# Project Report

# Mobile App Development

## 1. Project Title

App Name: Meal Muse
Team Members: [Komal Eijaz, 261933849] | [Soha Khan, 241556616]
Course: Mobile App Development | Instructor: Umber Nisar
Date: 01/12/2025

## 2. Overview

### Objective:

To develop a full-featured recipe discovery application that transforms household ingredients into personalized meal solutions. MealMuse enables users to intelligently manage their food inventory, receive expiry alerts, and find recipes tailored to their available ingredients and dietary restrictions. The application incorporates an AI-powered chatbot for culinary assistance and includes a complete shopping list system with seamless inventory integration. Built with robust user authentication and administrative controls, this cross-platform solution delivers an intuitive, theme-aware experience designed to reduce food waste and simplify meal planning through comprehensive digital kitchen management.

**Platform(s):** Cross-Platform (Flutter for Android)

**Tools:** Android Studio, Firebase (Auth, Firestore/Realtime DB, Storage), Spoonacular APi, and Grok API

## 3. Features

| Feature | Description | Status |
|---|---|---|
| Add inventory item | Add item with units, category, purchase date, expiry date, notes | Complete |
| Edit inventory item | Edit any field of an existing item | Complete |
| Remove inventory item | Delete item from inventory | Complete |
| Recipe finder | Find recipes using inventory items | Complete |
| Recipe filtering | Filter recipes based on intolerances or restrictions | Complete |

| | | |
|---|---|---|
| Save recipe | Save recipe for later | Complete |
| Add missing recipe items to shopping list | Move missing ingredients to shopping list | Complete |
| Manually add to shopping list | Add custom items to shopping list | Complete |
| Remove from shopping list | Delete items from shopping list | Complete |
| Move shopping list items to inventory | Convert purchased items into inventory items | Complete |
| AI chatbot | Ask AI for help inside the app | Complete |
| Dark/Light mode | Switch between themes | Complete |
| Expiry notifications | Notify user when items are expiring soon | Complete |
| Expiring soon screen | Show items nearing expiry | Complete |
| Language change | Switch app language | Incomplete |
| User registration | Create new account | Complete |
| User login | Login with credentials | Complete |
| Admin login | Login with admin privileges | Complete |
| Admin delete users | Admin can remove users | Complete |
| Admin delete user data | Admin can delete anything from user database | Complete |
| Change username | Update username | Complete |
| Change password | Update password | Complete |
| Logout | Sign out of user account | Complete |
| Reset/Forgot password | Reset password via email | Complete |

## 4. Screenshots/UI

HomeScreen, RecipesList, SavedRecipes, AddItem, Inventory, Recipe, Settings
Material Design principles with custom color schemes
Color-coded expiration system
Responsive layouts for various screen sizes
RTL support for Urdu language
Bottom sheet modals for item editing and filtering

## 5. Tech Stack

Frontend: Flutter
Backend: Firebase (Auth, Firestore).
API: **Spoonacular** (https://spoonacular.com/food-api), **Grok** (https://grok-api.apidog.io/)

## 6. Challenges & Solutions

### 1. Data Type Standardisation Issue

**Issue:**
Inconsistent data types when displaying items due to different input sources saving values differently.
**Solution:**
Implemented conversion logic to standardise all data types before displaying on screen.

### 2. App Scalability & Database Structure

**Issue:** Maintaining scalability without breaking existing features as more functionality was added.
**Solution:** Standardised data types and enforced a consistent Firestore database structure to keep future additions stable.

### 3. Recipe Filtering API Constraint Issue

**Issue:** Recipe filtering API failed due to overly strict request constraints.
**Solution:** Removed unnecessary constraints to ensure proper API responses.

### 4. Inventory Unit Conversion Logic

**Issue:** Incorrect unit conversion and quantity merging when adding items into inventory.
**Solution:** Implemented a conversion and addition system to correctly combine units before saving.

### 5. UI Responsiveness Across Screen Sizes

**Issue:** UI elements did not scale properly on devices with different screen sizes.
**Solution:** Implemented responsive layouts that automatically adjust based on screen dimensions.

### 6. Duplicate Recipe Saves

**Issue:** Users could save the same recipe multiple times, cluttering saved recipes.
**Solution:** Added a Firestore query check for existing recipeId and disabled the Save button with a “Saved” state.

### 7. Theme Preference Not Persisting

**Issue:** Theme (dark/light) reset to default after app restart.
**Solution:** Integrated shared_preferences with ThemeProvider to save theme locally and load it on app startup.

### 8. Language Switching Without Restart

**Issue:** Language change required app restart for changes to apply.
**Solution:** Implemented a LocaleProvider using ChangeNotifier so setLocale() rebuilds the UI instantly with updated translations.

## 7. Learning Outcomes

- Gained hands-on experience with Flutter, Firebase integration, state management, and handling real-time data.
- Improved skills in API integration, debugging, and solving data consistency issues across multiple features.
- Applied clean architecture principles to ensure scalability and maintainability as the app grew.
- Strengthened understanding of UI/UX considerations, theme management, and multi-language support.

- Practiced version control using Git, maintaining organised commits and structured project workflow.

## 8. Future Work

- Complete and polish multi-language translation across all screens
- Add collaboration and shared inventory features for households or groups
- Improve offline mode with full local caching and seamless sync
- Enhance recipe recommendations with smarter filtering and personalised suggestions
- Add analytics to track usage patterns and optimise user experience

## 9. Links

GitHub Repo: https://github.com/keijaz/MealMuseApp

## 10. Conclusion

Summary: MealMuse is an intelligent kitchen assistant that changes how users interact with their food inventory. By seamlessly connecting ingredient tracking, expiry management, and personalized recipe discovery, the app helps reduce food waste while simplifying meal planning. With features like dietary filtering, AI-powered culinary guidance, and integrated shopping lists, MealMuse provides a comprehensive solution for modern kitchen management. The clean, scalable architecture ensures an intuitive user experience while supporting future enhancements and expansion.

Team Contribution:

**Komal Eijaz:**

- Database Integration
- Authorization and Authentication
- Login, Register, and Forgot Password
- Add Users to database
- Add/Remove/Edit Items in Inventory
- Add inventory Items to database
- Spoonacular API integration to Find Recipes

- Recipe Filtering for dietary restrictions and preferences
- Grok AI API implementation and integration for Chatbot
- Save Recipes implementation and integration with Firebase
- Shopping List implementation and integration
- Move items from shopping list to inventory
- Search functionality for recipes and inventory items
- All Admin functionalities, UI, and logic
- App UI

**Soha Khan:**

- UI
- Firebase
- CRUD
- Notifications
- Scheduling and Logic
- Theme System
- Language Localization
- State Management
- Preference Retention
- Navigation
- Integration

## 11. Declaration

We confirm this is our original work.
Signatures: KomalEijaz, SohaKhan
Date: 01/12/2025

## Attachments

GitHub Repo: https://github.com/keijaz/MealMuseApp

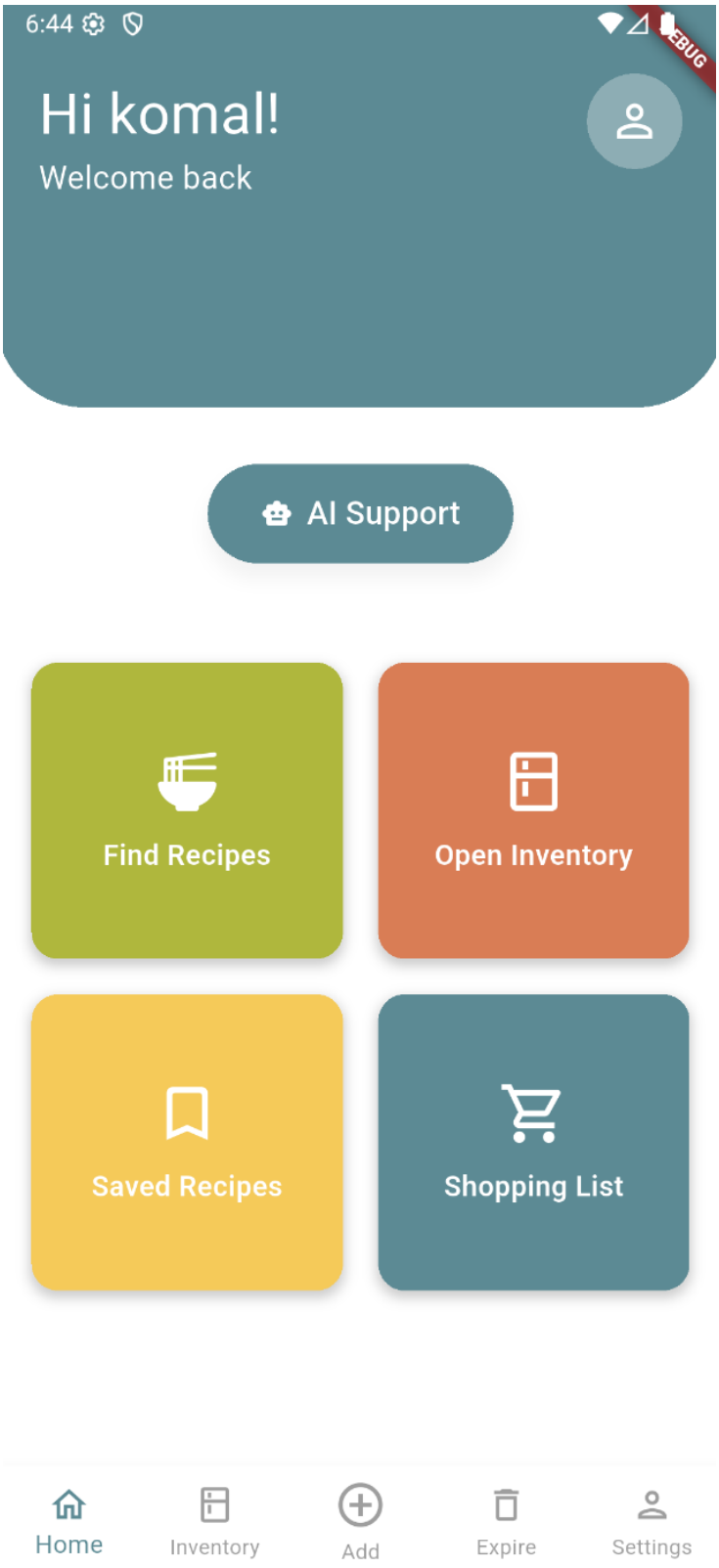


Fig 1.0: HomeScreen

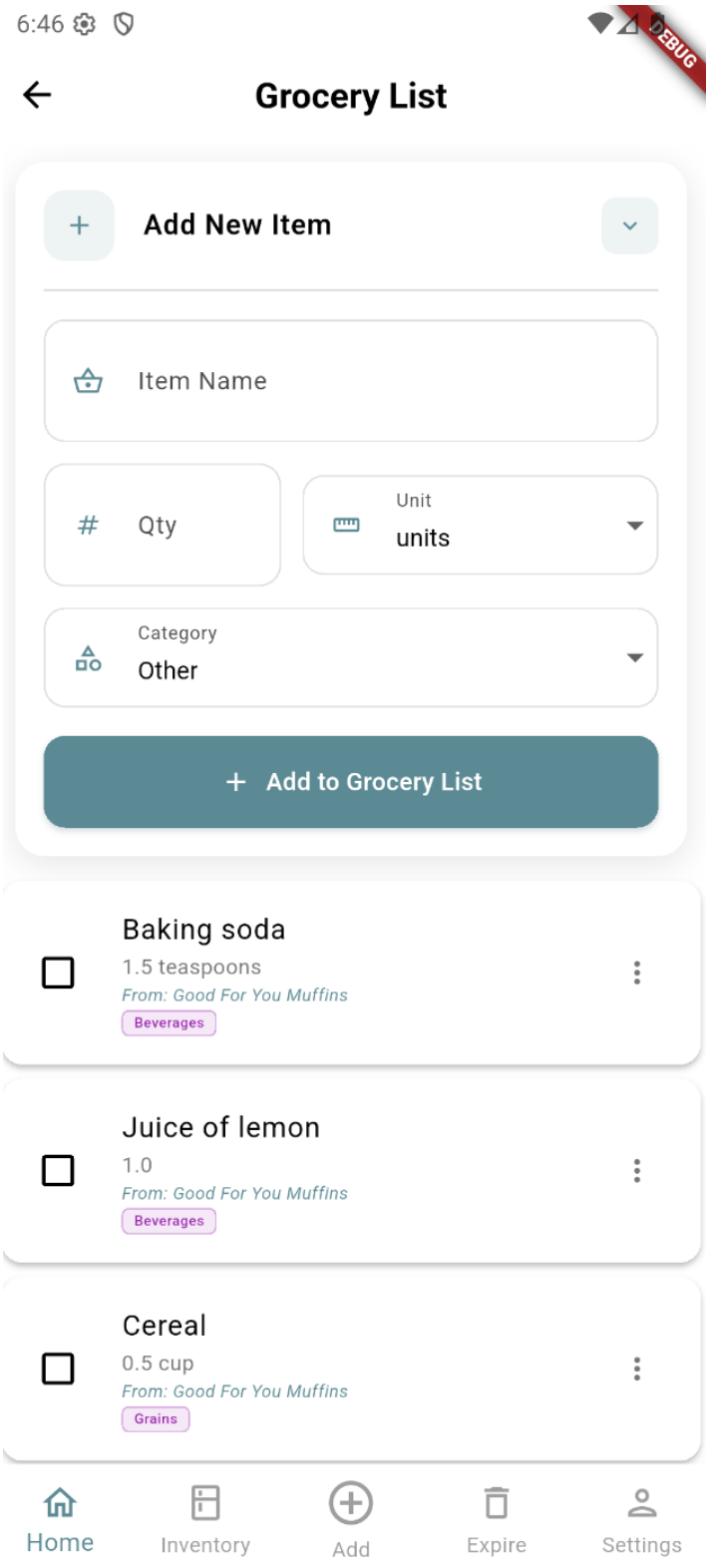


fig 1.1: Grocery List Screen

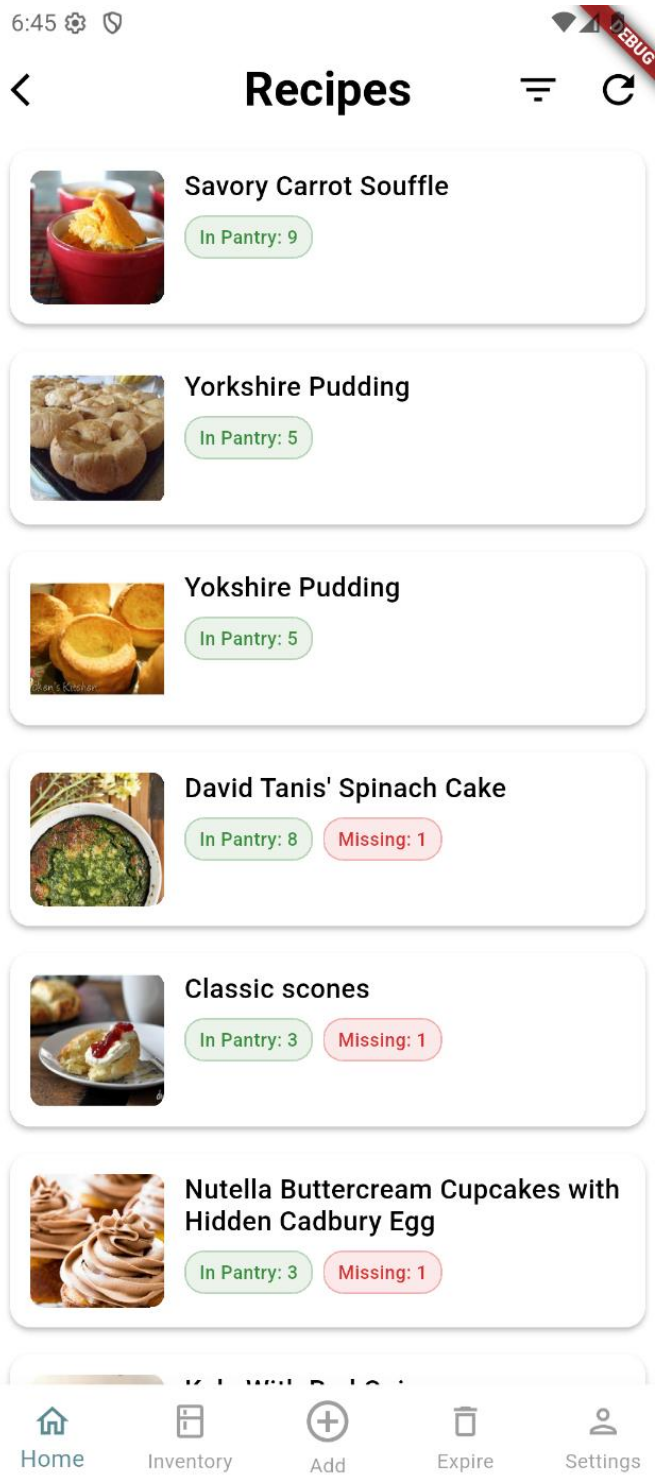


fig 1.2: Recipe Screen

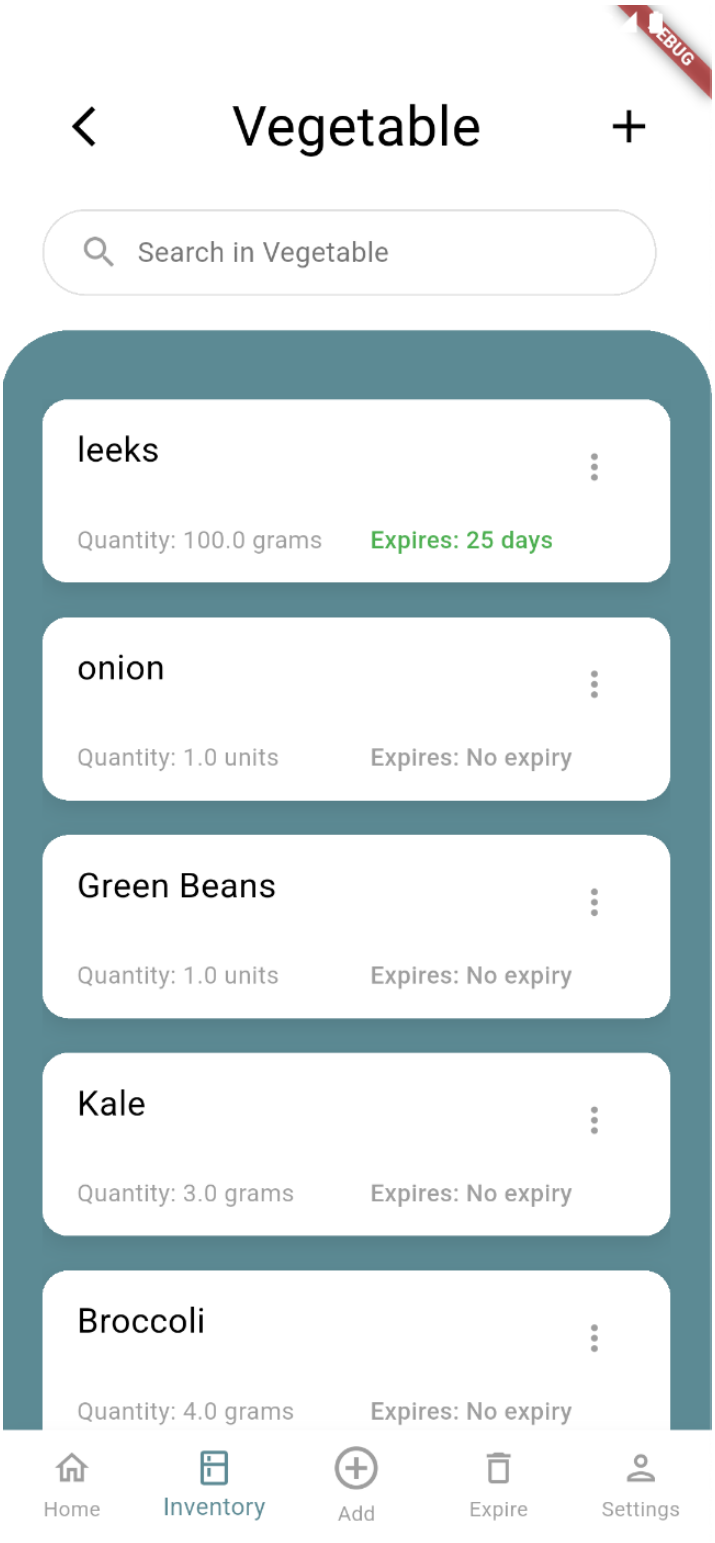


fig 1.3: Inventory -> Vegetable Screen

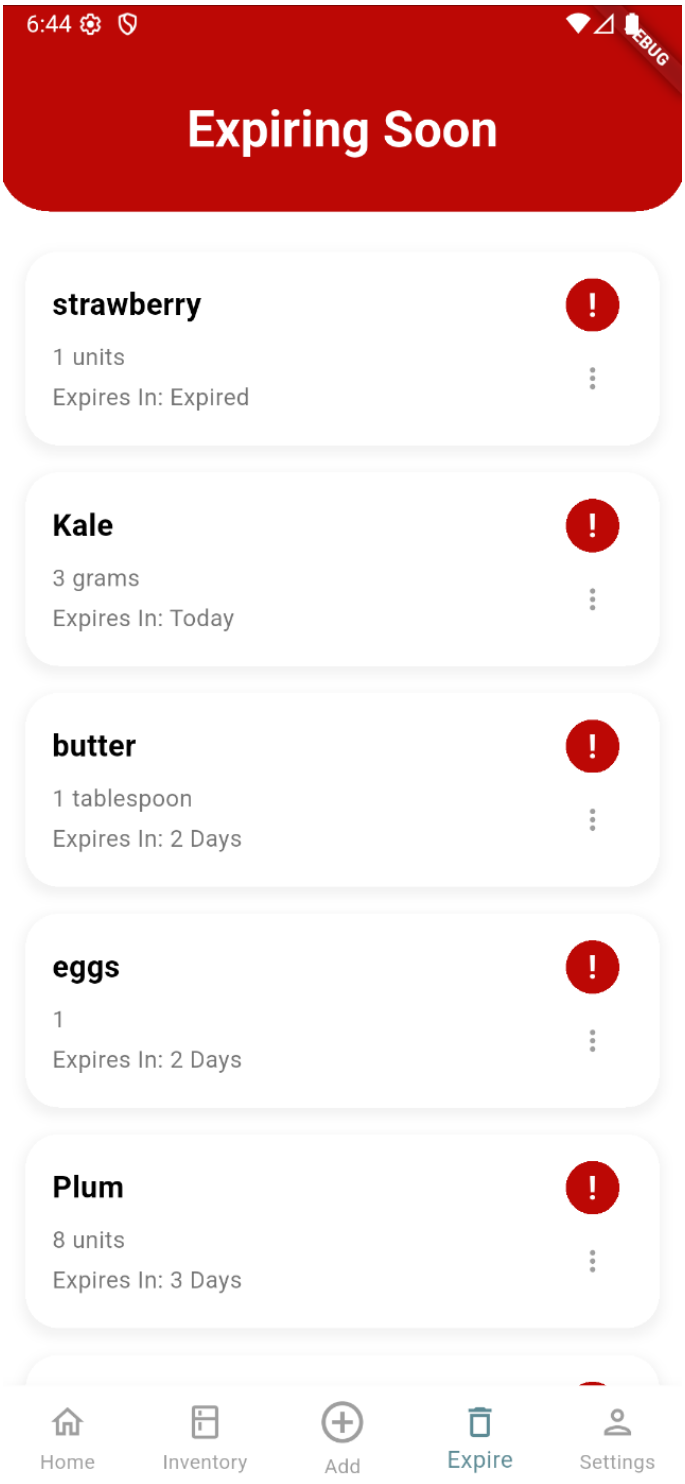


fig 1.4: Expiring Soon Screen

```dart
void _sendMessage() {
  final text = _textController.text.trim();
  if (text.isEmpty) return;

  // Add user message
  final userMessage = ChatMessage(
    text: text,
    isUser: true,
    timestamp: DateTime.now(),
  ); // ChatMessage
  setState(() {
    _messages.add(userMessage);
  });

  _textController.clear();
  _scrollToBottom();

  // Set loading state
  setState(() {
    _isLoading = true;
  });

  // Get AI response from Groq
  _groqService.generateResponse(text).then((response) {
    final aiMessage = ChatMessage(
      text: response,
      isUser: false,
      timestamp: DateTime.now(),
    ); // ChatMessage
    setState(() {
      _messages.add(aiMessage);
      _isLoading = false;
    });
    _scrollToBottom();
  }).catchError((e) {
    final errorMessage = ChatMessage(
      text: 'Error: $e',
      isUser: false,
      timestamp: DateTime.now(),
    ); // ChatMessage
    setState(() {
      _messages.add(errorMessage);
      _isLoading = false;
    });
    _scrollToBottom();
  });
}
```

fig 2.0: AI chatbot API integration code snippet

```dart
Future<void> _checkIfRecipeSaved() async {
  try {
    final user = FirebaseAuth.instance.currentUser;
    if (user == null) return;

    // Check both 'id' (from API) and 'recipeId' (from saved recipes)
    final recipeId = widget.recipe['id'] ?? widget.recipe['recipeId'];
    if (recipeId == null) return;

    final existingRecipeQuery = await FirebaseFirestore.instance
        .collection('users')
        .doc(user.uid)
        .collection('saved_recipes')
        .where('recipeId', isEqualTo: recipeId)
        .get();

    setState(() {
      _isSaved = existingRecipeQuery.docs.isNotEmpty;
    });
  } catch (e) {
    print('Error checking if recipe is saved: $e');
  }
}

Future<void> _fetchRecipeDetails() async {
  try {
    // Use the recipeId from the constructor
    final recipeId = widget.recipeId;
    if (recipeId.isEmpty) {
      throw Exception('Recipe ID not found');
    }

    // Convert String to int for the API calls
    final int recipeIdInt = int.tryParse(recipeId) ?? 0;
    if (recipeIdInt == 0) {
      throw Exception('Invalid Recipe ID: $recipeId');
    }

    // Fetch recipe details and instructions
    final details = await ApiService.fetchRecipeDetails(recipeIdInt);
    final instructions = await ApiService.fetchRecipeInstructions(recipeIdInt);

    // Calculate ingredient status
    _calculateIngredientStatus(details['extendedIngredients'] ?? []);

    setState(() {
      _recipeDetails = details;
      _recipeInstructions = instructions;
      _isLoading = false;
    });
  } catch (e) {
```

fig 2.1: Saved Recipes Screen code snippet

```dart
  // Fetch recipes using complexSearch with filters
  static Future<List<dynamic>> fetchRecipesWithComplexSearch({
  required String sort,
  List<String> diets = const [],
  List<String> intolerances = const [],
  int number = 10,
}) async {
  try {
    // Check internet connection first
    final hasConnection = await hasInternetConnection();
    if (!hasConnection) {
      throw Exception('No internet connection. Please check your network settings.');
    }

    // Get ingredients from inventory for query
    final ingredients = await getInventoryIngredients();

    // Build query parameters
    final params = <String, String>{
      'number': number.toString(),
      'apiKey': _apiKey,
      'sort': sort,
      'addRecipeInformation': 'true',
      'fillIngredients': 'true',
      'ranking': '2', // Maximize used ingredients
    };

    // Add diet filter if selected
    if (diets.isNotEmpty) {
      params['diet'] = diets.first;
      print('Applied diet filter: ${diets.first}');
    }

    // Add intolerance filter if selected
    if (intolerances.isNotEmpty) {
      params['intolerances'] = intolerances.join(',');
      print('Applied intolerances: ${intolerances.join(',')}');
    }

    // STRATEGY 1: Try with inventory ingredients first
    if (ingredients.isNotEmpty) {
      // Use a reasonable number of ingredients
      final uniqueIngredients = ingredients.toSet().take(8).toList();
      params['includeIngredients'] = uniqueIngredients.join(',');
      print('Trying with inventory ingredients: ${uniqueIngredients.join(', ')}');
    }

    // Build the URL
    final url = Uri.parse('$_baseUrl/recipes/complexSearch').replace(
      queryParameters: params,
    );
```

fig 3.2: Spoonacular API integration code snippet