

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	v
List of Tables	vi
Abbreviations	vii

1 Project Overview	1
1.1 Functional Requirements	1
1.2 Team Members	1
1.3 Introduction	1
1.4 Functional Requirements	3
1.5 Scope of your project	3
1.6 Tools and Technologies	5
2 Front-End Design and Implementation	7
2.1 Version Control-Access to Project Files	7
2.2 Flutter Application Prototype	7
2.3 Flutter Application Screen 1 - Login	8
2.4 Flutter Application Screen 2 - Sign Up	22
2.5 Flutter Application Screen 3 - Forgot Password	32
2.6 Flutter Application Screen 4 - Password Reset Verification	39
2.7 Flutter Application Screen 5 - Set New Password	48
2.8 Flutter Application Screen 6 - Home Page	57
2.9 Flutter Application Screen 7 - Recipe Creation	66
2.10 Flutter Application Screen 8 - Recipe Editing	72
2.11 Flutter Application Screen 9 - AI Recipe Generation	81
2.12 Flutter Application Screen 10 - AI Recipe Details	94
2.13 Flutter Application Screen 11 - Nutritional Recipe Recommendations	98
3 Backend Implementation	114
3.1 Python Files	114
3.2 Backend File 1 - Setting.py	114
3.3 Backend File 2 - urls.py	120
3.4 Backend File 3 - Accounts/urls.py	121
3.5 Backend File 4 - Accounts/models.py	122
3.6 Backend File 5 - Accounts/views.py	123
3.7 Backend File 6 - recipes/urls.py	128
3.8 Backend File 7 - Recipes/views.py	130
3.9 Backend File 8 - Recipes/models.py	136
3.10 Backend File 9 - Services/recipe_service.dart	137

3.11 Backend File 10 - auth service.dart	141
3.12 Backend File 11 - Recipe_detail_page.dart	148
3.13 Backend File 12 - App.dart	151
3.14 Backend File 13 - Main.dart	155
3.15 Backend File 14 - Ingridients_list.dart	157
4 Learning Outcomes and Challenges	176
4.1 Project Reflection	176
4.2 Learning Outcomes	176
4.3 Challenges Faced and How They Were Overcome	177
5 Conclusion	179
 References	 180

List of Figures

2.1	Login Screen	21
2.2	Sign Up Screen	31
2.3	Forgot Password Screen	38
2.4	Password Reset Verification Screen	47
2.5	Set New Password Screen	56
2.6	Home Screen	65
2.7	Recipe Creation Screen	71
2.8	Recipe Editing Screen	80
2.9	AI Recipe Generation Screen	93
2.10	AI Recipe Details Screen	97
2.11	Nutritional Recipe Recommendations Screen 1	112
2.12	Nutritional Recipe Recommendations Screen 2	113

List of Tables

1.1	Github Report Link	1
1.2	Functional Requirements	3
1.3	Project Scope	4
1.4	Technologies and Tools Used in the Project	5
2.1	Use Case 1 - Login Screen	8
2.2	Use Case 2 - User Registration	22
2.3	Use Case 3 - Password Recovery	32
2.4	Use Case 4 - Password Reset Verification	39
2.5	Use Case 5 - Password Update	48
2.6	Use Case 6 - Home Page	57
2.7	Use Case 7 - Recipe Creation	66
2.8	Use Case 8 - Recipe Editing	72
2.9	Use Case 9 - AI Recipe Generation	81
2.10	Use Case 10 - AI Recipe Details	94
2.11	Use Case 11 - Nutritional Recipe Recommendations	98
3.1	Setting File	114
3.2	Django URL Configuration – <code>urls.py</code>	120
3.3	Django URL Configuration – <code>accounts/urls.py</code>	121
3.4	Django Model – OTP in <code>accounts/models.py</code>	122
3.5	Django Views – <code>accounts/views.py</code>	123
3.6	Django URL Configuration – <code>recipes/urls.py</code>	128
3.7	Django Views – <code>recipes/views.py</code>	130
3.8	Django Models – <code>recipes/models.py</code>	136
3.9	Dart Service – <code>RecipeService</code>	137
3.10	Dart Service – <code>AuthService</code>	141
3.11	Flutter Widget – <code>RecipeDetailPage</code>	148
3.12	<code>App.dart</code>	151
3.13	Flutter Main Application – <code>main.dart</code>	155

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.
<i>Continued on next page</i>		

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	<ol style="list-style-type: none">1. Email Field:<ul style="list-style-type: none">• Text input for user's email address2. Password Field:<ul style="list-style-type: none">• Secure text input for user's password3. Continue Button:<ul style="list-style-type: none">• Submits the login credentials4. Forgot Password Link:<ul style="list-style-type: none">• Navigates to password recovery5. Sign Up Link:<ul style="list-style-type: none">• Navigates to registration screen
Output Elements	<ol style="list-style-type: none">1. Success Notification:<ul style="list-style-type: none">• User is authenticated and directed to main screen2. Error Messages:<ul style="list-style-type: none">• Invalid email format• Incorrect password• Account not found

Navigation Flow	<ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> • If "Forgot Password" is clicked, navigate to password recovery. • If "Sign Up" is clicked, navigate to registration. 5. Error Handling: <ul style="list-style-type: none"> • Invalid inputs display appropriate error messages.
------------------------	--

Code Implementation

```

1  import 'package:flutter/material.dart';
2  import 'package:flutter/gestures.dart';
3  import 'package:flutter_django_recipes_frontend/forgot_password.
    dart';
4  import 'package:flutter_django_recipes_frontend/app.dart';
5  import 'package:flutter_django_recipes_frontend/services/
    auth_service.dart';
6
7  class LoginPage extends StatefulWidget {
8    const LoginPage({super.key});
9
10   @override
11   LoginPageState createState() => LoginPageState();
12 }
13
14 class LoginPageState extends State<LoginPage> {
15   bool _isLoginView = true;
16   bool _isLoading = false;
17   bool _isPasswordVisible = false;
18
19   final TextEditingController _emailController =
     TextEditingController();
20   final TextEditingController _passwordController =
     TextEditingController();

```

```
21 final TextEditingController _firstNameController =
    TextEditingController();
22 final TextEditingController _lastNameController =
    TextEditingController();
23
24 @override
25 void dispose() {
26     _emailController.dispose();
27     _passwordController.dispose();
28     _firstNameController.dispose();
29     _lastNameController.dispose();
30     super.dispose();
31 }
32
33 bool _validateEmail() {
34     final email = _emailController.text.trim();
35     return RegExp(
36         r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
37     ).hasMatch(email);
38 }
39
40 Future<void> _handleLogin(BuildContext context) async {
41     if (!_validateEmail()) {
42         _showErrorDialog(context, 'Please enter a valid email');
43         return;
44     }
45
46     setState(() => _isLoading = true);
47
48     try {
49         final result = await AuthService.login(
50             email: _emailController.text,
51             password: _passwordController.text,
52         );
53
54         if (result['success'] == true) {
55             Navigator.pushReplacement(
56                 context,
57                 MaterialPageRoute(builder: (context) => const MainPage())
58             );
59         }
```

```
59     } else {
60         _showErrorDialog(context, result['message'] ?? 'Login
failed');
61     }
62     } catch (e) {
63         _showErrorDialog(context, 'An error occurred during login');
64     } finally {
65         setState(() => _isLoading = false);
66     }
67 }
68
69 Future<void> _handleRegister(BuildContext context) async {
70     if (!_validateEmail()) {
71         _showErrorDialog(context, 'Please enter a valid email');
72         return;
73     }
74
75     if (_firstNameController.text.isEmpty || _lastNameController.
text.isEmpty) {
76         _showErrorDialog(context, 'First and last name are required')
;
77         return;
78     }
79
80     if (_passwordController.text.length < 8) {
81         _showErrorDialog(context, 'Password must be at least 8
characters');
82         return;
83     }
84
85     setState(() => _isLoading = true);
86
87     try {
88         final result = await AuthService.register(
89             email: _emailController.text,
90             password: _passwordController.text,
91             firstName: _firstNameController.text,
92             lastName: _lastNameController.text,
93         );
94
95         if (result['success'] == true) {
```

```
96     Navigator.pushReplacement(  
97         context,  
98         MaterialPageRoute(builder: (context) => const MainPage()  
99     ),  
100     );  
101     } else {  
102         _showErrorDialog(context, result['message'] ?? '  
Registration failed');  
103     }  
104     } catch (e) {  
105         _showErrorDialog(context, 'An error occurred during  
registration');  
106     } finally {  
107         setState(() => _isLoading = false);  
108     }  
109 }  
110 void _showErrorDialog(BuildContext context, String message) {  
111     showDialog(  
112         context: context,  
113         builder: (context) => AlertDialog(  
114             title: const Text(  
115                 "Error",  
116                 style: TextStyle(  
117                     color: Colors.red,  
118                     fontWeight: FontWeight.bold,  
119                 ),  
120             ),  
121             content: Text(message),  
122             actions: [  
123                 TextButton(  
124                     onPressed: () => Navigator.of(context).pop(),  
125                     child: const Text("OK"),  
126                 ),  
127             ],  
128         ),  
129     );  
130 }  
131  
132 void _switchView() {  
133     setState(() {
```

```
134     _isLoginView = !_isLoginView;
135     _emailController.clear();
136     _passwordController.clear();
137     if (!_isLoginView) {
138         _firstNameController.clear();
139         _lastNameController.clear();
140     }
141     });
142 }
143
144 Widget _buildLoginForm(BuildContext context) {
145     return Padding(
146         padding: const EdgeInsets.symmetric(horizontal: 20.0,
147         vertical: 40.0),
148         child: Column(
149             mainAxisAlignment: MainAxisAlignment.center,
150             crossAxisAlignment: CrossAxisAlignment.start,
151             children: [
152                 const Text(
153                     "Email",
154                     style: TextStyle(fontWeight: FontWeight.bold, fontSize:
155                     16.0),
156                 ),
157                 const SizedBox(height: 8),
158                 TextField(
159                     controller: _emailController,
160                     keyboardType: TextInputType.emailAddress,
161                     decoration: _inputDecoration(),
162                 ),
163                 const SizedBox(height: 16),
164                 const Text(
165                     "Password",
166                     style: TextStyle(fontWeight: FontWeight.bold, fontSize:
167                     16.0),
168                 ),
169                 const SizedBox(height: 8),
170                 TextField(
171                     controller: _passwordController,
```

```
172         icon: Icon(  
173             _isPasswordVisible ? Icons.visibility : Icons.  
visibility_off,  
174         ),  
175         onPressed: () => setState(() => _isPasswordVisible  
= !_isPasswordVisible),  
176     ),  
177 ),  
178 ),  
179     const SizedBox(height: 8),  
180     Align(  
181         alignment: Alignment.centerRight,  
182         child: TextButton(  
183             onPressed: () => Navigator.push(  
184                 context,  
185                 MaterialPageRoute(builder: (context) => const  
ForgotPassword()),  
186             ),  
187             child: Text(  
188                 "Forgot Password?",  
189                 style: TextStyle(  
190                     fontWeight: FontWeight.bold,  
191                     color: Colors.blue.shade600,  
192                 ),  
193             ),  
194         ),  
195     ),  
196     const SizedBox(height: 24),  
197     SizedBox(  
198         width: double.infinity,  
199         height: 50,  
200         child: ElevatedButton(  
201             onPressed: _isLoading ? null : () => _handleLogin(  
context),  
202             style: ElevatedButton.styleFrom(  
203                 backgroundColor: Colors.blue,  
204                 shape: RoundedRectangleBorder(  
205                     borderRadius: BorderRadius.circular(12.0),  
206                 ),  
207             ),  
208             child: _isLoading
```

```

209         ? const CircularProgressIndicator(color: Colors.
white)
210         : const Text(
211             "Log In",
212             style: TextStyle(
213                 color: Colors.white,
214                 fontWeight: FontWeight.bold,
215                 fontSize: 16.0,
216             ),
217         ),
218     ),
219 ),
220 const SizedBox(height: 24),
221 Center(
222     child: RichText(
223         text: TextSpan(
224             text: "Don't have an account? ",
225             style: TextStyle(
226                 color: Colors.grey.shade600,
227                 fontWeight: FontWeight.bold,
228             ),
229             children: [
230                 TextSpan(
231                     text: "Sign Up",
232                     style: TextStyle(
233                         color: Colors.blue.shade600,
234                         fontWeight: FontWeight.bold,
235                     ),
236                     recognizer: TapGestureRecognizer()..onTap =
_switchView,
237                 ),
238             ],
239         ),
240     ),
241 ),
242 ],
243 ),
244 );
245 }
246
247 Widget _buildRegisterForm(BuildContext context) {

```

```
248     return Padding(  
249         padding: const EdgeInsets.symmetric(horizontal: 20.0,  
vertical: 20.0),  
250         child: Column(  
251             mainAxisAlignment: MainAxisAlignment.center,  
252             crossAxisAlignment: CrossAxisAlignment.start,  
253             children: [  
254                 const Text(  
255                     "First Name",  
256                     style: TextStyle(fontWeight: FontWeight.bold, fontSize:  
16.0),  
257                 ),  
258                 const SizedBox(height: 8),  
259                 TextField(  
260                     controller: _firstNameController,  
261                     decoration: _inputDecoration(),  
262                 ),  
263                 const SizedBox(height: 16),  
264                 const Text(  
265                     "Last Name",  
266                     style: TextStyle(fontWeight: FontWeight.bold, fontSize:  
16.0),  
267                 ),  
268                 const SizedBox(height: 8),  
269                 TextField(  
270                     controller: _lastNameController,  
271                     decoration: _inputDecoration(),  
272                 ),  
273                 const SizedBox(height: 16),  
274                 const Text(  
275                     "Email",  
276                     style: TextStyle(fontWeight: FontWeight.bold, fontSize:  
16.0),  
277                 ),  
278                 const SizedBox(height: 8),  
279                 TextField(  
280                     controller: _emailController,  
281                     keyboardType: TextInputType.emailAddress,  
282                     decoration: _inputDecoration(),  
283                 ),  
284                 const SizedBox(height: 16),
```



```

285         const Text(
286             "Password",
287             style: TextStyle(fontWeight: FontWeight.bold, fontSize:
16.0),
288         ),
289         const SizedBox(height: 8),
290         TextField(
291             controller: _passwordController,
292             obscureText: !_isPasswordVisible,
293             decoration: _inputDecoration(
294                 suffixIcon: IconButton(
295                     icon: Icon(
296                         _isPasswordVisible ? Icons.visibility : Icons.
visibility_off,
297                     ),
298                     onPressed: () => setState(() => _isPasswordVisible
= !_isPasswordVisible),
299                 ),
300             ),
301         ),
302         const SizedBox(height: 24),
303         SizedBox(
304             width: double.infinity,
305             height: 50,
306             child: ElevatedButton(
307                 onPressed: _isLoading ? null : () => _handleRegister(
context),
308                 style: ElevatedButton.styleFrom(
309                     backgroundColor: Colors.blue,
310                     shape: RoundedRectangleBorder(
311                         borderRadius: BorderRadius.circular(12.0),
312                     ),
313                 ),
314                 child: _isLoading
315                     ? const CircularProgressIndicator(color: Colors.
white)
316                     : const Text(
317                         "Register",
318                         style: TextStyle(
319                             color: Colors.white,
320                             fontWeight: FontWeight.bold,

```

```

321         fontSize: 16.0,
322     ),
323 ),
324 ),
325 ),
326 const SizedBox(height: 24),
327 Center(
328   child: RichText(
329     text: TextSpan(
330       text: "Already have an account? ",
331       style: TextStyle(
332         color: Colors.grey.shade600,
333         fontWeight: FontWeight.bold,
334       ),
335       children: [
336         TextSpan(
337           text: "Log In",
338           style: TextStyle(
339             color: Colors.blue.shade600,
340             fontWeight: FontWeight.bold,
341           ),
342           recognizer: TapGestureRecognizer()..onTap =
343             _switchView,
344         ),
345       ],
346     ),
347   ),
348 ],
349 ),
350 );
351 }
352
353 InputDecoration _inputDecoration({Widget? suffixIcon}) {
354   return InputDecoration(
355     border: OutlineInputBorder(
356       borderRadius: BorderRadius.circular(12.0),
357       borderSide: const BorderSide(color: Colors.black54, width:
358         1.0),
359     ),
360     focusedBorder: OutlineInputBorder(

```

```

360         borderRadius: BorderRadius.circular(12.0),
361         borderSide: const BorderSide(color: Colors.blue, width:
2.0),
362     ),
363     enabledBorder: OutlineInputBorder(
364         borderRadius: BorderRadius.circular(12.0),
365         borderSide: BorderSide(
366             color: Colors.grey.shade400,
367             width: 1.0,
368         ),
369     ),
370     contentPadding: const EdgeInsets.symmetric(horizontal: 16,
vertical: 14),
371     suffixIcon: suffixIcon,
372 );
373 }
374
375 @override
376 Widget build(BuildContext context) {
377     return Scaffold(
378         appBar: AppBar(
379             title: Row(
380                 mainAxisAlignment: MainAxisAlignment.spaceEvenly,
381                 children: [
382                     GestureDetector(
383                         onTap: _isLoginView ? null : () => setState(() =>
_isLoginView = true),
384                         child: Container(
385                             padding: const EdgeInsets.symmetric(vertical: 8.0,
horizontal: 24.0),
386                             decoration: _isLoginView
387                                 ? const BoxDecoration(
388                                     border: Border(
389                                         bottom: BorderSide(width: 2.0, color:
Colors.blue),
390                                     ),
391                                 )
392                                 : null,
393                             child: Text(
394                                 "Log In",
395                                 style: TextStyle(

```

```

396         color: _isLoginView ? Colors.blue : Colors.
black54,
397         fontWeight: FontWeight.bold,
398     ),
399 ),
400 ),
401 ),
402 GestureDetector(
403     onTap: _isLoginView ? () => setState(() =>
_isLoginView = false) : null,
404     child: Container(
405         padding: const EdgeInsets.symmetric(vertical: 8.0,
horizontal: 24.0),
406         decoration: !_isLoginView
407             ? const BoxDecoration(
408                 border: Border(
409                     bottom: BorderSide(width: 2.0, color:
Colors.blue),
410                 ),
411             )
412             : null,
413         child: Text(
414             "Sign Up",
415             style: TextStyle(
416                 color: !_isLoginView ? Colors.blue : Colors.
black54,
417                 fontWeight: FontWeight.bold,
418             ),
419         ),
420     ),
421 ),
422 ],
423 ),
424 ),
425 body: SingleChildScrollView(
426     child: _isLoginView ? _buildLoginForm(context) :
_buildRegisterForm(context),),),)}}

```

LISTING 2.1: Login Code

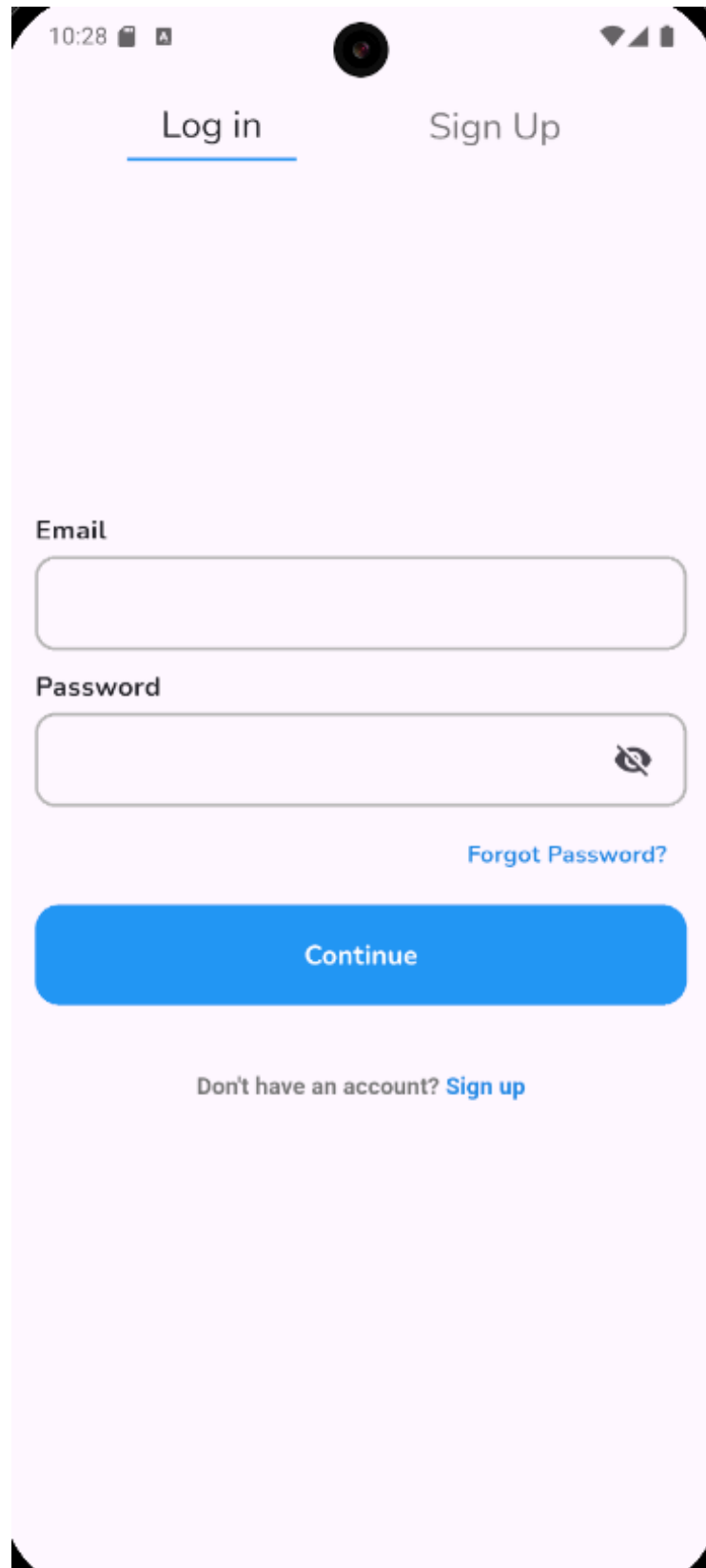


FIGURE 2.1: Login Screen

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	<ol style="list-style-type: none"> 1. First Name Field: <ul style="list-style-type: none"> • Text input for user's first name 2. Last Name Field: <ul style="list-style-type: none"> • Text input for user's last name 3. Email Field: <ul style="list-style-type: none"> • Text input for user's email address 4. Password Field: <ul style="list-style-type: none"> • Secure text input for account password 5. Continue Button: <ul style="list-style-type: none"> • Submits the registration form 6. Log In Link: <ul style="list-style-type: none"> • Navigates to login screen for existing users
Output Elements	<ol style="list-style-type: none"> 1. Success Notification: <ul style="list-style-type: none"> • Account created confirmation • Automatic login and redirection to main screen 2. Error Messages: <ul style="list-style-type: none"> • Missing required fields • Invalid email format • Password requirements not met • Email already registered

Navigation Flow	<ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> • If "Log In" is clicked, navigate to login screen. 5. Error Handling: <ul style="list-style-type: none"> • Invalid inputs display appropriate error messages. 6. Success Flow: <ul style="list-style-type: none"> • After successful registration, user is authenticated and directed to application's main screen.
------------------------	--

Code Implementation

```

1 import "package:flutter/material.dart";
2 import 'package:google_fonts/google_fonts.dart';
3 import 'package:flutter_django_recipes_frontend/app.dart';
4 import 'package:flutter_django_recipes_frontend/services/
  auth_service.dart';
5
6 class SignUpPage extends StatefulWidget {
7   const SignUpPage({super.key});
8
9   @override
10  SignUpPageState createState() => SignUpPageState();
11 }
12
13 class SignUpPageState extends State<SignUpPage> {
14   bool loggingInRn = false; // Loading indicator during
    registration
15   bool _isPassword = true; // Toggle password visibility
16   TextEditingController emailController = TextEditingController();
17   TextEditingController passwordController = TextEditingController
    ();
18   TextEditingController firstNameController = TextEditingController
    ();
19   TextEditingController lastNameController = TextEditingController
    ();

```

```
20
21 // Validates email format
22 bool emailValidation() {
23     String email = emailController.text.trim();
24     if (email.length < 5 || email.length > 50) {
25         return false;
26     }
27     return RegExp(
28         r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
29     ).hasMatch(email);
30 }
31
32 // Handles registration submission
33 Future<void> submitRegisterInformation(BuildContext context)
34     async {
35     if (!emailValidation()) {
36         showErrorDialog(context, 'Invalid email format');
37         return;
38     }
39     if (firstNameController.text.isEmpty || lastNameController.text
40     .isEmpty) {
41         showErrorDialog(context, 'First name and last name cannot be
42     empty');
43         return;
44     }
45     if (passwordController.text.length < 8) {
46         showErrorDialog(context, 'Password must be at least 8
47     characters long');
48         return;
49     }
50
51     final result = await AuthService.register(
52         email: emailController.text,
53         password: passwordController.text,
54         firstName: firstNameController.text,
55         lastName: lastNameController.text,
56     );
57
58     if (result['success']) {
59         Navigator.pushReplacement(
60             context,
```



```
57         MaterialPageRoute(builder: (context) => MainPage()),
58     );
59     } else {
60         showErrorDialog(context, result['message']);
61     }
62 }
63
64 // Shows error dialog
65 void showErrorDialog(BuildContext context, String message) {
66     showDialog(
67         context: context,
68         builder: (context) {
69             return AlertDialog(
70                 title: Text(
71                     "Error",
72                     style: TextStyle(
73                         color: Colors.red,
74                         fontWeight: FontWeight.bold,
75                         fontSize: 20,
76                     ),
77                 ),
78                 content: Text(message),
79                 actions: [
80                     TextButton(
81                         onPressed: () => Navigator.of(context).pop(),
82                         child: Text("OK"),
83                     ),
84                 ],
85             );
86         },
87     );
88 }
89
90 @override
91 void dispose() {
92     emailController.dispose();
93     passwordController.dispose();
94     firstNameController.dispose();
95     lastNameController.dispose();
96     super.dispose();
97 }
```

```
98
99  @override
100  Widget build(BuildContext context) {
101    return Scaffold(
102      appBar: AppBar(
103        title: Text("Sign Up", style: GoogleFonts.nunito()),
104      ),
105      body: Padding(
106        padding: EdgeInsets.symmetric(horizontal: 20.0, vertical:
107        20),
108        child: Column(
109          mainAxisAlignment: MainAxisAlignment.center,
110          crossAxisAlignment: CrossAxisAlignment.start,
111          children: [
112            // First Name Input
113            Text(
114              "First Name",
115              style: GoogleFonts.nunito(
116                fontWeight: FontWeight.bold,
117                fontSize: 16.0,
118              ),
119            ),
120            Padding(padding: EdgeInsets.all(2.0)),
121            TextField(
122              controller: firstNameController,
123              decoration: InputDecoration(
124                border: OutlineInputBorder(
125                  borderRadius: BorderRadius.circular(12.0),
126                focusedBorder: OutlineInputBorder(
127                  borderRadius: BorderRadius.circular(12.0),
128                enabledBorder: OutlineInputBorder(
129                  borderRadius: BorderRadius.circular(12.0),
130                  borderSide: BorderSide(
131                    width: 2.0,
132                    color: const Color.fromARGB(255, 184, 183, 183)
133                  ),
134                ),
135              ),
136            ),
137            Padding(padding: EdgeInsets.all(5.0)),
```

```
137
138         // Last Name Input
139         Text(
140             "Last Name",
141             style: GoogleFonts.nunito(
142                 fontWeight: FontWeight.bold,
143                 fontSize: 16.0,
144             ),
145         ),
146         Padding(padding: EdgeInsets.all(2.0)),
147         TextField(
148             controller: lastNameController,
149             decoration: InputDecoration(
150                 border: OutlineInputBorder(
151                     borderRadius: BorderRadius.circular(12.0),
152                 ),
153                 focusedBorder: OutlineInputBorder(
154                     borderRadius: BorderRadius.circular(12.0),
155                 ),
156                 enabledBorder: OutlineInputBorder(
157                     borderRadius: BorderRadius.circular(12.0),
158                     borderSide: BorderSide(
159                         width: 2.0,
160                         color: const Color.fromARGB(255, 184, 183, 183)
161                     ),
162                 ),
163             ),
164         ),
165         Padding(padding: EdgeInsets.all(5.0)),
166
167         // Email Input
168         Text(
169             "Email",
170             style: GoogleFonts.nunito(
171                 fontWeight: FontWeight.bold,
172                 fontSize: 16.0,
173             ),
174         ),
175         Padding(padding: EdgeInsets.all(2.0)),
176         TextField(
```

```

177         controller: emailController,
178         decoration: InputDecoration(
179             border: OutlineInputBorder(
180                 borderRadius: BorderRadius.circular(12.0),
181             ),
182             focusedBorder: OutlineInputBorder(
183                 borderRadius: BorderRadius.circular(12.0),
184             ),
185             enabledBorder: OutlineInputBorder(
186                 borderRadius: BorderRadius.circular(12.0),
187                 borderSide: BorderSide(
188                     width: 2.0,
189                     color: const Color.fromARGB(255, 184, 183, 183)
190                 ),
191             ),
192         ),
193     ),
194     Padding(padding: EdgeInsets.all(5.0)),
195
196     // Password Input
197     Text(
198         "Password",
199         style: GoogleFonts.nunito(
200             fontWeight: FontWeight.bold,
201             fontSize: 16.0,
202         ),
203     ),
204     Padding(padding: EdgeInsets.all(2.0)),
205     TextField(
206         controller: passwordController,
207         obscureText: _isPassword,
208         decoration: InputDecoration(
209             suffixIcon: Padding(
210                 padding: const EdgeInsets.symmetric(horizontal:
211                     8.0),
212                 child: IconButton(
213                     icon: Icon(
214                         _isPassword ? Icons.visibility_off : Icons.
215                         visibility,
216                     ),

```

```

215         onPressed: () => setState(() => _isPassword = !
_isPassword),
216     ),
217 ),
218     border: OutlineInputBorder(
219         borderRadius: BorderRadius.circular(12.0),
220     ),
221     enabledBorder: OutlineInputBorder(
222         borderRadius: BorderRadius.circular(12.0),
223         borderSide: BorderSide(
224             width: 2.0,
225             color: const Color.fromARGB(255, 184, 183, 183)
226         ),
227     ),
228     focusedBorder: OutlineInputBorder(
229         borderRadius: BorderRadius.circular(12.0),
230         borderSide: BorderSide(width: 2.0, color: Colors.
black54),
231     ),
232 ),
233     Padding(padding: EdgeInsets.symmetric(vertical: 10.0)),
234
235     // Register Button
236     SizedBox(height: 10),
237     OutlinedButton(
238         style: OutlinedButton.styleFrom(
239             backgroundColor: Colors.blue,
240             minimumSize: Size(double.infinity, 60.0),
241             side: BorderSide(color: Colors.blue),
242             shape: RoundedRectangleBorder(
243                 borderRadius: BorderRadius.circular(15.0),
244             ),
245         ),
246         onPressed: () async {
247             setState(() => loggingInRn = true);
248             await submitRegisterInformation(context);
249             setState(() => loggingInRn = false);
250         },
251         child: loggingInRn
252             ? CircularProgressIndicator()

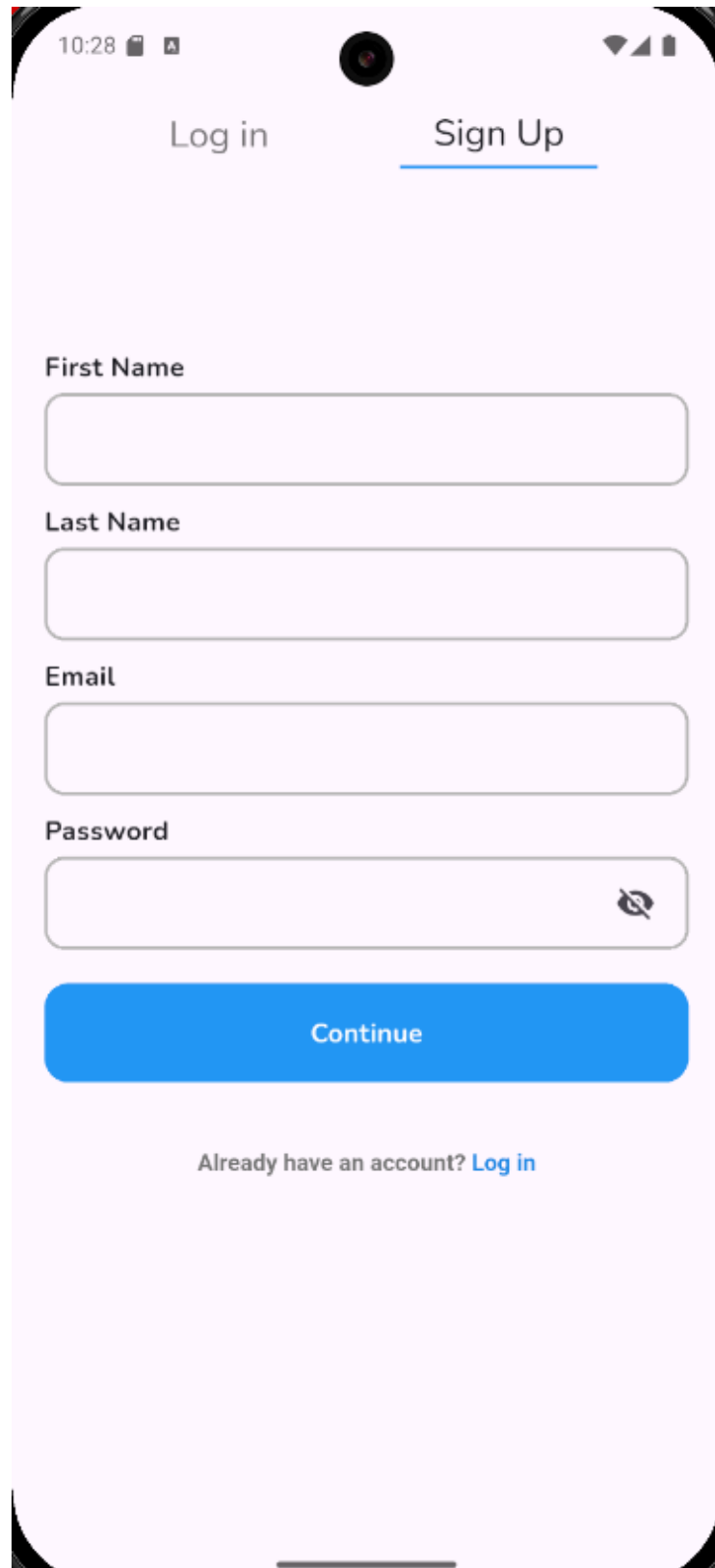
```

```

253         : Text(
254             "Continue",
255             style: GoogleFonts.nunito(
256                 color: Colors.white,
257                 fontWeight: FontWeight.bold,
258                 fontSize: 16.0,
259             ),
260         ),
261     ),
262
263     // Login Redirect
264     Padding(padding: EdgeInsets.all(20.0)),
265     Center(
266         child: RichText(
267             text: TextSpan(
268                 text: "Already have an account? ",
269                 style: TextStyle(
270                     color: Colors.grey.shade600,
271                     fontWeight: FontWeight.bold,
272                 ),
273                 children: [
274                     TextSpan(
275                         text: "Log in",
276                         style: TextStyle(
277                             color: Colors.blue.shade600,
278                             fontWeight: FontWeight.bold,
279                         ),
280                         recognizer: TapGestureRecognizer()
281                           ..onTap = () {
282                             Navigator.pushReplacement(
283                                 context,
284                                 MaterialPageRoute(builder: (context) =>
285                                     SignInPage()),
286                             );
287                         },
288                     ],
289                 ),
290             ),),),),),),);}}

```

LISTING 2.2: Sign Up Code



The image shows a mobile application screen for signing up. At the top, there is a status bar with the time 10:28 and various icons. Below the status bar, there are two tabs: "Log in" and "Sign Up". The "Sign Up" tab is selected and underlined. The main form consists of four input fields: "First Name", "Last Name", "Email", and "Password". Each field is a rounded rectangle with a light gray border. The "Password" field has a small eye icon on the right side, indicating a toggle for visibility. Below the input fields is a large blue button with the text "Continue". At the bottom of the screen, there is a link that says "Already have an account? Log in".

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: <ul style="list-style-type: none"> • Text input for user's registered email address 2. Reset Password Button: <ul style="list-style-type: none"> • Submits the password reset request 3. Back Button (Clicked): <ul style="list-style-type: none"> • Returns to login screen
Output Elements	1. Success Notification: <ul style="list-style-type: none"> • Confirmation email sent message 2. Error Messages: <ul style="list-style-type: none"> • Email not registered • Invalid email format • Network connection issues

Navigation Flow	<ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> • If email isn't registered, show error message • If invalid email format, show validation error 6. Success Flow: <ul style="list-style-type: none"> • User receives email with password reset instructions • After reset, user can return to login screen
------------------------	--

Code Implementation

```

1  import "package:flutter/material.dart";
2
3  import 'package:flutter_django_recipes_frontend/otp.dart';
4  import 'package:flutter_django_recipes_frontend/services/
   auth_service.dart';
5
6  class ForgotPassword extends StatefulWidget {
7    const ForgotPassword({super.key});
8
9    @override
10   ForgotPasswordState createState() => ForgotPasswordState();
11 }
12
13 class ForgotPasswordState extends State<ForgotPassword> {
14   @override
15   initState() {
16     super.initState();
17   }
18
19   final TextEditingController _emailController =
20     TextEditingController();
21
22   Future<void> _sendResetPasswordEmail(BuildContext context) async{
23     if(_emailController.text.isEmpty){
24       showErrorDialog(context, "Please enter your email");
25     }
26   }
27 }

```

```
24     }
25     final String email = _emailController.text.trim();
26     final bool? result = await AuthService.sendResetPasswordEmail(
27     email);
28     if (result == null){
29         showErrorDialog(context, "Failed to send reset password email
30 ");
31     } else if (result) {
32         Navigator.push(context, MaterialPageRoute(builder: (context)
33         => EmailVerificationScreen(email: email)));
34     } else {
35         showErrorDialog(context, "User does not exist");
36     }
37 }
38
39 void showErrorDialog(BuildContext context, String message) {
40     setState(() {
41         showDialog(
42             context: context,
43             builder: (context) {
44                 return AlertDialog(
45                     title: Text(
46                         "Error",
47                         style: TextStyle(
48                             color: Colors.red,
49                             fontWeight: FontWeight.bold,
50                             fontSize: 20,
51                         ),
52                     ),
53                     content: Text(message),
54                     actions: [
55                         TextButton(
56                             onPressed: () {
57                                 Navigator.of(context).pop();
58                             },
59                             child: Text("OK"),
60                         ),
61                     ],
62                 );
63             },
64         );
65     }
```

```
62     });
63   }
64
65   @override
66   void dispose() {
67     _emailController.dispose();
68     super.dispose();
69   }
70   @override
71   Widget build(BuildContext context) {
72     return Scaffold(
73       appBar: AppBar(),
74       body: ListView(
75         physics: NeverScrollableScrollPhysics(),
76         children: [
77           Padding(
78             padding: EdgeInsets.all(20.0),
79             child: Column(
80               mainAxisAlignment: MainAxisAlignment.start,
81               crossAxisAlignment: CrossAxisAlignment.start,
82               children: [
83                 Text(
84                   "Forgot Password",
85                   style: TextStyle(
86                     fontWeight: FontWeight.bold,
87                     fontSize: 24.0,
88                   ),
89               ),
90               Padding(padding: EdgeInsets.all(3.0)),
91               Text(
92                 "Please enter your email to reset the password",
93                 style: TextStyle(
94                   fontSize: 16.0,
95                 ),
96             ),
97             Padding(padding: EdgeInsets.all(8.0)),
98             Text(
99               "Email",
100               style: TextStyle(
101                 fontWeight: FontWeight.bold,
102                 fontSize: 16.0,
```

```

103         ),
104     ),
105     Padding(padding: EdgeInsets.all(2.0)),
106     TextField(
107         controller: _emailController,
108         decoration: InputDecoration(
109             border: OutlineInputBorder(
110                 borderRadius: BorderRadius.circular(12.0),
111                 borderSide: BorderSide(color: Colors.black54,
width: 2.0),
112             ),
113             focusedBorder: OutlineInputBorder(
114                 borderRadius: BorderRadius.circular(12.0),
115                 borderSide: BorderSide(color: Colors.black54,
width: 2.0),
116             ),
117             enabledBorder: OutlineInputBorder(
118                 borderRadius: BorderRadius.circular(12.0),
119                 borderSide: BorderSide(
120                     width: 2.0,
121                     color: const Color.fromARGB(255, 184, 183,
183),
122                 ),
123             ),
124         ),
125     ),
126     Padding(padding: EdgeInsets.all(10.0)),
127     OutlinedButton(
128         style: OutlinedButton.styleFrom(
129             backgroundColor: Colors.blue,
130             minimumSize: Size(double.infinity, 60.0),
131             side: BorderSide(color: Colors.blue),
132             shape: RoundedRectangleBorder(
133                 borderRadius: BorderRadius.circular(15.0),
134             ),
135         ),
136         onPressed: () async => await
_sendResetPasswordEmail(context),
137         child: Text(
138             "Reset Password",
139             style: TextStyle(

```

```
140         color: Colors.white,  
141         fontWeight: FontWeight.bold,  
142         fontSize: 16.0,  
143     ),  
144     ),  
145     ),  
146     ],  
147     ),  
148     ),  
149     ],  
150     ),  
151     );  
152 }  
153 }
```

LISTING 2.3: Forgot Password Code

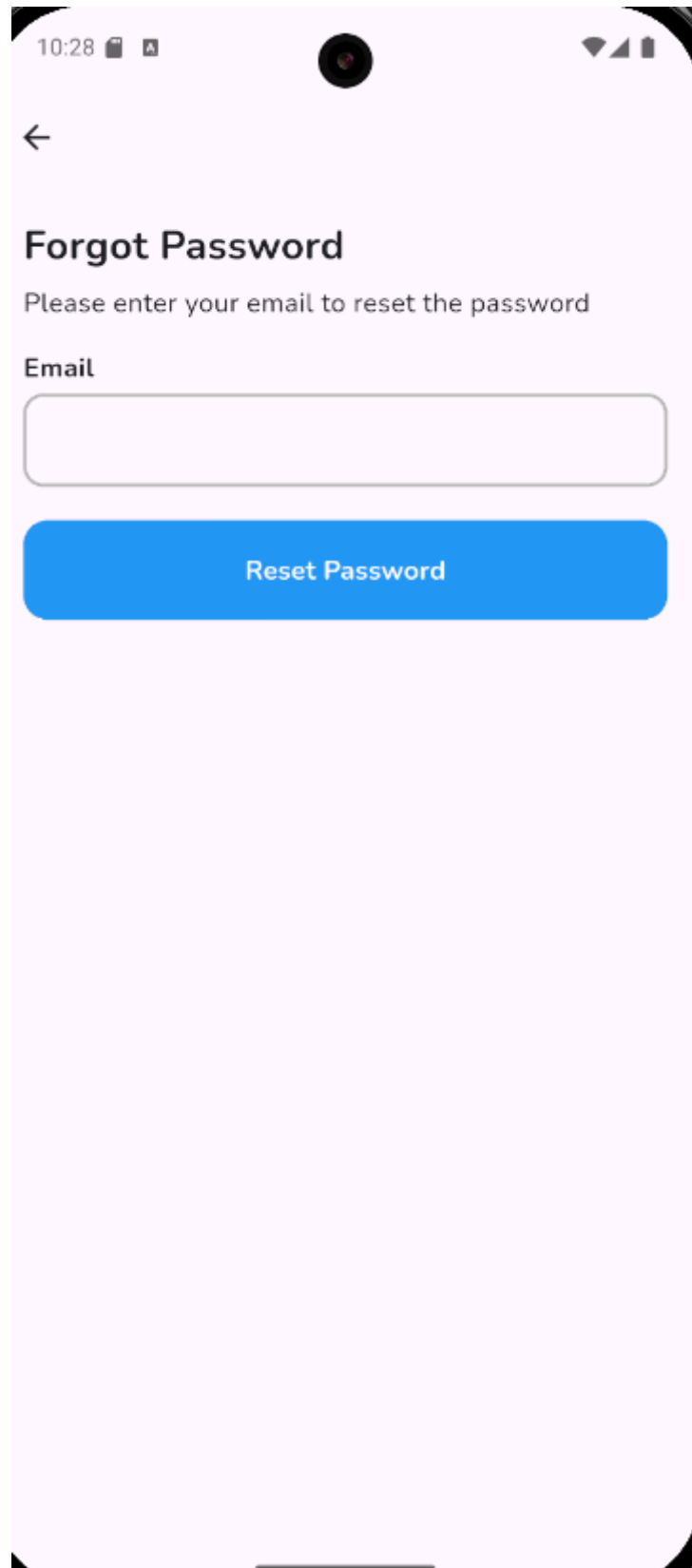


FIGURE 2.3: Forgot Password Screen

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	<ol style="list-style-type: none"> 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	<ol style="list-style-type: none"> 1. Verification Code Field: <ul style="list-style-type: none"> • Input for 5-digit verification code from email 2. Verify Code Button: <ul style="list-style-type: none"> • Submits the code for validation 3. Resend Email Link: <ul style="list-style-type: none"> • Re-sends verification code if not received 4. Back Button (Clicked): <ul style="list-style-type: none"> • Returns to previous screen
Output Elements	<ol style="list-style-type: none"> 1. Success Notification: <ul style="list-style-type: none"> • Code verified, proceed to password reset screen 2. Error Messages: <ul style="list-style-type: none"> • Invalid code • Expired code • Too many attempts 3. Resend Confirmation: <ul style="list-style-type: none"> • New code sent notification

Navigation Flow	<ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> • If code is invalid/expired, show error message • If "Resend email" clicked, send new code 6. Success Flow: <ul style="list-style-type: none"> • After successful verification, navigate to password reset screen 7. Failure Flow: <ul style="list-style-type: none"> • After multiple failures, may lock account temporarily
------------------------	---

Code Implementation

```

1 import 'package:flutter/material.dart';
2 import 'package:flutter_django_recipes_frontend/set_new_password.
  dart';
3 import 'package:flutter_django_recipes_frontend/services/
  auth_service.dart';
4 import 'package:http/http.dart' as http;
5 import 'dart:convert';
6
7 class EmailVerificationScreen extends StatefulWidget {
8   const EmailVerificationScreen({super.key, required this.email});
9   final String email;
10
11   @override
12   _EmailVerificationScreenState createState() =>
13     _EmailVerificationScreenState();
14 }
15 class _EmailVerificationScreenState extends State<
16   EmailVerificationScreen> {
17   final List<TextEditingController> _controllers = List.generate(
18     5,
19     (_) => TextEditingController(),

```



```
19     );
20     bool _isEnabled = false;
21     bool _isLoading = false;
22     String? _errorMessage;
23
24     @override
25     void initState() {
26         super.initState();
27         for (var controller in _controllers) {
28             controller.addListener(_checkIfAllFieldsFilled);
29         }
30     }
31
32     void _checkIfAllFieldsFilled() {
33         setState(() {
34             _isEnabled = _controllers.every(
35                 (controller) => controller.text.isNotEmpty,
36             );
37         });
38     }
39
40     Future<void> _verifyOTP() async {
41         setState(() {
42             _isLoading = true;
43             _errorMessage = null;
44         });
45
46         try {
47             final otp = _controllers.map((c) => c.text).join();
48             final csrfData = await AuthService.getCsrfToken();
49
50             if (csrfData == null) {
51                 setState(() {
52                     _errorMessage = 'Failed to get CSRF token';
53                     _isLoading = false;
54                 });
55                 return;
56             }
57
58             final client = http.Client();
59             final response = await client.post(
```

```
60     Uri.parse('${AuthService.baseUrl}/verifyOTP/'),
61     headers: {
62       'Content-Type': 'application/json',
63       'X-CSRFToken': csrfData['csrfToken'],
64       'Cookie': csrfData['csrfCookie'],
65     },
66     body: json.encode({
67       'email': widget.email,
68       'otp': otp,
69     }),
70   );
71
72   final responseData = json.decode(response.body);
73   client.close();
74
75   if (response.statusCode == 200) {
76     Navigator.push(
77       context,
78       MaterialPageRoute(
79         builder: (context) => SetNewPassword(email: widget.
80 email),
81       ),
82     );
83   } else {
84     setState(() {
85       _errorMessage = responseData['message'] ?? 'Verification
86 failed';
87     });
88   }
89   catch (e) {
90     setState(() {
91       _errorMessage = 'An error occurred. Please try again.';
92     });
93   }
94   finally {
95     setState(() {
96       _isLoading = false;
97     });
98   }
99
100 Future<void> _resendOTP() async {
```

```
99     setState(() {
100         _isLoading = true;
101         _errorMessage = null;
102     });
103
104     try {
105         final success = await AuthService.sendResetPasswordEmail(
106             widget.email);
107         if (success != true) {
108             setState(() {
109                 _errorMessage = 'Failed to resend OTP';
110             });
111         } catch (e) {
112             setState(() {
113                 _errorMessage = 'An error occurred. Please try again.';
114             });
115         } finally {
116             setState(() {
117                 _isLoading = false;
118             });
119         }
120     }
121
122     @override
123     void dispose() {
124         for (var controller in _controllers) {
125             controller.dispose();
126         }
127         super.dispose();
128     }
129
130     @override
131     Widget build(BuildContext context) {
132         return Scaffold(
133             appBar: AppBar(backgroundColor: Colors.transparent),
134             body: Padding(
135                 padding: const EdgeInsets.symmetric(horizontal: 24.0),
136                 child: Column(
137                     crossAxisAlignment: CrossAxisAlignment.start,
138                     children: [
```

```
139         Text(  
140             'Check your email',  
141             style: TextStyle(fontSize: 24, fontWeight: FontWeight  
142                 .bold),  
143         ),  
144         SizedBox(height: 8),  
145         Text(  
146             'We sent a reset link to ${widget.email}\nEnter 5  
147             digit code that is mentioned in the email',  
148             style: TextStyle(fontSize: 16, color: Colors.grey),  
149         ),  
150         Row(  
151             mainAxisAlignment: MainAxisAlignment.spaceBetween,  
152             children: List.generate(5, (index) {  
153                 return SizedBox(  
154                     width: 50,  
155                     height: 50,  
156                     child: TextField(  
157                         controller: _controllers[index],  
158                         maxLength: 1,  
159                         keyboardType: TextInputType.number,  
160                         textAlign: TextAlign.center,  
161                         decoration: InputDecoration(  
162                             counterText: '',  
163                             border: OutlineInputBorder(  
164                                 borderRadius: BorderRadius.circular(8),  
165                             ),  
166                             onChanged: (value) {  
167                                 if (value.length == 1 && index < 4) {  
168                                     FocusScope.of(context).nextFocus();  
169                                 }  
170                             },  
171                         ),  
172                     );  
173                 }  
174             ),  
175             if (_errorMessage != null)  
176                 Padding(  
177                     padding: const EdgeInsets.only(top: 16.0),
```

```

178         child: Text(
179             _errorMessage!,
180             style: TextStyle(color: Colors.red),
181         ),
182     ),
183     SizedBox(height: 32),
184     SizedBox(
185         width: double.infinity,
186         child: ElevatedButton(
187             onPressed: _isButtonEnabled && !_isLoading
188                 ? _verifyOTP
189                 : null,
190             style: ElevatedButton.styleFrom(
191                 backgroundColor: _isButtonEnabled && !_isLoading
192                     ? Colors.blue
193                     : Colors.grey[300],
194                 padding: EdgeInsets.symmetric(vertical: 16),
195                 shape: RoundedRectangleBorder(
196                     borderRadius: BorderRadius.circular(8),
197                 ),
198             ),
199             child: _isLoading
200                 ? CircularProgressIndicator(color: Colors.white
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217

```

```
218             style: TextStyle(color: Colors.blue),
219         ),
220     ),
221 ),
222 ],
223 ),
224 ),
225 );
226 }
227 }
```

LISTING 2.4: Password Reset Verification Code

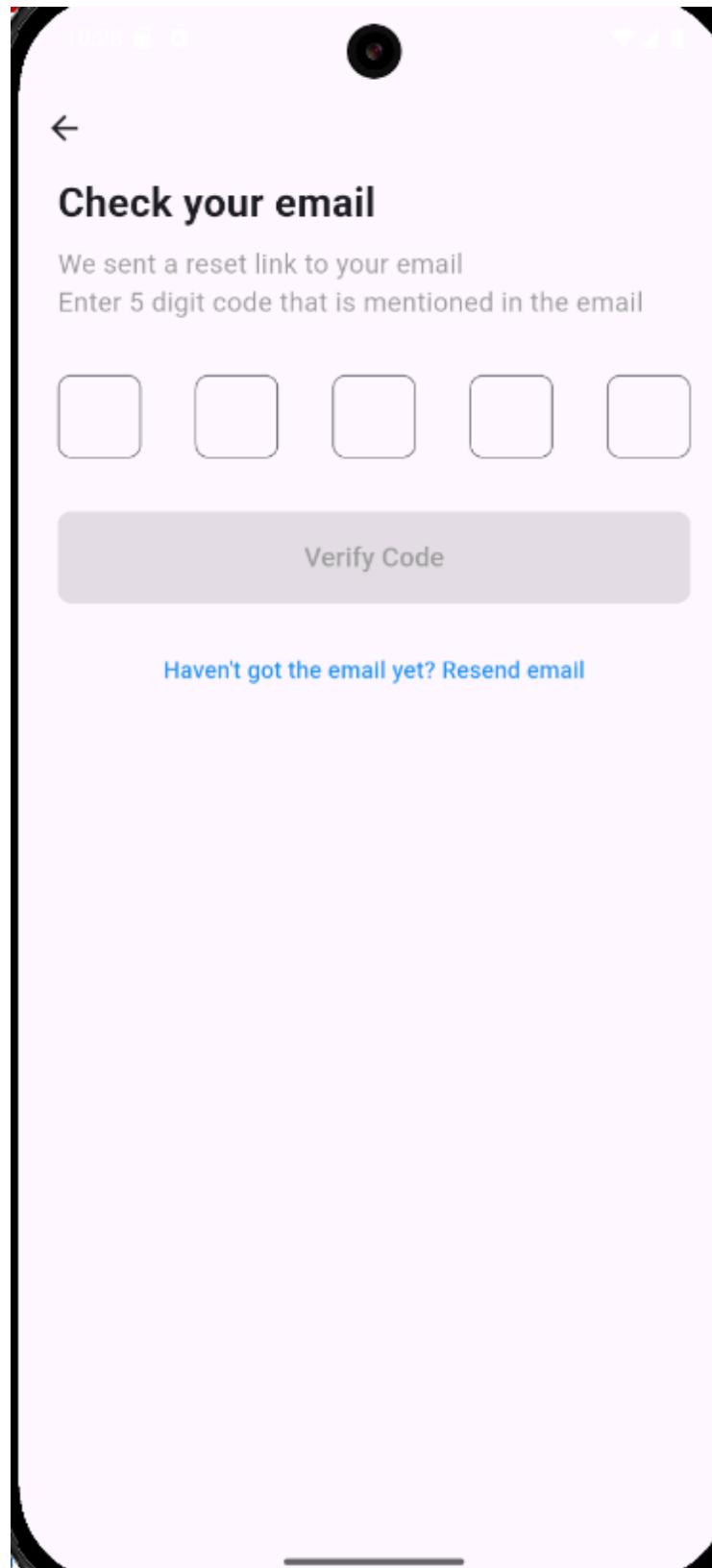


FIGURE 2.4: Password Reset Verification Screen

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	<ol style="list-style-type: none"> 1. User must have completed identity verification. 2. User must have valid access to this password reset flow.
Input Elements	<ol style="list-style-type: none"> 1. Password Field: <ul style="list-style-type: none"> • Secure input for new password 2. Confirm Password Field: <ul style="list-style-type: none"> • Re-entry of new password for verification 3. Update Password Button: <ul style="list-style-type: none"> • Submits the new password 4. Password Strength Indicator (Clicked): <ul style="list-style-type: none"> • Visual feedback on password complexity
Output Elements	<ol style="list-style-type: none"> 1. Success Notification: <ul style="list-style-type: none"> • Password updated confirmation • Automatic redirection to login screen 2. Error Messages: <ul style="list-style-type: none"> • Passwords don't match • Password same as previous • Invalid password format

<p>Navigation Flow</p>	<ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> • Checks password strength • Verifies password match • Compares with previous passwords 5. Success Flow: <ul style="list-style-type: none"> • Password updated • User redirected to login screen 6. Failure Flow: <ul style="list-style-type: none"> • Error messages shown for invalid inputs 7. Security Consideration: <ul style="list-style-type: none"> • Invalid attempts may trigger security timeout
-------------------------------	--

Code Implementation

```

1 import 'package:flutter/material.dart';
2 import 'package:flutter_django_recipes_frontend/services/
  auth_service.dart';
3 import 'package:http/http.dart' as http;
4 import 'dart:convert';
5
6 class SetNewPassword extends StatefulWidget {
7   const SetNewPassword({super.key, required this.email});
8   final String email;
9
10  @override
11  SetNewPasswordState createState() => SetNewPasswordState();
12 }
13
14 class SetNewPasswordState extends State<SetNewPassword> {
15   final _passwordController = TextEditingController();
16   final _confirmPasswordController = TextEditingController();
17   bool _isPassword = true;
18   bool _isPassword2 = true;
19   bool _isLoading = false;
20   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 ) {
33         setState(() {
34             _errorMessage = 'Passwords do not match';
35         });
36         return;
37     }
38
39     setState(() {
40         _isLoading = true;
41         _errorMessage = null;
42     });
43
44     try {
45         final csrfData = await AuthService.getCsrfToken();
46         if (csrfData == null) {
47             setState(() {
48                 _errorMessage = 'Failed to get CSRF token';
49                 _isLoading = false;
50             });
51             return;
52         }
53
54         final client = http.Client();
55         final response = await client.post(
56             Uri.parse('${AuthService.baseUrl}/updatePassword/'),
57             headers: {
58                 'Content-Type': 'application/json',
59                 'X-CSRFToken': csrfData['csrfToken'],
60                 'Cookie': csrfData['csrfCookie'],
61             },
62         );
63     } catch (e) {
64         setState(() {
65             _errorMessage = e.toString();
66             _isLoading = false;
67         });
68     }
69 }
```

```
61         body: json.encode({
62             'email': widget.email,
63             'new_password': _passwordController.text,
64         }),
65     );
66
67     final responseData = json.decode(response.body);
68     client.close();
69
70     if (response.statusCode == 200) {
71         Navigator.popUntil(context, (route) => route.isFirst);
72         ScaffoldMessenger.of(context).showSnackBar(
73             SnackBar(
74                 content: Text('Password updated successfully!'),
75                 backgroundColor: Colors.green,
76             ),
77         );
78     } else {
79         setState(() {
80             _errorMessage = responseData['message'] ?? 'Failed to
update password';
81         });
82     }
83     } catch (e) {
84         setState(() {
85             _errorMessage = 'An error occurred. Please try again.';
86         });
87     } finally {
88         setState(() {
89             _isLoading = false;
90         });
91     }
92 }
93
94 @override
95 void dispose() {
96     _passwordController.dispose();
97     _confirmPasswordController.dispose();
98     super.dispose();
99 }
100
```

```
101 @override
102 Widget build(BuildContext context) {
103   return Scaffold(
104     appBar: AppBar(),
105     body: ListView(
106       physics: NeverScrollableScrollPhysics(),
107       children: [
108         Padding(
109           padding: EdgeInsets.all(20.0),
110           child: Column(
111             mainAxisAlignment: MainAxisAlignment.start,
112             crossAxisAlignment: CrossAxisAlignment.start,
113             children: [
114               Text(
115                 "Set a new password",
116                 style: TextStyle(
117                   fontWeight: FontWeight.bold,
118                   fontSize: 24.0,
119                 ),
120             ),
121             Padding(padding: EdgeInsets.all(3.0)),
122             Text(
123               "Create a new password. Ensure it differs from
124               previous ones for security",
125               style: TextStyle(fontSize: 16.0),
126             ),
127             Padding(padding: EdgeInsets.all(8.0)),
128             if (_errorMessage != null)
129               Padding(
130                 padding: const EdgeInsets.only(bottom: 8.0),
131                 child: Text(
132                   _errorMessage!,
133                   style: TextStyle(
134                     color: Colors.red,
135                     fontSize: 14.0,
136                   ),
137                 ),
138             ),
139             Text(
140
```

```

141         "Password",
142         style: TextStyle(
143             fontWeight: FontWeight.bold,
144             fontSize: 16.0,
145         ),
146     ),
147     Padding(padding: EdgeInsets.all(2.0)),
148     TextField(
149         controller: _passwordController,
150         obscureText: _isPassword,
151         decoration: InputDecoration(
152             suffixIcon: Padding(
153                 padding: const EdgeInsets.symmetric(
horizontal: 8.0),
154                 child: IconButton(
155                     icon: Icon(
156                         _isPassword ? Icons.visibility_off :
Icons.visibility,
157                     ),
158                     onPressed: () => setState(() => _isPassword
= !_isPassword),
159                 ),
160             ),
161             border: OutlineInputBorder(
162                 borderRadius: BorderRadius.circular(12.0),
163                 borderSide: BorderSide(
164                     width: 2.0,
165                     color: const Color.fromARGB(255, 184, 183,
183),
166                 ),
167             ),
168             enabledBorder: OutlineInputBorder(
169                 borderRadius: BorderRadius.circular(12.0),
170                 borderSide: BorderSide(
171                     width: 2.0,
172                     color: const Color.fromARGB(255, 184, 183,
183),
173                 ),
174             ),
175             focusedBorder: OutlineInputBorder(
176                 borderRadius: BorderRadius.circular(12.0),

```

```

177         borderSide: BorderSide(width: 2.0, color:
Colors.black54),
178     ),
179     ),
180 ),
181     Padding(padding: EdgeInsets.all(5.0)),
182
183     Text(
184         "Confirm Password",
185         style: TextStyle(
186             fontWeight: FontWeight.bold,
187             fontSize: 16.0,
188         ),
189     ),
190     Padding(padding: EdgeInsets.all(2.0)),
191     TextField(
192         controller: _confirmPasswordController,
193         obscureText: _isPassword2,
194         decoration: InputDecoration(
195             suffixIcon: Padding(
196                 padding: const EdgeInsets.symmetric(
horizontal: 8.0),
197                 child: IconButton(
198                     icon: Icon(
199                         _isPassword2 ? Icons.visibility_off :
Icons.visibility,
200                     ),
201                     onPressed: () => setState(() =>
_isPassword2 = !_isPassword2),
202                 ),
203             ),
204             border: OutlineInputBorder(
205                 borderRadius: BorderRadius.circular(12.0),
206                 borderSide: BorderSide(
207                     width: 2.0,
208                     color: const Color.fromARGB(255, 184, 183,
183),
209             ),),
210             enabledBorder: OutlineInputBorder(
211                 borderRadius: BorderRadius.circular(12.0),
212                 borderSide: BorderSide(

```

```

213         width: 2.0,
214         color: const Color.fromARGB(255, 184, 183,
183),
215     ),
216 ),
217 focusedBorder: OutlineInputBorder(
218     borderRadius: BorderRadius.circular(12.0),
219     borderSide: BorderSide(width: 2.0, color:
Colors.black54),
220 ),
221 ),
222 ),
223 Padding(padding: EdgeInsets.all(10.0)),
224 OutlinedButton(
225     style: OutlinedButton.styleFrom(
226         backgroundColor: _isLoading ? Colors.blue[200]
: Colors.blue,
227         minimumSize: Size(double.infinity, 60.0),
228         side: BorderSide(color: Colors.blue),
229         shape: RoundedRectangleBorder(
230             borderRadius: BorderRadius.circular(15.0),
231         ),
232     ),
233     onPressed: _isLoading ? null : _updatePassword,
234     child: _isLoading
        ? CircularProgressIndicator(color: Colors.
white)
        : Text(
235         "Update Password",
236         style: TextStyle(
237             color: Colors.white,
238             fontWeight: FontWeight.bold,
239             fontSize: 16.0,
240         ),
241     ),
242 ),
243 ],
244 ],
245 ],
246 ),
247 ),],
248 ),,);}}

```

LISTING 2.5: Set New Password Code

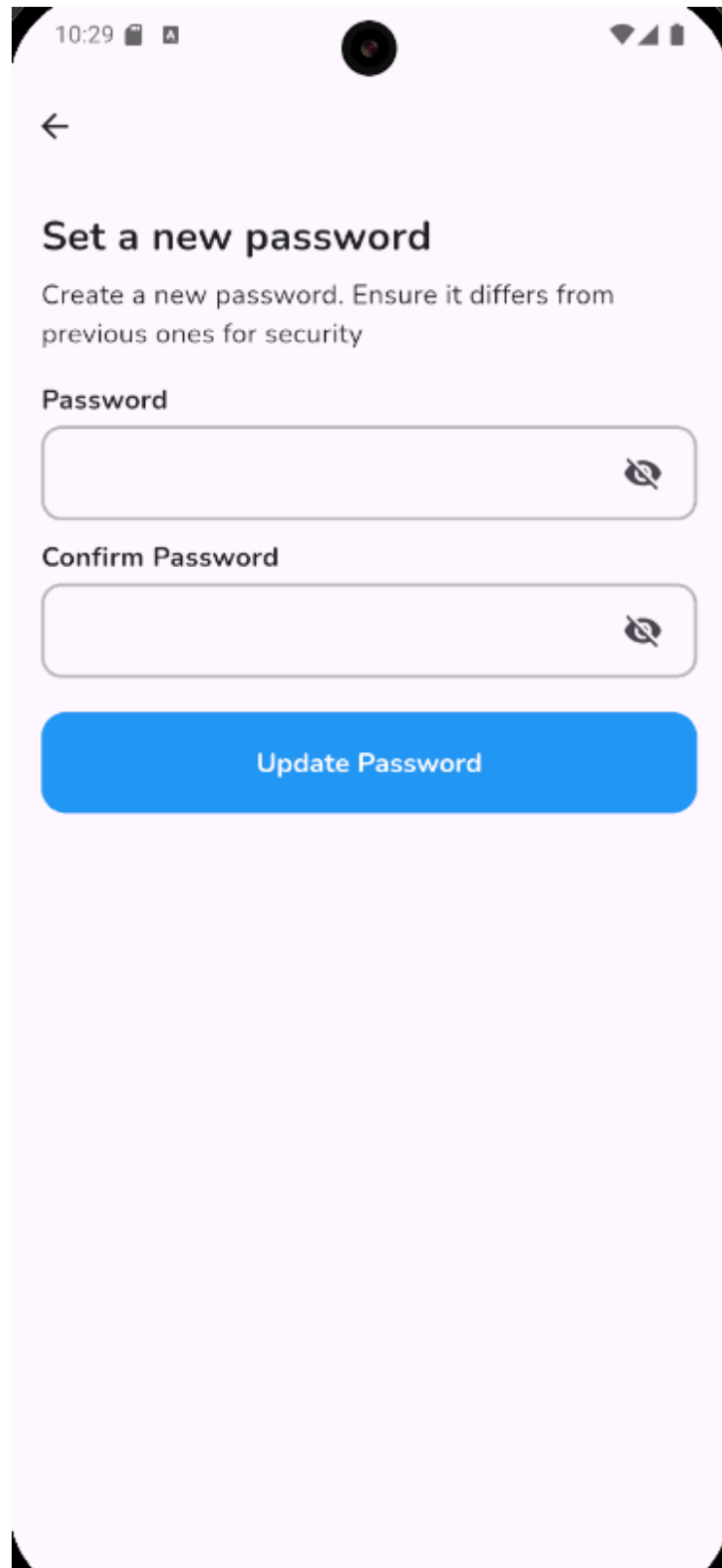


FIGURE 2.5: Set New Password Screen

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: <ul style="list-style-type: none"> • Returns to main menu 2. Generate Button: <ul style="list-style-type: none"> • Creates a new random recipe 3. Generate with Constraints Button: <ul style="list-style-type: none"> • Opens dialog for constrained recipe generation 4. Recipe Cards (tappable): <ul style="list-style-type: none"> • Displays recipe previews • Opens detailed recipe view when tapped
Output Elements	1. Recipe List: <ul style="list-style-type: none"> • Displays saved recipes with names and descriptions 2. New Recipe Dialog: <ul style="list-style-type: none"> • Appears when generating new recipes 3. Constraint Selection Interface: <ul style="list-style-type: none"> • Appears when generating with constraints

<p>Navigation Flow</p>	<ol style="list-style-type: none"> 1. User accesses the recipe management screen from main menu. 2. System displays saved recipes in card format. 3. User can: <ul style="list-style-type: none"> • Tap a recipe to view details • Press "Generate" for random recipe • Press "Generate with Constraints" for specific needs 4. For generation options: <ul style="list-style-type: none"> • System processes request • Displays new recipe • Offers save option 5. Alternative Flows: <ul style="list-style-type: none"> • No recipes: Shows empty state with prompt to generate • Network issues: Shows error message 6. Home button returns to main menu.
-------------------------------	---

Code Implementation

```

1 import 'package:flutter/material.dart';
2 import 'package:flutter_django_recipes_frontend/
  add_edit_recipe_page.dart';
3 import 'package:flutter_django_recipes_frontend/login_page.dart';
4 import 'package:flutter_django_recipes_frontend/recipe_detail_page.
  dart';
5 import 'package:flutter_django_recipes_frontend/services/
  recipe_service.dart';
6 import 'package:flutter_django_recipes_frontend/ingredients_list.
  dart';
7
8 class HomePage extends StatefulWidget {
9   const HomePage({super.key});
10
11   @override
12   State<HomePage> createState() => _HomePageState();
13 }
14

```

```
15 class _HomePageState extends State<HomePage> {
16   List<dynamic> recipes = [];
17   bool isLoading = true;
18   bool hasError = false;
19   String errorMessage = '';
20
21   @override
22   void initState() {
23     super.initState();
24     _loadRecipes();
25   }
26
27   Future<void> _loadRecipes() async {
28     setState(() {
29       isLoading = true;
30       hasError = false;
31     });
32
33     try {
34       final loadedRecipes = await RecipeService.getRecipes();
35       setState(() {
36         recipes = loadedRecipes;
37         isLoading = false;
38       });
39     } catch (e) {
40       setState(() {
41         isLoading = false;
42         hasError = true;
43         errorMessage = e.toString();
44       });
45       _showErrorSnackBar('Failed to load recipes: $e');
46     }
47   }
48
49   Future<void> _handleAddRecipe() async {
50     final result = await Navigator.push(
51       context,
52       MaterialPageRoute(
53         builder: (context) => const AddEditRecipePage(
54           availableIngredients: availableIngredients,
55           ),
```

```
56     ),
57   );
58
59   if (result != null) {
60     try {
61       setState(() => isLoading = true);
62       final newRecipe = await RecipeService.addRecipe(
63         name: result['name'],
64         description: result['description'],
65         ingredients: result['ingredients'],
66       );
67       setState(() {
68         recipes.add(newRecipe);
69         isLoading = false;
70       });
71     } catch (e) {
72       setState(() => isLoading = false);
73       _showErrorSnackBar('Failed to add recipe: $e');
74     }
75   }
76 }
77
78 Future<void> _handleEditRecipe(Map<String, dynamic> recipe, int
index) async {
79   final result = await Navigator.push(
80     context,
81     MaterialPageRoute(
82       builder: (context) => AddEditRecipePage(
83         recipe: recipe,
84         availableIngredients: availableIngredients,
85       ),
86     ),
87   );
88
89   if (result != null) {
90     try {
91       setState(() => isLoading = true);
92       final updatedRecipe = await RecipeService.updateRecipe(
93         id: recipe['id'],
94         name: result['name'],
95         description: result['description'],
```

```
196         ingredients: result['ingredients'],
197     );
198     setState(() {
199         recipes[index] = updatedRecipe;
200         isLoading = false;
201     });
202   } catch (e) {
203     setState(() => isLoading = false);
204     _showErrorSnackBar('Failed to update recipe: $e');
205   }
206 }
207 }
208
209 Future<void> _handleDeleteRecipe(int index) async {
210   final recipe = recipes[index];
211   try {
212     setState(() => isLoading = true);
213     await RecipeService.deleteRecipe(recipe['id']);
214     setState(() {
215       recipes.removeAt(index);
216       isLoading = false;
217     });
218   } catch (e) {
219     setState(() => isLoading = false);
220     _showErrorSnackBar('Failed to delete recipe: $e');
221   }
222 }
223
224 void _showErrorSnackBar(String message) {
225   ScaffoldMessenger.of(context).showSnackBar(
226     SnackBar(
227       content: Text(message),
228       backgroundColor: Colors.red,
229     ),
230   );
231 }
232
233 void _viewRecipeDetails(Map<String, dynamic> recipe) {
234   Navigator.push(
235     context,
236     MaterialPageRoute(
```

```
137         builder: (context) => RecipeDetailPage(recipe: recipe),
138       ),
139     );
140   }
141
142   void _logout() {
143     Navigator.of(context).pushReplacement(
144       MaterialPageRoute(builder: (context) => const LoginPage()),
145     );
146   }
147
148   Widget _buildRecipeList() {
149     if (hasError) {
150       return Center(
151         child: Column(
152           mainAxisAlignment: MainAxisAlignment.center,
153           children: [
154             const Icon(Icons.error_outline, size: 48, color: Colors
155               .red),
156             const SizedBox(height: 16),
157             Text(errorMessage, style: const TextStyle(color: Colors
158               .red)),
159             const SizedBox(height: 16),
160             ElevatedButton(
161               onPressed: _loadRecipes,
162               child: const Text('Retry'),
163             ),
164           ],
165         ),
166       );
167     }
168
169     if (recipes.isEmpty) {
170       return const Center(
171         child: Text('No recipes found. Add your first recipe!'),
172       );
173     }
174
175     return ListView.builder(
176       itemCount: recipes.length,
177       itemBuilder: (context, index) {
```

```
176     final recipe = recipes[index];
177     final ingredients = (recipe['ingredients'] as List<dynamic
178 >)
179         .map<String>((i) => i['name'] as String)
180         .join(', ');
181
182     return Card(
183         margin: const EdgeInsets.all(8),
184         elevation: 2,
185         shape: RoundedRectangleBorder(
186             borderRadius: BorderRadius.circular(12),
187         ),
188         child: ListTile(
189             title: Text(
190                 recipe['name'],
191                 style: const TextStyle(fontWeight: FontWeight.bold),
192             ),
193             subtitle: Column(
194                 crossAxisAlignment: CrossAxisAlignment.start,
195                 children: [
196                     Text(recipe['description']),
197                     const SizedBox(height: 4),
198                     Text(
199                         'Ingredients: $ingredients',
200                         style: const TextStyle(fontSize: 12, color:
201 Colors.grey),
202                     ),
203                 ],
204             ),
205             onTap: () => _viewRecipeDetails(recipe),
206             trailing: PopupMenuButton<String>(
207                 onSelected: (value) {
208                     if (value == 'edit') _handleEditRecipe(recipe,
209 index);
210                     if (value == 'delete') _handleDeleteRecipe(index);
211                 },
212                 itemBuilder: (context) => [
213                     const PopupMenuItem(
214                         value: 'edit',
215                         child: Text('Edit'),
216                     ),
217                 ],
218             ),
219         ),
220     );
```

```

214         const PopupMenuItem(
215             value: 'delete',
216             child: Text('Delete'),
217         ),
218     ],
219 ),
220 ),
221 );
222 },
223 );
224 }
225
226 @override
227 Widget build(BuildContext context) {
228     return Scaffold(
229         appBar: AppBar(
230             title: const Text('Your Recipes'),
231             actions: [
232                 IconButton(
233                     icon: const Icon(Icons.logout),
234                     onPressed: _logout,
235                     tooltip: 'Logout',
236                 ),
237                 IconButton(
238                     icon: const Icon(Icons.add),
239                     onPressed: _handleAddRecipe,
240                     tooltip: 'Add Recipe',
241                 ),
242             ],
243         ),
244         body: isLoading
245             ? const Center(child: CircularProgressIndicator())
246             : RefreshIndicator(
247                 onRefresh: _loadRecipes,
248                 child: _buildRecipeList(),
249             ),
250         floatingActionButton: FloatingActionButton(
251             onPressed: _handleAddRecipe,
252             child: const Icon(Icons.add),), ),);}

```

LISTING 2.6: Home Page Code

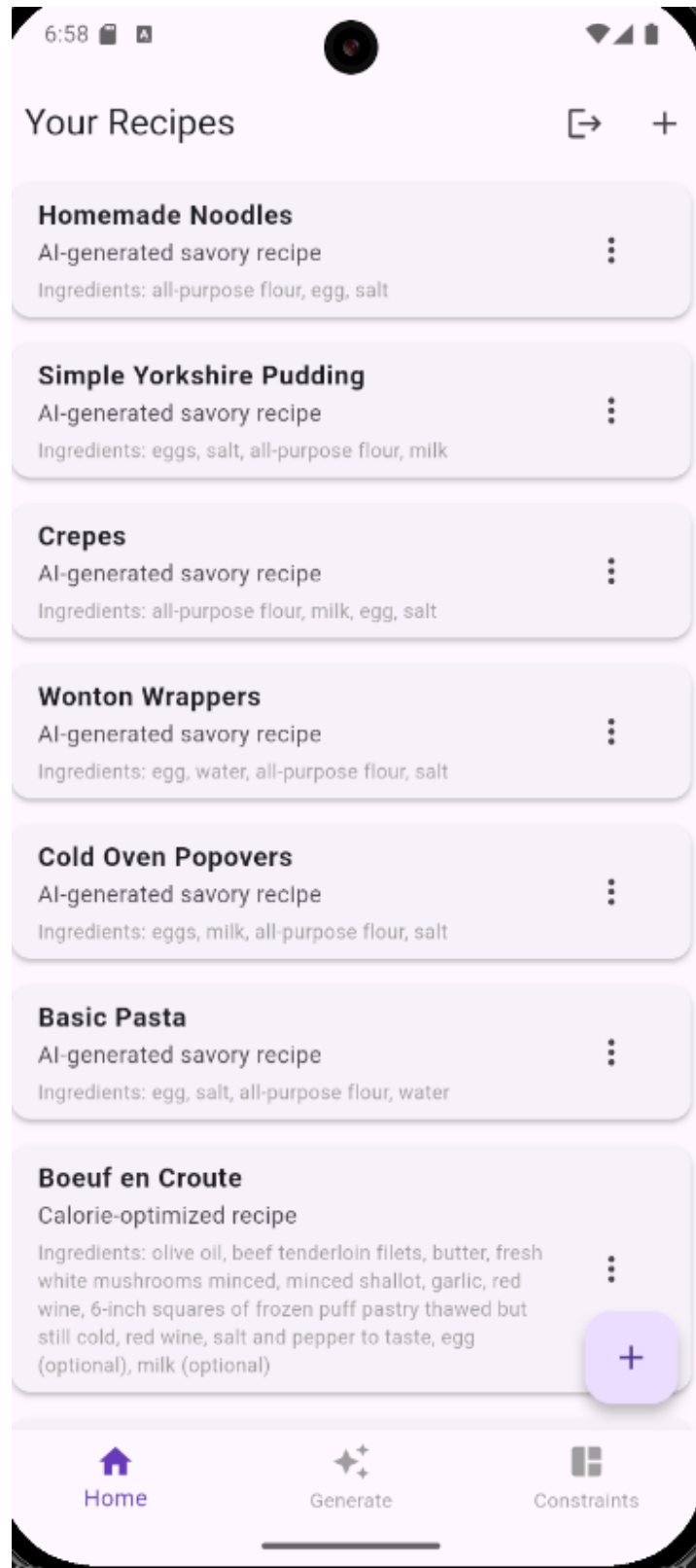


FIGURE 2.6: Home Screen

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: <ul style="list-style-type: none">• Text input for recipe title 2. Description Field: <ul style="list-style-type: none">• Multi-line input for recipe instructions/details 3. Ingredient Selection: <ul style="list-style-type: none">• Checkbox/tag system for selecting ingredients• Supports up to 30 ingredients 4. Add Recipe Button: <ul style="list-style-type: none">• Submits the completed recipe 5. Back Button (Clicked): <ul style="list-style-type: none">• Returns to previous screen

Output Elements	<ol style="list-style-type: none"> 1. Success Notification: <ul style="list-style-type: none"> • Recipe saved confirmation • Automatic return to recipe list 2. Error Messages: <ul style="list-style-type: none"> • Missing required fields • Too many ingredients selected • Duplicate recipe name 3. Ingredient Badges: <ul style="list-style-type: none"> • Visual display of selected ingredients 4. Character Counters: <ul style="list-style-type: none"> • For name and description fields
Navigation Flow	<ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> • Checks for required fields • Verifies ingredient count • Checks for duplicates 6. Success Flow: <ul style="list-style-type: none"> • Recipe saved to database • User returned to recipe list with new recipe visible 7. Alternative Flows: <ul style="list-style-type: none"> • Validation errors show specific messages • Network issues show retry option 8. Back button cancels creation and returns to list.

Code Implementation

```

1 import 'package:flutter/material.dart';
2
3 class AddEditRecipePage extends StatefulWidget {
4   final Map? recipe; // If null, were adding a new recipe. If not
   null, were editing.

```

```
5   final List<String> availableIngredients; // Ingredients to choose
      from (e.g., loaded via Django backend)
6
7   const AddEditRecipePage({super.key, this.recipe, required this.
      availableIngredients});
8
9   @override
10  _AddEditRecipePageState createState() => _AddEditRecipePageState
      ();
11 }
12
13 class _AddEditRecipePageState extends State<AddEditRecipePage> {
14   final _formKey = GlobalKey<FormState>();
15
16   // Controllers for input fields
17   late TextEditingController _nameController;
18   late TextEditingController _descriptionController;
19
20   // Stores selected ingredients from chips
21   List<String> selectedIngredients = [];
22
23   @override
24   void initState() {
25     super.initState();
26
27     // Pre-fill data if editing
28     _nameController = TextEditingController(text: widget.recipe?['
name'] ?? '');
29     _descriptionController = TextEditingController(
30       text: widget.recipe?['description'] ?? '',
31     );
32
33     // Initialize selected ingredients (if editing)
34     selectedIngredients = widget.recipe?['ingredientsList']?.cast<
String>() ?? [];
35   }
36
37   // Called when user presses "Add" or "Save Changes"
38   void _saveRecipe() {
39     if (_formKey.currentState!.validate()) {
40       final newRecipe = {
```

```

41         "name": _nameController.text,
42         "description": _descriptionController.text,
43         "ingredients": selectedIngredients.join(', '), // For
display
44         "ingredientsList": selectedIngredients,          // For
internal use or backend
45     };
46     // Return recipe to previous screen
47     Navigator.pop(context, newRecipe);
48 }
49 }
50
51 @override
52 Widget build(BuildContext context) {
53     return Scaffold(
54         appBar: AppBar(
55             title: Text(widget.recipe == null ? 'Add Recipe' : 'Edit
Recipe'),
56         ),
57         body: Padding(
58             padding: EdgeInsets.all(16),
59             child: Form(
60                 key: _formKey,
61                 child: ListView(
62                     children: [
63                         // Recipe Name Input
64                         TextFormField(
65                             controller: _nameController,
66                             decoration: InputDecoration(labelText: 'Recipe Name
'),
67                             validator: (value) => value!.isEmpty ? 'Enter
recipe name' : null,
68                         ),
69                         SizedBox(height: 16),
70
71                         // Description Input
72                         TextFormField(
73                             controller: _descriptionController,
74                             decoration: InputDecoration(labelText: 'Description
'),

```

```

75         validator: (value) => value!.isEmpty ? 'Enter
description' : null,
76     ),
77     SizedBox(height: 16),
78
79     // Ingredient Selection Section
80     Text(
81         'Select Ingredients (up to 30)',
82         style: TextStyle(fontWeight: FontWeight.bold),
83     ),
84     Wrap(
85         spacing: 8,
86         children: widget.availableIngredients.map((
ingredient) {
87             final isSelected = selectedIngredients.contains(
ingredient);
88             return FilterChip(
89                 label: Text(ingredient),
90                 selected: isSelected,
91                 onSelected: (selected) {
92                     setState(() {
93                         if (selected) {
94                             // Add if not more than 30
95                             if (selectedIngredients.length < 30) {
96                                 selectedIngredients.add(ingredient);
97                             }
98                         } else {
99                             // Remove if deselected
100                             selectedIngredients.remove(ingredient);
101                         }
102                     });
103                 },),)).toList(),
104     ),
105     SizedBox(height: 24),
106
107     // Submit Button
108     ElevatedButton(
109         onPressed: _saveRecipe,
110         child: Text(widget.recipe == null ? 'Add Recipe' :
'Save Changes'),),),),),    ),    );  }}

```

LISTING 2.7: Recipe Creation Code

6:59

← Add Recipe

Enter recipe name

Enter recipe description

Ingredients:

Available Ingredients

- ☐ olive oil
- ☐ canola oil
- ☐ all purpose flour
- ☐ sugar
- ☐ salt
- ☐ black pepper

Add Recipe

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: <ul style="list-style-type: none">• Editable text field for recipe title 2. Description Field: <ul style="list-style-type: none">• Editable multi-line input for recipe details 3. Ingredient Selection: <ul style="list-style-type: none">• Checkbox/tag system for modifying ingredients• Supports up to 30 ingredients 4. Save Changes Button: <ul style="list-style-type: none">• Submits the updated recipe 5. Back Button (Clicked): <ul style="list-style-type: none">• Returns to previous screen

Output Elements	<ol style="list-style-type: none"> 1. Success Notification: <ul style="list-style-type: none"> • Changes saved confirmation • Automatic return to recipe list 2. Error Messages: <ul style="list-style-type: none"> • Missing required fields • Too many ingredients selected 3. Original Recipe Display: <ul style="list-style-type: none"> • Shows pre-edit values for reference 4. Ingredient Badges: <ul style="list-style-type: none"> • Visual display of currently selected ingredients
Navigation Flow	<ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> • Checks for required fields • Verifies ingredient count 6. Success Flow: <ul style="list-style-type: none"> • Updated recipe saved to database • User returned to recipe list with changes visible 7. Alternative Flows: <ul style="list-style-type: none"> • Validation errors show specific messages • Cancellation returns without saving 8. Back button cancels editing if changes not saved.

Code Implementation

```

1 import 'package:flutter/material.dart';
2
3 class AddEditRecipePage extends StatefulWidget {
4   final Map? recipe;
5   final List<String> availableIngredients;
6
7   const AddEditRecipePage({
8     super.key,
9     this.recipe,

```

```

10     required this.availableIngredients,
11   });
12
13   @override
14   State<AddEditRecipePage> createState() => _AddEditRecipePageState
15     ();
16
17   class _AddEditRecipePageState extends State<AddEditRecipePage> {
18     late final TextEditingController _nameController;
19     late final TextEditingController _descController;
20     List<String> _selectedIngredients = [];
21     String? _nameError;
22     String? _descError;
23
24     @override
25     void initState() {
26       super.initState();
27       _nameController = TextEditingController(text: widget.recipe?['
28         name'] ?? '');
29       _descController = TextEditingController(
30         text: widget.recipe?['description'] ?? '',
31       );
32
33       if (widget.recipe != null && widget.recipe!['ingredients'] !=
34         null) {
35         final ingredientsData = widget.recipe!['ingredients'];
36         if (ingredientsData is List) {
37           _selectedIngredients =
38             ingredientsData.map<String>((e) => e['name'] as String)
39             .toList();
40         }
41       }
42
43       @override
44       void dispose() {
45         _nameController.dispose();
46         _descController.dispose();
47         super.dispose();
48       }

```

```
47
48 bool _validateFields() {
49     setState(() {
50         _nameError = _nameController.text.isEmpty ? 'Recipe name is
51         required' : null;
52         _descError = _descController.text.isEmpty ? 'Description is
53         required' : null;
54     });
55     return _nameError == null && _descError == null;
56 }
57
58 void _saveRecipe() {
59     if (_validateFields()) {
60         Navigator.pop(context, {
61             'name': _nameController.text,
62             'description': _descController.text,
63             'ingredients': _selectedIngredients,
64         });
65     }
66 }
67
68 @override
69 Widget build(BuildContext context) {
70     final screenHeight = MediaQuery.of(context).size.height;
71
72     return Scaffold(
73         appBar: AppBar(
74             title: Text(widget.recipe == null ? 'Add Recipe' : 'Edit
75             Recipe'),
76         ),
77         body: SafeArea(
78             child: ListView(
79
80                 padding: const EdgeInsets.all(16),
81                 children: [
82                     Column(
83                         children: [
84                             TextField(
85                                 controller: _nameController,
86                                 decoration: InputDecoration(
87                                     hintText: 'Enter recipe name',
```

```
85         errorText: _nameError,
86         border: OutlineInputBorder(
87           borderRadius: BorderRadius.circular(12),
88         ),
89         contentPadding: const EdgeInsets.symmetric(
90           horizontal: 16,
91           vertical: 14,
92         ),
93       ),
94       onChanged: (_) {
95         if (_nameError != null) {
96           setState(() => _nameError = null);
97         }
98       },
99     ),
100     const SizedBox(height: 16),
101
102     TextField(
103       controller: _descController,
104       maxLines: 3,
105       decoration: InputDecoration(
106         hintText: 'Enter recipe description',
107         errorText: _descError,
108         border: OutlineInputBorder(
109           borderRadius: BorderRadius.circular(12),
110         ),
111         contentPadding: const EdgeInsets.symmetric(
112           horizontal: 16,
113           vertical: 14,
114         ),
115       ),
116       onChanged: (_) {
117         if (_descError != null) {
118           setState(() => _descError = null);
119         }
120       },
121     ),
122     const SizedBox(height: 16),
123
124     const Text(
125       'Ingredients:',
```

```

126         style: TextStyle(fontWeight: FontWeight.bold),
127     ),
128     const SizedBox(height: 8),
129
130     Wrap(
131         spacing: 8,
132         runSpacing: 8,
133         children: _selectedIngredients.map((ingredient) {
134             return Chip(
135                 label: Text(ingredient),
136                 onDelete: () => setState(() {
137                     _selectedIngredients.remove(ingredient);
138                 }),
139                 deleteIcon: const Icon(Icons.close, size: 18)
140             ),
141         }).toList(),
142     ),
143     const SizedBox(height: 8),
144
145     SizedBox(
146         height: screenHeight * 0.4,
147         child: Card(
148             elevation: 2,
149             child: Column(
150                 children: [
151                     Padding(
152                         padding: const EdgeInsets.all(8.0),
153                         child: Text(
154                             'Available Ingredients',
155                             style: Theme.of(context).textTheme.
titleMedium,
156                         ),
157                     ),
158                     const Divider(height: 1),
159                     Expanded(
160                         child: ListView.builder(
161                             itemCount: widget.availableIngredients.
length,
162                             itemBuilder: (context, index) {

```

```

163         final ingredient = widget.
availableIngredients[index];
164         final isSelected =
_selectedIngredients.contains(ingredient);
165         return CheckboxListTile(
166             title: Text(ingredient),
167             value: isSelected,
168             onChanged: (selected) {
169                 setState(() {
170                     if (selected == true) {
171                         _selectedIngredients.add(
ingredient);
172                     } else {
173                         _selectedIngredients.remove(
ingredient);
174                     }
175                 });
176             },
177             controlAffinity:
ListTileControlAffinity.leading,
178         );
179     },
180 ),
181 ),
182 ],
183 ),
184 ),
185 ),
186
187     const SizedBox(height: 16),
188     ElevatedButton(
189         onPressed: _saveRecipe,
190         style: ElevatedButton.styleFrom(
191             minimumSize: const Size(double.infinity, 50),
192             shape: RoundedRectangleBorder(
193                 borderRadius: BorderRadius.circular(12),
194             ),
195         ),
196         child: Text(
197             widget.recipe == null ? 'Add Recipe' : 'Save
Changes',

```

```
198         style: const TextStyle(fontSize: 16),  
199     ),  
200 ),  
201 ],  
202 ),  
203 ],  
204 ),  
205 ),  
206 );  
207 }  
208 }
```

LISTING 2.8: Recipe Editing Code

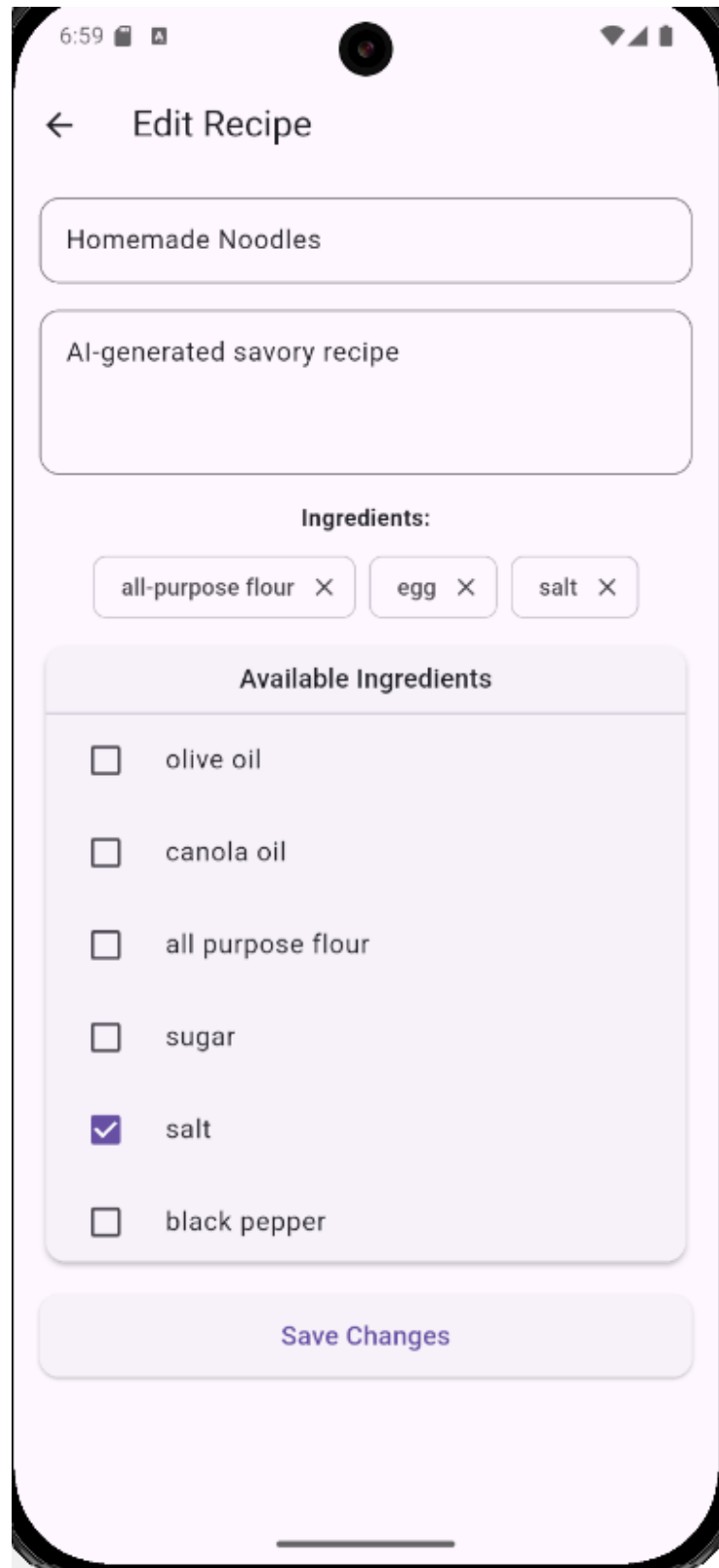


FIGURE 2.8: Recipe Editing Screen

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	<ol style="list-style-type: none"> 1. System displays AI-generated recipe. 2. User can: <ul style="list-style-type: none"> • Accept to save to their collection • Discard to remove the suggestion • Generate new random recipe • Generate with specific constraints 3. For accepted recipes: <ul style="list-style-type: none"> • Saved to user's collection • Confirmation message displayed 4. Alternative Flows: <ul style="list-style-type: none"> • Network issues show error and retry options • Empty results show appropriate message 5. Home button returns to main menu.
------------------------	---

Code Implementation

```

1 import 'package:flutter/material.dart';
2 import 'package:flutter_django_recipes_frontend/recipe_detail_page.
  dart';
3 import 'package:flutter_django_recipes_frontend/services/
  recipe_service.dart';
4 import 'package:flutter_django_recipes_frontend/services/
  auth_service.dart';
5 import 'package:http/http.dart' as http;
6 import 'dart:convert';
7
8 class GenerateRecipePage extends StatefulWidget {
9   const GenerateRecipePage({super.key});
10
11   @override
12   State<GenerateRecipePage> createState() =>
13     _GenerateRecipePageState();
14 }
15
16 class _GenerateRecipePageState extends State<GenerateRecipePage> {
17   final List<String> paletteList = [
18     'savory',
19     'spicy',

```

```
19     'sweet',
20     'sour',
21     'herby',
22     'umami',
23     'earthy',
24     'fruity',
25     'smoky',
26     'neutral',
27     'bitter',
28 ];
29
30 String? selectedPalette;
31 List<Map<String, dynamic>> recommendedRecipes = [];
32 bool isLoading = false;
33 bool isGenerating = false;
34 String? errorMessage;
35
36 @override
37 void initState() {
38     super.initState();
39     _loadUserIngredients();
40 }
41
42 Future<void> _loadUserIngredients() async {
43     setState(() {
44         isLoading = true;
45         errorMessage = null;
46     });
47
48     try {
49         final ingredients = await _getUserTopIngredients();
50         if (ingredients.isEmpty) {
51             setState(() {
52                 errorMessage =
53                 'No ingredients found in your recipes. Add some
54                 recipes first.';
55                 isLoading = false;
56             });
57         } else {
58             setState(() => isLoading = false);
59         }
60     } catch (e) {
61         setState(() {
62             errorMessage = e.toString();
63             isLoading = false;
64         });
65     }
66 }
```

```
59     } catch (e) {
60         setState(() {
61             errorMessage = 'Failed to load your ingredients: $e';
62             isLoading = false;
63         });
64         debugPrint('Error loading ingredients: $e');
65     }
66 }
67
68 Future<List<String>> _getUserTopIngredients() async {
69     final client = http.Client();
70     try {
71         final headers = await AuthService.getAuthHeaders();
72
73         final response = await client.get(
74             Uri.parse('${RecipeService.baseUrl}/recipes/'),
75             headers: headers,
76         );
77
78         if (response.statusCode == 200) {
79             final recipes = jsonDecode(response.body) as List;
80
81             if (recipes.isEmpty) return [];
82
83             final ingredientCounts = <String, int>{};
84
85             for (final recipe in recipes) {
86                 final ingredients = recipe['ingredients'] as List;
87                 for (final ing in ingredients) {
88                     final name = ing['name'].toString().toLowerCase().trim
89 ();
90                     ingredientCounts[name] = (ingredientCounts[name] ?? 0)
91 + 1;
92                 }
93             }
94
95             final sortedIngredients =
96                 ingredientCounts.entries.toList()
97                     ..sort((a, b) => b.value.compareTo(a.value));
98
99             final topIngredients =
```

```
108         sortedIngredients.take(4).map((e) => e.key).toList();
109         return topIngredients;
110     } else {
111         throw Exception('Failed to load recipes: ${response.
112         statusCode}');
113     }
114 } finally {
115     client.close();
116 }
117 }
118
119 Future<void> _generateRecipes() async {
120     if (selectedPalette == null) {
121         setState(() => errorMessage = 'Please select a palette');
122         return;
123     }
124
125     setState(() {
126         isGenerating = true;
127         errorMessage = null;
128         recommendedRecipes = [];
129     });
130
131     try {
132         final ingredients = await _getUserTopIngredients();
133         if (ingredients.isEmpty) {
134             throw Exception('No ingredients found in your recipes');
135         }
136
137         final headers = await AuthService.getAuthHeaders();
138         final response = await http.post(
139             Uri.parse('${RecipeService.baseUrl}/recipes/recommend/'),
140             headers: headers,
141             body: jsonEncode({
142                 'ingredients': ingredients,
143                 'palette': [selectedPalette],
144             }),
145         );
146
147         if (response.statusCode == 200) {
148             final data = jsonDecode(response.body) as List;
```

```

138         setState(() {
139             recommendedRecipes = List<Map<String, dynamic>>.from(data
140         );
141             isGenerating = false;
142         });
143     } else {
144         throw Exception(
145             'Failed to generate recipes: ${response.statusCode} - ${
146             response.body}',
147         );
148     }
149 } catch (e) {
150     setState(() {
151         errorMessage = 'Failed to generate recipes: $e';
152         isGenerating = false;
153     });
154 }
155
156 Future<void> _saveRecipe(Map<String, dynamic> recipe) async {
157     try {
158         final recipeName = recipe['recipe_name'] ?? recipe['name'] ?? '
159         Unnamed Recipe';
160
161         List<String> ingredients = [];
162
163         if (recipe['ingredients_list'] != null) {
164             if (recipe['ingredients_list'] is String) {
165                 String ingredientsStr = recipe['ingredients_list'] as
166                 String;
167                 ingredientsStr = ingredientsStr
168                     .replaceAll('[', ' ')
169                     .replaceAll(']', ' ')
170                     .replaceAll('"', "")
171                     .trim();
172
173                 if (ingredientsStr.isNotEmpty) {
174                     ingredients = ingredientsStr.split(',').map((e) => e.trim
175                     ()).toList();
176                 }
177             }
178         }
179     }
180 }

```

```

174
175     else if (recipe['ingredients_list'] is List) {
176         ingredients = (recipe['ingredients_list'] as List)
177             .map((e) => e.toString().trim())
178             .toList();
179     }
180 }
181 else if (recipe['ingredients'] != null) {
182     if (recipe['ingredients'] is String) {
183         ingredients = (recipe['ingredients'] as String)
184             .split(',')
185             .map((e) => e.trim())
186             .toList();
187     } else if (recipe['ingredients'] is List) {
188         ingredients = (recipe['ingredients'] as List)
189             .map((e) => e.toString().trim())
190             .toList();
191     }
192 }
193
194
195 final savedRecipe = await RecipeService.addRecipe(
196     name: recipeName,
197     description: 'AI-generated $selectedPalette recipe',
198     ingredients: ingredients,
199 );
200
201 ScaffoldMessenger.of(context).showSnackBar(
202     SnackBar(
203         content: Text('Saved ${savedRecipe['name']} to your recipes
204         !'),
205         backgroundColor: Colors.green,
206     ),
207 );
208
209 setState(() {
210     recommendedRecipes.removeWhere((r) =>
211         (r['recipe_name'] ?? r['name']) == recipeName);
212 });
213 } catch (e) {
214     ScaffoldMessenger.of(context).showSnackBar(

```

```

214     Snackbar(
215         content: Text('Failed to save recipe: ${e.toString()}'),
216         backgroundColor: Colors.red,
217     ),
218 );
219     debugPrint('Error saving recipe: $e');
220 }
221 }
222
223 void _viewRecipeDetails(Map<String, dynamic> recipe) {
224     final recipeName = recipe['recipe_name'] ?? recipe['name'] ?? '
    Unnamed Recipe';
225
226     List<Map<String, dynamic>> ingredientsList = [];
227
228     if (recipe['ingredients_list'] != null) {
229         if (recipe['ingredients_list'] is String) {
230             String ingredientsStr = recipe['ingredients_list'] as String;
231
232             ingredientsStr = ingredientsStr
233                 .replaceAll('[', ' ')
234                 .replaceAll(']', ' ')
235                 .replaceAll("'", "")
236                 .trim();
237
238             if (ingredientsStr.isNotEmpty) {
239                 ingredientsList = ingredientsStr
240                     .split(',')
241                     .map((ingredient) => {'name': ingredient.trim()})
242                     .toList();
243             }
244         }
245         else if (recipe['ingredients_list'] is List) {
246             ingredientsList = (recipe['ingredients_list'] as List)
247                 .map((ingredient) => {'name': ingredient.toString().trim
248                     ()})
249                 .toList();
250         }
251         else if (recipe['ingredients'] != null) {
252             if (recipe['ingredients'] is String) {

```



```

253     ingredientsList = (recipe['ingredients'] as String)
254         .split(',')
255         .map((ingredient) => {'name': ingredient.trim()})
256         .toList();
257   } else if (recipe['ingredients'] is List) {
258     ingredientsList = (recipe['ingredients'] as List).map((
259 ingredient) {
260       if (ingredient is Map) {
261         return {'name': ingredient['name']?.toString().trim() ??
262 'Unknown'};
263       } else {
264         return {'name': ingredient.toString().trim()};
265       }
266     }).toList();
267   }
268
269   Navigator.push(
270     context,
271     MaterialPageRoute(
272       builder: (context) => RecipeDetailPage(recipe: {
273         'name': recipeName,
274         'description': 'AI-generated $selectedPalette recipe with $
275 {recipe['palette'] ?? selectedPalette ?? 'various'} flavors',
276         'ingredients': ingredientsList,
277       }),
278     ),
279   );
280
281   @override
282   Widget build(BuildContext context) {
283     return Scaffold(
284       appBar: AppBar(title: const Text('AI Recipe Generator')),
285       body:
286         isLoading
287           ? const Center(child: CircularProgressIndicator())
288           : Padding(
289             padding: const EdgeInsets.all(16),
290             child: Column(

```

```

291         crossAxisAlignment: CrossAxisAlignment.start,
292         children: [
293             if (errorMessage != null)
294                 Padding(
295                     padding: const EdgeInsets.only(bottom: 16),
296                     child: Text(
297                         errorMessage!,
298                         style: const TextStyle(color: Colors.red)
299                     ),
300                 ),
301
302             const Text(
303                 'Select Flavor Palette:',
304                 style: TextStyle(
305                     fontSize: 18,
306                     fontWeight: FontWeight.bold,
307                 ),
308             ),
309             const SizedBox(height: 8),
310
311             Wrap(
312                 spacing: 8,
313                 runSpacing: 8,
314                 children:
315                     paletteList.map((palette) {
316                         return ChoiceChip(
317                             label: Text(palette.capitalize()),
318                             selected: selectedPalette == palette,
319                             onSelected: (selected) {
320                                 setState(() {
321                                     selectedPalette = selected ?
palette : null;
322                                     errorMessage = null;
323                                 });
324                             },
325                             selectedColor: Colors.blue.shade300,
326                             backgroundColor: Colors.grey.shade200
327                         ),
328                     labelStyle: TextStyle(
color:

```

```

329         selectedPalette == palette
330         ? Colors.white
331         : Colors.black,
332     ),
333 );
334     }).toList(),
335 ),
336
337     const SizedBox(height: 24),
338
339     Center(
340       child: ElevatedButton(
341         onPressed: isGenerating ? null :
_generateRecipes,
342         style: ElevatedButton.styleFrom(
343           padding: const EdgeInsets.symmetric(
344             horizontal: 32,
345             vertical: 16,
346           ),
347         ),
348         child:
349           isGenerating
350             ? const CircularProgressIndicator(
351               color: Colors.white,
352             )
353             : const Text('Generate Recipes'),
354       ),
355     ),
356
357     const SizedBox(height: 24),
358
359     if (recommendedRecipes.isNotEmpty)
360       const Text(
361         'Recommended Recipes:',
362         style: TextStyle(
363           fontSize: 18,
364           fontWeight: FontWeight.bold,
365         ),
366       ),
367
368     Expanded(

```

```

369         child: ListView.builder(
370             itemCount: recommendedRecipes.length,
371             itemBuilder: (context, index) {
372                 final recipe = recommendedRecipes[index];
373                 return Card(
374                     margin: const EdgeInsets.symmetric(
375                         vertical: 8),
376                     child: ListTile(
377                         title: Text(
378                             recipe['recipe_name'] ?? recipe['
379                             name'],
380                         ),
381                         subtitle: Text(
382                             recipe['matched_palette'] ??
383                             'Flavor: $selectedPalette',
384                         ),
385                         trailing: IconButton(
386                             icon: const Icon(Icons.save),
387                             onPressed: () => _saveRecipe(recipe
388                                 ),
389                             tooltip: 'Save Recipe',
390                         ),
391                         onTap: () => _viewRecipeDetails(
392                             recipe),
393                     ),
394                 );
395             },
396         ],
397     ),
398 );
399
400 extension StringExtension on String {
401     String capitalize() {
402         return "${this[0].toUpperCase()}${substring(1).toLowerCase()}";
403     }
404 }

```

LISTING 2.9: AI Recipe Generation Code

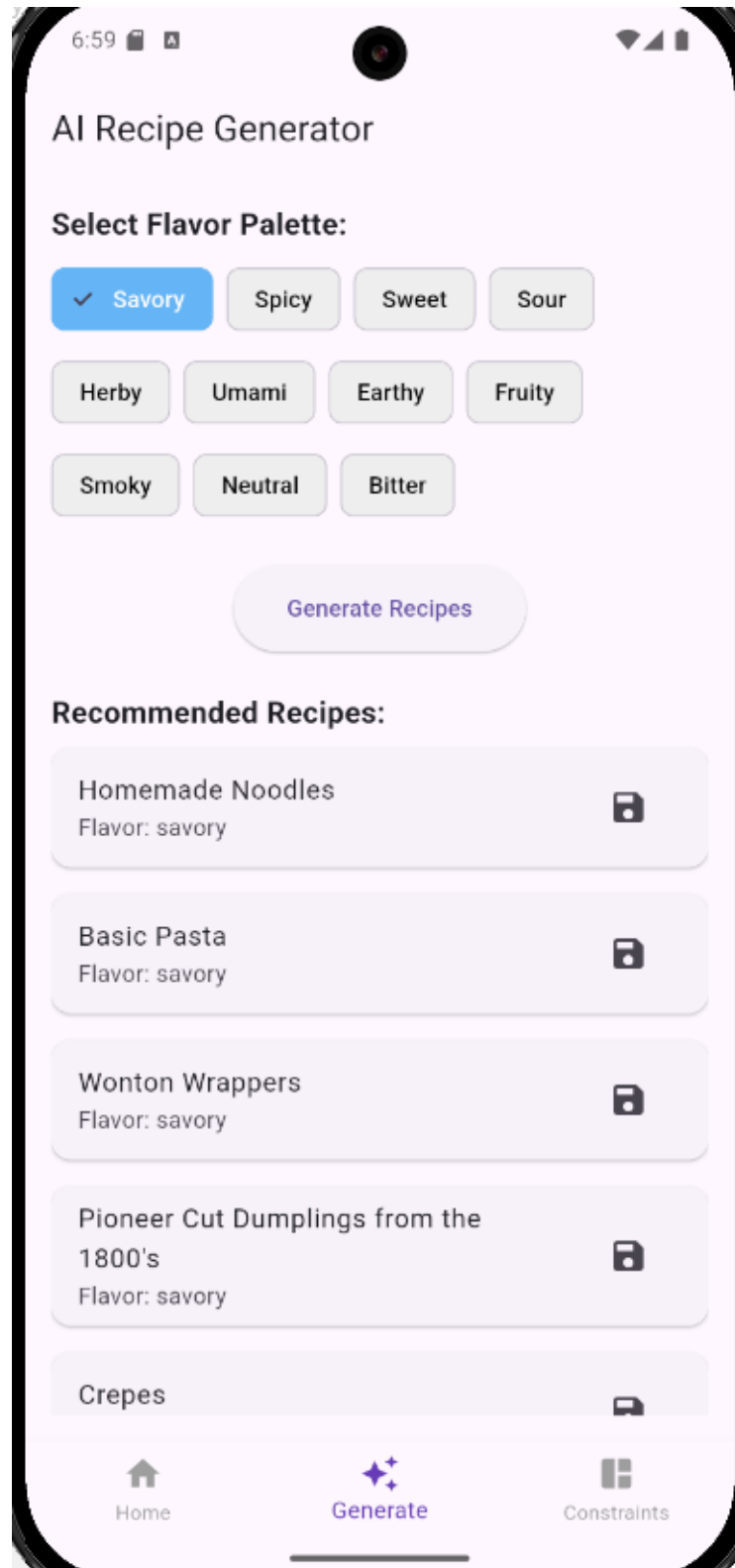


FIGURE 2.9: AI Recipe Generation Screen

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): <ul style="list-style-type: none"> • Returns to recipe list/generation screen 2. Save Button (implied): <ul style="list-style-type: none"> • Adds recipe to user's collection 3. Share Button (Clicked): <ul style="list-style-type: none"> • Shares recipe with others
Output Elements	1. Recipe Title: <ul style="list-style-type: none"> • Clear display of recipe name 2. Recipe Description: <ul style="list-style-type: none"> • Detailed preparation instructions 3. Ingredients List: <ul style="list-style-type: none"> • Bulleted list of required ingredients 4. AI Generation Badge: <ul style="list-style-type: none"> • Indicates recipe is AI-generated 5. Success Notifications: <ul style="list-style-type: none"> • For save/share actions

<p>Navigation Flow</p>	<ol style="list-style-type: none"> 1. User accesses from recipe generation or list screen. 2. System displays full recipe details. 3. User can: <ul style="list-style-type: none"> • Save recipe to their collection • Share recipe via other apps • Return to previous screen 4. Success Flow: <ul style="list-style-type: none"> • Saved recipes added to user collection • Shared recipes open share dialog 5. Alternative Flows: <ul style="list-style-type: none"> • Network issues prevent saving • Missing data shows error message
-------------------------------	---

Code Implementation

```

1  import 'package:flutter/material.dart';
2
3  // Stateless widget to show detailed view of a selected recipe
4  class RecipeDetailPage extends StatelessWidget {
5      // Recipe data is passed as a Map (contains keys like 'name', '
6      // description', 'ingredients')
7      final Map recipe;
8
9      // Constructor with required recipe data
10     const RecipeDetailPage({super.key, required this.recipe});
11
12     @override
13     Widget build(BuildContext context) {
14         return Scaffold(
15             // App bar displaying the recipe name
16             appBar: AppBar(
17                 title: Text(recipe['name']),
18             ),
19             body: Padding(
20                 padding: EdgeInsets.all(16),
21                 child: Column(
22                     crossAxisAlignment: CrossAxisAlignment.start, // Align
23                     content to the start (left)

```

```
22         children: [
23             // Section: Description header
24             Text(
25                 'Description',
26                 style: TextStyle(fontSize: 18, fontWeight: FontWeight
27                 .bold),
28             ),
29             SizedBox(height: 8),
30             // Section: Description content
31             Text(recipe['description']),
32             SizedBox(height: 16),
33             // Section: Ingredients header
34             Text(
35                 'Ingredients',
36                 style: TextStyle(fontSize: 18, fontWeight: FontWeight
37                 .bold),
38             ),
39             SizedBox(height: 8),
40             // Section: Ingredients content
41             Text(recipe['ingredients']),
42         ],
43     ),
44 ),
45 );
46 };
47 }
48 }
```

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	<ol style="list-style-type: none"> Nutritional Preference Selectors: <ul style="list-style-type: none"> Fat, Carbs, Protein, Cholesterol, Sodium, Fiber Ingredient Grid: <ul style="list-style-type: none"> Checkbox selection for available ingredients Supports up to 30 ingredients Home Button: <ul style="list-style-type: none"> Returns to main menu Generate Button: <ul style="list-style-type: none"> Creates recommendations with current settings Generate with Constraints Button: <ul style="list-style-type: none"> Opens advanced constraint dialog

Output Elements	<ol style="list-style-type: none"> 1. Nutritional Summary: <ul style="list-style-type: none"> • Visual indicators for selected nutrition profile 2. Selected Ingredients Display: <ul style="list-style-type: none"> • Shows count of selected ingredients 3. Recipe Recommendations: <ul style="list-style-type: none"> • Displays matching recipes after generation 4. Error Messages: <ul style="list-style-type: none"> • For invalid combinations
Navigation Flow	<ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> • Analyzes nutritional constraints • Matches with selected ingredients • Returns appropriate recipes 5. Alternative Flows: <ul style="list-style-type: none"> • No matches: Suggests ingredient adjustments • Advanced constraints: Opens detailed dialog 6. Home button returns to main menu.

Code Implementation

```

1 import 'package:flutter/material.dart';
2 import 'package:flutter_django_recipes_frontend/recipe_detail_page.
  dart';
3 import 'package:flutter_django_recipes_frontend/services/
  recipe_service.dart';
4 import 'package:flutter_django_recipes_frontend/services/
  auth_service.dart';
5 import 'package:http/http.dart' as http;
6 import 'dart:convert';
7 import 'ingredients_list.dart';
8
9 class ConstraintsRecipe extends StatefulWidget {
10   const ConstraintsRecipe({super.key});
11

```

```
12  @override
13  _ConstraintsRecipeState createState() => _ConstraintsRecipeState
14  ();
15  }
16  class _ConstraintsRecipeState extends State<ConstraintsRecipe> {
17    final TextEditingController caloriesController =
18      TextEditingController();
19    final TextEditingController fatController = TextEditingController
20      ();
21    final TextEditingController carbsController =
22      TextEditingController();
23    final TextEditingController proteinController =
24      TextEditingController();
25    final TextEditingController cholesterolController =
26      TextEditingController();
27    final TextEditingController sodiumController =
28      TextEditingController();
29    final TextEditingController fiberController =
30      TextEditingController();
31
32    List<String> selectedIngredients = [];
33    List<Map<String, dynamic>> recommendations = [];
34    bool isLoading = false;
35    String? errorMessage;
36
37    Future<void> getRecommendations() async {
38      if (selectedIngredients.isEmpty) {
39        setState(() => errorMessage = 'Please select at least one
40        ingredient');
41        return;
42      }
43
44      setState(() {
45        isLoading = true;
46        errorMessage = null;
47        recommendations = [];
48      });
49
50      try {
51        final headers = await AuthService.getAuthHeaders();
```

```
44     final response = await http.post(  
45         Uri.parse('${RecipeService.baseUrl}/recipes/  
recommend_calorie_based/'),  
46         headers: headers,  
47         body: jsonEncode(  
48             'calories': caloriesController.text.isEmpty ? 0 : double.  
parse(caloriesController.text),  
49             'fat': fatController.text.isEmpty ? 0 : double.parse(  
fatController.text),  
50             'carbs': carbsController.text.isEmpty ? 0 : double.parse(  
carbsController.text),  
51             'protein': proteinController.text.isEmpty ? 0 : double.  
parse(proteinController.text),  
52             'cholesterol': cholesterolController.text.isEmpty ? 0 :  
double.parse(cholesterolController.text),  
53             'sodium': sodiumController.text.isEmpty ? 0 : double.  
parse(sodiumController.text),  
54             'fiber': fiberController.text.isEmpty ? 0 : double.parse(  
fiberController.text),  
55             'ingredients': selectedIngredients,  
56         }),  
57     );  
58  
59     if (response.statusCode == 200) {  
60         final data = jsonDecode(response.body) as List;  
61         setState(() {  
62             recommendations = List<Map<String, dynamic>>.from(data);  
63         });  
64     } else {  
65         throw Exception('Failed to get recommendations: ${response.  
statusCode}');  
66     }  
67     } catch (e) {  
68         setState(() => errorMessage = e.toString());  
69     } finally {  
70         setState(() => isLoading = false);  
71     }  
72 }  
73  
74 Future<void> _saveRecipe(Map<String, dynamic> recipe) async {  
75     try {
```

```

76     final recipeName = recipe['recipe_name'] ?? recipe['name'] ??
    'Unnamed Recipe';
77
78     List<String> ingredients = [];
79     if (recipe['ingredients_list'] != null) {
80         if (recipe['ingredients_list'] is String) {
81             String ingredientsStr = recipe['ingredients_list'].
toString();
82             ingredientsStr = ingredientsStr
83                 .replaceAll('[', ' ')
84                 .replaceAll(']', ' ')
85                 .replaceAll('"', "");
86             ingredients = ingredientsStr.split(',').map((e) => e.trim
()).toList();
87         } else if (recipe['ingredients_list'] is List) {
88             ingredients = (recipe['ingredients_list'] as List)
89                 .map((e) => e.toString().trim())
90                 .toList();
91         }
92     }
93
94     final savedRecipe = await RecipeService.addRecipe(
95         name: recipeName,
96         description: 'Calorie-optimized recipe',
97         ingredients: ingredients,
98     );
99
100     ScaffoldMessenger.of(context).showSnackBar(
101         SnackBar(
102             content: Text('Saved ${savedRecipe['name']} to your
recipes!'),
103             backgroundColor: Colors.green,
104         ),
105     );
106 } catch (e) {
107     ScaffoldMessenger.of(context).showSnackBar(
108         SnackBar(
109             content: Text('Failed to save recipe: $e'),
110             backgroundColor: Colors.red,
111         ),
112     );

```

```

113     }
114 }
115
116 void _viewRecipeDetails(Map<String, dynamic> recipe) {
117
118
119     List<Map<String, String>> ingredientsList = [];
120
121     if (recipe['ingredients_list'] != null) {
122         if (recipe['ingredients_list'] is String) {
123             String ingredientsStr = recipe['ingredients_list'].toString
124             ();
125             ingredientsStr = ingredientsStr
126                 .replaceAll('[', ' ')
127                 .replaceAll(']', ' ')
128                 .replaceAll('"', "");
129             ingredientsList = ingredientsStr
130                 .split(',')
131                 .map((ingredient) => {'name': ingredient.trim()})
132                 .toList();
133         } else if (recipe['ingredients_list'] is List) {
134             ingredientsList = (recipe['ingredients_list'] as List)
135                 .map((ingredient) => {'name': ingredient.toString().
136                 trim()})
137                 .toList();
138         }
139     }
140
141     Widget buildIngredientSelection() {
142         final screenHeight = MediaQuery.of(context).size.height;
143
144         return Column(
145             crossAxisAlignment: CrossAxisAlignment.start,
146             children: [
147                 Center(
148                     child: const Text(
149                         'Ingredients:',
150                         style: TextStyle(fontWeight: FontWeight.bold),
151                 ),
152             ),
153             const SizedBox(height: 8),

```

```
152
153     Wrap(
154         spacing: 8,
155         runSpacing: 8,
156         children: selectedIngredients.map((ingredient) {
157             return Chip(
158                 label: Text(ingredient),
159                 onDelete: () => setState(() {
160                     selectedIngredients.remove(ingredient);
161                 }),
162                 deleteIcon: const Icon(Icons.close, size: 18),
163             );
164         }).toList(),
165     ),
166     const SizedBox(height: 8),
167
168     SizedBox(
169         height: screenHeight * 0.4,
170         child: Card(
171             elevation: 2,
172             child: Column(
173                 children: [
174                     Padding(
175                         padding: const EdgeInsets.all(8.0),
176                         child: Text(
177                             'Available Ingredients',
178                             style: Theme.of(context).textTheme.titleMedium,
179                         ),
180                     ),
181                     const Divider(height: 1),
182                     Expanded(
183                         child: ListView.builder(
184                             itemCount: availableIngredients.length,
185                             itemBuilder: (context, index) {
186                                 final ingredient = availableIngredients[index]
187
188                                 final isSelected = selectedIngredients.
189 contains(ingredient);
190
191                                 return CheckboxListTile(
192                                     title: Text(ingredient),
193                                     value: isSelected,
```



```
191         onChanged: (selected) {
192             setState(() {
193                 if (selected == true) {
194                     selectedIngredients.add(ingredient);
195                 } else {
196                     selectedIngredients.remove(ingredient
197                 );
198             }
199         });
200         controlAffinity: ListTileControlAffinity.
201         leading,
202     );
203     },
204     ),
205 ],
206 ),
207 ),
208 ),
209 ],
210 );
211 }
212
213 @override
214 Widget build(BuildContext context) {
215     return Scaffold(
216         appBar: AppBar(
217             title: const Text('Calorie-Based Recipes'),
218             elevation: 0,
219         ),
220         body: SafeArea(
221             child: ListView(
222                 padding: const EdgeInsets.all(16),
223                 children: [
224                     if (errorMessage != null)
225                         Container(
226                             padding: const EdgeInsets.all(12),
227                             margin: const EdgeInsets.only(bottom: 16),
228                             decoration: BoxDecoration(
229                                 color: Colors.red.shade50,
```

```

230         borderRadius: BorderRadius.circular(8),
231         border: Border.all(color: Colors.red.shade200),
232     ),
233     child: Row(
234         children: [
235             const Icon(Icons.error_outline, color: Colors.
red),
236
237             const SizedBox(width: 8),
238             Expanded(
239                 child: Text(
240                     errorMessage!,
241                     style: TextStyle(color: Colors.red.shade800
),
242                 ),
243             ),
244         ],
245     ),
246
247     Card(
248         elevation: 0,
249         margin: const EdgeInsets.only(bottom: 16),
250         shape: RoundedRectangleBorder(
251             borderRadius: BorderRadius.circular(12),
252         ),
253         child: Padding(
254             padding: const EdgeInsets.all(16),
255             child: Column(
256                 crossAxisAlignment: CrossAxisAlignment.start,
257                 children: [
258                     const Text(
259                         'Nutrition Constraints',
260                         style: TextStyle(fontSize: 18, fontWeight:
FontWeight.bold),
261                     ),
262                     const SizedBox(height: 12),
263                     buildTextField('Calories (kcal)',
caloriesController, TextInputType.number),
264                     buildTextField('Fat (g)', fatController,
TextInputType.number),

```

```

265         buildTextField('Carbs (g)', carbsController,
    TextInputType.number),
266         buildTextField('Protein (g)', proteinController
    , TextInputType.number),
267         buildTextField('Cholesterol (mg)',
    cholesterolController, TextInputType.number),
268         buildTextField('Sodium (mg)', sodiumController,
    TextInputType.number),
269         buildTextField('Fiber (g)', fiberController,
    TextInputType.number),
270     ],
271     ),
272   ),
273 ),
274
275   Card(
276     elevation: 0,
277     margin: const EdgeInsets.only(bottom: 16),
278     shape: RoundedRectangleBorder(
279       borderRadius: BorderRadius.circular(12),
280     ),
281     child: Padding(
282       padding: const EdgeInsets.all(16),
283       child: buildIngredientSelection(),
284     ),
285   ),
286
287   SizedBox(
288     width: double.infinity,
289     child: ElevatedButton(
290       style: ElevatedButton.styleFrom(
291         padding: const EdgeInsets.symmetric(vertical: 16)
292       ,
293         shape: RoundedRectangleBorder(
294           borderRadius: BorderRadius.circular(10),
295         ),
296       onPressed: isLoading ? null : getRecommendations,
297       child: isLoading
298         ? const SizedBox(
299           width: 20,

```

```

300         height: 20,
301         child: CircularProgressIndicator(
302             strokeWidth: 2,
303             color: Colors.white,
304         ),
305     ),
306     : const Text(
307         'Get Recommendations',
308         style: TextStyle(fontSize: 16),
309     ),
310 ),
311 ),
312
313     const SizedBox(height: 24),
314     const Text(
315         'Recommended Recipes',
316         style: TextStyle(fontSize: 18, fontWeight: FontWeight
317         .bold),
318     ),
319     const SizedBox(height: 12),
320     buildRecommendationList(),
321     const SizedBox(height: 16),
322 ],
323 ),
324 );
325 }
326
327 Widget buildTextField(String label, TextEditingController
328 controller, TextInputType keyboardType) {
329     return Padding(
330         padding: const EdgeInsets.only(bottom: 12),
331         child: TextField(
332             controller: controller,
333             keyboardType: keyboardType,
334             decoration: InputDecoration(
335                 contentPadding: const EdgeInsets.symmetric(horizontal:
336                 12, vertical: 14),
337                 border: OutlineInputBorder(
338                     borderRadius: BorderRadius.circular(8),
339                     borderSide: BorderSide(color: Colors.grey.shade300),

```

```
338         ),
339         enabledBorder: OutlineInputBorder(
340             borderRadius: BorderRadius.circular(8),
341             borderSide: BorderSide(color: Colors.grey.shade300),
342         ),
343         labelText: label,
344         floatingLabelBehavior: FloatingLabelBehavior.auto,
345     ),
346 ),
347 );
348 }
349
350 Widget buildRecommendationList() {
351     if (recommendations.isEmpty) {
352         return Container(
353             padding: const EdgeInsets.all(16),
354             decoration: BoxDecoration(
355                 color: Colors.grey.shade100,
356                 borderRadius: BorderRadius.circular(10),
357             ),
358             child: const Center(
359                 child: Text(
360                     'No recommendations yet. Adjust your constraints and
361                     try again.',
362                     style: TextStyle(color: Colors.grey),
363                 ),
364             );
365         }
366
367         return ListView.builder(
368             shrinkWrap: true,
369             physics: const NeverScrollableScrollPhysics(),
370             itemCount: recommendations.length,
371             itemBuilder: (context, index) {
372                 final recipe = recommendations[index];
373                 return Card(
374                     elevation: 2,
375                     margin: const EdgeInsets.symmetric(vertical: 8),
376                     shape: RoundedRectangleBorder(
377                         borderRadius: BorderRadius.circular(10),
```

```

378     ),
379     child: InkWell(
380       borderRadius: BorderRadius.circular(10),
381       onTap: () => _viewRecipeDetails(recipe),
382       child: Padding(
383         padding: const EdgeInsets.symmetric(vertical: 12,
horizontal: 5),
384         child: Column(
385           crossAxisAlignment: CrossAxisAlignment.start,
386           children: [
387             Row(
388               mainAxisAlignment: MainAxisAlignment.
spaceBetween,
389               children: [
390                 Text(
391                   recipe['recipe_name'],
392                   style: const TextStyle(
393                     fontSize: 16,
394                     fontWeight: FontWeight.bold,
395                   ),
396                 ),
397                 IconButton(
398                   icon: const Icon(Icons.save),
399                   onPressed: () => _saveRecipe(recipe),
400                   tooltip: 'Save Recipe',
401                 ),
402               ],
403             ),
404             const SizedBox(height: 8),
405             Text(
406               (recipe['ingredients_list'] is String
407                 ? recipe['ingredients_list'].toString()
408                   .replaceAll('[', ' ')
409                   .replaceAll(']', ' ')
410                   .replaceAll("'", '"')
411                 : (recipe['ingredients_list'] as List).join
(' ', ' ')),
412               style: TextStyle(color: Colors.grey.shade700),
413             ),
414           ],
415         ),

```

```
416         ),  
417     ),  
418 );  
419 },  
420 );  
421 }  
422 }
```

LISTING 2.11: Nutritional Recipe Recommendations Code

10:39

Recipe Recommendations

Fat

Carbohydrates

Protein

Cholesterol

Sodium

Fiber

Select Ingredients (up to 30)

Tomato Cheese Lettuce Onion

Chicken Beef Pasta Garlic

Potato Pepper Spinach Mushroom

Basil Parsley Carrot Peas

Broccoli Corn Rice Egg Milk

Home Generate Generate with Const...

FIGURE 2.11: Nutritional Recipe Recommendations Screen 1

10:39

Recipe Recommendations

Sodium

Fiber

Select Ingredients (up to 30)

Tomato Cheese Lettuce Onion

Chicken Beef Pasta Garlic

Potato Pepper Spinach Mushroom

Basil Parsley Carrot Peas

Broccoli Corn Rice Egg Milk

Yogurt Shrimp Fish Lamb

Turkey Cucumber Avocado Apple

Banana

Get Recommendations

Home Generate Generate with Const.

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 <ul style="list-style-type: none"> – Input for 5-digit verification code from email Verify Code Button <ul style="list-style-type: none"> – Submits the code for validation Resend Email Link <ul style="list-style-type: none"> – Re-sends verification code if not received Back Button (Clicked) <ul style="list-style-type: none"> – Returns to previous screen
Output Elements	Success Notification <ul style="list-style-type: none"> – Code verified, proceed to password reset screen Error Messages <ul style="list-style-type: none"> – Invalid code – Expired code – Too many attempts Resend Confirmation <ul style="list-style-type: none"> – New code sent notification
Navigation Flow	<ol style="list-style-type: none"> 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: <ul style="list-style-type: none"> – If code is invalid/expired, show error message – If "Resend email" clicked, send new code 6. Success Flow: <ul style="list-style-type: none"> – After successful verification, navigate to password reset screen 7. Failure Flow: <ul style="list-style-type: none"> – After multiple failures, may lock account temporarily

Code Implementation

```

1
2  """
3  Django settings for FlutterDjangoRecipes project.

```

```
4
5 Generated by 'django-admin startproject' using Django 5.1.6.
6
7 For more information on this file, see
8 https://docs.djangoproject.com/en/5.1/topics/settings/
9
10 For the full list of settings and their values, see
11 https://docs.djangoproject.com/en/5.1/ref/settings/
12 """
13
14 from pathlib import Path
15
16 # Build paths inside the project like this: BASE_DIR / 'subdir'.
17 BASE_DIR = Path(__file__).resolve().parent.parent
18
19 # Quick-start development settings - unsuitable for production
20 # See https://docs.djangoproject.com/en/5.1/howto/deployment/
21 # checklist/
22
23 # SECURITY WARNING: keep the secret key used in production secret!
24 SECRET_KEY = 'django-insecure-$3c8z4!qorf6f9n22o85ne)u&tyueek&
25 kvxsip)nlklm7tg*d0'
26
27 # SECURITY WARNING: don't run with debug turned on in production!
28 DEBUG = True
29
30 CORS_ALLOW_CREDENTIALS = True
31 CSRF_COOKIE_HTTPONLY = False
32 CSRF_COOKIE_SAMESITE = 'Lax'
33 ALLOWED_HOSTS = ["localhost", "127.0.0.1", '10.0.2.2']
34
35 CORS_ALLOWED_ORIGINS = [
36     "http://localhost:61797", # Default for local React/Flutter
37     apps running on port 3000
38     "http://127.0.0.1:61797", # Equivalent localhost with
39     127.0.0.1
40     "http://10.0.2.2:61797", # For Android emulators to access
41     your local Django backend
42 ]
43
44 EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
```

```
40 EMAIL_HOST = 'smtp.gmail.com'
41 EMAIL_PORT = 587
42 EMAIL_USE_TLS = True
43 EMAIL_HOST_USER = 'khawajafahadzia1@gmail.com'
44 EMAIL_HOST_PASSWORD = 'vamx jlpw bren opfs'
45 DEFAULT_FROM_EMAIL = 'khawajafahadzia1@gmail.com'
46
47 #CORS_ALLOW_ALL_ORIGINS = True
48 # Application definition
49
50 INSTALLED_APPS = [
51     'django.contrib.admin',
52     'django.contrib.auth',
53     'django.contrib.contenttypes',
54     'django.contrib.sessions',
55     'django.contrib.messages',
56     'django.contrib.staticfiles',
57     'accounts',
58     'recipes',
59     'rest_framework',
60     'corsheaders',
61 ]
62
63 REST_FRAMEWORK = {
64     'DEFAULT_AUTHENTICATION_CLASSES': [
65         'rest_framework.authentication.SessionAuthentication',
66     ],
67     'DEFAULT_PERMISSION_CLASSES': [
68         'rest_framework.permissions.IsAuthenticated',
69     ],
70 }
71
72 MIDDLEWARE = [
73     'django.middleware.security.SecurityMiddleware',
74     'django.contrib.sessions.middleware.SessionMiddleware',
75     'django.middleware.common.CommonMiddleware',
76     'django.middleware.csrf.CsrfViewMiddleware',
77     'django.contrib.auth.middleware.AuthenticationMiddleware',
78     'django.contrib.messages.middleware.MessageMiddleware',
79     'django.middleware.clickjacking.XFrameOptionsMiddleware',
80     'corsheaders.middleware.CorsMiddleware',
```

```
81
82 ]
83
84 ROOT_URLCONF = 'FlutterDjangoRecipes.urls'
85
86 TEMPLATES = [
87     {
88         'BACKEND': 'django.template.backends.django.DjangoTemplates
89     },
90     'DIRS': ['templates'],
91     'APP_DIRS': True,
92     'OPTIONS': {
93         'context_processors': [
94             'django.template.context_processors.debug',
95             'django.template.context_processors.request',
96             'django.contrib.auth.context_processors.auth',
97             'django.contrib.messages.context_processors.
98         messages',
99     ],
100     },
101 ]
102
103 WSGI_APPLICATION = 'FlutterDjangoRecipes.wsgi.application'
104
105 # Database
106 # https://docs.djangoproject.com/en/5.1/ref/settings/#databases
107
108 DATABASES = {
109     'default': {
110         'ENGINE': 'django.db.backends.sqlite3',
111         'NAME': BASE_DIR / 'db.sqlite3',
112     }
113 }
114
115 # Password validation
116 # https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password
117 # -validators
118
119 AUTH_PASSWORD_VALIDATORS = [
120     {
```

```

119         'NAME': 'django.contrib.auth.password_validation.
UserAttributeSimilarityValidator',
120     },
121     {
122         'NAME': 'django.contrib.auth.password_validation.
MinimumLengthValidator',
123     },
124     {
125         'NAME': 'django.contrib.auth.password_validation.
CommonPasswordValidator',
126     },
127     {
128         'NAME': 'django.contrib.auth.password_validation.
NumericPasswordValidator',
129     },
130 ]
131 # Internationalization
132 # https://docs.djangoproject.com/en/5.1/topics/i18n/
133
134 LANGUAGE_CODE = 'en-us'
135 TIME_ZONE = 'UTC'
136
137 USE_I18N = True
138
139 USE_TZ = True
140 # Static files (CSS, JavaScript, Images)
141 # https://docs.djangoproject.com/en/5.1/howto/static-files/
142 STATIC_URL = 'static/'
143 MEDIA_URL = 'media/'
144 STATIC_ROOT = BASE_DIR / 'assets'
145 MEDIA_ROOT = BASE_DIR / 'media'
146
147 STATICFILES_DIRS = [
148     BASE_DIR / 'static'
149 ]
150 # Default primary key field type
151 # https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-
field
152 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

```

1  from django.contrib import admin
2  from django.urls import path, include#, re_path
3  urlpatterns = [
4      path('admin/', admin.site.urls),
5      path('', include("accounts.urls")),
6      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 <code>accounts</code> app, mapping specific endpoints to view functions for user authentication and account management features.
URL Patterns	<p><code>''</code> – <code>views.loginPage</code>: Loads the login page.</p> <p><code>gettingcsrftoken/</code> – <code>views.getCsrfToken</code>: Retrieves CSRF token for frontend security.</p> <p><code>register/</code> – <code>views.registerPage</code>: Loads the registration page.</p> <p><code>resetPassword/</code> – <code>views.resetPassword</code>: Initiates password reset process.</p> <p><code>verifyOTP/</code> – <code>views.verifyOTP</code>: Verifies OTP during account actions.</p> <p><code>updatePassword/</code> – <code>views.updatePassword</code>: Updates user password after verification.</p>
App Namespace	<code>accounts</code> – Enables namespacing of URLs to avoid collisions across Django apps.

Code Implementation

```

1 from django.urls import path
2 from . import views
3 app_name = 'accounts'
4 urlpatterns = [
5     path('', views.loginPage, name='login-page'),
6     path('gettingcsrftoken/', views.getCsrfToken, name='get-csrf-
    token'), path('register/', views.registerPage, name='register-
    page'), path('resetPassword/', views.resetPassword, name='reset-
    password'),
7     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	<p><code>email</code> – <code>EmailField(unique=True)</code>: Stores the user's email, must be unique.</p> <p><code>otp</code> – <code>CharField(max_length=6)</code>: Stores a 6-digit OTP.</p> <p><code>created_at</code> – <code>DateTimeField(auto_now_add=True)</code>: Stores the time when the OTP was generated.</p>
Method	<code>is_expired(self)</code> – Checks whether the OTP is expired by comparing current time with <code>created_at + 60 seconds</code> .

Code Implementation

```

1 from django.db import models
2 from django.utils import timezone
3 class OTP(models.Model):
4     email = models.EmailField(unique=True)
5     otp = models.CharField(max_length=6)
6     created_at = models.DateTimeField(auto_now_add=True)
7     def is_expired(self):
8         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 <code>@permission_classes([AllowAny])</code> , allowing unauthenticated access.
CSRF Protection	<code>@ensure_csrf_cookie</code> for GET request, <code>@csrf_protect</code> for all POST requests to prevent cross-site request forgery.
Email Functionality	Uses <code>send_mail()</code> from Django to email OTPs during password reset process.

Code Implementation

```

1 from django.contrib.auth import get_user_model, authenticate, login
   , logout
2 from django.http import JsonResponse
3 from django.middleware.csrf import get_token
4 from rest_framework.decorators import api_view, permission_classes
5 from rest_framework.response import Response
6 from django.views.decorators.csrf import ensure_csrf_cookie,
   csrf_protect
7 import random
8 from .models import OTP
9 from django.core.mail import send_mail
10 from django.conf import settings
11 from django.utils import timezone
12 from rest_framework.permissions import AllowAny
13
14 User = get_user_model()
15
16 @api_view(['GET'])
17 @ensure_csrf_cookie
18 @permission_classes([AllowAny])
19 def getCsrftoken(request):
20     """Endpoint to get CSRF token for frontend"""
21     return JsonResponse({'csrfToken': get_token(request)})
22
23 @api_view(['POST'])
24 @csrf_protect
25 @permission_classes([AllowAny])
26 def loginPage(request):

```

```
27     """Handle user login with email and password"""
28     logout(request)
29     username = request.data.get('email')
30     password = request.data.get('password')
31
32     user = authenticate(request, username=username, password=
password)
33
34     if user is not None:
35         login(request, user)
36         return Response({
37             "success": True,
38             "message": "Login successful",
39             "user": {
40                 "id": user.id,
41                 "email": user.username,
42                 "first_name": user.first_name,
43                 "last_name": user.last_name
44             }
45         })
46     return Response({"success": False, "message": "Invalid
credentials"})
47
48 @api_view(['POST'])
49 @csrf_protect
50 @permission_classes([AllowAny])
51 def registerPage(request):
52     """Handle new user registration"""
53     logout(request)
54     username = request.data.get('email')
55     password = request.data.get('password')
56     first_name = request.data.get('first_name')
57     last_name = request.data.get('last_name')
58
59     if User.objects.filter(username=username).exists():
60         return Response({'success': False, 'message': 'Email
already exists'})
61
62     user = User.objects.create_user(
63         username=username,
64         password=password,
```

```
65         first_name=first_name,
66         last_name=last_name
67     )
68     login(request, user)
69     return Response({
70         'success': True,
71         'message': 'Registration successful',
72         'user': {
73             'id': user.id,
74             'email': user.username,
75             'first_name': user.first_name,
76             'last_name': user.last_name
77         }
78     })
79
80 @api_view(['POST'])
81 @csrf_protect
82 @permission_classes([AllowAny])
83 def resetPassword(request):
84     """Initiate password reset process"""
85     email = request.data.get('email')
86     if not User.objects.filter(email=email).exists():
87         return Response({'success': False})
88
89     otp = str(random.randint(10000, 99999))
90     OTP.objects.update_or_create(
91         email=email,
92         defaults={'otp': otp, 'created_at': timezone.now()}
93     )
94     send_mail(
95         "Password Reset OTP",
96         f"Your OTP is {otp}. It will expire in 60 seconds.",
97         settings.EMAIL_HOST_USER,
98         [email]
99     )
100     return Response({'success': True})
101
102 @api_view(['POST'])
103 @csrf_protect
104 @permission_classes([AllowAny])
105 def verifyOTP(request):
```

```
106     """Verify OTP for password reset"""
107     email = request.data.get('email')
108     otp_entered = request.data.get('otp')
109
110     try:
111         otp_record = OTP.objects.get(email=email)
112         if otp_record.is_expired():
113             return Response({'success': False, 'message': 'OTP
114 expired'}, status=400)
115
116         if otp_record.otp == otp_entered:
117             return Response({'success': True, 'message': 'OTP
118 verified'})
119         return Response({'success': False, 'message': 'Invalid OTP'
120 }, status=400)
121     except OTP.DoesNotExist:
122         return Response({'success': False, 'message': 'OTP not
123 found'}, status=400)
124
125 @api_view(['POST'])
126 @csrf_protect
127 @permission_classes([AllowAny])
128 def updatePassword(request):
129     """Update user password after OTP verification"""
130     email = request.data.get('email')
131     new_password = request.data.get('new_password')
132
133     try:
134         user = User.objects.get(username=email)
135         user.set_password(new_password)
136         user.save()
137         return Response({'success': True, 'message': 'Password
138 updated'})
139     except User.DoesNotExist:
140         return Response({'success': False, 'message': 'User not
141 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	<pre>'' – 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.</pre>
App Namespace	recipes – Used for namespacing URLs to prevent naming conflicts with other Django apps.

Code Implementation

```

1 from django.urls import path
2 from . import views
3
4 app_name = 'recipes'
5
6 urlpatterns = [
7     # GET: List all recipes | POST: Create new recipe

```



```
8     path('', views.recipe_list, name='recipe-list'),
9
10    # GET: Get single recipe | PUT: Update recipe | DELETE: Delete
    recipe
11    path('<int:pk>', views.recipe_detail, name='recipe-detail'),
12
13    path('recommend/', views.recommend_recipes, name='recipe-
    recommend'),
14    path('recommend_calorie_based/', views.recommend_calorie_based,
        name='calorie-recommend'),
15 ]
```

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 (<code>api_view</code> , <code>permission_classes</code> , <code>Response</code> , <code>IsAuthenticated</code> , <code>status</code>), Django shortcuts (<code>get_object_or_404</code>), models (<code>Recipes</code> , <code>RecipeIngredients</code>), CSRF protection decorator, and external recommendation modules (<code>recipe_recommender</code> , <code>calorie_model</code>).
Function: <code>recipe_list</code>	Handles GET and POST requests: <ul style="list-style-type: none"> - 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: <code>recipe_detail</code>	Handles GET, PUT, and DELETE requests for a recipe specified by primary key: <ul style="list-style-type: none"> - GET: Returns details of the specific recipe with ingredients. - PUT: Updates recipe's name, description, and ingredients. - DELETE: Deletes the recipe. Uses <code>get_object_or_404</code> to ensure recipe belongs to user. Protected by authentication and CSRF.
Function: <code>recommend_recipes</code>	Handles POST requests to provide recipe recommendations based on input ingredients and palette: <ul style="list-style-type: none"> - Validates inputs are non-empty lists. - Calls external <code>recipe_recommender.recommend_recipes</code> function. - Returns list of recommended recipes or error on failure. Protected by authentication and CSRF.

Function: <code>recommend_calorie_based</code>	<p>Handles POST requests to provide calorie-based recipe recommendations:</p> <ul style="list-style-type: none"> - Extracts numeric nutrition features and ingredients list from request data. - Calls external <code>calorie_model.recommend_recipes</code> function. - Returns recommendations or error on failure. <p>Protected by authentication and CSRF.</p>
--	---

Code Implementation

```

1 from rest_framework.decorators import api_view, permission_classes
2 from rest_framework.response import Response
3 from rest_framework.permissions import IsAuthenticated
4 from rest_framework import status
5 from django.shortcuts import get_object_or_404
6 from .models import Recipes, RecipeIngredients
7 from django.views.decorators.csrf import csrf_protect
8 from .palatte_based import recipe_recommender
9 from .calorie_based import app as calorie_model
10
11 @api_view(['GET', 'POST'])
12 @permission_classes([IsAuthenticated])
13 @csrf_protect
14 def recipe_list(request):
15     """List all recipes or create new recipe"""
16     if request.method == 'GET':
17         recipes = Recipes.objects.filter(user=request.user)
18         data = [{
19             'id': recipe.id,
20             'name': recipe.name,
21             'description': recipe.description,
22             'ingredients': [
23                 {'id': i.id, 'name': i.name}
24                 for i in recipe.recipeingredients_set.all()
25             ]
26         } for recipe in recipes]
27     return Response(data)
28

```

```
29     elif request.method == 'POST':
30         recipe = Recipes.objects.create(
31             user=request.user,
32             name=request.data.get('name'),
33             description=request.data.get('description')
34         )
35
36         for ingredient in request.data.get('ingredients', []):
37             RecipeIngredients.objects.create(
38                 recipe=recipe,
39                 name=ingredient
40             )
41
42         return Response({
43             'id': recipe.id,
44             'name': recipe.name,
45             'description': recipe.description,
46             'ingredients': [
47                 {'id': i.id, 'name': i.name}
48                 for i in recipe.recipeingredients_set.all()
49             ]
50         }, status=201)
51
52 @api_view(['GET', 'PUT', 'DELETE'])
53 @permission_classes([IsAuthenticated])
54 @csrf_protect
55 def recipe_detail(request, pk):
56     """Get, update or delete specific recipe"""
57     recipe = get_object_or_404(Recipes, pk=pk, user=request.user)
58
59     if request.method == 'GET':
60         return Response({
61             'id': recipe.id,
62             'name': recipe.name,
63             'description': recipe.description,
64             'ingredients': [
65                 {'id': i.id, 'name': i.name}
66                 for i in recipe.recipeingredients_set.all()
67             ]
68         })
69
```

```
70     elif request.method == 'PUT':
71         recipe.name = request.data.get('name', recipe.name)
72         recipe.description = request.data.get('description', recipe
73         .description)
74         recipe.save()
75
76         recipe.recipeingredients_set.all().delete()
77         for ingredient in request.data.get('ingredients', []):
78             RecipeIngredients.objects.create(
79                 recipe=recipe,
80                 name=ingredient
81             )
82
83         return Response({
84             'id': recipe.id,
85             'name': recipe.name,
86             'description': recipe.description,
87             'ingredients': [
88                 {'id': i.id, 'name': i.name}
89                 for i in recipe.recipeingredients_set.all()
90             ]
91         })
92
93     elif request.method == 'DELETE':
94         recipe.delete()
95         return Response(status=204)
96
97 @api_view(['POST'])
98 @permission_classes([IsAuthenticated])
99 @csrf_protect
100 def recommend_recipes(request):
101     """Handle recipe recommendations"""
102     try:
103
104
105         data = request.data
106         ingredients = data.get('ingredients', [])
107         palette = data.get('palette', [])
108
109
```

```

110         if not isinstance(ingredients, list) or not isinstance(
palette, list):
111             return Response(
112                 {'error': 'Ingredients and palette must be lists'},
113                 status=status.HTTP_400_BAD_REQUEST
114             )
115
116         if not ingredients or not palette:
117             return Response(
118                 {'error': 'Both ingredients and palette are
required'},
119                 status=status.HTTP_400_BAD_REQUEST
120             )
121
122
123         ingredients_str = ', '.join([str(i) for i in ingredients])
124         palette_str = ', '.join([str(p) for p in palette])
125
126
127         recommendations = recipe_recommender.recommend_recipes(
128             input_ingredients_str=ingredients_str,
129             input_palette_str=palette_str,
130             top_k=5
131         )
132
133
134         results = recommendations.to_dict('records')
135
136         return Response(results)
137
138     except Exception as e:
139         return Response(
140             {'error': f'Recommendation failed: {str(e)}'},
141             status=status.HTTP_500_INTERNAL_SERVER_ERROR
142         )
143
144 @api_view(['POST'])
145 @permission_classes([IsAuthenticated])
146 @csrf_protect
147 def recommend_calorie_based(request):
148     """Get calorie-based recipe recommendations"""
149     try:

```

```
149     data = request.data
150     input_features = [
151         float(data.get('calories', 0)),
152         float(data.get('fat', 0)),
153         float(data.get('carbs', 0)),
154         float(data.get('protein', 0)),
155         float(data.get('cholesterol', 0)),
156         float(data.get('sodium', 0)),
157         float(data.get('fiber', 0)),
158         ','.join(data.get('ingredients', [])),]
159     recommendations = calorie_model.recommend_recipes(
input_features)
160     results = recommendations.to_dict('records')
161     return Response(results)
162 except Exception as e:
163     return Response(
164         {'error': str(e)},
165         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

```

1 from django.db import models
2 from django.contrib.auth.models import User
3
4 class Recipes(models.Model):
5     id = models.BigAutoField(db_column="id", primary_key=True)
6     name = models.CharField(max_length=300, blank=True)
7     description = models.TextField(blank=True)
8     user = models.ForeignKey(User, on_delete=models.CASCADE)
9
10 class RecipeIngredients(models.Model):
11     id = models.BigAutoField(db_column="id", primary_key=True)
12     name = models.CharField(max_length=300, blank=True)
13     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	<ul style="list-style-type: none"> • getRecipes(): <ul style="list-style-type: none"> - Makes GET request to '/recipes/' endpoint - Returns list of recipes if status code is 200 - Throws exception on failure • addRecipe({required params}): <ul style="list-style-type: none"> - 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}): <ul style="list-style-type: none"> - 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): <ul style="list-style-type: none"> - Makes DELETE request to '/recipes/{id}/' - Expects status code 204 for success - Throws exception on failure

Common Patterns	<ul style="list-style-type: none"> - All methods use <code>http.Client()</code> with try-finally to ensure cleanup - All requests include auth headers from <code>AuthService.getAuthHeaders()</code> - All methods throw exceptions for non-success status codes - Request bodies are JSON encoded with <code>jsonEncode()</code>
------------------------	--

Code Implementation

```

1 import 'dart:convert';
2 import 'package:http/http.dart' as http;
3 import 'auth_service.dart';
4
5 class RecipeService {
6   static const String baseUrl = 'http://10.0.2.2:9091';
7
8   static Future<List<dynamic>> getRecipes() async {
9     final client = http.Client();
10    try {
11      final headers = await AuthService.getAuthHeaders();
12      final response = await client.get(
13        Uri.parse('$baseUrl/recipes/'),
14        headers: headers,
15      );
16
17      if (response.statusCode == 200) {
18        return jsonDecode(response.body);
19      } else {
20        throw Exception('Failed to load recipes (${response.
21          statusCode})');
22      }
23    } finally {
24      client.close();
25    }
26
27    static Future<Map<String, dynamic>> addRecipe({
28      required String name,

```

```
29     required String description,
30     required List<String> ingredients,
31 }) async {
32     final client = http.Client();
33     try {
34         final headers = await AuthService.getAuthHeaders();
35         final response = await client.post(
36             Uri.parse('$baseUrl/recipes/'),
37             headers: headers,
38             body: jsonEncode({
39                 'name': name,
40                 'description': description,
41                 'ingredients': ingredients,
42             })),
43         );
44
45         if (response.statusCode == 201) {
46             return jsonDecode(response.body);
47         } else {
48             throw Exception('Failed to add recipe (${response.
49                 statusCode})');
50         }
51     } finally {
52         client.close();
53     }
54
55     static Future<Map<String, dynamic>> updateRecipe({
56         required int id,
57         required String name,
58         required String description,
59         required List<String> ingredients,
60     }) async {
61         final client = http.Client();
62         try {
63             final headers = await AuthService.getAuthHeaders();
64             final response = await client.put(
65                 Uri.parse('$baseUrl/recipes/$id/'),
66                 headers: headers,
67                 body: jsonEncode({
68                     'name': name,
```

```
69         'description': description,
70         'ingredients': ingredients,
71     }},
72 );
73
74     if (response.statusCode == 200) {
75         return jsonDecode(response.body);
76     } else {
77         throw Exception('Failed to update recipe (${response.
78             statusCode})');
79     }
80     } finally {
81         client.close();
82     }
83 }
84
85 static Future<void> deleteRecipe(int id) async {
86     final client = http.Client();
87     try {
88         final headers = await AuthService.getAuthHeaders();
89         final response = await client.delete(
90             Uri.parse('$baseUrl/recipes/$id/'),
91             headers: headers,
92         );
93
94         if (response.statusCode != 204) {
95             throw Exception('Failed to delete recipe (${response.
96                 statusCode})');
97         }
98     } finally {
99         client.close();
100     }
101 }
```

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	<ul style="list-style-type: none"> • <code>baseUrl</code>: String constant set to <code>'http://10.0.2.2:9091'</code> • <code>sessionCookie</code>: Static nullable String for session storage • <code>csrfToken</code>: Static nullable String for CSRF token storage
Helper Methods	<ul style="list-style-type: none"> • <code>_extractCookies(headers)</code>: <ul style="list-style-type: none"> - Parses Set-Cookie headers to extract sessionid and csrftoken - Returns map with extracted cookies

Main Methods	<ul style="list-style-type: none">• getCsrftoken():<ul style="list-style-type: none">- Makes GET request to <code>'/gettingcsrftoken/'</code>- Extracts and stores CSRF token and session cookie- Returns token and cookie string• login({required email, password}):<ul style="list-style-type: none">- First obtains CSRF token- Makes POST request with credentials- Returns success status and message- Updates session cookie on success• register({required params}):<ul style="list-style-type: none">- 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):<ul style="list-style-type: none">- Makes POST request to reset password endpoint- Returns success status or null on failure• getAuthHeaders():<ul style="list-style-type: none">- Ensures CSRF token and session cookie exist- Returns headers with auth tokens for authenticated requests
---------------------	--

Common Patterns	<ul style="list-style-type: none"> • Uses <code>http.Client()</code> 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 <code>jsonEncode()</code> for request bodies
Security Features	<ul style="list-style-type: none"> • CSRF token handling for all mutating requests • Session cookie management Secure credential transmission • Token refresh mechanism

Code Implementation

```

1 import 'dart:convert';
2 import 'package:http/http.dart' as http;
3
4 class AuthService {
5   static const String baseUrl = 'http://10.0.2.2:9091';
6   static String? sessionCookie;
7   static String? csrfToken;
8
9   static Map<String, String> _extractCookies(Map<String, String>
10     headers) {
11     final setCookie = headers['set-cookie'] ?? '';
12     final cookies = setCookie.split(',');
13
14     String? session;
15     String? csrf;

```

```
15
16     for (var cookie in cookies) {
17         if (cookie.contains('sessionid=')) {
18             session = cookie.split(';')[0].trim().split('=').last;
19         } else if (cookie.contains('csrftoken=')) {
20             csrf = cookie.split(';')[0].trim().split('=').last;
21         }
22     }
23
24     return {
25         if (session != null) 'sessionid': session,
26         if (csrf != null) 'csrftoken': csrf,
27     };
28 }
29
30 static Future<Map<String, dynamic>?> getCsrftoken() async {
31     final client = http.Client();
32     try {
33         final response = await client.get(
34             Uri.parse('$baseUrl/gettingcsrftoken/'),
35             headers: {'Content-Type': 'application/json'},
36         );
37
38         if (response.statusCode == 200) {
39             final cookies = _extractCookies(response.headers);
40             csrfToken = cookies['csrftoken'];
41             sessionCookie = cookies['sessionid'];
42             return {
43                 'csrfToken': csrfToken,
44                 'csrfCookie': 'csrftoken=$csrfToken; sessionid=
45 $sessionCookie',
46             };
47         }
48         return null;
49     } finally {
50         client.close();
51     }
52 }
53
54 static Future<Map<String, dynamic>> login({
55     required String email,
```



```
55     required String password,
56   }) async {
57     final csrfData = await getCsrftoken();
58     if (csrfData == null) {
59       return {'success': false, 'message': 'Failed to get CSRF
60       token'};
61     }
62
63     final client = http.Client();
64     try {
65       final response = await client.post(
66         Uri.parse('$baseUrl/'),
67         headers: {
68           'Content-Type': 'application/json',
69           'X-CSRFToken': csrfData['csrfToken']!,
70           'Cookie': csrfData['csrfCookie']!,
71         },
72         body: jsonEncode({'email': email, 'password': password}),
73       );
74
75       if (response.statusCode == 200) {
76         if (jsonDecode(response.body)['success']) {
77           final cookies = _extractCookies(response.headers);
78           sessionCookie = cookies['sessionid'] ?? sessionCookie;
79           return {'success': true, 'message': 'Login successful'};
80         }
81         return {'success': false, 'message': 'Invalid credentials'};
82       } else {
83         return {'success': false, 'message': 'Invalid credentials'};
84       }
85     } finally {
86       client.close();
87     }
88
89     static Future<Map<String, dynamic>> register({
90       required String email,
91       required String password,
92       required String firstName,
```

```
93     required String lastName,
94   }) async {
95     final csrfData = await getCsrftoken();
96     if (csrfData == null) {
97       return {'success': false, 'message': 'Failed to get CSRF
98       token'};
99     }
100
101     final client = http.Client();
102     try {
103       final response = await client.post(
104         Uri.parse('$baseUrl/register/'),
105         headers: {
106           'Content-Type': 'application/json',
107           'X-CSRFToken': csrfData['csrfToken']!,
108           'Cookie': csrfData['csrfCookie'] ?? '',
109         },
110         body: jsonEncode({
111           'email': email,
112           'password': password,
113           'first_name': firstName,
114           'last_name': lastName,
115         })),
116       );
117
118       if (response.statusCode == 200) {
119         if (jsonDecode(response.body)['success']){
120           final cookies = _extractCookies(response.headers);
121           sessionCookie = cookies['sessionid'] ?? sessionCookie;
122           csrfToken = cookies['csrftoken'] ?? csrfToken;
123           return {'success': true, 'message': 'Registration
124           successful'};
125         }
126         return {'success': false, 'message': 'Registration failed'
127         };
128       } else {
129         return {'success': false, 'message': 'Registration failed'
130         };
131       }
132     } finally {
133       client.close();
134     }
135   }
136 }
```

```

130     }
131   }
132
133   static Future<bool?> sendResetPasswordEmail(String email) async {
134     final client = http.Client();
135     final csrfData = await getCsrfToken();
136     if (csrfData == null) {
137       print("Failed to get CSRF token");
138       return null;
139     }
140     try {
141       final response = await client.post(
142         Uri.parse('$baseUrl/resetPassword/'),
143         headers: {
144           'Content-Type': 'application/json',
145           'X-CSRFToken': csrfData['csrfToken']!,
146           'Cookie': '${csrfData['csrfCookie']}; ${sessionCookie ??
147             ''}',
148           },
149         body: jsonEncode({'email': email}),
150       );
151       if (response.statusCode == 200) {
152         return jsonDecode(response.body)['success'] as bool?;
153       } else {
154         print('Failed with status: ${response.statusCode}');
155         return null;
156       }
157     } finally {
158       client.close();
159     }
160   }
161
162   static Future<Map<String, String>> getAuthHeaders() async {
163     if (csrfToken == null || sessionCookie == null) {
164       await getCsrfToken();
165     }
166     return {
167       'Content-Type': 'application/json',
168       'X-CSRFToken': csrfToken ?? '',
169       'Cookie': 'csrftoken=$csrfToken; sessionid=$sessionCookie',
170     };
171   }

```

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	<ul style="list-style-type: none"> • Requires a recipe parameter (Map type) • Optional key parameter passed to super
Build Method	<p>Returns a Scaffold widget containing:</p> <ul style="list-style-type: none"> • An AppBar with recipe name as title • A Padding widget with 16px padding • A Column with cross-axis start alignment containing: <ul style="list-style-type: none"> – "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	<ul style="list-style-type: none">• Simple vertical layout with clear section headings• Consistent spacing between elements• Basic typography with bold headings• Displays recipe name, description, and ingredients
Data Handling	<ul style="list-style-type: none">• Expects recipe data as a Map with:<ul style="list-style-type: none">– <code>name</code> (String) - for app bar title– <code>description</code> (String) - main content– <code>ingredients</code> (List<Map>) - with <code>name</code> fields• Transforms ingredients list to comma-separated string
Key Features	<ul style="list-style-type: none">• Stateless widget (immutable)• Simple recipe detail display• Responsive layout with padding• Clear visual hierarchy

Code Implementation

```
1 import 'package:flutter/material.dart';
2
3 class RecipeDetailPage extends StatelessWidget {
4   final Map recipe;
5
6   const RecipeDetailPage({super.key, required this.recipe});
7
8   @override
```

```
9 Widget build(BuildContext context) {
10   return Scaffold(
11     appBar: AppBar(title: Text(recipe['name'])),
12     body: Padding(
13       padding: EdgeInsets.all(16),
14       child: Column(
15         crossAxisAlignment: CrossAxisAlignment.start,
16         children: [
17           Text(
18             'Description',
19             style: TextStyle(fontSize: 18, fontWeight: FontWeight
20               .bold),
21           ),
22           SizedBox(height: 8),
23           Text(recipe['description']),
24           SizedBox(height: 16),
25           Text(
26             'Ingredients',
27             style: TextStyle(fontSize: 18, fontWeight: FontWeight
28               .bold),
29           ),
30           SizedBox(height: 8),
31           Text(
32             (recipe['ingredients'] as List)
33               .map((e) => e['name'] as String)
34               .join(', '),
35           ),
36         ],
37       ),
38     ),
39   );
}
```

LISTING 3.11: Recipe_detail_page.dart File

3.13 Backend File 12 - App.dart

TABLE 3.12: App.dart

File	ssApp.dart
Imports	<ul style="list-style-type: none"> • flutter/material.dart • home_page.dart • generate_recipe_page.dart • constraints_recipe.dart
Class	MainPage (extends StatefulWidget)
State Class	_MainPageState
State Variables	<ul style="list-style-type: none"> • <code>_selectedIndex</code>: Tracks current bottom nav index (initial 0) • <code>_isLoading</code>: Loading state flag (initial true) • <code>_pages</code>: List of page widgets (HomePage, GenerateRecipePage, ConstraintsRecipe)
Lifecycle Methods	<ul style="list-style-type: none"> • <code>initState()</code>: Calls <code>_verifyAuthStatus()</code> on initialization • <code>_verifyAuthStatus()</code>: Mock authentication check (sets loading to false)
Interaction Methods	<ul style="list-style-type: none"> • <code>_onItemTapped(index)</code>: Updates <code>_selectedIndex</code> when bottom nav item tapped

Build Method	<p>Returns either:</p> <ul style="list-style-type: none"> • Loading indicator scaffold when <code>_isLoading</code> is true • Main scaffold with: <ul style="list-style-type: none"> – Current page from <code>_pages</code> list – Bottom navigation bar with 3 items: <ul style="list-style-type: none"> * Home (house icon) * Generate (auto_awesome icon) * Constraints (auto_awesome_mosaic icon) – Purple selected item color – Grey unselected item color
Navigation Structure	<ul style="list-style-type: none"> • Three-page navigation system: <ul style="list-style-type: none"> – HomePage (default view) – GenerateRecipePage – ConstraintsRecipe • BottomNavigationBar for page switching • State maintains current page index
Key Features	<ul style="list-style-type: none"> • Stateful widget with loading state • Simple bottom navigation pattern • Placeholder auth verification • Color-coded navigation indicators

Code Implementation

```
1 import 'package:flutter/material.dart';
2 import 'package:flutter_django_recipes_frontend/home_page.dart';
3 import 'package:flutter_django_recipes_frontend/
  generate_recipe_page.dart';
4 import 'package:flutter_django_recipes_frontend/constraints_recipe.
  dart';
5
6 class MainPage extends StatefulWidget {
7   const MainPage({super.key});
8
9   @override
10  State<MainPage> createState() => _MainPageState();
11 }
12
13 class _MainPageState extends State<MainPage> {
14   int _selectedIndex = 0;
15   bool _isLoading = true;
16
17   static final List<Widget> _pages = <Widget>[
18     const HomePage(),
19     GenerateRecipePage(),
20     ConstraintsRecipe(),
21   ];
22
23   @override
24   void initState() {
25     super.initState();
26     _verifyAuthStatus();
27   }
28
29   Future<void> _verifyAuthStatus() async {
30     setState(() => _isLoading = false);
31   }
32
33   void _onItemTapped(int index) {
34     setState(() => _selectedIndex = index);
35   }
36
37   @override
```

```
38 Widget build(BuildContext context) {
39   if (!_isLoading) {
40     return const Scaffold(
41       body: Center(child: CircularProgressIndicator()),
42     );
43   }
44
45   return Scaffold(
46     body: _pages[_selectedIndex],
47     bottomNavigationBar: BottomNavigationBar(
48       items: const <BottomNavigationBarItem>[
49         BottomNavigationBarItem(
50           icon: Icon(Icons.home),
51           label: 'Home',
52         ),
53         BottomNavigationBarItem(
54           icon: Icon(Icons.auto_awesome),
55           label: 'Generate',
56         ),
57         BottomNavigationBarItem(
58           icon: Icon(Icons.auto_awesome_mosaic),
59           label: 'Constraints',
60         ),
61       ],
62       currentIndex: _selectedIndex,
63       selectedItemColor: Colors.deepPurple,
64       unselectedItemColor: Colors.grey,
65       onTap: _onItemTapped,
66     ),
67   );
68 }
69 }
```

LISTING 3.12: App.dart File

3.14 Backend File 13 - Main.dart

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

File	<code>main.dart</code> (Application Entry Point)
Imports	<ul style="list-style-type: none"> • <code>flutter/material.dart</code> • <code>login_page.dart</code>
Main Function	<ul style="list-style-type: none"> • Entry point of the Flutter application • Calls <code>runApp()</code> with <code>MyApp</code> widget
Widget: MyApp	<ul style="list-style-type: none"> • <code>StatelessWidget</code> that serves as root application widget • Constructor with optional key parameter
Build Method	Returns a <code>MaterialApp</code> with: <ul style="list-style-type: none"> • <code>debugShowCheckedModeBanner</code>: <code>false</code> (disables debug banner) • <code>home</code>: Set to <code>LoginPage()</code> (initial route)
Configuration	<ul style="list-style-type: none"> • Basic <code>MaterialApp</code> configuration • No theme or routes specified (minimal setup) • Direct navigation to <code>LoginPage</code>

Key Features	<ul style="list-style-type: none"> • Minimal application setup • Clean debug-free appearance • LoginPage as entry point • Standard Flutter application structure
Purpose	<ul style="list-style-type: none"> • Serves as the root of the application • Initializes Flutter framework • Sets up basic MaterialApp configuration • Defines initial screen (LoginPage)

Code Implementation

```

1 import "package:flutter/material.dart";
2 import "package:flutter_django_recipes_frontend/login_page.dart";
3
4 void main() {
5   runApp(MyApp());
6 }
7
8 class MyApp extends StatelessWidget {
9   const MyApp({super.key});
10
11   @override
12   Widget build(BuildContext context) {
13     return MaterialApp(
14       debugShowCheckedModeBanner: false,
15       home: LoginPage(),
16     );
17   }
18 }

```

LISTING 3.13: Main.dart

3.15 Backend File 14 - Ingredients_list.dart

Code Implementation

```
1  const List<String> paletteList = [  
2    'savory',  
3    'spicy',  
4    'sweet',  
5    'sour',  
6    'herby',  
7    'umami',  
8    'earthy',  
9    'fruity',  
10   'smoky',  
11   'neutral',  
12   'bitter',  
13 ];  
14  
15  
16  const List<String> availableIngredients = [  
17    'olive oil',  
18    'canola oil',  
19    'all purpose flour',  
20    'sugar',  
21    'salt',  
22    'black pepper',  
23    'garlic',  
24    'onion',  
25    'butter',  
26    'eggs',  
27    'milk',  
28    'water',  
29    'chicken broth',  
30    'vegetable oil',  
31    'baking powder',  
32    'baking soda',  
33    'vanilla extract',  
34    'lemon juice',  
35    'soy sauce',  
36    'rice',
```

```
37      'pasta',
38      'tomato',
39      'tomato sauce',
40      'tomato paste',
41      'cheese',
42      'cheddar cheese',
43      'parmesan cheese',
44      'mozzarella cheese',
45      'heavy cream',
46      'sour cream',
47      'yogurt',
48      'mayonnaise',
49      'honey',
50      'brown sugar',
51      'powdered sugar',
52      'cornstarch',
53      'vinegar',
54      'apple cider vinegar',
55      'balsamic vinegar',
56      'red wine vinegar',
57      'white wine vinegar',
58      'mustard',
59      'dijon mustard',
60      'yellow mustard',
61      'ketchup',
62      'worcestershire sauce',
63      'hot sauce',
64      'sriracha',
65      'bay leaf',
66      'oregano',
67      'basil',
68      'thyme',
69      'rosemary',
70      'parsley',
71      'cilantro',
72      'cumin',
73      'coriander',
74      'paprika',
75      'smoked paprika',
76      'cayenne pepper',
77      'red pepper flakes',
```

```
78     'chili powder',
79     'curry powder',
80     'turmeric',
81     'ginger',
82     'nutmeg',
83     'cinnamon',
84     'cloves',
85     'allspice',
86     'cardamom',
87     'star anise',
88     'fennel seed',
89     'sesame seeds',
90     'poppy seeds',
91     'caraway seeds',
92     'celery seed',
93     'mustard seeds',
94     'bread crumbs',
95     'panko breadcrumbs',
96     'oats',
97     'cornmeal',
98     'flour tortillas',
99     'corn tortillas',
100    'potatoes',
101    'sweet potatoes',
102    'carrots',
103    'celery',
104    'bell peppers',
105    'green bell peppers',
106    'red bell peppers',
107    'yellow bell peppers',
108    'jalape o peppers',
109    'serrano peppers',
110    'habanero peppers',
111    'cucumber',
112    'zucchini',
113    'yellow squash',
114    'eggplant',
115    'mushrooms',
116    /* 'white mushrooms',
117    'cremini mushrooms',
118    'portobello mushrooms',
```

```
119     'shiitake mushrooms',
120     'spinach',
121     'kale',
122     'lettuce',
123     'romaine lettuce',
124     'iceberg lettuce',
125     'arugula',
126     'swiss chard',
127     'collard greens',
128     'cabbage',
129     'green cabbage',
130     'red cabbage',
131     'bok choy',
132     'brussels sprouts',
133     'cauliflower',
134     'broccoli',
135     'asparagus',
136     'green beans',
137     'peas',
138     'corn',
139     'avocado',
140     'lime',
141     'lemon',
142     'orange',
143     'apple',
144     'banana',
145     'strawberries',
146     'blueberries',
147     'raspberries',
148     'blackberries',
149     'peaches',
150     'pears',
151     'plums',
152     'cherries',
153     'grapes',
154     'pineapple',
155     'mango',
156     'kiwi',
157     'coconut',
158     'raisins',
159     'dates',
```



```
160     'prunes',
161     'apricots',
162     'figs',
163     'almonds',
164     'walnuts',
165     'pecans',
166     'cashews',
167     'peanuts',
168     'hazelnuts',
169     'pistachios',
170     'sunflower seeds',
171     'pumpkin seeds',
172     'chia seeds',
173     'flax seeds',
174     'quinoa',
175     'couscous',
176     'bulgur',
177     'barley',
178     'lentils',
179     'black beans',
180     'kidney beans',
181     'pinto beans',
182     'navy beans',
183     'garbanzo beans',
184     'cannellini beans',
185     'split peas',
186     'black eyed peas',
187     'tofu',
188     'tempeh',
189     'seitan',
190     'chicken',
191     'chicken breast',
192     'chicken thighs',
193     'chicken wings',
194     'chicken drumsticks',
195     'turkey',
196     'ground turkey',
197     'turkey breast',
198     'beef',
199     'ground beef',
200     'steak',
```

```
201     'ribeye steak',
202     'sirloin steak',
203     'flank steak',
204     'short ribs',
205     'beef chuck',
206     'pork',
207     'pork chops',
208     'pork tenderloin',
209     'pork shoulder',
210     'bacon',
211     'ham',
212     'sausage',
213     'italian sausage',
214     'chorizo',
215     'hot dogs',
216     'lamb',
217     'lamb chops',
218     'ground lamb',
219     'fish',
220     'salmon',
221     'tuna',
222     'cod',
223     'halibut',
224     'tilapia',
225     'trout',
226     'shrimp',
227     'prawns',
228     'scallops',
229     'lobster',
230     'crab',
231     'clams',
232     'mussels',
233     'oysters',
234     'anchovies',
235     'sardines',
236     'bread',
237     'white bread',
238     'whole wheat bread',
239     'baguette',
240     'ciabatta',
241     'sourdough bread',
```

```
242     'pita bread',
243     'naan',
244     'tortilla chips',
245     'potato chips',
246     'pretzels',
247     'crackers',
248     'saltine crackers',
249     'graham crackers',
250     'chocolate',
251     'dark chocolate',
252     'milk chocolate',
253     'white chocolate',
254     'chocolate chips',
255     'cocoa powder',
256     'vanilla bean',
257     'maple syrup',
258     'molasses',
259     'agave nectar',
260     'corn syrup',
261     'peanut butter',
262     'almond butter',
263     'jam',
264     'jelly',
265     'marmalade',
266     'hummus',
267     'tahini',
268     'soy milk',
269     'almond milk',
270     'coconut milk',
271     'rice milk',
272     'oat milk',
273     'whipping cream',
274     'half and half',
275     'buttermilk',
276     'evaporated milk',
277     'condensed milk',
278     'cream cheese',
279     'ricotta cheese',
280     'feta cheese',
281     'goat cheese',
282     'blue cheese',
```

```
283     'gorgonzola cheese',
284     'provolone cheese',
285     'swiss cheese',
286     'monterey jack cheese',
287     'pepper jack cheese',
288     'colby cheese',
289     'american cheese',
290     'velveeta cheese',
291     'wine',
292     'red wine',
293     'white wine',
294     'sherry',
295     'beer',
296     'rum',
297     'vodka',
298     'gin',
299     'tequila',
300     'whiskey',
301     'bourbon',
302     'scotch',
303     'cognac',
304     'brandy',
305     'coffee',
306     'tea',
307     'green tea',
308     'black tea',
309     'herbal tea',
310     'sparkling water',
311     'club soda',
312     'tonic water',
313     'lemonade',
314     'orange juice',
315     'apple juice',
316     'cranberry juice',
317     'grapefruit juice',
318     'pineapple juice',
319     'tomato juice',
320     'vegetable juice',
321     'soda',
322     'cola',
323     'ginger ale',
```

```
324     'root beer',
325     'lemon lime soda',
326     'gelatin',
327     'yeast',
328     'active dry yeast',
329     'instant yeast',
330     'shortening',
331     'lard',
332     'bacon fat',
333     'duck fat',
334     'chicken fat',
335     'beef fat',
336     'vegetable shortening',
337     'margarine',
338     'vegetable spread',
339     'cooking spray',
340     'nonstick spray',
341     'ice',
342     'dry ice',
343     'liquid smoke',
344     'food coloring',
345     'vanilla pudding',
346     'chocolate pudding',
347     'butterscotch pudding',
348     'tapioca pudding',
349     'custard',
350     'flan',
351     'ice cream',
352     'vanilla ice cream',
353     'chocolate ice cream',
354     'strawberry ice cream',
355     'sherbet',
356     'sorbet',
357     'whipped topping',
358     'cool whip',
359     'marshmallows',
360     'sprinkles',
361     'candy',
362     'caramels',
363     'toffee',
364     'gumdrops',
```

```
365     'licorice',
366     'gummy bears',
367     'jelly beans',
368     'mints',
369     'peppermint',
370     'cinnamon candy',
371     'popcorn',
372     'caramel corn',
373     'cheese popcorn',
374     'peanut brittle',
375     'trail mix',
376     'granola',
377     'protein powder',
378     'whey protein',
379     'soy protein',
380     'nutritional yeast',
381     'wheat germ',
382     'bran',
383     'wheat bran',
384     'oat bran',
385     'psyllium husk',
386     'arrowroot powder',
387     'tapioca flour',
388     'almond flour',
389     'coconut flour',
390     'rice flour',
391     'chickpea flour',
392     'buckwheat flour',
393     'corn flour',
394     'potato flour',
395     'semolina',
396     'bread flour',
397     'cake flour',
398     'pastry flour',
399     'self rising flour',
400     'whole wheat flour',
401     'gluten free flour',
402     'pancake mix',
403     'biscuit mix',
404     'cornbread mix',
405     'cake mix',
```

```
406     'brownie mix',
407     'cookie mix',
408     'muffin mix',
409     'pie crust',
410     'phyllo dough',
411     'puff pastry',
412     'pizza dough',
413     'bread dough',
414     'cookie dough',
415     'brownie batter',
416     'cake batter',
417     'pancake batter',
418     'waffle batter',
419     'crepe batter',
420     'french toast batter',
421     'tempura batter',
422     'beer batter',
423     'marinade',
424     'rub',
425     'dry rub',
426     'seasoning salt',
427     'garlic salt',
428     'onion salt',
429     'celery salt',
430     'seasoning blend',
431     'italian seasoning',
432     'poultry seasoning',
433     'pumpkin pie spice',
434     'apple pie spice',
435     'chai spice',
436     'five spice powder',
437     'adobo seasoning',
438     'creole seasoning',
439     'cajun seasoning',
440     'blackening seasoning',
441     'jerk seasoning',
442     'taco seasoning',
443     'fajita seasoning',
444     'ranch seasoning',
445     'sazon seasoning',
446     'old bay seasoning',
```

```
447     'everything bagel seasoning',
448     'furikake',
449     'zaatar',
450     'sumac',
451     'harissa',
452     'sambal oelek',
453     'gochujang',
454     'miso paste',
455     'hoisin sauce',
456     'oyster sauce',
457     'fish sauce',
458     'teriyaki sauce',
459     'ponzu sauce',
460     'duck sauce',
461     'plum sauce',
462     'sweet chili sauce',
463     'peanut sauce',
464     'alfredo sauce',
465     'marinara sauce',
466     'pesto',
467     'tzatziki sauce',
468     'aioli',
469     'remoulade sauce',
470     'tartar sauce',
471     'cocktail sauce',
472     'bbq sauce',
473     'honey mustard',
474     'ranch dressing',
475     'blue cheese dressing',
476     'caesar dressing',
477     'italian dressing',
478     'french dressing',
479     'thousand island dressing',
480     'vinaigrette',
481     'balsamic vinaigrette',
482     'raspberry vinaigrette',
483     'honey vinaigrette',
484     'mustard vinaigrette',
485     'citrus vinaigrette',
486     'herb vinaigrette',
487     'shallot vinaigrette',
```



```
488     'garlic vinaigrette',
489     'ginger vinaigrette',
490     'sesame vinaigrette',
491     'soy vinaigrette',
492     'wasabi',
493     'horseradish',
494     'ginger paste',
495     'garlic paste',
496     'tomato puree',
497     'apple puree',
498     'banana puree',
499     'pumpkin puree',
500     'sweet potato puree',
501     'butternut squash puree',
502     'cauliflower puree',
503     'avocado puree',
504     'mango puree',
505     'pear puree',
506     'peach puree',
507     'berry puree',
508     'fruit puree',
509     'vegetable puree',
510     'bean puree',
511     'lentil puree',
512     'hummus',
513     'guacamole',
514     'salsa',
515     'pico de gallo',
516     'salsa verde',
517     'corn salsa',
518     'mango salsa',
519     'pineapple salsa',
520     'peach salsa',
521     'black bean salsa',
522     'tomato salsa',
523     'fruit salsa',
524     'chutney',
525     'mango chutney',
526     'apple chutney',
527     'tomato chutney',
528     'onion chutney',
```

```
529     'cranberry chutney',
530     'pear chutney',
531     'peach chutney',
532     'plum chutney',
533     'fig chutney',
534     'relish',
535     'pickle relish',
536     'corn relish',
537     'pepper relish',
538     'onion relish',
539     'cucumber relish',
540     'zucchini relish',
541     'eggplant relish',
542     'tomato relish',
543     'fruit relish',
544     'vegetable relish',
545     'pickles',
546     'dill pickles',
547     'bread and butter pickles',
548     'sweet pickles',
549     'gherkins',
550     'cornichons',
551     'pickled jalape os',
552     'pickled onions',
553     'pickled beets',
554     'pickled carrots',
555     'pickled cucumbers',
556     'pickled ginger',
557     'pickled vegetables',
558     'kimchi',
559     'sauerkraut',
560     'olives',
561     'black olives',
562     'green olives',
563     'kalamata olives',
564     'olive tapenade',
565     'capers',
566     'artichoke hearts',
567     'sun dried tomatoes',
568     'roasted red peppers',
569     'pepperoncini',
```

```
570     'giardiniera',
571     'banana peppers',
572     'cherry peppers',
573     'pepperoni',
574     'salami',
575     'prosciutto',
576     'pancetta',
577     'capicola',
578     'soppressata',
579     'mortadella',
580     'corned beef',
581     'pastrami',
582     'roast beef',
583     'turkey breast',
584     'chicken breast',
585     'ham',
586     'bologna',
587     'liverwurst',
588     'braunschweiger',
589     'head cheese',
590     'blood sausage',
591     'andouille sausage',
592     'kielbasa',
593     'bratwurst',
594     'italian sausage',
595     'breakfast sausage',
596     'chorizo',
597     'linguica',
598     'loukaniko',
599     'merguez',
600     'sucuk',
601     'noodles',
602     'ramen noodles',
603     'udon noodles',
604     'soba noodles',
605     'rice noodles',
606     'vermicelli',
607     'angel hair pasta',
608     'spaghetti',
609     'linguine',
610     'fettuccine',
```

```
611     'penne',
612     'rigatoni',
613     'ziti',
614     'macaroni',
615     'fusilli',
616     'rotini',
617     'farfalle',
618     'orzo',
619     'lasagna noodles',
620     'manicotti',
621     'cannelloni',
622     'ravioli',
623     'tortellini',
624     'gnocchi',
625     'pierogi',
626     'dumplings',
627     'wonton wrappers',
628     'egg roll wrappers',
629     'spring roll wrappers',
630     'phyllo dough',
631     'puff pastry',
632     'pie crust',
633     'tart shell',
634     'graham cracker crust',
635     'cookie crust',
636     'shortbread crust',
637     'pastry dough',
638     'biscuit dough',
639     'pizza dough',
640     'bread dough',
641     'roll dough',
642     'croissant dough',
643     'danish dough',
644     'brioche dough',
645     'challah dough',
646     'bagel dough',
647     'pretzel dough',
648     'doughnut dough',
649     'fritter batter',
650     'pancake batter',
651     'waffle batter',
```

```
652     'crepe batter',
653     'french toast batter',
654     'tempura batter',
655     'beer batter',
656     'cake batter',
657     'brownie batter',
658     'cookie dough',
659     'muffin batter',
660     'quick bread batter',
661     'yeast bread dough',
662     'sourdough starter',
663     'poolish',
664     'biga',
665     'sponge',
666     'levain',
667     'mother dough',
668     'wild yeast starter',
669     'sourdough culture',
670     'sourdough discard',
671     'active sourdough starter',
672     'ripe sourdough starter',
673     'fed sourdough starter',
674     'unfed sourdough starter',
675     'liquid levain',
676     'stiff levain',
677     'firm levain',
678     'liquid sourdough starter',
679     'stiff sourdough starter',
680     'firm sourdough starter',
681     '100% hydration starter',
682     '50% hydration starter',
683     '60% hydration starter',
684     '70% hydration starter',
685     '80% hydration starter',
686     '90% hydration starter',
687     'whole wheat starter',
688     'rye starter',
689     'spelt starter',
690     'einkorn starter',
691     'kamut starter',
692     'emmer starter',
```

```
693     'farro starter',
694     'barley starter',
695     'oat starter',
696     'buckwheat starter',
697     'quinoa starter',
698     'amaranth starter',
699     'teff starter',
700     'millet starter',
701     'sorghum starter',
702     'rice starter',
703     'corn starter',
704     'potato starter',
705     'sweet potato starter',
706     'pumpkin starter',
707     'squash starter',
708     'beet starter',
709     'carrot starter',
710     'parsnip starter',
711     'turnip starter',
712     'radish starter',
713     'daikon starter',
714     'ginger starter',
715     'turmeric starter',
716     'garlic starter',
717     'onion starter',
718     'shallot starter',
719     'leek starter',
720     'scallion starter',
721     'chive starter',
722     'herb starter',
723     'spice starter',
724     'fruit starter',
725     'vegetable starter',
726     'seed starter',
727     'nut starter',
728     'bean starter',
729     'lentil starter',
730     'pea starter',
731     'soy starter',
732     'tofu starter',
733     'tempeh starter',
```

```
734     'seitan starter',  
735     'miso starter',  
736     'soy sauce starter',  
737     'fish sauce starter',  
738     'oyster sauce starter',  
739     'hoisin sauce starter',  
740     'teriyaki sauce starter',  
741     'ponzu sauce starter',  
742     ' Worcestershire sauce starter',  
743     'vinegar starter',  
744     'kombucha starter',  
745     'water kefir starter',  
746     'milk kefir starter',  
747     'yogurt starter',  
748     'buttermilk starter', */  
749 ];
```

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.”