```

```
      SnackBar(
        content: Text('Failed to save recipe: ${e.toString()}'),
        backgroundColor: Colors.red,
      ),
    );
    debugPrint('Error saving recipe: $e');
  }
}

  void _viewRecipeDetails(Map<String, dynamic> recipe) {
  final recipeName = recipe['recipe_name'] ?? recipe['name'] ?? 'Unnamed Recipe';

  List<Map<String, dynamic>> ingredientsList = [];

  if (recipe['ingredients_list'] != null) {
    if (recipe['ingredients_list'] is String) {
      String ingredientsStr = recipe['ingredients_list'] as String;

      ingredientsStr = ingredientsStr
          .replaceAll('[', '')
          .replaceAll(']', '')
          .replaceAll("'", "")
          .trim();

      if (ingredientsStr.isNotEmpty) {
        ingredientsList = ingredientsStr
            .split(',')
            .map((ingredient) => {'name': ingredient.trim()})
            .toList();
      }
    }
    else if (recipe['ingredients_list'] is List) {
      ingredientsList = (recipe['ingredients_list'] as List)
          .map((ingredient) => {'name': ingredient.toString().trim()})
          .toList();
    }
  }
  else if (recipe['ingredients'] != null) {
    if (recipe['ingredients'] is String) {
```

```
      ingredientsList = (recipe['ingredients'] as String)
          .split(',')
          .map((ingredient) => {'name': ingredient.trim()})
          .toList();
    } else if (recipe['ingredients'] is List) {
      ingredientsList = (recipe['ingredients'] as List).map((
    ingredient) {
        if (ingredient is Map) {
          return {'name': ingredient['name']?.toString().trim() ??
    'Unknown'};
        } else {
          return {'name': ingredient.toString().trim()};
        }
      }).toList();
    }
  }


  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => RecipeDetailPage(recipe: {
        'name': recipeName,
        'description': 'AI-generated $selectedPalette recipe with $
    {recipe['palette'] ?? selectedPalette ?? 'various'} flavors',
        'ingredients': ingredientsList,
      }),
    ),
  );
}

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('AI Recipe Generator')),
      body:
          isLoading
              ? const Center(child: CircularProgressIndicator())
              : Padding(
                padding: const EdgeInsets.all(16),
                child: Column(
```

```
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      if (errorMessage != null)
                        Padding(
                          padding: const EdgeInsets.only(bottom: 16),
                          child: Text(
                            errorMessage!,
                            style: const TextStyle(color: Colors.red),
                          ),
                        ),

                      const Text(
                        'Select Flavor Palette:',
                        style: TextStyle(
                          fontSize: 18,
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                      const SizedBox(height: 8),

                      Wrap(
                        spacing: 8,
                        runSpacing: 8,
                        children:
                            paletteList.map((palette) {
                              return ChoiceChip(
                                label: Text(palette.capitalize()),
                                selected: selectedPalette == palette,
                                onSelected: (selected) {
                                  setState(() {
                                    selectedPalette = selected ? palette : null;
                                    errorMessage = null;
                                  });
                                },
                                selectedColor: Colors.blue.shade300,
                                backgroundColor: Colors.grey.shade200,
                                labelStyle: TextStyle(
                                  color:
```

```
                                    selectedPalette == palette
                                        ? Colors.white
                                        : Colors.black,
                                ),
                            );
                        }).toList(),
                    ),

                    const SizedBox(height: 24),

                    Center(
                      child: ElevatedButton(
                        onPressed: isGenerating ? null :
    _generateRecipes,
                        style: ElevatedButton.styleFrom(
                          padding: const EdgeInsets.symmetric(
                            horizontal: 32,
                            vertical: 16,
                          ),
                        ),
                        child:
                            isGenerating
                                ? const CircularProgressIndicator(
                                  color: Colors.white,
                                )
                                : const Text('Generate Recipes'),
                      ),
                    ),

                    const SizedBox(height: 24),

                    if (recommendedRecipes.isNotEmpty)
                      const Text(
                        'Recommended Recipes:',
                        style: TextStyle(
                          fontSize: 18,
                          fontWeight: FontWeight.bold,
                        ),
                      ),

                    Expanded(
```

```
                        child: ListView.builder(
                          itemCount: recommendedRecipes.length,
                          itemBuilder: (context, index) {
                            final recipe = recommendedRecipes[index];
                            return Card(
                              margin: const EdgeInsets.symmetric(vertical: 8),
                              child: ListTile(
                                title: Text(
                                  recipe['recipe_name'] ?? recipe['name'],
                                ),
                                subtitle: Text(
                                  recipe['matched_palette'] ??
                                      'Flavor: $selectedPalette',
                                ),
                                trailing: IconButton(
                                  icon: const Icon(Icons.save),
                                  onPressed: () => _saveRecipe(recipe),
                                  tooltip: 'Save Recipe',
                                ),
                                onTap: () => _viewRecipeDetails(recipe),
                              ),
                            );
                          },
                        ),
                      ),
                    ],
                  ),
                ),
    );
  }}
extension StringExtension on String {
  String capitalize() {
    return "${this[0].toUpperCase()}${substring(1).toLowerCase()}";
  }
}
```

Listing 2.9: AI Recipe Generation Code

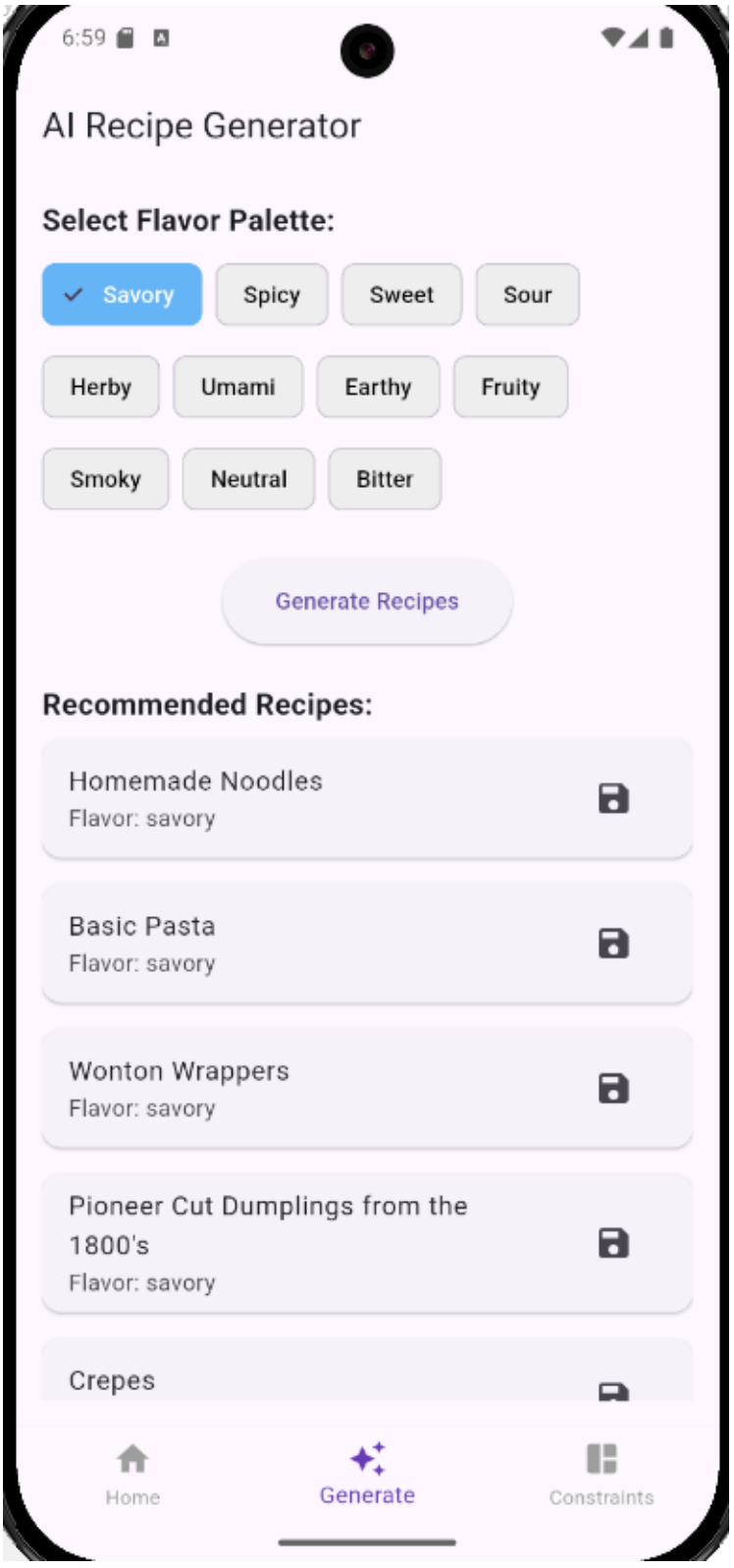

FIGURE 2.9: AI Recipe Generation Screen

2.12 Flutter Application Screen 10 - AI Recipe Details

Table 2.10: Use Case 10 - AI Recipe Details

Use Case ID	1010
Reference	Figure 6.10
Use Case Name	AI Recipe Details
Actors	User Flutter Application AI Recipe Service
Purpose	This screen displays the complete details of an AI-generated recipe including description and ingredients, allowing users to view the full recipe.
Preconditions	1. User must have generated or selected an AI recipe. 2. Recipe data must be available from AI service.
Input Elements	1. Back Button (Clicked): • Returns to recipe list/generation screen 2. Save Button (implied): • Adds recipe to user's collection 3. Share Button (Clicked): • Shares recipe with others
Output Elements	1. Recipe Title: • Clear display of recipe name 2. Recipe Description: • Detailed preparation instructions 3. Ingredients List: • Bulleted list of required ingredients 4. AI Generation Badge: • Indicates recipe is AI-generated 5. Success Notifications: • For save/share actions

Navigation Flow	1. User accesses from recipe generation or list screen. 2. System displays full recipe details. 3. User can: • Save recipe to their collection • Share recipe via other apps • Return to previous screen 4. Success Flow: • Saved recipes added to user collection • Shared recipes open share dialog 5. Alternative Flows: • Network issues prevent saving • Missing data shows error message

Code Implementation

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

// Stateless widget to show detailed view of a selected recipe
class RecipeDetailPage extends StatelessWidget {
  // Recipe data is passed as a Map (contains keys like 'name', 'description', 'ingredients')
  final Map recipe;

  // Constructor with required recipe data
  const RecipeDetailPage({super.key, required this.recipe});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      // App bar displaying the recipe name
      appBar: AppBar(
        title: Text(recipe['name']),
      ),
      body: Padding(
        padding: EdgeInsets.all(16),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start, // Align content to the start (left)
```

```
          children: [
            // Section: Description header
            Text(
              'Description',
              style: TextStyle(fontSize: 18, fontWeight: FontWeight
    .bold),
            ),
            SizedBox(height: 8),

            // Section: Description content
            Text(recipe['description']),
            SizedBox(height: 16),

            // Section: Ingredients header
            Text(
              'Ingredients',
              style: TextStyle(fontSize: 18, fontWeight: FontWeight
    .bold),
            ),
            SizedBox(height: 8),

            // Section: Ingredients content
            Text(recipe['ingredients']),
          ],
        ),
      ),
    );
  }
}
```

Listing 2.10: AI Recipe Details Code

FIGURE 2.10: AI Recipe Details Screen

2.13 Flutter Application Screen 11 - Nutritional Recipe Recommendations

Table 2.11: Use Case 11 - Nutritional Recipe Recommendations

Use Case ID	1011
Reference	Figure 6.11
Use Case Name	Nutritional Recipe Recommendations
Actors	User Flutter Application Nutrition AI Recipe Engine
Purpose	This screen allows users to get recipe recommendations based on nutritional preferences and available ingredients, with options to customize both nutritional parameters and ingredient selections.
Preconditions	1. User must be logged in. 2. Nutritional AI recipe engine must be accessible.
Input Elements	1. Nutritional Preference Selectors: • Fat, Carbs, Protein, Cholesterol, Sodium, Fiber 2. Ingredient Grid: • Checkbox selection for available ingredients • Supports up to 30 ingredients 3. Home Button: • Returns to main menu 4. Generate Button: • Creates recommendations with current settings 5. Generate with Constraints Button: • Opens advanced constraint dialog

Output Elements	1. Nutritional Summary: • Visual indicators for selected nutrition profile 2. Selected Ingredients Display: • Shows count of selected ingredients 3. Recipe Recommendations: • Displays matching recipes after generation 4. Error Messages: • For invalid combinations
Navigation Flow	1. User sets nutritional preferences using sliders/-selectors. 2. User selects available ingredients from the grid. 3. User clicks Generate to get recommendations. 4. System: • Analyzes nutritional constraints • Matches with selected ingredients • Returns appropriate recipes 5. Alternative Flows: • No matches: Suggests ingredient adjustments • Advanced constraints: Opens detailed dialog 6. Home button returns to main menu.

Code Implementation

```
import 'package:flutter/material.dart';
import 'package:flutter_django_recipes_frontend/recipe_detail_page.
    dart';
import 'package:flutter_django_recipes_frontend/services/
    recipe_service.dart';
import 'package:flutter_django_recipes_frontend/services/
    auth_service.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
import 'ingredients_list.dart';

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

```

```
  @override
  _ConstraintsRecipeState createState() => _ConstraintsRecipeState
    ();
}

class _ConstraintsRecipeState extends State<ConstraintsRecipe> {
  final TextEditingController caloriesController =
    TextEditingController();
  final TextEditingController fatController = TextEditingController
    ();
  final TextEditingController carbsController =
    TextEditingController();
  final TextEditingController proteinController =
    TextEditingController();
  final TextEditingController cholesterolController =
    TextEditingController();
  final TextEditingController sodiumController =
    TextEditingController();
  final TextEditingController fiberController =
    TextEditingController();

  List<String> selectedIngredients = [];
  List<Map<String, dynamic>> recommendations = [];
  bool isLoading = false;
  String? errorMessage;

  Future<void> getRecommendations() async {
    if (selectedIngredients.isEmpty) {
      setState(() => errorMessage = 'Please select at least one
    ingredient');
      return;
    }

    setState(() {
      isLoading = true;
      errorMessage = null;
      recommendations = [];
    });

    try {
      final headers = await AuthService.getAuthHeaders();
```

```
      final response = await http.post(
        Uri.parse('${RecipeService.baseUrl}/recipes/recommend_calorie_based/'),
        headers: headers,
        body: jsonEncode({
          'calories': caloriesController.text.isEmpty ? 0 : double.parse(caloriesController.text),
          'fat': fatController.text.isEmpty ? 0 : double.parse(fatController.text),
          'carbs': carbsController.text.isEmpty ? 0 : double.parse(carbsController.text),
          'protein': proteinController.text.isEmpty ? 0 : double.parse(proteinController.text),
          'cholesterol': cholesterolController.text.isEmpty ? 0 : double.parse(cholesterolController.text),
          'sodium': sodiumController.text.isEmpty ? 0 : double.parse(sodiumController.text),
          'fiber': fiberController.text.isEmpty ? 0 : double.parse(fiberController.text),
          'ingredients': selectedIngredients,
        }),
      );

      if (response.statusCode == 200) {
        final data = jsonDecode(response.body) as List;
        setState(() {
          recommendations = List<Map<String, dynamic>>.from(data);
        });
      } else {
        throw Exception('Failed to get recommendations: ${response.statusCode}');
      }
    } catch (e) {
      setState(() => errorMessage = e.toString());
    } finally {
      setState(() => isLoading = false);
    }
  }

    Future<void> _saveRecipe(Map<String, dynamic> recipe) async {
    try {
```

```
      final recipeName = recipe['recipe_name'] ?? recipe['name'] ?? 'Unnamed Recipe';

      List<String> ingredients = [];
      if (recipe['ingredients_list'] != null) {
        if (recipe['ingredients_list'] is String) {
          String ingredientsStr = recipe['ingredients_list'].toString();
          ingredientsStr = ingredientsStr
              .replaceAll('[', '')
              .replaceAll(']', '')
              .replaceAll("'", "");
          ingredients = ingredientsStr.split(',').map((e) => e.trim()).toList();
        } else if (recipe['ingredients_list'] is List) {
          ingredients = (recipe['ingredients_list'] as List)
              .map((e) => e.toString().trim())
              .toList();
        }
      }

      final savedRecipe = await RecipeService.addRecipe(
        name: recipeName,
        description: 'Calorie-optimized recipe',
        ingredients: ingredients,
      );

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Saved ${savedRecipe['name']} to your recipes!'),
          backgroundColor: Colors.green,
        ),
      );
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Failed to save recipe: $e'),
          backgroundColor: Colors.red,
        ),
      );
```

```
    }
  }

  void _viewRecipeDetails(Map<String, dynamic> recipe) {


    List<Map<String, String>> ingredientsList = [];

    if (recipe['ingredients_list'] != null) {
      if (recipe['ingredients_list'] is String) {
        String ingredientsStr = recipe['ingredients_list'].toString
    ();
        ingredientsStr = ingredientsStr
            .replaceAll('[', '')
            .replaceAll(']', '')
            .replaceAll("'", "");
        ingredientsList = ingredientsStr
            .split(',')
            .map((ingredient) => {'name': ingredient.trim()})
            .toList();
      } else if (recipe['ingredients_list'] is List) {
        ingredientsList = (recipe['ingredients_list'] as List)
            .map((ingredient) => {'name': ingredient.toString().
    trim()})
            .toList();
      }
    }
  }
  Widget buildIngredientSelection() {
    final screenHeight = MediaQuery.of(context).size.height;

    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Center(
          child: const Text(
            'Ingredients:',
            style: TextStyle(fontWeight: FontWeight.bold),
          ),
        ),
        const SizedBox(height: 8),
```

```

        Wrap(
          spacing: 8,
          runSpacing: 8,
          children: selectedIngredients.map((ingredient) {
            return Chip(
              label: Text(ingredient),
              onDeleted: () => setState(() {
                selectedIngredients.remove(ingredient);
              }),
              deleteIcon: const Icon(Icons.close, size: 18),
            );
          }).toList(),
        ),
        const SizedBox(height: 8),

        SizedBox(
          height: screenHeight * 0.4,
          child: Card(
            elevation: 2,
            child: Column(
              children: [
                Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Text(
                    'Available Ingredients',
                    style: Theme.of(context).textTheme.titleMedium,
                  ),
                ),
                const Divider(height: 1),
                Expanded(
                  child: ListView.builder(
                    itemCount: availableIngredients.length,
                    itemBuilder: (context, index) {
                      final ingredient = availableIngredients[index
    ];
                      final isSelected = selectedIngredients.
    contains(ingredient);
                      return CheckboxListTile(
                        title: Text(ingredient),
                        value: isSelected,
```

```
                        onChanged: (selected) {
                          setState(() {
                            if (selected == true) {
                              selectedIngredients.add(ingredient);
                            } else {
                              selectedIngredients.remove(ingredient
  );
                            }
                          });
                        },
                        controlAffinity: ListTileControlAffinity.
  leading,
                      );
                    },
                  ),
                ),
              ],
            ),
          ),
        ),
      ],
    );
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Calorie-Based Recipes'),
        elevation: 0,
      ),
      body: SafeArea(
        child: ListView(
          padding: const EdgeInsets.all(16),
          children: [
            if (errorMessage != null)
              Container(
                padding: const EdgeInsets.all(12),
                margin: const EdgeInsets.only(bottom: 16),
                decoration: BoxDecoration(
                  color: Colors.red.shade50,
```

```
                  borderRadius: BorderRadius.circular(8),
                  border: Border.all(color: Colors.red.shade200),
                ),
                child: Row(
                  children: [
                    const Icon(Icons.error_outline, color: Colors.red),
                    const SizedBox(width: 8),
                    Expanded(
                      child: Text(
                        errorMessage!,
                        style: TextStyle(color: Colors.red.shade800),
                      ),
                    ),
                  ],
                ),
              ),

            Card(
              elevation: 0,
              margin: const EdgeInsets.only(bottom: 16),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(12),
              ),
              child: Padding(
                padding: const EdgeInsets.all(16),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    const Text(
                      'Nutrition Constraints',
                      style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),
                    ),
                    const SizedBox(height: 12),
                    buildTextField('Calories (kcal)', caloriesController, TextInputType.number),
                    buildTextField('Fat (g)', fatController, TextInputType.number),
```

```
                    buildTextField('Carbs (g)', carbsController, TextInputType.number),
                    buildTextField('Protein (g)', proteinController, TextInputType.number),
                    buildTextField('Cholesterol (mg)', cholesterolController, TextInputType.number),
                    buildTextField('Sodium (mg)', sodiumController, TextInputType.number),
                    buildTextField('Fiber (g)', fiberController, TextInputType.number),
                  ],
                ),
              ),
            ),

            Card(
              elevation: 0,
              margin: const EdgeInsets.only(bottom: 16),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(12),
              ),
              child: Padding(
                padding: const EdgeInsets.all(16),
                child: buildIngredientSelection(),
              ),
            ),

            SizedBox(
              width: double.infinity,
              child: ElevatedButton(
                style: ElevatedButton.styleFrom(
                  padding: const EdgeInsets.symmetric(vertical: 16),
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(10),
                  ),
                ),
                onPressed: isLoading ? null : getRecommendations,
                child: isLoading
                    ? const SizedBox(
                        width: 20,
```

```
                        height: 20,
                        child: CircularProgressIndicator(
                          strokeWidth: 2,
                          color: Colors.white,
                        ),
                      )
                    : const Text(
                        'Get Recommendations',
                        style: TextStyle(fontSize: 16),
                      ),
              ),
            ),

            const SizedBox(height: 24),
            const Text(
              'Recommended Recipes',
              style: TextStyle(fontSize: 18, fontWeight: FontWeight
  .bold),
            ),
            const SizedBox(height: 12),
            buildRecommendationList(),
            const SizedBox(height: 16),
          ],
        ),
      ),
    );
  }

  Widget buildTextField(String label, TextEditingController
   controller, TextInputType keyboardType) {
    return Padding(
      padding: const EdgeInsets.only(bottom: 12),
      child: TextField(
        controller: controller,
        keyboardType: keyboardType,
        decoration: InputDecoration(
          contentPadding: const EdgeInsets.symmetric(horizontal:
  12, vertical: 14),
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(8),
            borderSide: BorderSide(color: Colors.grey.shade300),
```

```
          ),
          enabledBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(8),
            borderSide: BorderSide(color: Colors.grey.shade300),
          ),
          labelText: label,
          floatingLabelBehavior: FloatingLabelBehavior.auto,
        ),
      ),
    );
  }

  Widget buildRecommendationList() {
    if (recommendations.isEmpty) {
      return Container(
        padding: const EdgeInsets.all(16),
        decoration: BoxDecoration(
          color: Colors.grey.shade100,
          borderRadius: BorderRadius.circular(10),
        ),
        child: const Center(
          child: Text(
            'No recommendations yet. Adjust your constraints and try again.',
            style: TextStyle(color: Colors.grey),
          ),
        ),
      );
    }

    return ListView.builder(
      shrinkWrap: true,
      physics: const NeverScrollableScrollPhysics(),
      itemCount: recommendations.length,
      itemBuilder: (context, index) {
        final recipe = recommendations[index];
        return Card(
          elevation: 2,
          margin: const EdgeInsets.symmetric(vertical: 8),
          shape: RoundedRectangleBorder(
            borderRadius: BorderRadius.circular(10),
```

```
          ),
          child: InkWell(
            borderRadius: BorderRadius.circular(10),
            onTap: () => _viewRecipeDetails(recipe),
            child: Padding(
              padding: const EdgeInsets.symmetric(vertical: 12, horizontal: 5),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Row(
                    mainAxisAlignment: MainAxisAlignment.spaceBetween,
                    children: [
                      Text(
                        recipe['recipe_name'],
                        style: const TextStyle(
                          fontSize: 16,
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                      IconButton(
                        icon: const Icon(Icons.save),
                        onPressed: () => _saveRecipe(recipe),
                        tooltip: 'Save Recipe',
                      ),
                    ],
                  ),
                  const SizedBox(height: 8),
                  Text(
                    (recipe['ingredients_list'] is String
                        ? recipe['ingredients_list'].toString()
                            .replaceAll('[', '')
                            .replaceAll(']', '')
                            .replaceAll("'", "")
                        : (recipe['ingredients_list'] as List).join(', ')),
                    style: TextStyle(color: Colors.grey.shade700),
                  ),
                ],
              ),
```

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

Listing 2.11: Nutritional Recipe Recommendations Code

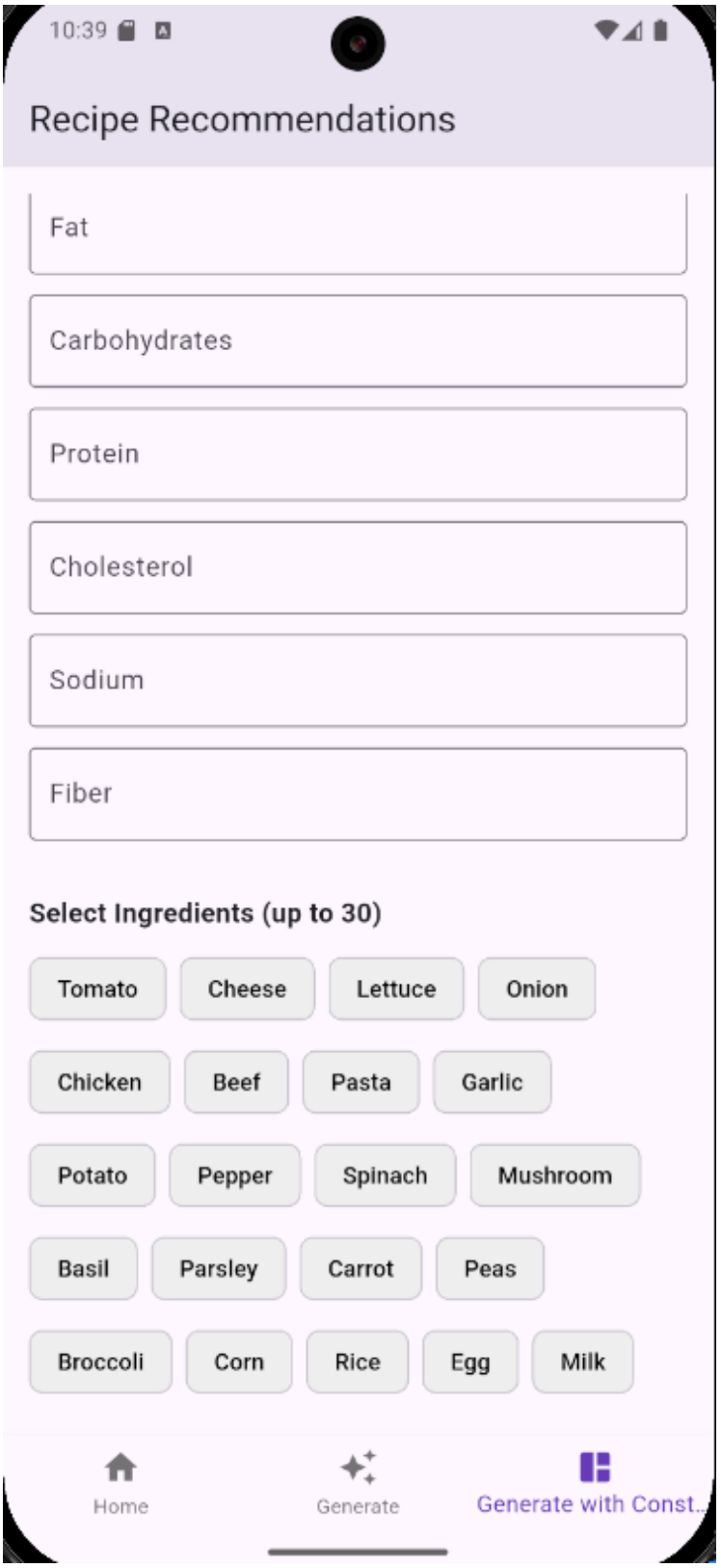

FIGURE 2.11: Nutritional Recipe Recommendations Screen 1

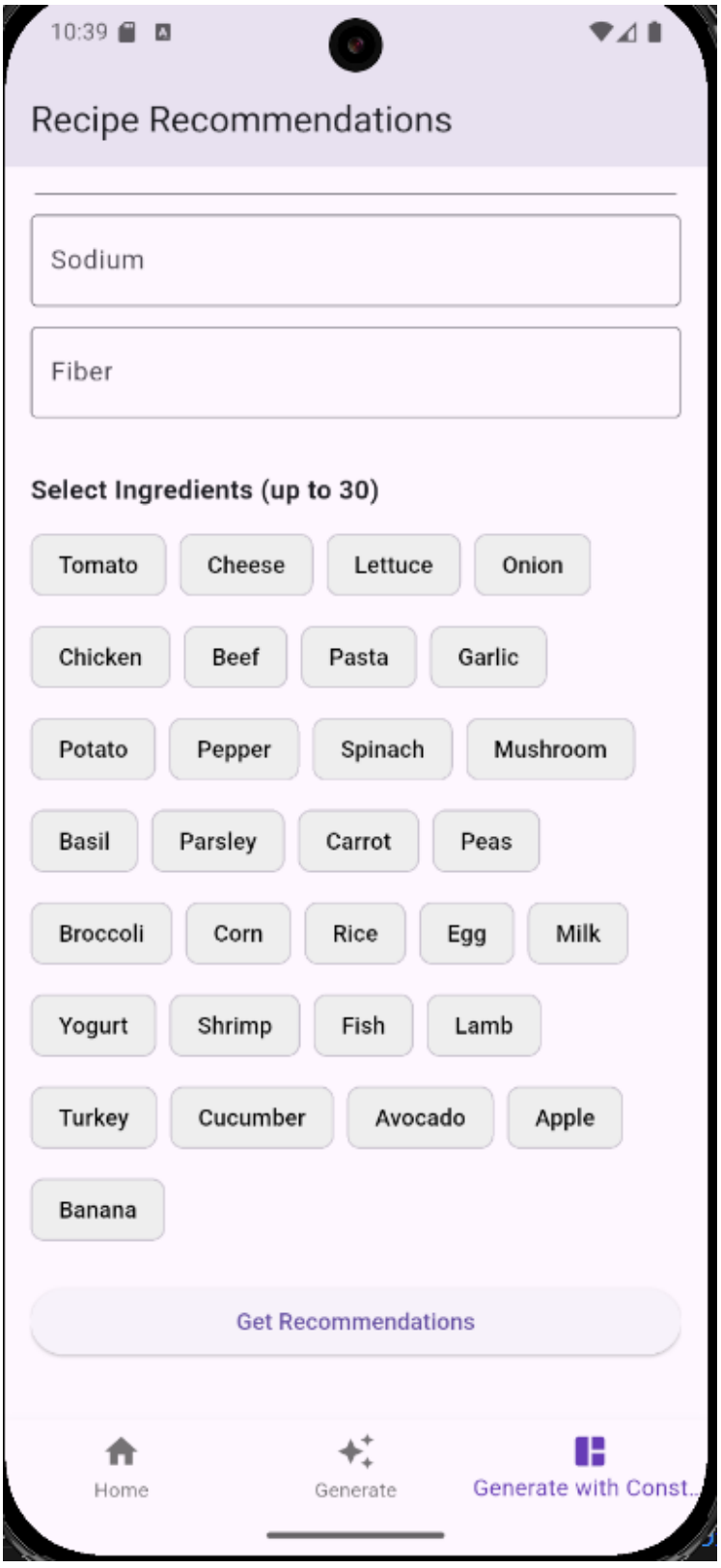

FIGURE 2.12: Nutritional Recipe Recommendations Screen 2

Chapter 3

Backend Implementation

3.1 Python Files

To implement the backend of our Recipe Recommendation System, we employed Python and dart due to its robust libraries and flexibility in handling data processing, recommendation logic, and integration with machine learning modules. Python allowed seamless manipulation of datasets, classification of ingredients, and dynamic generation of recipe recommendations based on user preferences, authentication services and main startup files. Below is the code for each respective Python and dart filessss used in the backend:

3.2 Backend File 1 - Setting.py

Table 3.1: Setting File

File Name	Settings.py

Actors	`User`
	`Flutter Application`
	`Email Service`

Purpose	This screen verifies the user's identity through a 5-digit code sent to their email, completing the password reset process.
Preconditions	1. User must have requested password reset. 2. User must have received the verification email. 3. The 5-digit code must be valid and unexpired.

Input Elements	`Verification Code Field`
	– Input for 5-digit verification code from email
	`Verify Code Button`
	– Submits the code for validation
	`Resend Email Link`
	– Re-sends verification code if not received
	`Back Button (Clicked)`
	– Returns to previous screen

Output Elements	`Success Notification`
	– Code verified, proceed to password reset screen
	`Error Messages`
	– Invalid code
	– Expired code
	– Too many attempts
	`Resend Confirmation`
	– New code sent notification

Navigation Flow	1. User arrives from password reset request screen. 2. User checks email for 5-digit verification code. 3. User enters the code in the verification field. 4. User clicks "Verify Code" to submit. 5. `Alternative Flows`: – If code is invalid/expired, show error message – If "Resend email" clicked, send new code 6. `Success Flow`: – After successful verification, navigate to password reset screen 7. `Failure Flow`: – After multiple failures, may lock account temporarily

Code Implementation

```

"""
Django settings for FlutterDjangoRecipes project.
```

```python

Generated by 'django-admin startproject' using Django 5.1.6.

For more information on this file, see
https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.1/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-$3c8z4!qorf6f9n22o85ne)u&tyueek&kvxsip)nlklm7tg*d0'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

CORS_ALLOW_CREDENTIALS = True
CSRF_COOKIE_HTTPONLY = False
CSRF_COOKIE_SAMESITE = 'Lax'
ALLOWED_HOSTS = ["localhost", "127.0.0.1", '10.0.2.2']

CORS_ALLOWED_ORIGINS = [
    "http://localhost:61797",  # Default for local React/Flutter apps running on port 3000
    "http://127.0.0.1:61797",  # Equivalent localhost with 127.0.0.1
    "http://10.0.2.2:61797",   # For Android emulators to access your local Django backend
]

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
```

```
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = 'khawajafahadzia1@gmail.com'
EMAIL_HOST_PASSWORD = 'vamx jlpw bren opfs'
DEFAULT_FROM_EMAIL = 'khawajafahadzia1@gmail.com'

#CORS_ALLOW_ALL_ORIGINS = True
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'accounts',
    'recipes',
    'rest_framework',
    'corsheaders',
]

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.SessionAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
}

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'corsheaders.middleware.CorsMiddleware',
```

```

]

ROOT_URLCONF = 'FlutterDjangoRecipes.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': ['templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'FlutterDjangoRecipes.wsgi.application'

# Database
# https://docs.djangoproject.com/en/5.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
```

```
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
# Internationalization
# https://docs.djangoproject.com/en/5.1/topics/i18n/

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.1/howto/static-files/
STATIC_URL = 'static/'
MEDIA_URL = 'media/'
STATIC_ROOT = BASE_DIR / 'assets'
MEDIA_ROOT = BASE_DIR / 'media'

STATICFILES_DIRS = [
    BASE_DIR / 'static'
]
# Default primary key field type
# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

Listing 3.1: Password Reset Verification

3.3 Backend File 2 - urls.py

TABLE 3.2: Django URL Configuration – `urls.py`

File Name	`urls.py`
Location	Root project directory
Purpose	This file defines the main URL routing configuration for the Django project. It maps different URL patterns to their respective applications and includes the admin panel.
Imports	`from django.contrib import admin` `from django.urls import path, include`

URL Patterns	`path('admin/', admin.site.urls)`
	– Routes to Django admin interface.
	`path('', include("accounts.urls"))`
	– Routes the root URL to the `accounts` app.
	`path('recipes/', include("recipes.urls"))`
	– Routes URLs starting with `recipes/` to the `recipes` app.

Comments	The file also contains a commented-out import for `re_path`, which may be used for regex-based URL patterns if needed.

Code Implementation

```
 from django.contrib import admin
from django.urls import path, include#, re_path
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include("accounts.urls")),
    path('recipes/', include("recipes.urls"))]
```

LISTING 3.2: URL File

3.4 Backend File 3 - Accounts/urls.py

TABLE 3.3: Django URL Configuration – `accounts/urls.py`

File Name	`accounts/urls.py`
Variable Name	`urlpatterns`
Purpose	Defines URL patterns for the `accounts` app, mapping specific endpoints to view functions for user authentication and account management features.
URL Patterns	`''` – `views.loginPage`: Loads the login page. `gettingcsrftoken/` – `views.getCsrfToken`: Retrieves CSRF token for frontend security. `register/` – `views.registerPage`: Loads the registration page. `resetPassword/` – `views.resetPassword`: Initiates password reset process. `verifyOTP/` – `views.verifyOTP`: Verifies OTP during account actions. `updatePassword/` – `views.updatePassword`: Updates user password after verification.
App Namespace	`accounts` – Enables namespacing of URLs to avoid collisions across Django apps.

Code Implementation

```
from django.urls import path
from . import views
app_name = 'accounts'
urlpatterns = [
    path('', views.loginPage, name='login-page'),
    path('gettingcsrftoken/', views.getCsrfToken, name='get-csrf-token'),path('register/', views.registerPage, name='register-page'),path('resetPassword/', views.resetPassword, name='reset-password'),
    path('verifyOTP/', views.verifyOTP, name='verify-otp'),path('updatePassword/',views.updatePassword, name='update-password'),]
```

LISTING 3.3: Accounts/URL File

3.5 Backend File 4 - Accounts/models.py

TABLE 3.4: Django Model – `OTP` in `accounts/models.py`

File Name	`accounts/models.py`
Model Name	`OTP`
Purpose	Stores one-time passwords (OTPs) associated with user emails for actions like registration or password reset. Includes expiration logic.
Fields	`email` – `EmailField(unique=True)`: Stores the user's email, must be unique. `otp` – `CharField(max_length=6)`: Stores a 6-digit OTP. `created_at` – `DateTimeField(auto_now_add=True)`: Stores the time when the OTP was generated.
Method	`is_expired(self)` – Checks whether the OTP is expired by comparing current time with `created_at + 60 seconds`.

Code Implementation

```
from django.db import models
from django.utils import timezone
class OTP(models.Model):
    email = models.EmailField(unique=True)
    otp = models.CharField(max_length=6)
    created_at = models.DateTimeField(auto_now_add=True)
    def is_expired(self):
        return timezone.now() > self.created_at + timezone.
timedelta(seconds=60)
```

LISTING 3.4: Accounts/models File

3.6 Backend File 5 - Accounts/views.py

Table 3.5: Django Views – `accounts/views.py`

File Name	`accounts/views.py`
Purpose	Contains API endpoints for user authentication and account management functionalities such as login, registration, CSRF token retrieval, password reset, OTP verification, and password update.
Function Name	**Description**
`getCsrfToken`	**Method: GET** Returns a CSRF token to the frontend. Ensures CSRF protection for subsequent POST requests.
`loginPage`	**Method: POST** Authenticates the user using email and password. Logs the user in and returns their basic profile if successful. Logs out any existing session first.
`registerPage`	**Method: POST** Registers a new user with email, password, first name, and last name. Automatically logs in the new user. Fails if email already exists.
`resetPassword`	**Method: POST** Generates a 5-digit OTP and sends it to the user's email for password reset. Stores the OTP and timestamp in the database.
`verifyOTP`	**Method: POST** Verifies the OTP submitted by the user. Checks for expiry and correctness of the OTP. Returns appropriate success/failure messages.
`updatePassword`	**Method: POST** Updates the user's password after OTP verification. Fetches the user by email and updates the password securely using Django's built-in methods.

Permissions	All views are decorated with `@permission_classes([AllowAny])`, allowing unauthenticated access.
CSRF Protection	`@ensure_csrf_cookie` for GET request, `@csrf_protect` for all POST requests to prevent cross-site request forgery.
Email Functionality	Uses `send_mail()` from Django to email OTPs during password reset process.

Code Implementation

```
from django.contrib.auth import get_user_model, authenticate, login
    , logout
from django.http import JsonResponse
from django.middleware.csrf import get_token
from rest_framework.decorators import api_view,permission_classes
from rest_framework.response import Response
from django.views.decorators.csrf import ensure_csrf_cookie,
    csrf_protect
import random
from .models import OTP
from django.core.mail import send_mail
from django.conf import settings
from django.utils import timezone
from rest_framework.permissions import AllowAny

User = get_user_model()

@api_view(['GET'])
@ensure_csrf_cookie
@permission_classes([AllowAny])
def getCsrfToken(request):
    """Endpoint to get CSRF token for frontend"""
    return JsonResponse({'csrfToken': get_token(request)})

@api_view(['POST'])
@csrf_protect
@permission_classes([AllowAny])
def loginPage(request):
```

```
    """Handle user login with email and password"""
    logout(request)
    username = request.data.get('email')
    password = request.data.get('password')

    user = authenticate(request, username=username, password=
   password)

    if user is not None:
        login(request, user)
        return Response({
            "success": True,
            "message": "Login successful",
            "user": {
                "id": user.id,
                "email": user.username,
                "first_name": user.first_name,
                "last_name": user.last_name
            }
        })
    return Response({"success": False, "message": "Invalid
   credentials"})

@api_view(['POST'])
@csrf_protect
@permission_classes([AllowAny])
def registerPage(request):
    """Handle new user registration"""
    logout(request)
    username = request.data.get('email')
    password = request.data.get('password')
    first_name = request.data.get('first_name')
    last_name = request.data.get('last_name')

    if User.objects.filter(username=username).exists():
        return Response({'success': False, 'message': 'Email
   already exists'})

    user = User.objects.create_user(
        username=username,
        password=password,
```

```
        first_name=first_name,
        last_name=last_name
    )
    login(request, user)
    return Response({
        'success': True,
        'message': 'Registration successful',
        'user': {
            'id': user.id,
            'email': user.username,
            'first_name': user.first_name,
            'last_name': user.last_name
        }
    })

@api_view(['POST'])
@csrf_protect
@permission_classes([AllowAny])
def resetPassword(request):
    """Initiate password reset process"""
    email = request.data.get('email')
    if not User.objects.filter(email=email).exists():
        return Response({'success': False})

    otp = str(random.randint(10000, 99999))
    OTP.objects.update_or_create(
        email=email,
        defaults={'otp': otp, 'created_at': timezone.now()}
    )
    send_mail(
        "Password Reset OTP",
        f"Your OTP is {otp}. It will expire in 60 seconds.",
        settings.EMAIL_HOST_USER,
        [email]
    )
    return Response({'success': True})

@api_view(['POST'])
@csrf_protect
@permission_classes([AllowAny])
def verifyOTP(request):
```

```
    """Verify OTP for password reset"""
    email = request.data.get('email')
    otp_entered = request.data.get('otp')

    try:
        otp_record = OTP.objects.get(email=email)
        if otp_record.is_expired():
            return Response({'success': False, 'message': 'OTP expired'}, status=400)

        if otp_record.otp == otp_entered:
            return Response({'success': True, 'message': 'OTP verified'})
        return Response({'success': False, 'message': 'Invalid OTP'}, status=400)
    except OTP.DoesNotExist:
        return Response({'success': False, 'message': 'OTP not found'}, status=400)

@api_view(['POST'])
@csrf_protect
@permission_classes([AllowAny])
def updatePassword(request):
    """Update user password after OTP verification"""
    email = request.data.get('email')
    new_password = request.data.get('new_password')

    try:
        user = User.objects.get(username=email)
        user.set_password(new_password)
        user.save()
        return Response({'success': True, 'message': 'Password updated'})
    except User.DoesNotExist:
        return Response({'success': False, 'message': 'User not found'}, status=400)
```

Listing 3.5: Accounts/views File

3.7 Backend File 6 - recipes/urls.py

TABLE 3.6: Django URL Configuration – `recipes/urls.py`

File Name	`recipes/urls.py`
Variable Name	`urlpatterns`
Purpose	Maps URL patterns to view functions for handling recipe-related operations such as listing, creating, retrieving, updating, deleting, and recommending recipes.
URL Patterns	`''` – `views.recipe_list`: GET: List all recipes POST: Create a new recipe `<int:pk>/` – `views.recipe_detail`: GET: Retrieve a specific recipe by primary key PUT: Update an existing recipe DELETE: Delete a recipe `recommend/` – `views.recommend_recipes`: Return recipe recommendations based on user preferences. `recommend_calorie_based/` – `views.recommend_calorie_based`: Return recipe recommendations based on calorie constraints.
App Namespace	`recipes` – Used for namespacing URLs to prevent naming conflicts with other Django apps.

Code Implementation

```
from django.urls import path
from . import views

app_name = 'recipes'

urlpatterns = [
    # GET: List all recipes | POST: Create new recipe
```

```
    path('', views.recipe_list, name='recipe-list'),

    # GET: Get single recipe | PUT: Update recipe | DELETE: Delete recipe
    path('<int:pk>/', views.recipe_detail, name='recipe-detail'),

    path('recommend/', views.recommend_recipes, name='recipe-recommend'),
    path('recommend_calorie_based/', views.recommend_calorie_based, name='calorie-recommend'),
]
```

Listing 3.6: Accounts/models File

3.8 Backend File 7 - Recipes/views.py

Table 3.7: Django Views – `recipes/views.py`

File Name	`recipes/views.py`
Imports	DRF decorators and classes (`api_view`, `permission_classes`, `Response`, `IsAuthenticated`, `status`), Django shortcuts (`get_object_or_404`), models (`Recipes`, `RecipeIngredients`), CSRF protection decorator, and external recommendation modules (`recipe_recommender`, `calorie_model`).
Function: `recipe_list`	Handles `GET` and `POST` requests: - `GET`: Returns list of all recipes for authenticated user with their ingredients. - `POST`: Creates a new recipe and associated ingredients for the authenticated user. Protected by authentication and CSRF.
Function: `recipe_detail`	Handles `GET`, `PUT`, and `DELETE` requests for a recipe specified by primary key: - `GET`: Returns details of the specific recipe with ingredients. - `PUT`: Updates recipe's name, description, and ingredients. - `DELETE`: Deletes the recipe. Uses `get_object_or_404` to ensure recipe belongs to user. Protected by authentication and CSRF.
Function: `recommend_recipes`	Handles `POST` requests to provide recipe recommendations based on input ingredients and palette: - Validates inputs are non-empty lists. - Calls external `recipe_recommender.recommend_recipes` function. - Returns list of recommended recipes or error on failure. Protected by authentication and CSRF.

Function: `recommend_calorie_based`	Handles `POST` requests to provide calorie-based recipe recommendations: - Extracts numeric nutrition features and ingredients list from request data. - Calls external `calorie_model.recommend_recipes` function. - Returns recommendations or error on failure. Protected by authentication and CSRF.

Code Implementation

```
from rest_framework.decorators import api_view, permission_classes
from rest_framework.response import Response
from rest_framework.permissions import IsAuthenticated
from rest_framework import status
from django.shortcuts import get_object_or_404
from .models import Recipes, RecipeIngredients
from django.views.decorators.csrf import csrf_protect
from .palatte_based import recipe_recommender
from .calorie_based import app as calorie_model

@api_view(['GET', 'POST'])
@permission_classes([IsAuthenticated])
@csrf_protect
def recipe_list(request):
    """List all recipes or create new recipe"""
    if request.method == 'GET':
        recipes = Recipes.objects.filter(user=request.user)
        data = [{
            'id': recipe.id,
            'name': recipe.name,
            'description': recipe.description,
            'ingredients': [
                {'id': i.id, 'name': i.name}
                for i in recipe.recipeingredients_set.all()
            ]
        } for recipe in recipes]
        return Response(data)

```

```
    elif request.method == 'POST':
        recipe = Recipes.objects.create(
            user=request.user,
            name=request.data.get('name'),
            description=request.data.get('description')
        )

        for ingredient in request.data.get('ingredients', []):
            RecipeIngredients.objects.create(
                recipe=recipe,
                name=ingredient
            )

        return Response({
            'id': recipe.id,
            'name': recipe.name,
            'description': recipe.description,
            'ingredients': [
                {'id': i.id, 'name': i.name}
                for i in recipe.recipeingredients_set.all()
            ]
        }, status=201)

@api_view(['GET', 'PUT', 'DELETE'])
@permission_classes([IsAuthenticated])
@csrf_protect
def recipe_detail(request, pk):
    """Get, update or delete specific recipe"""
    recipe = get_object_or_404(Recipes, pk=pk, user=request.user)

    if request.method == 'GET':
        return Response({
            'id': recipe.id,
            'name': recipe.name,
            'description': recipe.description,
            'ingredients': [
                {'id': i.id, 'name': i.name}
                for i in recipe.recipeingredients_set.all()
            ]
        })

```

```
    elif request.method == 'PUT':
        recipe.name = request.data.get('name', recipe.name)
        recipe.description = request.data.get('description', recipe
    .description)
        recipe.save()

        recipe.recipeingredients_set.all().delete()
        for ingredient in request.data.get('ingredients', []):
            RecipeIngredients.objects.create(
                recipe=recipe,
                name=ingredient
            )

        return Response({
            'id': recipe.id,
            'name': recipe.name,
            'description': recipe.description,
            'ingredients': [
                {'id': i.id, 'name': i.name}
                for i in recipe.recipeingredients_set.all()
            ]
        })

    elif request.method == 'DELETE':
        recipe.delete()
        return Response(status=204)


@api_view(['POST'])
@permission_classes([IsAuthenticated])
@csrf_protect
def recommend_recipes(request):
    """Handle recipe recommendations"""
    try:


        data = request.data
        ingredients = data.get('ingredients', [])
        palette = data.get('palette', [])


```

```
        if not isinstance(ingredients, list) or not isinstance(palette, list):
            return Response(
                {'error': 'Ingredients and palette must be lists'},
                status=status.HTTP_400_BAD_REQUEST
            )

        if not ingredients or not palette:
            return Response(
                {'error': 'Both ingredients and palette are required'},
                status=status.HTTP_400_BAD_REQUEST
            )


        ingredients_str = ', '.join([str(i) for i in ingredients])
        palette_str = ', '.join([str(p) for p in palette])


        recommendations = recipe_recommender.recommend_recipes(
            input_ingredients_str=ingredients_str,
            input_palette_str=palette_str,
            top_k=5
        )


        results = recommendations.to_dict('records')

        return Response(results)

    except Exception as e:
        return Response(
            {'error': f'Recommendation failed: {str(e)}'},
            status=status.HTTP_500_INTERNAL_SERVER_ERROR
        )
@api_view(['POST'])
@permission_classes([IsAuthenticated])
@csrf_protect
def recommend_calorie_based(request):
    """Get calorie-based recipe recommendations"""
    try:
```

```
        data = request.data
        input_features = [
            float(data.get('calories', 0)),
            float(data.get('fat', 0)),
            float(data.get('carbs', 0)),
            float(data.get('protein', 0)),
            float(data.get('cholesterol', 0)),
            float(data.get('sodium', 0)),
            float(data.get('fiber', 0)),
            ','.join(data.get('ingredients', [])),]
        recommendations = calorie_model.recommend_recipes(
    input_features)
        results = recommendations.to_dict('records')
        return Response(results)
    except Exception as e:
        return Response(
            {'error': str(e)},
  status=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

Listing 3.7: Recipes/views File

3.9 Backend File 8 - Recipes/models.py

TABLE 3.8: Django Models – `recipes/models.py`

File Name	`recipes/models.py`
Imports	Django `models` module, Django `User` model from `auth` contrib.
Model: `Recipes`	Represents a recipe entity with the following fields: - `id`: `BigAutoField`, primary key. - `name`: `CharField` with max length 300, optional. - `description`: `TextField`, optional. - `user`: ForeignKey to `User`, cascade on delete. Links recipe to its creator.
Model: `RecipeIngredients`	Represents ingredients linked to a recipe with the following fields: - `id`: `BigAutoField`, primary key. - `name`: `CharField` with max length 300, optional. - `recipe`: ForeignKey to `Recipes`, cascade on delete. Establishes many-to-one relationship with recipe.

Code Implementation

```python
from django.db import models
from django.contrib.auth.models import User

class Recipes(models.Model):
    id = models.BigAutoField(db_column="id", primary_key=True)
    name = models.CharField(max_length=300, blank=True)
    description = models.TextField(blank=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE)

class RecipeIngredients(models.Model):
    id = models.BigAutoField(db_column="id", primary_key=True)
    name = models.CharField(max_length=300,blank=True)
    recipe = models.ForeignKey(Recipes, on_delete=models.CASCADE)
```

LISTING 3.8: reviews/models File

3.10 Backend File 9 - Services/recipe_service.dart

Table 3.9: Dart Service – `RecipeService`

File	`recipe_service.dart`
Imports	`dart:convert`, `http` package (as http), `auth_service.dart`
Class	`RecipeService`
Constants	- `baseUrl`: String constant set to `'http://10.0.2.2:9091'`
Methods	• `getRecipes()`: - Makes GET request to `'/recipes/'` endpoint - Returns list of recipes if status code is 200 - Throws exception on failure • `addRecipe({required params})`: - Makes POST request to `'/recipes/'` with recipe data - Accepts name, description, and ingredients list - Returns created recipe if status code is 201 - Throws exception on failure • `updateRecipe({required params})`: - Makes PUT request to `'/recipes/{id}/'` - Accepts id, name, description, and ingredients list - Returns updated recipe if status code is 200 - Throws exception on failure • `deleteRecipe(id)`: - Makes DELETE request to `'/recipes/{id}/'` - Expects status code 204 for success - Throws exception on failure

Common Patterns	- All methods use `http.Client()` with try-finally to ensure cleanup - All requests include auth headers from `AuthService.getAuthHeaders()` - All methods throw exceptions for non-success status codes - Request bodies are JSON encoded with `jsonEncode()`

Code Implementation

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

class RecipeService {
  static const String baseUrl = 'http://10.0.2.2:9091';

  static Future<List<dynamic>> getRecipes() async {
    final client = http.Client();
    try {
      final headers = await AuthService.getAuthHeaders();
      final response = await client.get(
        Uri.parse('$baseUrl/recipes/'),
        headers: headers,
      );

      if (response.statusCode == 200) {
        return jsonDecode(response.body);
      } else {
        throw Exception('Failed to load recipes (${response.
    statusCode})');
      }
    } finally {
      client.close();
    }
  }

  static Future<Map<String, dynamic>> addRecipe({
    required String name,
```

```
    required String description,
    required List<String> ingredients,
  }) async {
    final client = http.Client();
    try {
      final headers = await AuthService.getAuthHeaders();
      final response = await client.post(
        Uri.parse('$baseUrl/recipes/'),
        headers: headers,
        body: jsonEncode({
          'name': name,
          'description': description,
          'ingredients': ingredients,
        }),
      );

      if (response.statusCode == 201) {
        return jsonDecode(response.body);
      } else {
        throw Exception('Failed to add recipe (${response.
  statusCode})');
      }
    } finally {
      client.close();
    }
  }

  static Future<Map<String, dynamic>> updateRecipe({
    required int id,
    required String name,
    required String description,
    required List<String> ingredients,
  }) async {
    final client = http.Client();
    try {
      final headers = await AuthService.getAuthHeaders();
      final response = await client.put(
        Uri.parse('$baseUrl/recipes/$id/'),
        headers: headers,
        body: jsonEncode({
          'name': name,
```

```
          'description': description,
          'ingredients': ingredients,
        }),
      );

      if (response.statusCode == 200) {
        return jsonDecode(response.body);
      } else {
        throw Exception('Failed to update recipe (${response.
    statusCode})');
      }
    } finally {
      client.close();
    }
  }

  static Future<void> deleteRecipe(int id) async {
    final client = http.Client();
    try {
      final headers = await AuthService.getAuthHeaders();
      final response = await client.delete(
        Uri.parse('$baseUrl/recipes/$id/'),
        headers: headers,
      );

      if (response.statusCode != 204) {
        throw Exception('Failed to delete recipe (${response.
    statusCode})');
      }
    } finally {
      client.close();
    }
  }
}
```

Listing 3.9: Services/recipe$_service.dartFile$

3.11 Backend File 10 - auth service.dart

Table 3.10: Dart Service – `AuthService`

File	`auth_service.dart`
Imports	`dart:convert`, `http` package (as http)
Class	`AuthService`
Constants & Variables	• `baseUrl`: String constant set to `'http://10.0.2.2:9091'` • `sessionCookie`: Static nullable String for session storage • `csrfToken`: Static nullable String for CSRF token storage
Helper Methods	• `_extractCookies(headers)`: - Parses Set-Cookie headers to extract sessionid and csrftoken - Returns map with extracted cookies

Main Methods	• `getCsrfToken()`: - Makes GET request to `'/gettingcsrftoken/'` - Extracts and stores CSRF token and session cookie - Returns token and cookie string • `login({required email, password})`: - First obtains CSRF token - Makes POST request with credentials - Returns success status and message - Updates session cookie on success • `register({required params})`: - First obtains CSRF token - Makes POST request with registration data - Returns success status and message - Updates session cookie and CSRF token on success • `sendResetPasswordEmail(email)`: - Makes POST request to reset password endpoint - Returns success status or null on failure • `getAuthHeaders()`: - Ensures CSRF token and session cookie exist - Returns headers with auth tokens for authenticated requests

Common Patterns	• Uses `http.Client()` with try-finally for resource cleanup • Handles CSRF token management for Django backend • Maintains session state through static variables • Returns structured response maps with success status • Uses `jsonEncode()` for request bodies
Security Features	• CSRF token handling for all mutating requests • Session cookie management Secure credential transmission • Token refresh mechanism

Code Implementation

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

class AuthService {
  static const String baseUrl = 'http://10.0.2.2:9091';
  static String? sessionCookie;
  static String? csrfToken;

  static Map<String, String> _extractCookies(Map<String, String>
   headers) {
    final setCookie = headers['set-cookie'] ?? '';
    final cookies = setCookie.split(',');

    String? session;
    String? csrf;
```

```

    for (var cookie in cookies) {
      if (cookie.contains('sessionid=')) {
        session = cookie.split(';')[0].trim().split('=').last;
      } else if (cookie.contains('csrftoken=')) {
        csrf = cookie.split(';')[0].trim().split('=').last;
      }
    }

    return {
      if (session != null) 'sessionid': session,
      if (csrf != null) 'csrftoken': csrf,
    };
  }

  static Future<Map<String, dynamic>?> getCsrfToken() async {
    final client = http.Client();
    try {
      final response = await client.get(
        Uri.parse('$baseUrl/gettingcsrftoken/'),
        headers: {'Content-Type': 'application/json'},
      );

      if (response.statusCode == 200) {
        final cookies = _extractCookies(response.headers);
        csrfToken = cookies['csrftoken'];
        sessionCookie = cookies['sessionid'];
        return {
          'csrfToken': csrfToken,
          'csrfCookie': 'csrftoken=$csrfToken; sessionid=
  $sessionCookie',
        };
      }
      return null;
    } finally {
      client.close();
    }
  }

  static Future<Map<String, dynamic>> login({
    required String email,
```

```
    required String password,
  }) async {
    final csrfData = await getCsrfToken();
    if (csrfData == null) {
      return {'success': false, 'message': 'Failed to get CSRF
  token'};
    }

    final client = http.Client();
    try {
      final response = await client.post(
        Uri.parse('$baseUrl/'),
        headers: {
          'Content-Type': 'application/json',
          'X-CSRFToken': csrfData['csrfToken']!,
          'Cookie': csrfData['csrfCookie']!,
        },
        body: jsonEncode({'email': email, 'password': password}),
      );

      if (response.statusCode == 200) {
        if (jsonDecode(response.body)['success']) {
          final cookies = _extractCookies(response.headers);
          sessionCookie = cookies['sessionid'] ?? sessionCookie;
          return {'success': true, 'message': 'Login successful'};
        }
        return {'success': false, 'message': 'Invalid credentials'
  };
      } else {
        return {'success': false, 'message': 'Invalid credentials'
  };
      }
    } finally {
      client.close();
    }
  }

  static Future<Map<String, dynamic>> register({
    required String email,
    required String password,
    required String firstName,
```

```
    required String lastName,
  }) async {
    final csrfData = await getCsrfToken();
    if (csrfData == null) {
      return {'success': false, 'message': 'Failed to get CSRF
  token'};
    }

    final client = http.Client();
    try {
      final response = await client.post(
        Uri.parse('$baseUrl/register/'),
        headers: {
          'Content-Type': 'application/json',
          'X-CSRFToken': csrfData['csrfToken']!,
          'Cookie': csrfData['csrfCookie'] ?? '',
        },
        body: jsonEncode({
          'email': email,
          'password': password,
          'first_name': firstName,
          'last_name': lastName,
        }),
      );

      if (response.statusCode == 200) {
        if (jsonDecode(response.body)['success']){
          final cookies = _extractCookies(response.headers);
        sessionCookie = cookies['sessionid'] ?? sessionCookie;
        csrfToken = cookies['csrftoken'] ?? csrfToken;
        return {'success': true, 'message': 'Registration
  successful'};
        }
        return {'success': false, 'message': 'Registration failed'
  };
      } else {
        return {'success': false, 'message': 'Registration failed'
  };
      }
    } finally {
      client.close();
```

```
    }
  }

  static Future<bool?> sendResetPasswordEmail(String email) async {
    final client = http.Client();
    final csrfData = await getCsrfToken();
    if (csrfData == null) {
      print("Failed to get CSRF token");
      return null;
    }
    try {
      final response = await client.post(
        Uri.parse('$baseUrl/resetPassword/'),
        headers: {
          'Content-Type': 'application/json',
          'X-CSRFToken': csrfData['csrfToken']!,
          'Cookie': '${csrfData['csrfCookie']}; ${sessionCookie ??
  ''}',
        },
        body: jsonEncode({'email': email}),
      );
      if (response.statusCode == 200) {
        return jsonDecode(response.body)['success'] as bool?;
      } else {
        print('Failed with status: ${response.statusCode}');
        return null;
      }
    } finally {
      client.close();
    }}
  static Future<Map<String, String>> getAuthHeaders() async {
    if (csrfToken == null || sessionCookie == null) {
      await getCsrfToken();
    }
    return {
      'Content-Type': 'application/json',
      'X-CSRFToken': csrfToken ?? '',
      'Cookie': 'csrftoken=$csrfToken; sessionid=$sessionCookie',
    };}}
```

Listing 3.10: auth service.dart File

3.12 Backend File 11 - Recipe_detail_page.dart

Table 3.11: Flutter Widget – `RecipeDetailPage`

File	`recipe_detail_page.dart`
Imports	`flutter/material.dart`
Class	`RecipeDetailPage` (extends `StatelessWidget`)
Constructor	• Requires a `recipe` parameter (Map type) • Optional `key` parameter passed to super
Build Method	Returns a `Scaffold` widget containing: • An `AppBar` with recipe name as title • A `Padding` widget with 16px padding • A `Column` with cross-axis start alignment containing: – "Description" heading (bold 18px text) – 8px vertical spacing (`SizedBox`) – Recipe description text – 16px vertical spacing – "Ingredients" heading (bold 18px text) – 8px vertical spacing – Ingredients list (joined comma-separated string)

UI Structure	• Simple vertical layout with clear section headings • Consistent spacing between elements • Basic typography with bold headings • Displays recipe name, description, and ingredients
Data Handling	• Expects recipe data as a Map with: – `name` (String) - for app bar title – `description` (String) - main content – `ingredients` (List¡Map¿) - with `name` fields • Transforms ingredients list to comma-separated string
Key Features	• Stateless widget (immutable) • Simple recipe detail display • Responsive layout with padding • Clear visual hierarchy

Code Implementation

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

class RecipeDetailPage extends StatelessWidget {
  final Map recipe;

  const RecipeDetailPage({super.key, required this.recipe});

  @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text(recipe['name'])),
      body: Padding(
        padding: EdgeInsets.all(16),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              'Description',
              style: TextStyle(fontSize: 18, fontWeight: FontWeight
    .bold),
            ),
            SizedBox(height: 8),
            Text(recipe['description']),
            SizedBox(height: 16),
            Text(
              'Ingredients',
              style: TextStyle(fontSize: 18, fontWeight: FontWeight
    .bold),
            ),
            SizedBox(height: 8),
            Text(
              (recipe['ingredients'] as List)
                  .map((e) => e['name'] as String)
                  .join(', ')
            ),
          ],
        ),
      ),
    );
  }
}
```

Listing 3.11: Recipe_detail_page.dart File

3.13 Backend File 12 - App.dart

Table 3.12: App.dart

File	`ssApp.dart`
Imports	• `flutter/material.dart` • `home_page.dart` • `generate_recipe_page.dart` • `constraints_recipe.dart`
Class	`MainPage` (extends `StatefulWidget`)
State Class	`_MainPageState`
State Variables	• `_selectedIndex`: Tracks current bottom nav index (initial 0) • `_isLoading`: Loading state flag (initial true) • `_pages`: List of page widgets (HomePage, GenerateRecipePage, ConstraintsRecipe)
Lifecycle Methods	• `initState()`: Calls `_verifyAuthStatus()` on initialization • `_verifyAuthStatus()`: Mock authentication check (sets loading to false)
Interaction Methods	• `_onItemTapped(index)`: Updates `_selectedIndex` when bottom nav item tapped

Build Method	Returns either: • Loading indicator scaffold when `_isLoading` is true • Main scaffold with: – Current page from `_pages` list – Bottom navigation bar with 3 items: ∗ Home (house icon) ∗ Generate (auto_awesome icon) ∗ Constraints (auto_awesome_mosaic icon) – Purple selected item color – Grey unselected item color
Navigation Structure	• Three-page navigation system: – HomePage (default view) – GenerateRecipePage – ConstraintsRecipe • BottomNavigationBar for page switching • State maintains current page index
Key Features	• Stateful widget with loading state • Simple bottom navigation pattern • Placeholder auth verification • Color-coded navigation indicators

Code Implementation

```
import 'package:flutter/material.dart';
import 'package:flutter_django_recipes_frontend/home_page.dart';
import 'package:flutter_django_recipes_frontend/
    generate_recipe_page.dart';
import 'package:flutter_django_recipes_frontend/constraints_recipe.
    dart';

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

  @override
  State<MainPage> createState() => _MainPageState();
}

class _MainPageState extends State<MainPage> {
  int _selectedIndex = 0;
  bool _isLoading = true;

  static final List<Widget> _pages = <Widget>[
    const HomePage(),
    GenerateRecipePage(),
    ConstraintsRecipe(),
  ];

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

  Future<void> _verifyAuthStatus() async {
    setState(() => _isLoading = false);
  }

  void _onItemTapped(int index) {
    setState(() => _selectedIndex = index);
  }

  @override
```

```
  Widget build(BuildContext context) {
    if (_isLoading) {
      return const Scaffold(
        body: Center(child: CircularProgressIndicator()),
      );
    }

    return Scaffold(
      body: _pages[_selectedIndex],
      bottomNavigationBar: BottomNavigationBar(
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(Icons.home),
            label: 'Home',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.auto_awesome),
            label: 'Generate',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.auto_awesome_mosaic),
            label: 'Constraints',
          ),
        ],
        currentIndex: _selectedIndex,
        selectedItemColor: Colors.deepPurple,
        unselectedItemColor: Colors.grey,
        onTap: _onItemTapped,
      ),
    );
  }
}
```

Listing 3.12: App.dart File

3.14 Backend File 13 - Main.dart

Table 3.13: Flutter Main Application – `main.dart`

File	`main.dart` (Application Entry Point)
Imports	• `flutter/material.dart` • `login_page.dart`
Main Function	• Entry point of the Flutter application • Calls `runApp()` with `MyApp` widget
Widget: `MyApp`	• StatelessWidget that serves as root application widget • Constructor with optional key parameter
Build Method	Returns a `MaterialApp` with: • `debugShowCheckedModeBanner`: false (disables debug banner) • `home`: Set to `LoginPage()` (initial route)
Configuration	• Basic MaterialApp configuration • No theme or routes specified (minimal setup) • Direct navigation to LoginPage

Key Features	• Minimal application setup • Clean debug-free appearance • LoginPage as entry point • Standard Flutter application structure
Purpose	• Serves as the root of the application • Initializes Flutter framework • Sets up basic MaterialApp configuration • Defines initial screen (LoginPage)

Code Implementation

```
import "package:flutter/material.dart";
import "package:flutter_django_recipes_frontend/login_page.dart";

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: LoginPage(),
    );
  }
}
```

Listing 3.13: Main.dart

3.15 Backend File 14 - Ingridients_list.dart

Code Implementation

```
const List<String> paletteList = [
  'savory',
  'spicy',
  'sweet',
  'sour',
  'herby',
  'umami',
  'earthy',
  'fruity',
  'smoky',
  'neutral',
  'bitter',
];


const List<String> availableIngredients = [
      'olive oil',
      'canola oil',
      'all purpose flour',
      'sugar',
      'salt',
      'black pepper',
      'garlic',
      'onion',
      'butter',
      'eggs',
      'milk',
      'water',
      'chicken broth',
      'vegetable oil',
      'baking powder',
      'baking soda',
      'vanilla extract',
      'lemon juice',
      'soy sauce',
      'rice',
```

```
        'pasta',
        'tomato',
        'tomato sauce',
        'tomato paste',
        'cheese',
        'cheddar cheese',
        'parmesan cheese',
        'mozzarella cheese',
        'heavy cream',
        'sour cream',
        'yogurt',
        'mayonnaise',
        'honey',
        'brown sugar',
        'powdered sugar',
        'cornstarch',
        'vinegar',
        'apple cider vinegar',
        'balsamic vinegar',
        'red wine vinegar',
        'white wine vinegar',
        'mustard',
        'dijon mustard',
        'yellow mustard',
        'ketchup',
        'worcestershire sauce',
        'hot sauce',
        'sriracha',
        'bay leaf',
        'oregano',
        'basil',
        'thyme',
        'rosemary',
        'parsley',
        'cilantro',
        'cumin',
        'coriander',
        'paprika',
        'smoked paprika',
        'cayenne pepper',
        'red pepper flakes',
```

```
        'chili powder',
        'curry powder',
        'turmeric',
        'ginger',
        'nutmeg',
        'cinnamon',
        'cloves',
        'allspice',
        'cardamom',
        'star anise',
        'fennel seed',
        'sesame seeds',
        'poppy seeds',
        'caraway seeds',
        'celery seed',
        'mustard seeds',
        'bread crumbs',
        'panko breadcrumbs',
        'oats',
        'cornmeal',
        'flour tortillas',
        'corn tortillas',
        'potatoes',
        'sweet potatoes',
        'carrots',
        'celery',
        'bell peppers',
        'green bell peppers',
        'red bell peppers',
        'yellow bell peppers',
        'jalape o peppers',
        'serrano peppers',
        'habanero peppers',
        'cucumber',
        'zucchini',
        'yellow squash',
        'eggplant',
        'mushrooms',
        /* 'white mushrooms',
        'cremini mushrooms',
        'portobello mushrooms',
```

```
        'shiitake mushrooms',
        'spinach',
        'kale',
        'lettuce',
        'romaine lettuce',
        'iceberg lettuce',
        'arugula',
        'swiss chard',
        'collard greens',
        'cabbage',
        'green cabbage',
        'red cabbage',
        'bok choy',
        'brussels sprouts',
        'cauliflower',
        'broccoli',
        'asparagus',
        'green beans',
        'peas',
        'corn',
        'avocado',
        'lime',
        'lemon',
        'orange',
        'apple',
        'banana',
        'strawberries',
        'blueberries',
        'raspberries',
        'blackberries',
        'peaches',
        'pears',
        'plums',
        'cherries',
        'grapes',
        'pineapple',
        'mango',
        'kiwi',
        'coconut',
        'raisins',
        'dates',
```

```
        'prunes',
        'apricots',
        'figs',
        'almonds',
        'walnuts',
        'pecans',
        'cashews',
        'peanuts',
        'hazelnuts',
        'pistachios',
        'sunflower seeds',
        'pumpkin seeds',
        'chia seeds',
        'flax seeds',
        'quinoa',
        'couscous',
        'bulgur',
        'barley',
        'lentils',
        'black beans',
        'kidney beans',
        'pinto beans',
        'navy beans',
        'garbanzo beans',
        'cannellini beans',
        'split peas',
        'black eyed peas',
        'tofu',
        'tempeh',
        'seitan',
        'chicken',
        'chicken breast',
        'chicken thighs',
        'chicken wings',
        'chicken drumsticks',
        'turkey',
        'ground turkey',
        'turkey breast',
        'beef',
        'ground beef',
        'steak',
```

```
        'ribeye steak',
        'sirloin steak',
        'flank steak',
        'short ribs',
        'beef chuck',
        'pork',
        'pork chops',
        'pork tenderloin',
        'pork shoulder',
        'bacon',
        'ham',
        'sausage',
        'italian sausage',
        'chorizo',
        'hot dogs',
        'lamb',
        'lamb chops',
        'ground lamb',
        'fish',
        'salmon',
        'tuna',
        'cod',
        'halibut',
        'tilapia',
        'trout',
        'shrimp',
        'prawns',
        'scallops',
        'lobster',
        'crab',
        'clams',
        'mussels',
        'oysters',
        'anchovies',
        'sardines',
        'bread',
        'white bread',
        'whole wheat bread',
        'baguette',
        'ciabatta',
        'sourdough bread',
```

```
        'pita bread',
        'naan',
        'tortilla chips',
        'potato chips',
        'pretzels',
        'crackers',
        'saltine crackers',
        'graham crackers',
        'chocolate',
        'dark chocolate',
        'milk chocolate',
        'white chocolate',
        'chocolate chips',
        'cocoa powder',
        'vanilla bean',
        'maple syrup',
        'molasses',
        'agave nectar',
        'corn syrup',
        'peanut butter',
        'almond butter',
        'jam',
        'jelly',
        'marmalade',
        'hummus',
        'tahini',
        'soy milk',
        'almond milk',
        'coconut milk',
        'rice milk',
        'oat milk',
        'whipping cream',
        'half and half',
        'buttermilk',
        'evaporated milk',
        'condensed milk',
        'cream cheese',
        'ricotta cheese',
        'feta cheese',
        'goat cheese',
        'blue cheese',
```

```
        'gorgonzola cheese',
        'provolone cheese',
        'swiss cheese',
        'monterey jack cheese',
        'pepper jack cheese',
        'colby cheese',
        'american cheese',
        'velveeta cheese',
        'wine',
        'red wine',
        'white wine',
        'sherry',
        'beer',
        'rum',
        'vodka',
        'gin',
        'tequila',
        'whiskey',
        'bourbon',
        'scotch',
        'cognac',
        'brandy',
        'coffee',
        'tea',
        'green tea',
        'black tea',
        'herbal tea',
        'sparkling water',
        'club soda',
        'tonic water',
        'lemonade',
        'orange juice',
        'apple juice',
        'cranberry juice',
        'grapefruit juice',
        'pineapple juice',
        'tomato juice',
        'vegetable juice',
        'soda',
        'cola',
        'ginger ale',
```

```
        'root beer',
        'lemon lime soda',
        'gelatin',
        'yeast',
        'active dry yeast',
        'instant yeast',
        'shortening',
        'lard',
        'bacon fat',
        'duck fat',
        'chicken fat',
        'beef fat',
        'vegetable shortening',
        'margarine',
        'vegetable spread',
        'cooking spray',
        'nonstick spray',
        'ice',
        'dry ice',
        'liquid smoke',
        'food coloring',
        'vanilla pudding',
        'chocolate pudding',
        'butterscotch pudding',
        'tapioca pudding',
        'custard',
        'flan',
        'ice cream',
        'vanilla ice cream',
        'chocolate ice cream',
        'strawberry ice cream',
        'sherbet',
        'sorbet',
        'whipped topping',
        'cool whip',
        'marshmallows',
        'sprinkles',
        'candy',
        'caramels',
        'toffee',
        'gumdrops',
```

```
        'licorice',
        'gummy bears',
        'jelly beans',
        'mints',
        'peppermint',
        'cinnamon candy',
        'popcorn',
        'caramel corn',
        'cheese popcorn',
        'peanut brittle',
        'trail mix',
        'granola',
        'protein powder',
        'whey protein',
        'soy protein',
        'nutritional yeast',
        'wheat germ',
        'bran',
        'wheat bran',
        'oat bran',
        'psyllium husk',
        'arrowroot powder',
        'tapioca flour',
        'almond flour',
        'coconut flour',
        'rice flour',
        'chickpea flour',
        'buckwheat flour',
        'corn flour',
        'potato flour',
        'semolina',
        'bread flour',
        'cake flour',
        'pastry flour',
        'self rising flour',
        'whole wheat flour',
        'gluten free flour',
        'pancake mix',
        'biscuit mix',
        'cornbread mix',
        'cake mix',
```

```
        'brownie mix',
        'cookie mix',
        'muffin mix',
        'pie crust',
        'phyllo dough',
        'puff pastry',
        'pizza dough',
        'bread dough',
        'cookie dough',
        'brownie batter',
        'cake batter',
        'pancake batter',
        'waffle batter',
        'crepe batter',
        'french toast batter',
        'tempura batter',
        'beer batter',
        'marinade',
        'rub',
        'dry rub',
        'seasoning salt',
        'garlic salt',
        'onion salt',
        'celery salt',
        'seasoning blend',
        'italian seasoning',
        'poultry seasoning',
        'pumpkin pie spice',
        'apple pie spice',
        'chai spice',
        'five spice powder',
        'adobo seasoning',
        'creole seasoning',
        'cajun seasoning',
        'blackening seasoning',
        'jerk seasoning',
        'taco seasoning',
        'fajita seasoning',
        'ranch seasoning',
        'sazon seasoning',
        'old bay seasoning',
```

```
        'everything bagel seasoning',
        'furikake',
        'zaatar',
        'sumac',
        'harissa',
        'sambal oelek',
        'gochujang',
        'miso paste',
        'hoisin sauce',
        'oyster sauce',
        'fish sauce',
        'teriyaki sauce',
        'ponzu sauce',
        'duck sauce',
        'plum sauce',
        'sweet chili sauce',
        'peanut sauce',
        'alfredo sauce',
        'marinara sauce',
        'pesto',
        'tzatziki sauce',
        'aioli',
        'remoulade sauce',
        'tartar sauce',
        'cocktail sauce',
        'bbq sauce',
        'honey mustard',
        'ranch dressing',
        'blue cheese dressing',
        'caesar dressing',
        'italian dressing',
        'french dressing',
        'thousand island dressing',
        'vinaigrette',
        'balsamic vinaigrette',
        'raspberry vinaigrette',
        'honey vinaigrette',
        'mustard vinaigrette',
        'citrus vinaigrette',
        'herb vinaigrette',
        'shallot vinaigrette',
```

```
        'garlic vinaigrette',
        'ginger vinaigrette',
        'sesame vinaigrette',
        'soy vinaigrette',
        'wasabi',
        'horseradish',
        'ginger paste',
        'garlic paste',
        'tomato puree',
        'apple puree',
        'banana puree',
        'pumpkin puree',
        'sweet potato puree',
        'butternut squash puree',
        'cauliflower puree',
        'avocado puree',
        'mango puree',
        'pear puree',
        'peach puree',
        'berry puree',
        'fruit puree',
        'vegetable puree',
        'bean puree',
        'lentil puree',
        'hummus',
        'guacamole',
        'salsa',
        'pico de gallo',
        'salsa verde',
        'corn salsa',
        'mango salsa',
        'pineapple salsa',
        'peach salsa',
        'black bean salsa',
        'tomato salsa',
        'fruit salsa',
        'chutney',
        'mango chutney',
        'apple chutney',
        'tomato chutney',
        'onion chutney',
```

```
        'cranberry chutney',
        'pear chutney',
        'peach chutney',
        'plum chutney',
        'fig chutney',
        'relish',
        'pickle relish',
        'corn relish',
        'pepper relish',
        'onion relish',
        'cucumber relish',
        'zucchini relish',
        'eggplant relish',
        'tomato relish',
        'fruit relish',
        'vegetable relish',
        'pickles',
        'dill pickles',
        'bread and butter pickles',
        'sweet pickles',
        'gherkins',
        'cornichons',
        'pickled jalape os',
        'pickled onions',
        'pickled beets',
        'pickled carrots',
        'pickled cucumbers',
        'pickled ginger',
        'pickled vegetables',
        'kimchi',
        'sauerkraut',
        'olives',
        'black olives',
        'green olives',
        'kalamata olives',
        'olive tapenade',
        'capers',
        'artichoke hearts',
        'sun dried tomatoes',
        'roasted red peppers',
        'pepperoncini',
```

```
        'giardiniera',
        'banana peppers',
        'cherry peppers',
        'pepperoni',
        'salami',
        'prosciutto',
        'pancetta',
        'capicola',
        'soppressata',
        'mortadella',
        'corned beef',
        'pastrami',
        'roast beef',
        'turkey breast',
        'chicken breast',
        'ham',
        'bologna',
        'liverwurst',
        'braunschweiger',
        'head cheese',
        'blood sausage',
        'andouille sausage',
        'kielbasa',
        'bratwurst',
        'italian sausage',
        'breakfast sausage',
        'chorizo',
        'linguica',
        'loukaniko',
        'merguez',
        'sucuk',
        'noodles',
        'ramen noodles',
        'udon noodles',
        'soba noodles',
        'rice noodles',
        'vermicelli',
        'angel hair pasta',
        'spaghetti',
        'linguine',
        'fettuccine',
```

```
        'penne',
        'rigatoni',
        'ziti',
        'macaroni',
        'fusilli',
        'rotini',
        'farfalle',
        'orzo',
        'lasagna noodles',
        'manicotti',
        'cannelloni',
        'ravioli',
        'tortellini',
        'gnocchi',
        'pierogi',
        'dumplings',
        'wonton wrappers',
        'egg roll wrappers',
        'spring roll wrappers',
        'phyllo dough',
        'puff pastry',
        'pie crust',
        'tart shell',
        'graham cracker crust',
        'cookie crust',
        'shortbread crust',
        'pastry dough',
        'biscuit dough',
        'pizza dough',
        'bread dough',
        'roll dough',
        'croissant dough',
        'danish dough',
        'brioche dough',
        'challah dough',
        'bagel dough',
        'pretzel dough',
        'doughnut dough',
        'fritter batter',
        'pancake batter',
        'waffle batter',
```

```
        'crepe batter',
        'french toast batter',
        'tempura batter',
        'beer batter',
        'cake batter',
        'brownie batter',
        'cookie dough',
        'muffin batter',
        'quick bread batter',
        'yeast bread dough',
        'sourdough starter',
        'poolish',
        'biga',
        'sponge',
        'levain',
        'mother dough',
        'wild yeast starter',
        'sourdough culture',
        'sourdough discard',
        'active sourdough starter',
        'ripe sourdough starter',
        'fed sourdough starter',
        'unfed sourdough starter',
        'liquid levain',
        'stiff levain',
        'firm levain',
        'liquid sourdough starter',
        'stiff sourdough starter',
        'firm sourdough starter',
        '100% hydration starter',
        '50% hydration starter',
        '60% hydration starter',
        '70% hydration starter',
        '80% hydration starter',
        '90% hydration starter',
        'whole wheat starter',
        'rye starter',
        'spelt starter',
        'einkorn starter',
        'kamut starter',
        'emmer starter',
```

```
        'farro starter',
        'barley starter',
        'oat starter',
        'buckwheat starter',
        'quinoa starter',
        'amaranth starter',
        'teff starter',
        'millet starter',
        'sorghum starter',
        'rice starter',
        'corn starter',
        'potato starter',
        'sweet potato starter',
        'pumpkin starter',
        'squash starter',
        'beet starter',
        'carrot starter',
        'parsnip starter',
        'turnip starter',
        'radish starter',
        'daikon starter',
        'ginger starter',
        'turmeric starter',
        'garlic starter',
        'onion starter',
        'shallot starter',
        'leek starter',
        'scallion starter',
        'chive starter',
        'herb starter',
        'spice starter',
        'fruit starter',
        'vegetable starter',
        'seed starter',
        'nut starter',
        'bean starter',
        'lentil starter',
        'pea starter',
        'soy starter',
        'tofu starter',
        'tempeh starter',
```

```
      'seitan starter',
      'miso starter',
      'soy sauce starter',
      'fish sauce starter',
      'oyster sauce starter',
      'hoisin sauce starter',
      'teriyaki sauce starter',
      'ponzu sauce starter',
      'worcestershire sauce starter',
      'vinegar starter',
      'kombucha starter',
      'water kefir starter',
      'milk kefir starter',
      'yogurt starter',
      'buttermilk starter', */
    ];
```

Listing 3.14: Ingridient_list.dart

Chapter 4

Learning Outcomes and Challenges

4.1 Project Reflection

Developing the Recipe Recommendation System using Flutter was a comprehensive and insightful experience that integrated multiple aspects of mobile app development, data handling, and user-centered design. The goal of the project was to create a smart system that could suggest recipes based on a user's taste preferences (such as spicy, sweet, or fruity) as well as nutritional needs (like calories, carbohydrates, fats, and proteins). This required not only technical implementation using Flutter but also significant work in dataset enhancement, algorithm design, and user experience optimization. Through the process, several challenges were encountered and valuable lessons were learned, both technically and in terms of problem-solving and collaboration.

4.2 Learning Outcomes

During the development of the Recipe Recommendation System using Flutter, several valuable skills and concepts were learned.

1. Flutter App Development

Gained hands-on experience in building cross-platform mobile applications using Flutter and Dart, including state management, navigation, and UI/UX design principles.

2. Recommendation Algorithms

Learned how to implement and fine-tune filtering algorithms based on user preferences and nutritional content using conditional logic and basic ML principles.

3. Data Preprocessing & Classification

Developed the ability to preprocess raw data by extracting meaningful features like taste palette and nutritional values.

4. Integration of External Datasets

Understood the challenges and best practices in working with real-world recipe datasets, including cleaning, extending, and formatting them for efficient use in a Flutter application.

5. User-Centric Design

Focused on building an intuitive and dynamic interface that reflects user preferences in real-time, improving user satisfaction and engagement.

6. Team Collaboration & Problem Solving

Improved communication and coordination within the team, especially while addressing technical bottlenecks like data inconsistency and API limitations.

4.3 Challenges Faced and How They Were Overcome

1. Integrating Taste Palette into the Dataset

Challenge: The original recipe dataset lacked explicit classification of ingredients by taste palette (e.g., spicy, sweet, fruity). This made it difficult to recommend recipes based on user taste preferences.

Solution: We manually classified all ingredients in the dataset according to common taste palettes (e.g., `"pepper"` as spicy, `"mango"` as fruity, `"lemon"` as sour). This was done by creating a mapping dictionary and programmatically tagging each recipe with relevant taste labels based on its ingredients. This enriched dataset enabled personalized recommendations based on a user's taste profile.

2. Handling Inconsistent Nutritional Data

Challenge: Some recipes in the dataset had missing or inconsistent nutritional information (calories, carbs, fats, etc.), which compromised the accuracy of nutrient-based filtering.

Solution: We applied data imputation techniques using average values for similar ingredients or categories to fill in missing values. In some cases, we sourced additional nutritional data from publicly available APIs and merged it with our dataset.

3. Efficient Filtering for Nutrient and Taste-Based Queries

Challenge: Filtering recipes dynamically based on both taste palette and nutritional constraints (e.g., low-carb, high-protein + spicy) introduced performance issues as the dataset grew.

Solution: We optimized data queries using indexed lists and categorized filters. Additionally, we implemented lazy loading and pagination for performance improvement on the UI side.

4. UI/UX Complexity with Dynamic Filtering

Challenge: Designing an interface that allows users to simultaneously filter by taste palette and nutritional needs without overwhelming them was a complex UX challenge.

Solution: We used multi-select dropdowns and interactive sliders (e.g., for calorie range) to make the filtering intuitive. Flutter's flexible widget system helped us keep the layout clean and responsive.

5. Limited Dataset Coverage for Local/Regional Foods

Challenge: The dataset lacked support for regional or cultural recipes which some users might prefer, limiting the system's inclusivity.

Solution: We allowed the system to import user-submitted recipes and tagged them with relevant taste and nutrient metadata. This crowdsourced approach helped expand coverage and personalization.

6. Debugging and State Management in Flutter

Challenge: Managing complex UI states during live filtering led to bugs and unexpected behaviors.

Solution: We adopted the `Provider` package for state management, which helped separate business logic from UI and made debugging and scalability more manageable.

Chapter 5

Conclusion

This Flutter recipe app offers an intuitive way to manage, view, and edit recipes. It enhances user experience with smart features like **AI-based taste palette recommendations** and **nutritional value-based suggestions**, helping users make healthier and personalized food choices. The project lays a strong foundation for a modern, intelligent cooking assistant.

References

[1] T. Team, “Tensorflow installation guide.”

[2] Documentations, “Flutter docs.”

[3] Nvidia, “Cuda toolkit.”

[4] Nvidia, “cudnn library.”

[5] D. with Bappy, “Cuda setup tutorial,” 2023.

[6] Kaggle, “Dataset-foodrecsys v1.”

[7] V. Paradigm, “Visual paradigm online.”

[8] Draw.io, “Draw.io online diagram tool.”

[9] Food.com, “Food.com - recipes and reviews.”