AI Chef Recipe Generator – SDLC

Project By: Eiad Alsafadi, Oleg Vasiliev, Moied Ahmed

1. Project Overview

Project Name: AI Chef Recipe Generator
Objective:
To help users find recipes they can cook with the ingredients they already have, taking into account available cooking methods, time, and dietary restrictions.

Target Demographic:

- Home cooks
- Food enthusiasts
- People with dietary restrictions

Platform:

- Web and Mobile application

2. Project Scope

- Generate recipes using AI based on user inputs
- Accept user inputs including:
    - Ingredients (from list, personal additions, or photo recognition)
    - Available cooking time
    - Cooking method (Stove, Microwave, Oven)

    - Dietary restrictions

- Provide step-by-step cooking instructions with images
- Sign up and sign in using Email/Password, Google, or Facebook
- Save user ingredient lists and preferences
- Option to generate multiple recipe suggestions

---

3. SDLC Phases

3.1 Requirement Analysis

Functional Requirements:

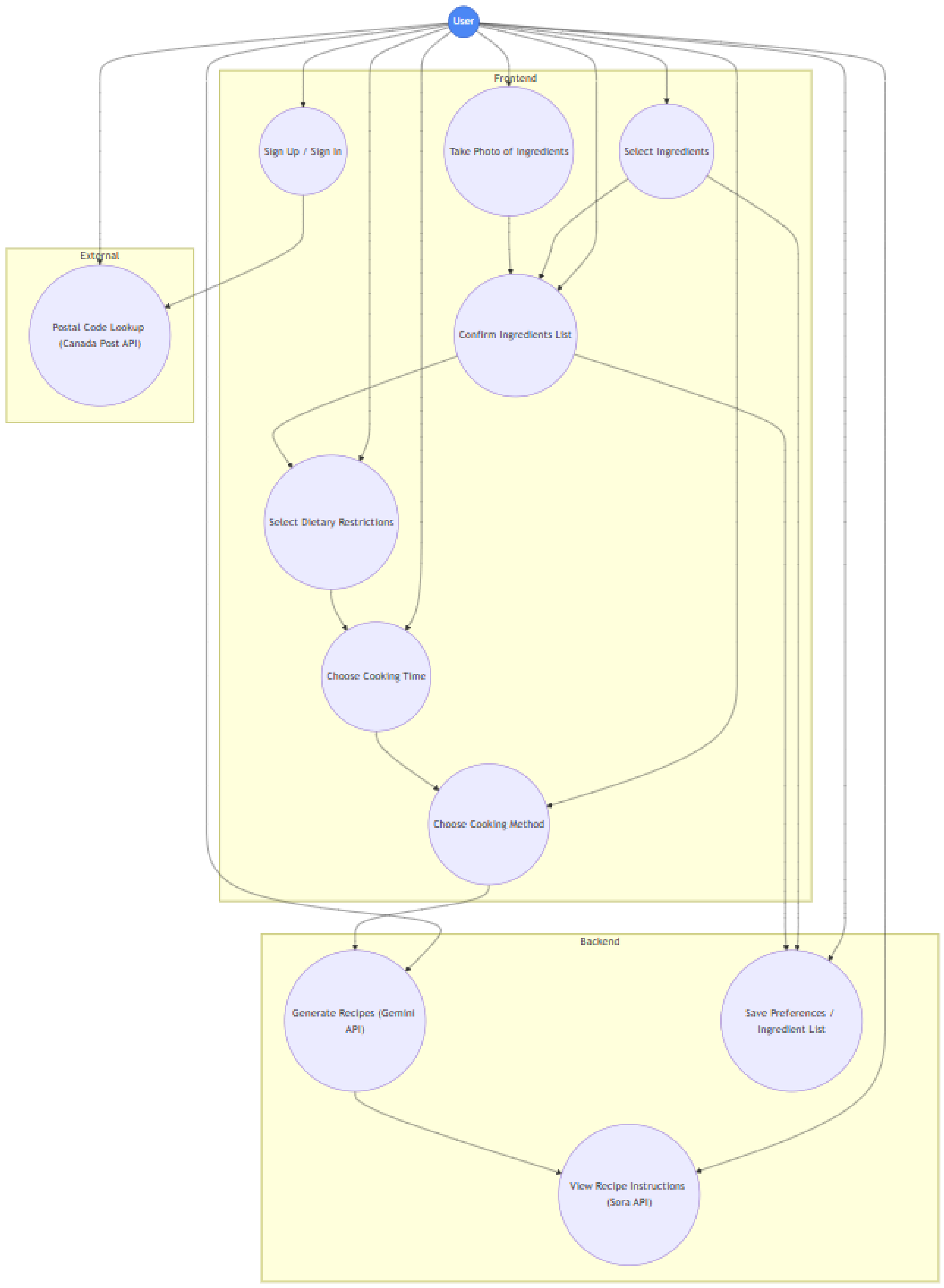
- Sign Up / Sign In functionality with multiple login options
- Ingredient selection from generic or personal lists
- Cooking time and method selection
- Dietary restriction selection
- Recipe generation using AI (Gemini API)
- Display cooking instructions with images (Sora API)
- Address auto-generation via Canada Post API

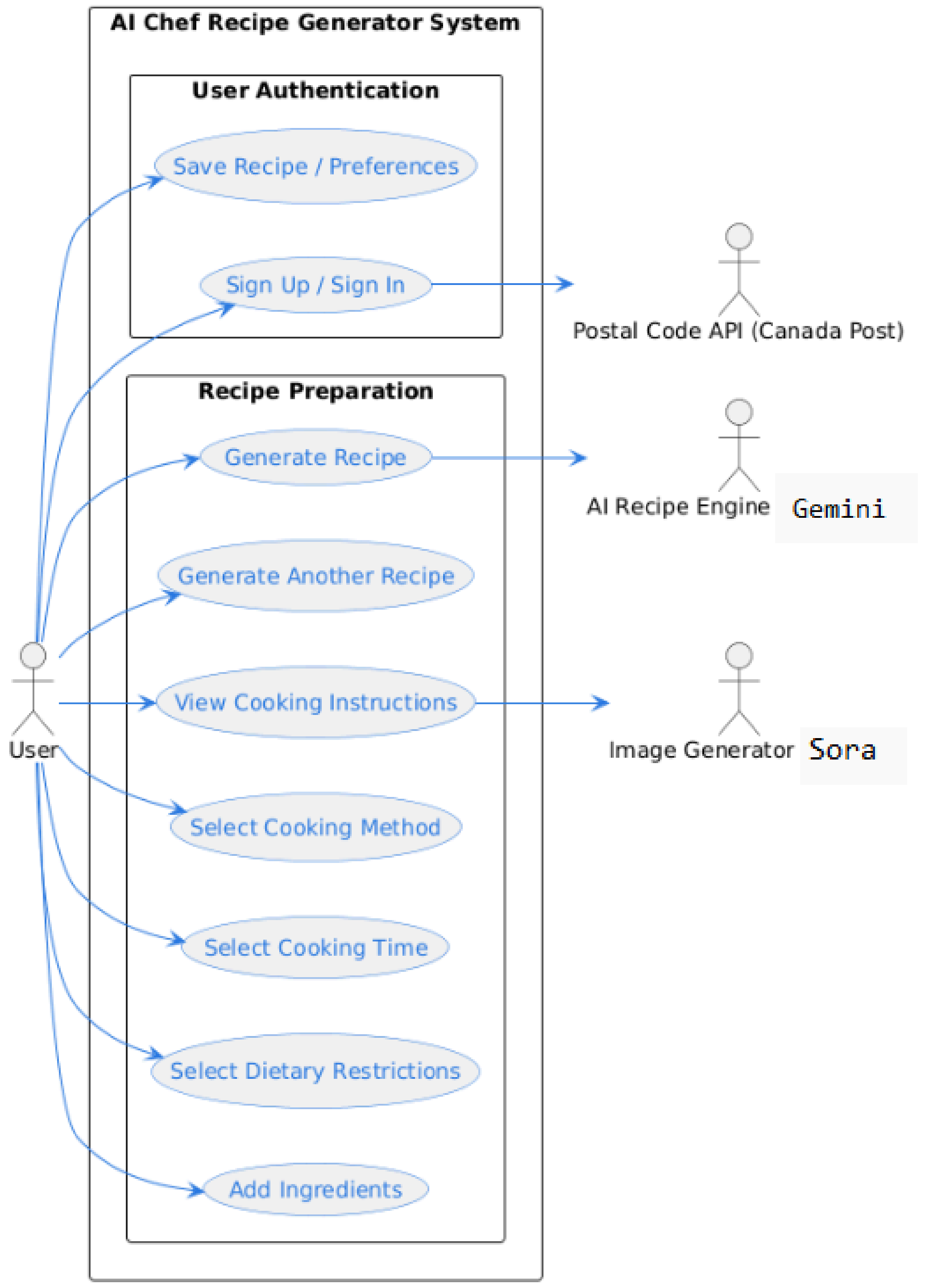
Non-Functional Requirements:

- Fast response time for recipe generation
- User-friendly and intuitive UI
- Secure login and user data handling
- Support remote and distributed development

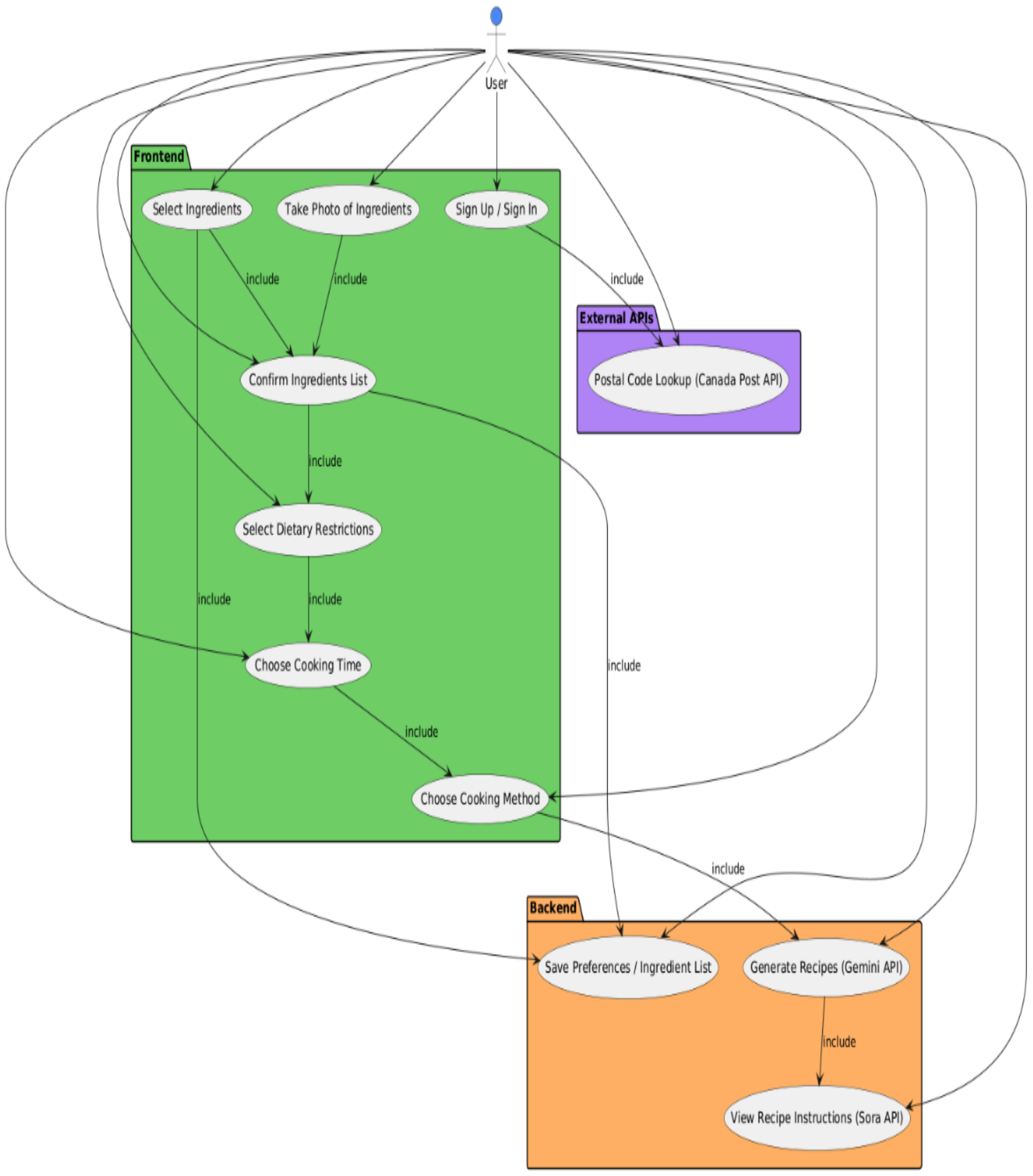
Deliverables: UML

- **Use Case Diagrams**

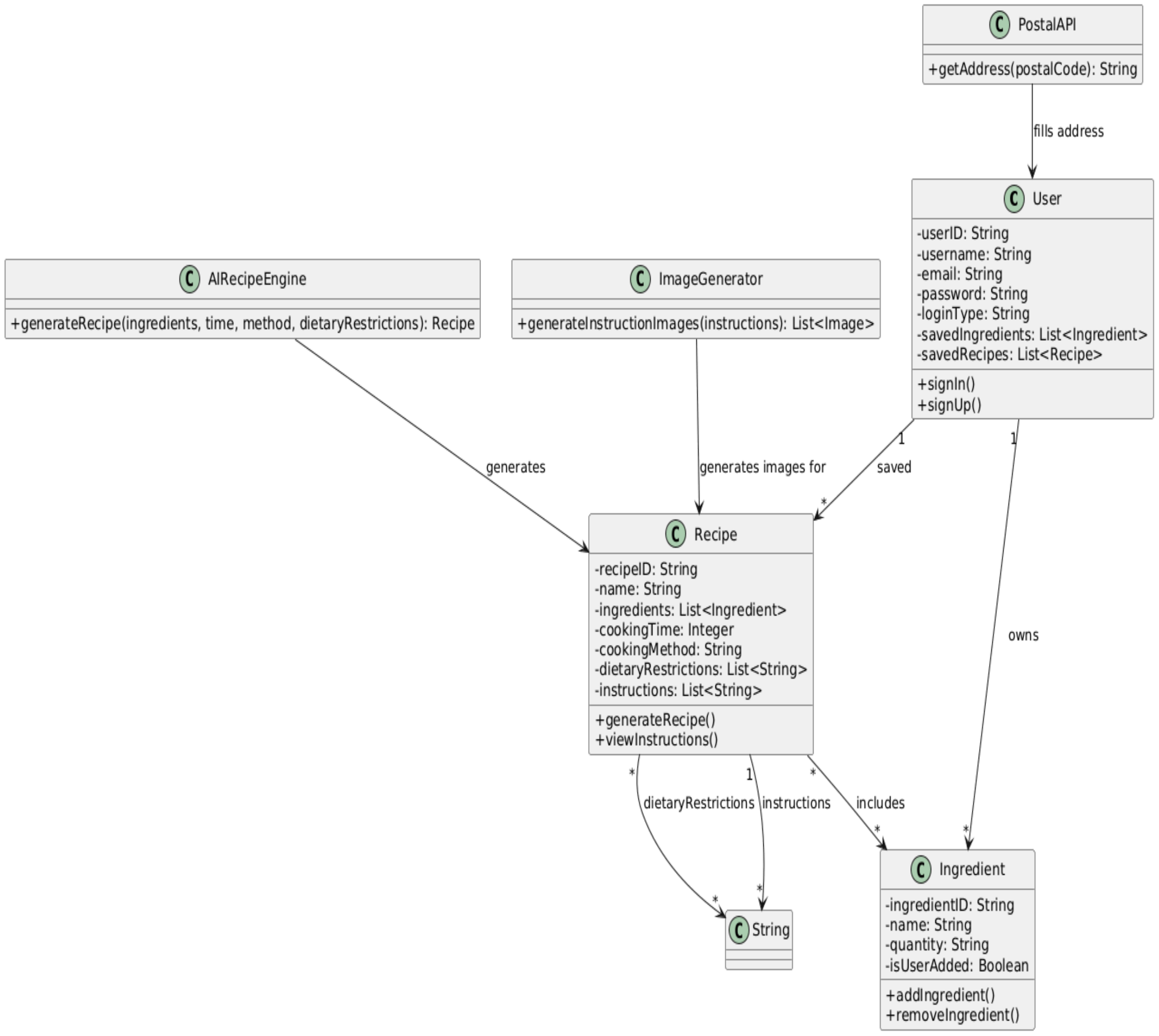

User
Frontend
Sign Up / Sign In
Take Photo of Ingredients
Select Ingredients
External
Postal Code Lookup (Canada Post API)
Confirm Ingredients List
Select Dietary Restrictions
Choose Cooking Time
Choose Cooking Method
Backend
Generate Recipes (Gemini API)
Save Preferences / Ingredient List
View Recipe Instructions (Sora API)


AI Chef Recipe Generator System
User Authentication
Save Recipe / Preferences
Sign Up / Sign In
Recipe Preparation
Generate Recipe
Generate Another Recipe
View Cooking Instructions
Select Cooking Method
Select Cooking Time
Select Dietary Restrictions
Add Ingredients
User
Postal Code API (Canada Post)
AI Recipe Engine
Gemini
Image Generator
Sora

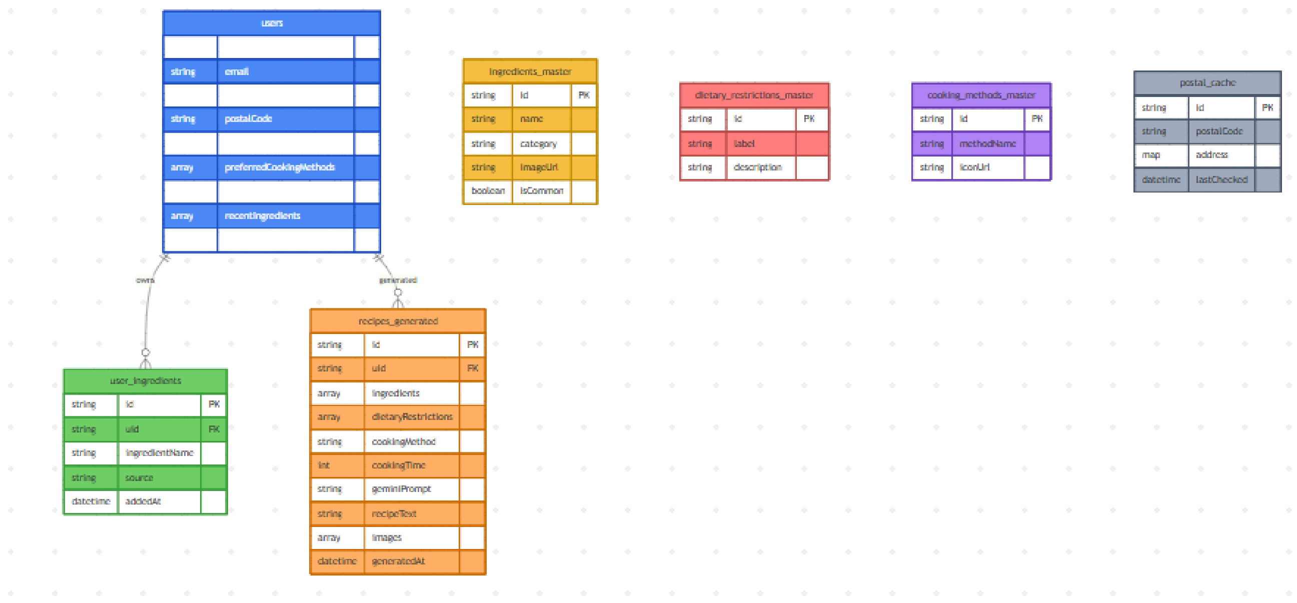
AI Chef Recipe Generator - Use Case Diagram



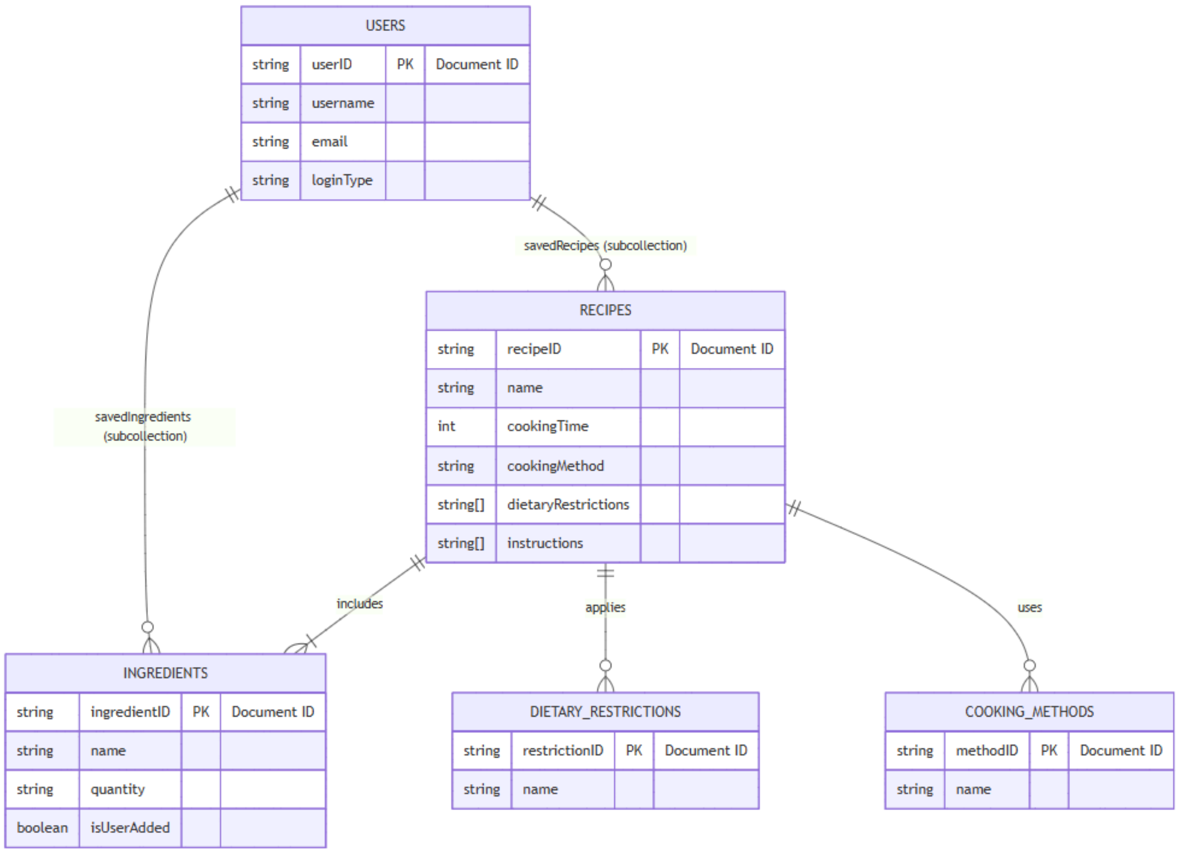
Class Diagram



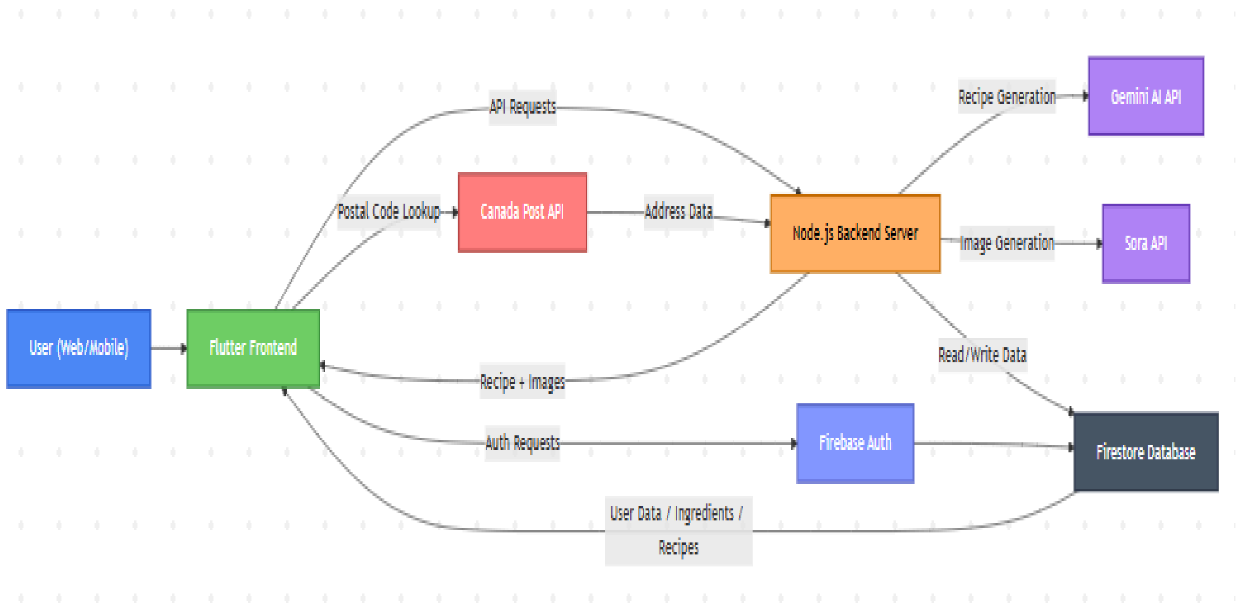
Firebase Firestore Database ERD Diagram



USER ERD Diagram



3.2 System Design:<u>System Architecture</u>



High-Level Design:

- Client-Server architecture
- Frontend: Mobile & Web interface
- Backend: Firebase
- Database: Firebase Firestore for user data, ingredients, and recipe history
- Authentication: Firebase Auth for Email/Password, Google, Facebook

Low-Level Design:

- Screen Flow (9–10 screens)
- API endpoints for recipe generation, image generation, and postal code lookup
- Data models for users, ingredients, recipes, and dietary restrictions

Data Structure for Integration

<u>Front-end (Flutter/Dart)</u> → reads/writes from Firestore

- **Gemini AI** → receives ingredients, dietary restrictions, time, cooking method and generates recipeText → save to recipes_generated

- **Sora API** → receives the recipe steps and generates images → store URLs directly in recipes_generated

- **Canada Post API** → receives postalCode → return address → store URL in users

<u>Back-end (Firestore Flow)</u>

User logs in → users collection

User selects ingredients → user_ingredients

User selects dietary restrictions → references dietary_restrictions

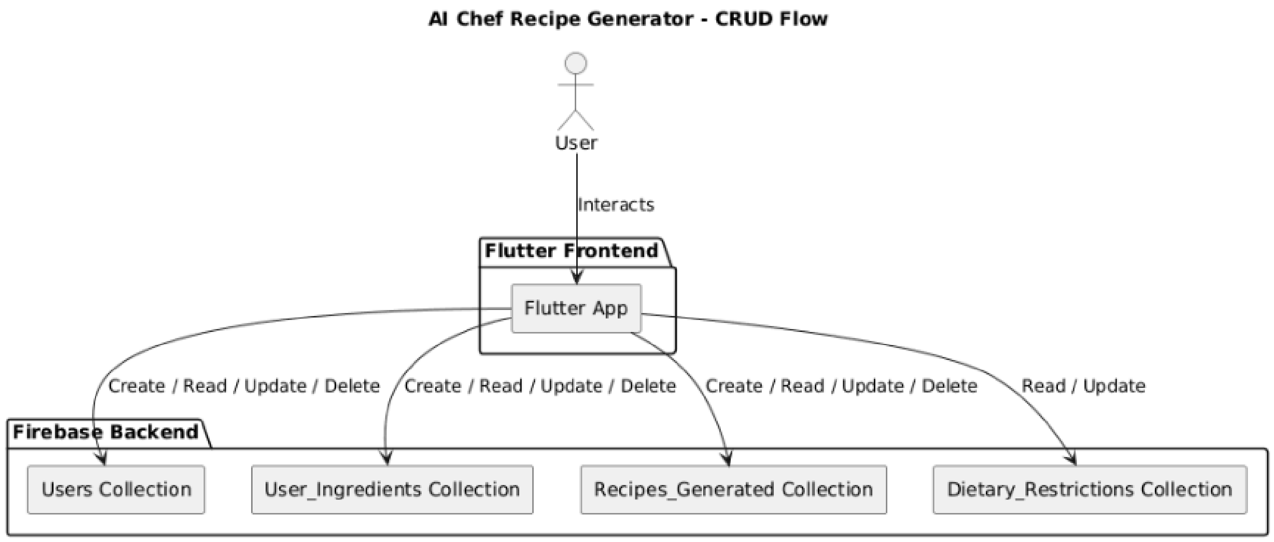
User clicks "Generate Recipe" → send ingredients, restrictions, time, method to Gemini AI

Gemini AI returns recipe → store in recipes_generated

Sora generates images → store URLs in recipes_generated or separate recipe_images

Frontend displays recipe and images

CRUD (Create,Read,Update,Delete Flow)


AI Chef Recipe Generator - CRUD Flow
User
Interacts
Flutter Frontend
Flutter App
Create / Read / Update / Delete
Create / Read / Update / Delete
Create / Read / Update / Delete
Read / Update
Firebase Backend
Users Collection
User_Ingredients Collection
Recipes_Generated Collection
Dietary_Restrictions Collection

CRUD Implementation in AI Chef Recipe Generator

Our app uses Firebase Firestore to implement full CRUD functionality:

- **Create: Users can add new ingredients to their personal list, register accounts, and generate recipes, which are stored in the users, user_ingredients, and recipes_generated collections.**

- **Read: The frontend reads data from Firestore to display available ingredients, dietary restrictions, saved recipes, and user preferences in real time.**

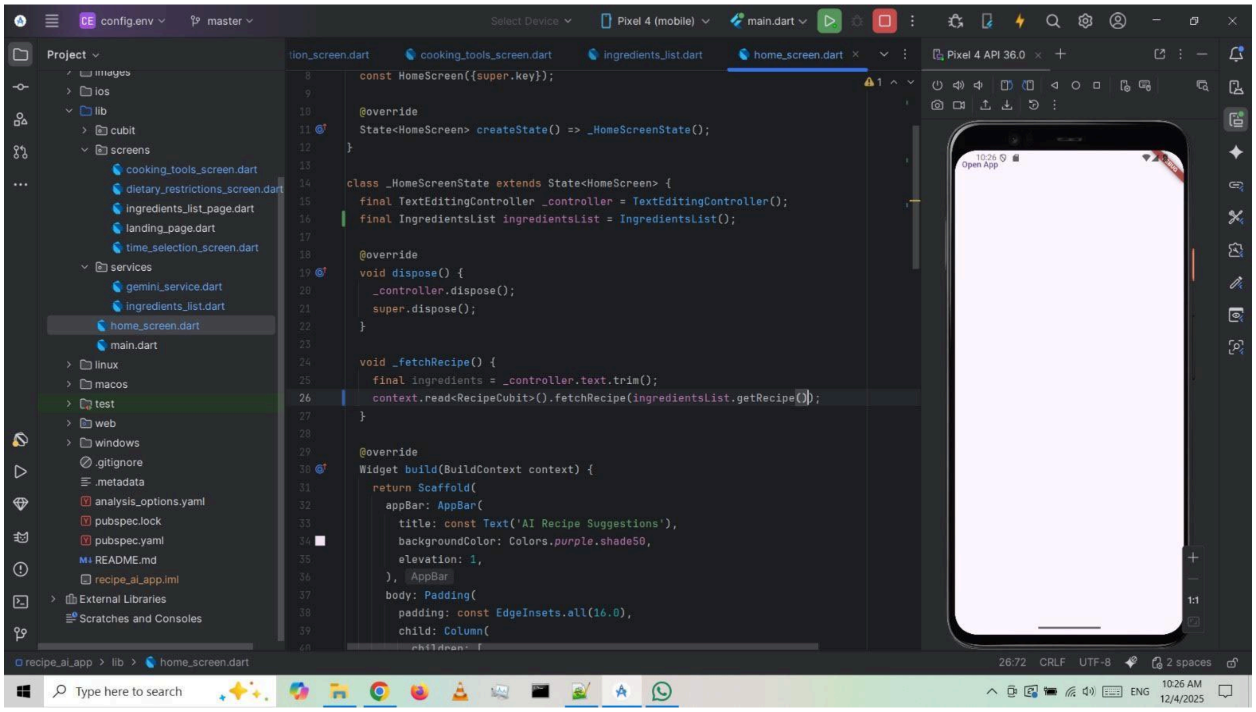
- **Update: Users can update their ingredient list, modify dietary restrictions, and save edits to recipes or preferences. Firestore automatically syncs these changes across devices.**

- **Delete: Users can remove ingredients, delete saved recipes, or clear preferences, which deletes the corresponding documents or fields from Firestore.**

All operations are performed via Flutter/Dart using Firestore SDK, ensuring secure and efficient data access without the need for a separate backend server.

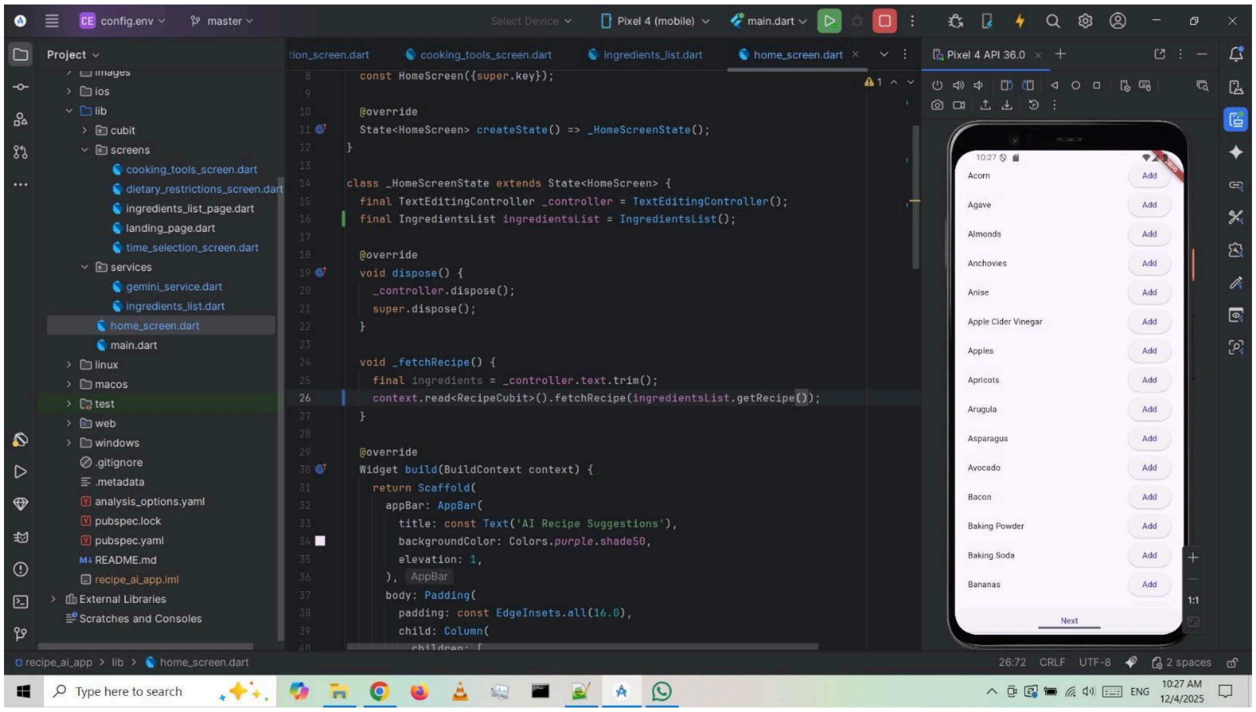
3.3 Implementation / Coding Frontend: Flutter

1) Open App



```dart
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final TextEditingController _controller = TextEditingController();
  final IngredientsList ingredientsList = IngredientsList();

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void _fetchRecipe() {
    final ingredients = _controller.text.trim();
    context.read<RecipeCubit>().fetchRecipe(ingredientsList.getRecipe());
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('AI Recipe Suggestions'),
        backgroundColor: Colors.purple.shade50,
        elevation: 1,
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
```

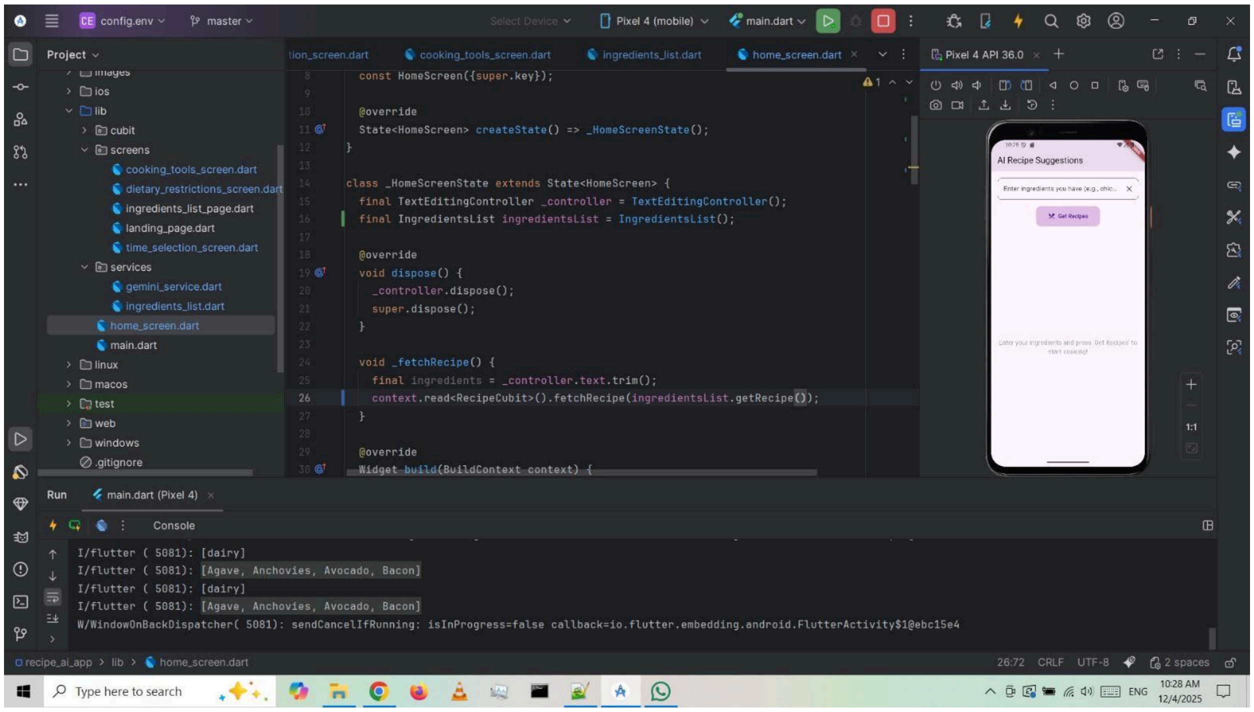
2) Choose Ingredients from collection user_ingredients for example: [Agave, Anchovies, Avocado, Bacon]



```dart
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final TextEditingController _controller = TextEditingController();
  final IngredientsList ingredientsList = IngredientsList();

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void _fetchRecipe() {
    final ingredients = _controller.text.trim();
    context.read<RecipeCubit>().fetchRecipe(ingredientsList.getRecipe());
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('AI Recipe Suggestions'),
        backgroundColor: Colors.purple.shade50,
        elevation: 1,
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
```

Acorn Add
Agave Add
Almonds Add
Anchovies Add
Anise Add
Apple Cider Vinegar Add
Apples Add
Apricots Add
Arugula Add
Asparagus Add
Avocado Add
Bacon Add
Baking Powder Add
Baking Soda Add
Bananas Add
Next

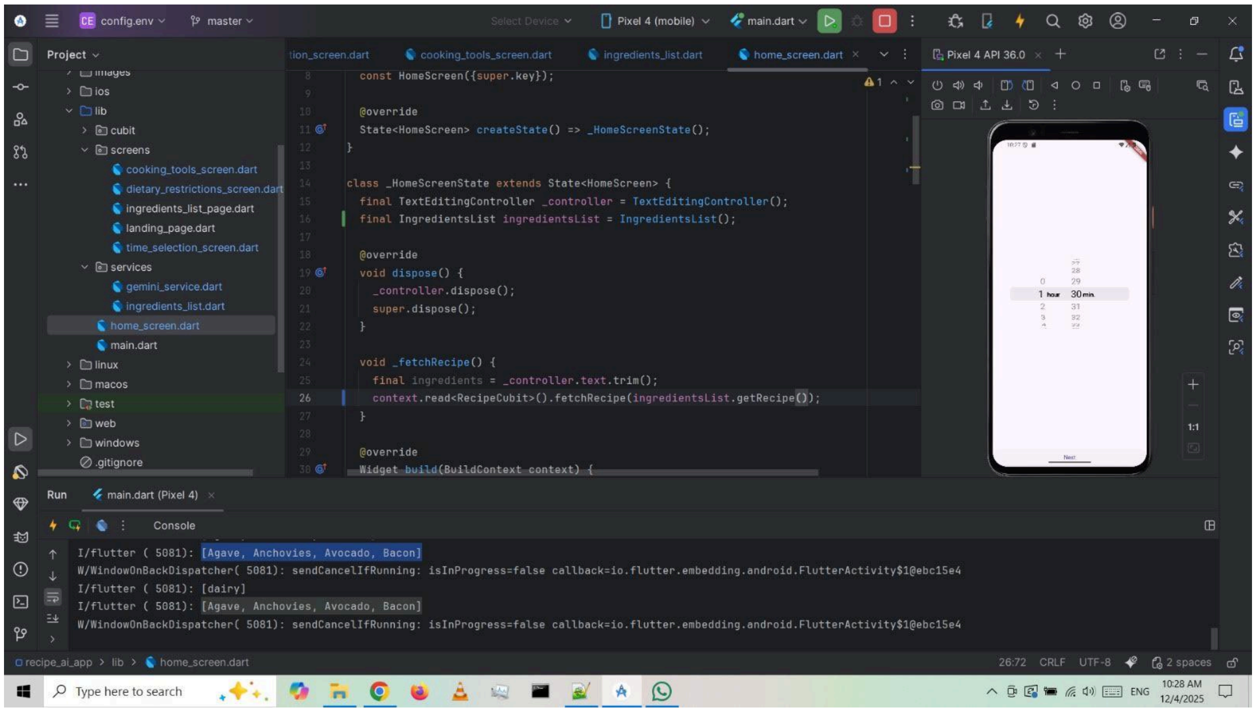
3) Choose dietary restriction(s) from collection dietary_restrictions for example: [Dairy-free]



```dart
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final TextEditingController _controller = TextEditingController();
  final IngredientsList ingredientsList = IngredientsList();

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void _fetchRecipe() {
    final ingredients = _controller.text.trim();
    context.read<RecipeCubit>().fetchRecipe(ingredientsList.getRecipe());
  }

  @override
  Widget build(BuildContext context) {
```

```
I/flutter ( 5081): [dairy]
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
I/flutter ( 5081): [dairy]
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
W/WindowOnBackDispatcher( 5081): sendCancelIfRunning: isInProgress=false callback=io.flutter.embedding.android.FlutterActivity$1@ebc15e4
```

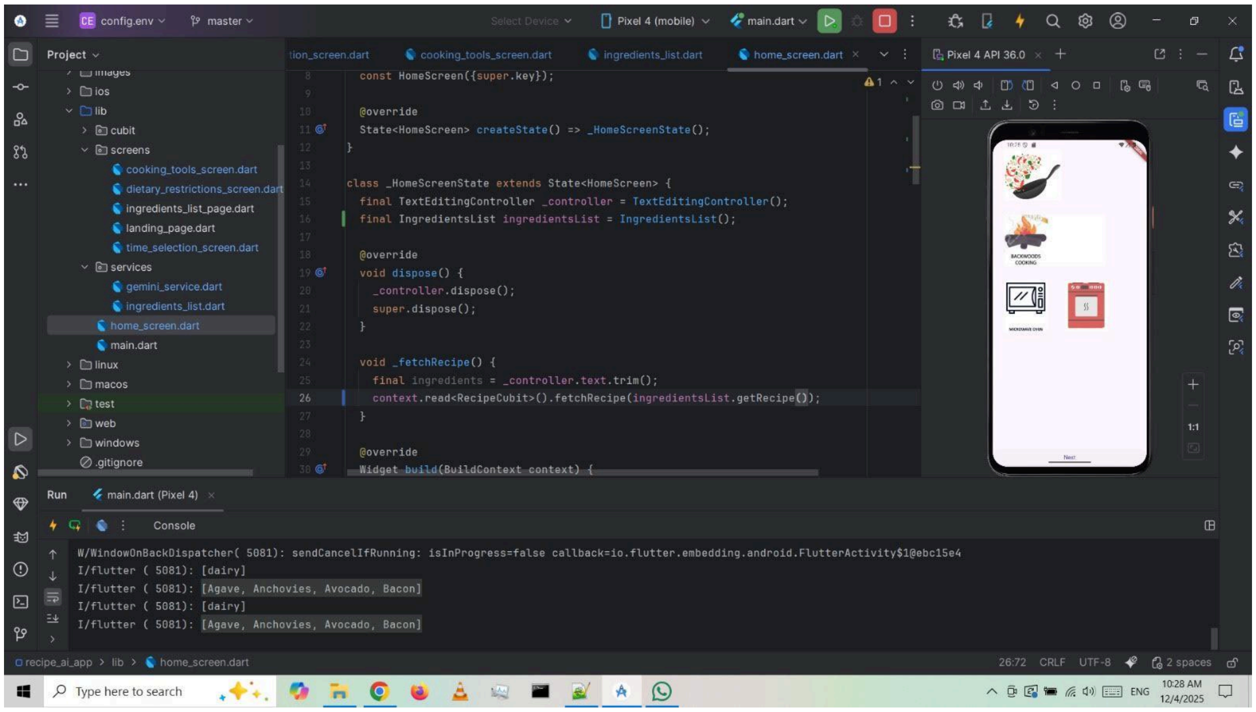
4) Cook Timer for example: 1 hour and 30 min



```dart
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final TextEditingController _controller = TextEditingController();
  final IngredientsList ingredientsList = IngredientsList();

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void _fetchRecipe() {
    final ingredients = _controller.text.trim();
    context.read<RecipeCubit>().fetchRecipe(ingredientsList.getRecipe());
  }

  @override
  Widget build(BuildContext context) {
```

```
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
W/WindowOnBackDispatcher( 5081): sendCancelIfRunning: isInProgress=false callback=io.flutter.embedding.android.FlutterActivity$1@ebc15e4
I/flutter ( 5081): [dairy]
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
W/WindowOnBackDispatcher( 5081): sendCancelIfRunning: isInProgress=false callback=io.flutter.embedding.android.FlutterActivity$1@ebc15e4
```

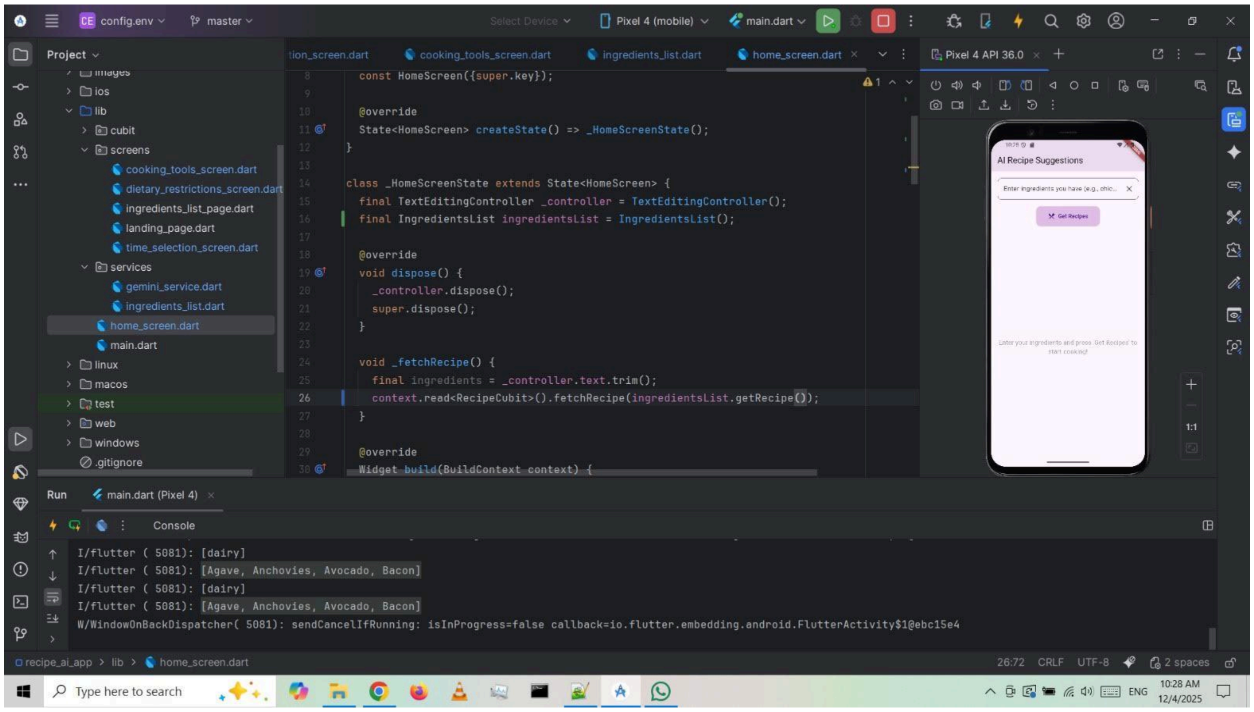
5) Select Cooking Method for example, Stove and pan



```
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final TextEditingController _controller = TextEditingController();
  final IngredientsList ingredientsList = IngredientsList();

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void _fetchRecipe() {
    final ingredients = _controller.text.trim();
    context.read<RecipeCubit>().fetchRecipe(ingredientsList.getRecipe());
  }

  @override
  Widget build(BuildContext context) {
```

```
W/WindowOnBackDispatcher( 5081): sendCancelIfRunning: isInProgress=false callback=io.flutter.embedding.android.FlutterActivity$1@ebc15e4
I/flutter ( 5081): [dairy]
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
I/flutter ( 5081): [dairy]
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
```

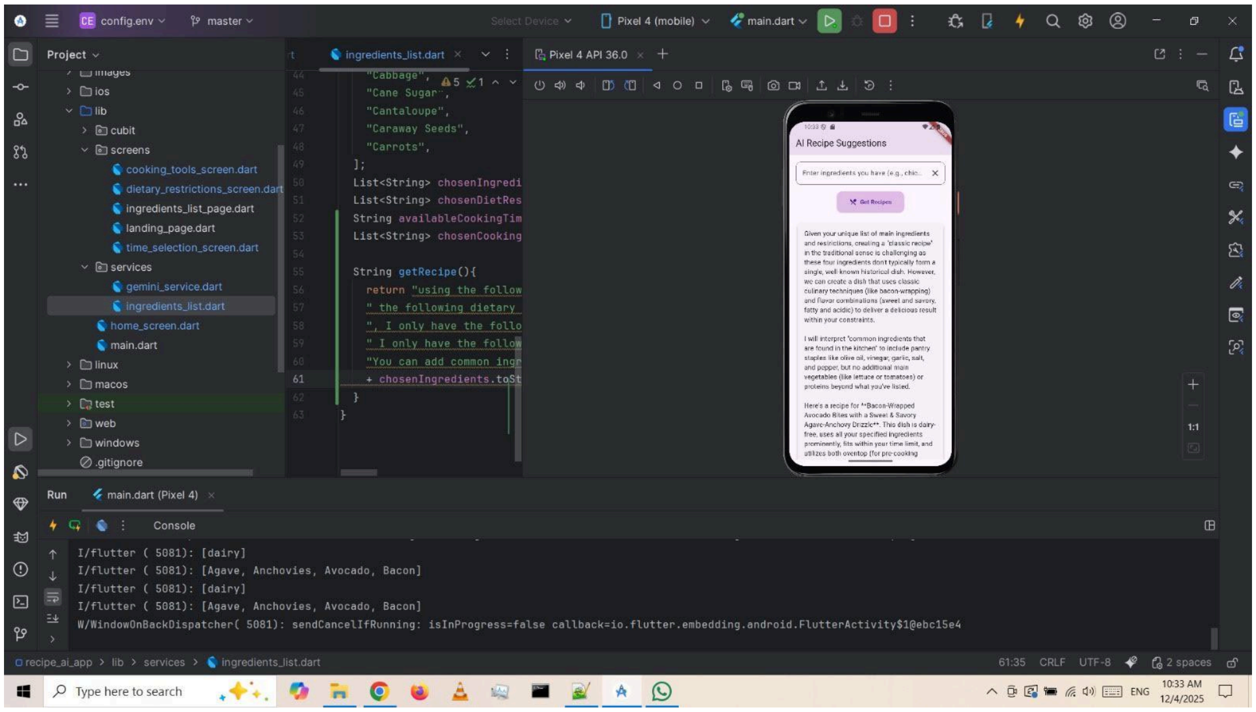
6) Click Get Recipe


AI Recipe Suggestions
Enter ingredients you have (e.g., chic...
Get Recipes
Enter your ingredients and press 'Get Recipes' to start cooking!

```
  const HomeScreen({super.key});

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  final TextEditingController _controller = TextEditingController();
  final IngredientsList ingredientsList = IngredientsList();

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void _fetchRecipe() {
    final ingredients = _controller.text.trim();
    context.read<RecipeCubit>().fetchRecipe(ingredientsList.getRecipe());
  }

  @override
  Widget build(BuildContext context) {
```

```
I/flutter ( 5081): [dairy]
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
I/flutter ( 5081): [dairy]
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
W/WindowOnBackDispatcher( 5081): sendCancelIfRunning: isInProgress=false callback=io.flutter.embedding.android.FlutterActivity$1@ebc15e4
```

7) User gets recipes based on inputs



```
  "Cabbage",
  "Cane Sugar",
  "Cantaloupe",
  "Caraway Seeds",
  "Carrots",
];
List<String> chosenIngredi
List<String> chosenDietRes
String availableCookingTim
List<String> chosenCooking

String getRecipe(){
  return "using the follow
  " the following dietary
  ", I only have the follo
  " I only have the follow
  "You can add common ingr
  + chosenIngredients.toSt
}
}
```

AI Recipe Suggestions

Enter ingredients you have (e.g., chic...

Get Recipes

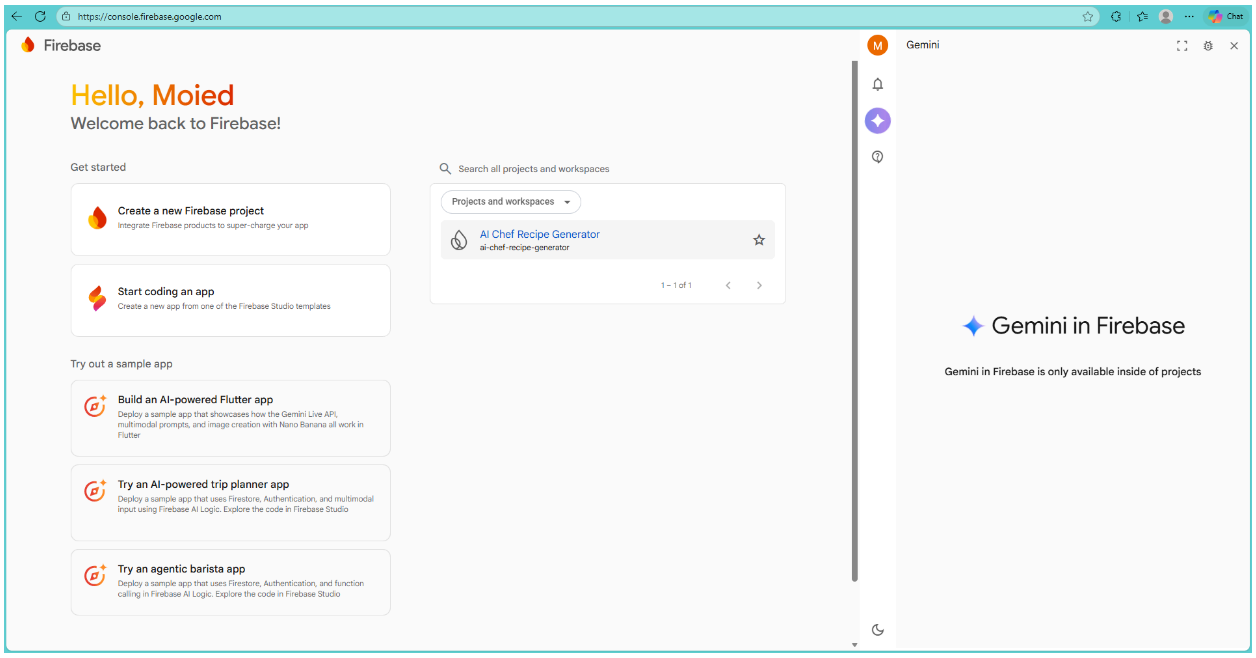
Given your unique list of main ingredients and restrictions, creating a 'classic recipe' in the traditional sense is challenging as these four ingredients dont typically form a single, well known historical dish. However, we can create a dish that uses classic culinary techniques (like bacon-wrapping) and flavor combinations (sweet and savory, fatty and acidic) to deliver a delicious result within your constraints.

I will interpret 'common ingredients that are found in the kitchen' to include pantry staples like olive oil, vinegar, garlic, salt, and pepper, but no additional main vegetables (like lettuce or tomatoes) or proteins beyond what you've listed.

Here's a recipe for **Bacon-Wrapped Avocado Bites with a Sweet & Savory Agave-Anchovy Drizzle**. This dish is dairy-free, uses all your specified ingredients prominently, fits within your time limit, and utilizes both oventop (for pre-cooking

```
I/flutter ( 5081): [dairy]
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
I/flutter ( 5081): [dairy]
I/flutter ( 5081): [Agave, Anchovies, Avocado, Bacon]
W/WindowOnBackDispatcher( 5081): sendCancelIfRunning: isInProgress=false callback=io.flutter.embedding.android.FlutterActivity$1@ebc15e4
```

Backend: Firebase Database AI Chef Recipe Generator



https://console.firebase.google.com

Firebase

Hello, Moied

Welcome back to Firebase!

Get started

Create a new Firebase project
Integrate Firebase products to super-charge your app

Start coding an app
Create a new app from one of the Firebase Studio templates

Try out a sample app

Build an AI-powered Flutter app
Deploy a sample app that showcases how the Gemini Live API, multimodal prompts, and image creation with Nano Banana all work in Flutter

Try an AI-powered trip planner app
Deploy a sample app that uses Firestore, Authentication, and multimodal input using Firebase AI Logic. Explore the code in Firebase Studio

Try an agentic barista app
Deploy a sample app that uses Firestore, Authentication, and function calling in Firebase AI Logic. Explore the code in Firebase Studio

Search all projects and workspaces

Projects and workspaces

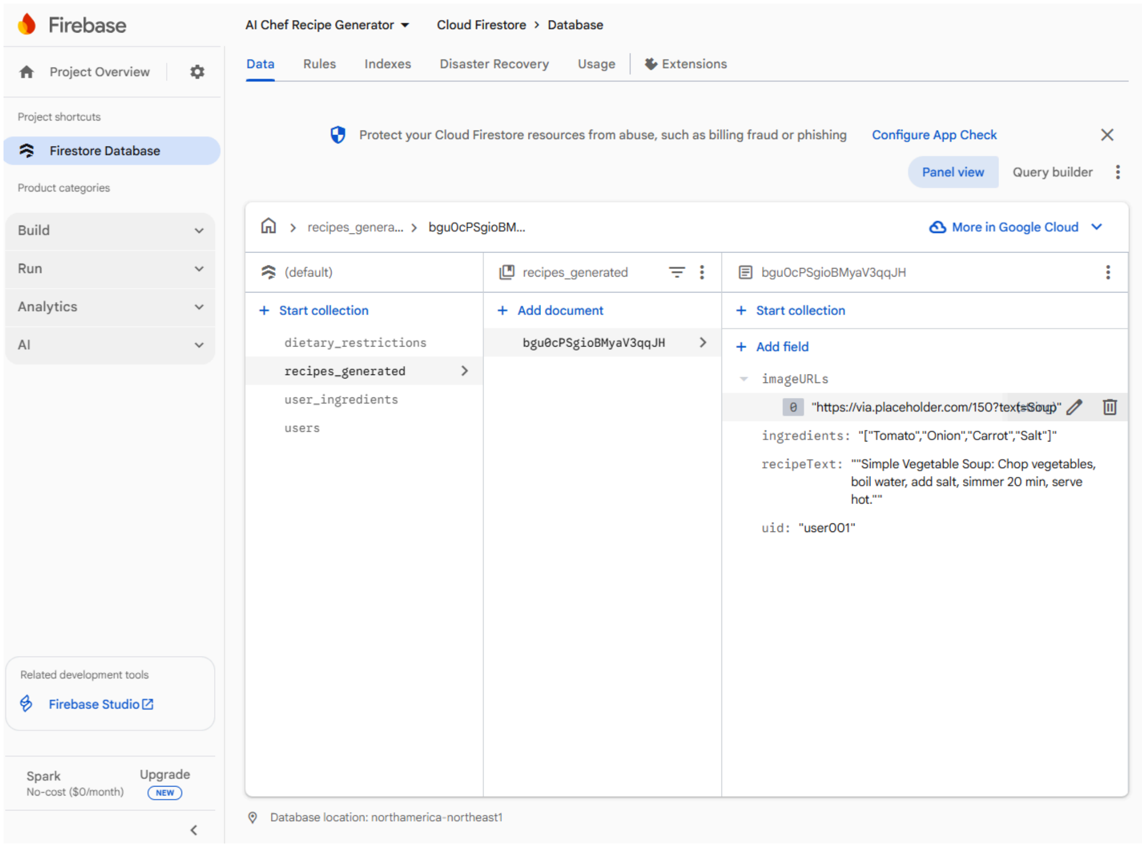
AI Chef Recipe Generator
ai-chef-recipe-generator

1 – 1 of 1

Gemini

Gemini in Firebase

Gemini in Firebase is only available inside of projects


Firebase

AI Chef Recipe Generator › Cloud Firestore › Database

Data | Rules | Indexes | Disaster Recovery | Usage | Extensions

Project Overview

Project shortcuts

Firestore Database

Product categories

Build

Run

Analytics

AI

Protect your Cloud Firestore resources from abuse, such as billing fraud or phishing Configure App Check

Panel view | Query builder

recipes_genera... › bguOcPSgioBM...

More in Google Cloud

(default)	recipes_generated	bguOcPSgioBMyaV3qqJH
+ Start collection	+ Add document	+ Start collection
dietary_restrictions	bgu0cPSgioBMyaV3qqJH	+ Add field
recipes_generated		imageURLs
user_ingredients		0 "https://via.placeholder.com/150?text=Soup"
users		ingredients: "["Tomato","Onion","Carrot","Salt"]"
		recipeText: ""Simple Vegetable Soup: Chop vegetables, boil water, add salt, simmer 20 min, serve hot.""
		uid: "user001"

Related development tools

Firebase Studio

Spark
No-cost ($0/month)

Upgrade NEW

Database location: northamerica-northeast1

AI Integration: Gemini API for recipe generation

Image Integration: Sora API for cooking instructions

Address auto-generation via Canada Post API

Database: Firebase Firestore

Authentication: Firebase Auth (Email/Password, Google, Facebook)

3.4 Testing

- Unit Testing: Frontend components, backend APIs, AI calls

- Integration Testing: Frontend-backend-AI interaction
- Functional Testing: Login, ingredient selection, recipe generation
- User Acceptance Testing (UAT): Validating generated recipes
- Performance Testing: Recipe generation response times

3.5 Deployment

- Hosting Platform: Firebase Hosting Cloud Firestore
- CI/CD Pipeline: GitHub Actions or equivalent
- Monitoring: Firebase Crashlytics or logging tools

3.6 Screens & User Flow

1. Sign In Screen

- Login via Email/Password, Google, or Facebook
- Option to navigate to Sign Up Screen

2. Sign Up Screen

- Register with username/password
- Postal code auto-generates street information via Canada Post API
- After signup, navigates to Landing Page

3. Landing Page Screen

- Options:

    - Make a recipe with current ingredients
    - Make a recipe by taking a photo of ingredients
    - Add new ingredients to personal list
- Leads to Ingredient List Screen

4. Recipe With Ingredients List Screen

- Users select ingredients from generic list or add personal ingredients
- Selected ingredients are highlighted
- Users can remove ingredients if needed
- Proceed to Confirm Ingredients Screen

5. Confirming Ingredient List Screen

- Review selected ingredients in concise format
- Options to remove ingredients or return to add more
- Proceed to Dietary Restrictions Screen

6. Choosing Dietary Restrictions Screen

- Users select dietary restrictions for the recipe
- Proceed to Available Cooking Time Screen

7. Choosing Available Cooking Time Screen

- Users select the time they have to cook
- Proceed to Choosing Cooking Method Screen

3.7 Task Distribution

Team Member	Role & Responsibilities	Key Tasks
Eiad Alsafadi	Backend & AI Integration	- Integrate Front-end with Back-end server - Integrate Gemini API for recipe generation - Integrate Sora API for cooking instruction images - Develop API endpoints (ingredient management, recipe requests, postal code lookup) - Ensure secure API communication and data validation
Oleg Vasiliev	Frontend & UI/UX Design	- Design and implement UI screens (Sign In/Sign Up, Landing Page, Recipe screens) - Connect frontend to backend APIs - Implement responsive design for web/mobile - Ensure smooth navigation and user interactions (ingredients, dietary restrictions, cooking time/method)
Moied Ahmed	Database, Integration & Testing	- Design and manage Firebase Firestore database (users, ingredients, recipes) - Implement Firebase Auth (Email/Password, Google, Facebook) - Conduct unit, integration, and UAT testing - Document APIs, data flow, and SDLC updates - Support remote collaboration and deployment

END OF DELIVERABLE DOCUMENT