Project Report Template

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: Build a pantry management app where users can enter their pantry ingredients and find recipes based on those ingredients. Users are also be able to create a shopping list through either manual addition of items or moving missing items from recipes into their shopping list. There is also an expiry section that notifies users of ingredients expiring soon.

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/Dart
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/MealMuse/tree/final

10. Conclusion

Summary: This app helps users manage their food inventory efficiently by tracking items, monitoring expiry dates, organising shopping lists, and discovering recipes based on what they already have. It also includes an AI assistant, admin controls, and a clean, scalable design to support future feature 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 idea, name, logo, and logic
- App UI

Soha Khan:

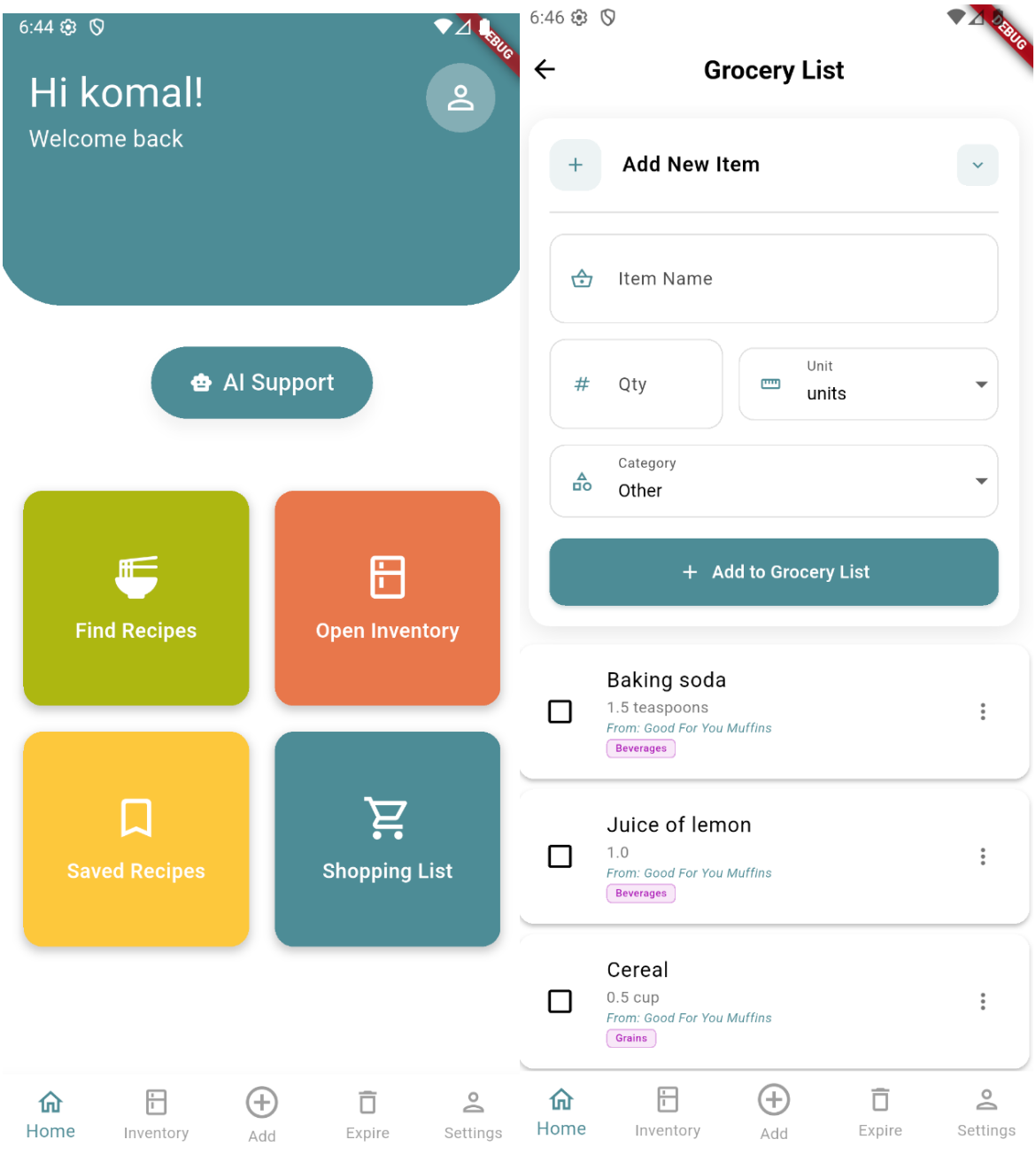
- UI
- Firebase
- CRUD
- Notifications
- Scheduling and Logic
- Theme System
- Language Localization
- State Management
- Preference Retention

11. Declaration

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

Attachments

GitHub Repo: https://github.com/keijaz/MealMuse/tree/final
Screenshots: Next page


6:44
DEBUG
Hi komal!
Welcome back
AI Support
Find Recipes
Open Inventory
Saved Recipes
Shopping List
Home
Inventory
Add
Expire
Settings
6:46
DEBUG
Grocery List
Add New Item
Item Name
Qty
Unit
units
Category
Other
Add to Grocery List
Baking soda
1.5 teaspoons
From: Good For You Muffins
Beverages
Juice of lemon
1.0
From: Good For You Muffins
Beverages
Cereal
0.5 cup
From: Good For You Muffins
Grains
Home
Inventory
Add
Expire
Settings

6:45

DEBUG

Recipes

Savory Carrot Souffle

In Pantry: 9

Yorkshire Pudding

In Pantry: 5

Yokshire Pudding

In Pantry: 5

David Tanis' Spinach Cake

In Pantry: 8 Missing: 1

Classic scones

In Pantry: 3 Missing: 1

Nutella Buttercream Cupcakes with Hidden Cadbury Egg

In Pantry: 3 Missing: 1

Home Inventory Add Expire Settings

DEBUG

Vegetable

Search in Vegetable

leeks

Quantity: 100.0 grams Expires: 25 days

onion

Quantity: 1.0 units Expires: No expiry

Green Beans

Quantity: 1.0 units Expires: No expiry

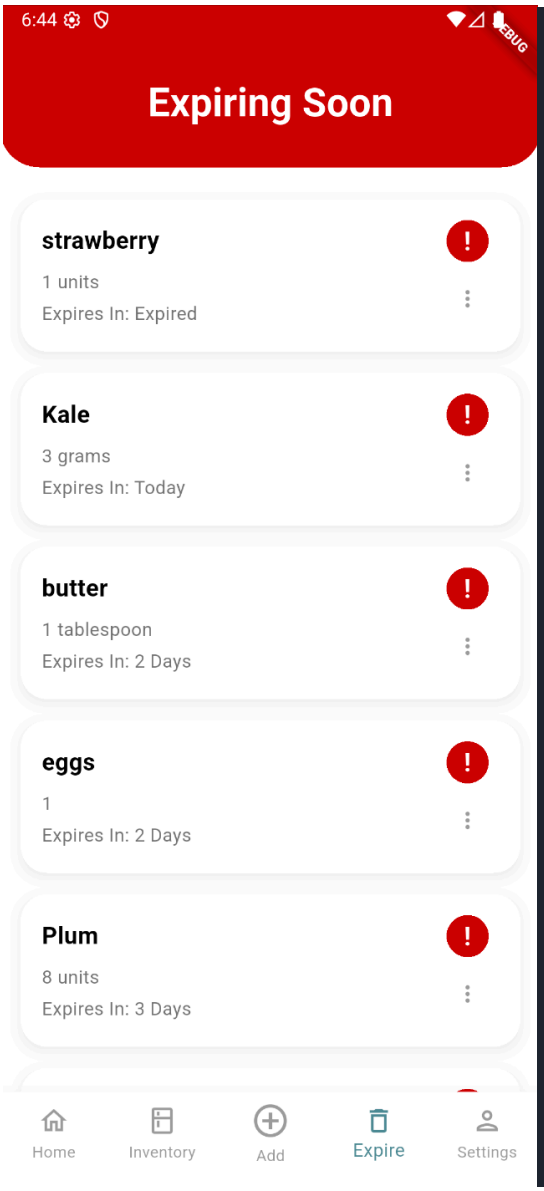
Kale

Quantity: 3.0 grams Expires: No expiry

Broccoli

Quantity: 4.0 grams Expires: No expiry

Home Inventory Add Expire Settings




```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();
  });
}
```

```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) {
```

```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,
    );
```