Software Requirements Specification (SRS) Document

Project: Pick My Dish – Recipe & Cooking Assistant

Version: 1.0

Development Team

November 2025

Abstract

This document outlines the software requirements for **Pick My Dish**, a Flutter-based mobile application that provides personalized recipe recommendations based on user preferences, ingredients, cooking time, and mood. It serves as a comprehensive guide for developers, testers, project managers, and stakeholders.

Approval Signatures

Product Owner	**Project Manager**
Name:KAMDEU YAMDJEUSON NEIL MARSHALL Date:1/11/2025	Name:TUHEU TCHOUBI PEMPEME MOUSSA FAHDIL Date:1/11/2025

Contents

1 Introduction

1.1 Purpose

This document outlines the software requirements for **Pick My Dish**, a Flutter-based mobile application that provides personalized recipe recommendations based on user preferences, ingredients, cooking time, and mood. It serves as a comprehensive guide for developers, testers, project managers, and stakeholders.

1.2 Scope

Pick My Dish is a cross-platform mobile application that allows users to:

- Browse, search, and filter recipes.
- Upload, edit, and delete their own recipes.
- Personalize recipe recommendations based on ingredients, cooking time, and emotions.
- Mark recipes as favorites.
- Manage user profiles with profile pictures and usernames.
- View detailed recipe instructions, ingredients, and nutritional information.

The system interfaces with a RESTful backend server (Node.js + MySQL) for data storage and retrieval. The app is developed using **Flutter** and follows the **MVVM architecture** with **Provider** for state management.

1.3 Definitions, Acronyms, and Abbreviations

Term	Description
API	Application Programming Interface
UI	User Interface
SRS	Software Requirements Specification
MVVM	Model-View-ViewModel
JWT	JSON Web Token (for authentication)
VPS	Virtual Private Server
SQLite	Local database for offline storage
HTTP	Hypertext Transfer Protocol
FR	Functional Requirement
NFR	Non-Functional Requirement

Table 1: Definitions and Acronyms

1.4 References

- Flutter Documentation: `https://flutter.dev`
- Provider Package: `https://pub.dev/packages/provider`
- REST API Design Guidelines
- Project Source Code Documentation

1.5 Overview

This document is structured into sections that describe:

- Overall system functionality
- User roles and characteristics
- Functional and non-functional requirements
- Interface requirements
- Constraints and assumptions

2 Overall Description

2.1 Product Perspective

Pick My Dish is an independent mobile application that communicates with a backend server hosted on a VPS. The app is built using **Flutter** for cross-platform compatibility (iOS & Android). It uses **SQLite** for local caching and **HTTP** for server communication.

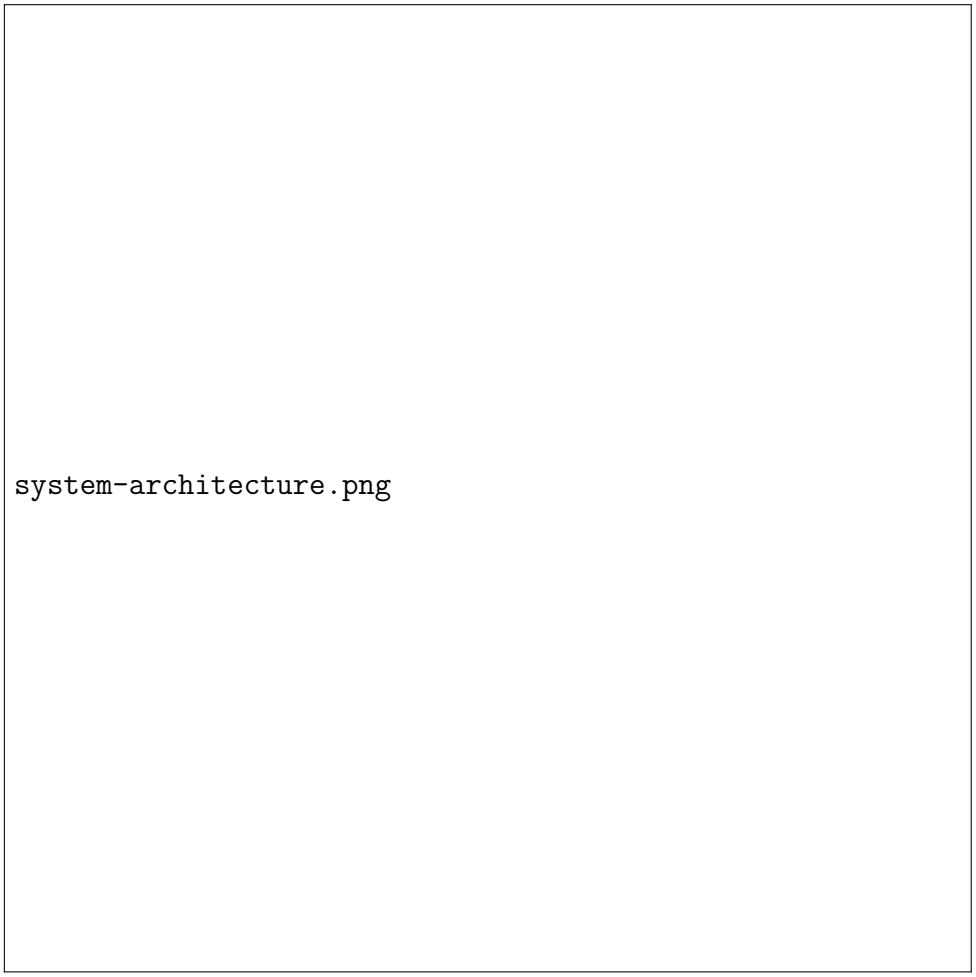

Figure 1: System Architecture Overview

2.2 Product Functions

The primary functions of the system include:

1. **User Authentication**: Register, login, and guest access.
2. **Recipe Management**: Browse, search, filter, upload, edit, and delete recipes.
3. **Personalization**: Generate recipe recommendations based on selected ingredients, cooking time, and mood.
4. **Favorites Management**: Add/remove recipes to/from favorites.
5. **User Profile Management**: Update username and profile picture.
6. **Admin Features**: Edit/delete any recipe (admin role).
7. **Offline Access**: Cached recipes and images for offline viewing.

2.3 User Characteristics

User Type	Description
End Users	Home cooks, cooking enthusiasts, beginners. Regular users who browse, cook, and share recipes.
Admin Users	Content moderators or administrators with extended privileges to manage all recipes and users.
Guests	Users without an account. Have limited access to browse recipes but cannot save favorites or upload recipes.

Table 2: User Characteristics

2.4 Constraints

- Must support **iOS 12+** and **Android 8.0+**.
- Backend server must be reachable via public IP.
- Maximum image upload size: **5 MB**.
- Must comply with GDPR for user data (if applicable).
- Development must follow Flutter best practices and Dart style guide.

2.5 Assumptions and Dependencies

- Users have a stable internet connection for real-time data sync.
- Backend server is maintained and available with 99% uptime.
- Users have basic familiarity with mobile applications.
- The app will be distributed via Google Play Store and Apple App Store.
- Third-party packages used are maintained and compatible with Flutter updates.

3 System Features and Requirements

3.1 Functional Requirements

ID	Requirement Description	Priority
FR1	Users can register with email, username, and password with validation	High
FR2	Users can log in with email/password or as guest	High
FR3	Users can view a list of recipes with images, names, and cooking times	High
FR4	Users can search recipes by name, category, or mood	Medium
FR5	Users can filter recipes by ingredients, cooking time, and mood	Medium
FR6	Logged-in users can upload new recipes with image, ingredients, steps, and mood tags	High
FR7	Recipe owners or admins can edit recipe details	Medium
FR8	Recipe owners or admins can delete recipes	Medium
FR9	Users can add/remove recipes to/from favorites	High
FR10	Users can update username and profile picture	Medium
FR11	Users can generate personalized recipes based on selected criteria	High
FR12	Users can view cached recipes without internet connection	Low
FR13	Admins can edit/delete any recipe	Low
FR14	System validates email format and password strength during registration	Medium
FR15	System provides ingredient selector with search and add-new functionality	Medium
FR16	System caches images for offline viewing	Low

Table 3: Functional Requirements

3.2 Non-Functional Requirements

ID	Requirement Description
NFR1	**Performance**: App should load recipes within **2 seconds**. Image caching should reduce bandwidth usage.
NFR2	**Usability**: Intuitive UI with consistent navigation, clear icons, and responsive design.
NFR3	**Reliability**: 99% uptime for backend server; graceful error handling with user-friendly messages.
NFR4	**Security**: Passwords hashed using bcrypt; JWT for session management; input validation to prevent injection attacks.
NFR5	**Compatibility**: Supports Flutter **3.9+**, iOS 12+, Android 8.0+, various screen sizes and resolutions.
NFR6	**Scalability**: Backend supports up to **10,000 concurrent users** with horizontal scaling capability.
NFR7	**Maintainability**: Modular code structure with clear separation of concerns (Models, Providers, Services, Screens).
NFR8	**Availability**: System available 24/7 with maintenance windows announced in advance.
NFR9	**Portability**: Single codebase runs on both iOS and Android without platform-specific modifications.
NFR10	**Testability**: Comprehensive unit and integration test coverage; testable architecture patterns.

Table 4: Non-Functional Requirements

4 External Interface Requirements

4.1 User Interfaces

The application includes the following main screens:

- **Splash Screen**: Displays app logo and brief loading animation (3 seconds).
- **Login/Register Screens**: Form-based authentication with validation and password strength indicator.
- **Home Screen**: Personalized recommendations, today's recipes, side menu navigation.
- **Recipe Screens**: List view, grid view, and detailed recipe view with step-by-step instructions.
- **Profile Screen**: User information, edit options, profile picture management, logout.
- **Upload/Edit Screens**: Comprehensive forms for recipe data entry, image picker, and ingredient selector.
- **Favorites Screen**: Grid/list of favorited recipes with swipe-to-remove functionality.

UI Design Principles:

- Consistent color scheme (black/orange/white) $\times Times New Roman font family for titles and text$

- Responsive layouts for various screen sizes
- Intuitive navigation with clear visual hierarchy

4.2 Hardware Interfaces

- **Camera/Gallery**: For uploading profile and recipe images via `image_picker` package.
- **Internet Connectivity**: Wi-Fi or mobile data for API calls and real-time updates.
- **Storage**: Local device storage for caching images and SQLite database.
- **Memory**: Minimum 1GB RAM recommended for smooth operation.

4.3 Software Interfaces

Component	Description
Operating System	iOS 12+, Android 8.0+ via Flutter framework
Backend API	RESTful endpoints hosted at `http://38.242.246.126:3000` (or localhost for development)
Local Database	SQLite for caching recipes and user preferences
Third-Party Libraries	• `cached_network_image`: Image caching • `image_picker`: Photo selection • `http`: API communication • `provider`: State management • `sqflite`: SQLite database • `intl`: Internationalization • `flutter_cache_manager`: Cache management
Development Tools	Flutter SDK, Dart, Android Studio, Xcode

Table 5: Software Interfaces

4.4 Communication Interfaces

- **Protocol**: HTTP/HTTPS for all client-server communication.
- **Data Format**: JSON for request/response payloads.
- **Authentication**: JWT tokens sent in Authorization header.
- **Error Handling**: Standard HTTP status codes with descriptive error messages.
- **Rate Limiting**: Implemented on server-side to prevent abuse.

5 Other Non-Functional Requirements

5.1 Performance Requirements

Metric	Requirement	Target
App Startup Time	Time from launch to home screen display	¡ 3 seconds
Recipe List Loading	Time to load and display recipe list	2 seconds
Image Loading	Time to load cached images	¡ 1 second
API Response Time	Server response time for API calls	¡ 500 ms
Database Operations	Local SQLite read/write operations	¡ 100 ms
Search Response	Time to filter/search recipes	¡ 1 second

Table 6: Performance Requirements

5.2 Security Requirements

- **Authentication**:
  - Password hashing using bcrypt algorithm
  - JWT token-based authentication with 24-hour expiration
  - Secure token storage using Flutter Secure Storage
- **Authorization**:
  - Role-based access control (User, Admin)
  - Recipe ownership validation for edit/delete operations
  - Server-side authorization checks for all protected endpoints
- **Data Protection**:
  - HTTPS for all network communications
  - Input validation and sanitization
  - SQL injection prevention via parameterized queries
  - File upload validation (type, size, virus scanning)
- **Privacy**:
  - GDPR compliance for EU users
  - Clear privacy policy and data usage terms
  - User data encryption at rest

5.3 Software Quality Attributes

- **Maintainability**:
  - Clean architecture with separation of concerns
  - Comprehensive code documentation
  - Modular design with reusable components
  - Consistent coding standards (Dart style guide)
- **Testability**:
  - Unit tests for models and services
  - Integration tests for UI components

  - Mock server for API testing
  - Test coverage target: 80%+

- **Portability**:
  - Single codebase for iOS and Android
  - Responsive design for various screen sizes
  - Platform-specific adaptations where necessary

- **Reliability**:
  - Graceful error handling and recovery
  - Offline capability with data synchronization
  - Automatic retry for failed network requests
  - Comprehensive logging and monitoring

- **Usability**:
  - Intuitive navigation and workflow
  - Consistent UI patterns and components
  - Accessibility features (font scaling, contrast)
  - Clear error messages and user guidance

6 Appendices

6.1 Appendix A: Glossary

Term	Definition
Recipe Model	Data structure containing recipe details (name, ingredients, steps, cooking time, calories, etc.)
Provider	State management class in Flutter that notifies listeners of changes (e.g., RecipeProvider, UserProvider)
IngredientSelector	Custom widget for selecting ingredients from a searchable list with add-new capability
CachedProfileImage	Widget for displaying cached network images or local assets with fallback UI
ApiService	Service class containing all API communication methods
DatabaseService	Service class for local SQLite database operations
ChangeNotifier	Flutter class that provides change notification to widgets
MultiProvider	Widget that provides multiple providers to the widget tree

Table 7: Technical Glossary

6.2 Appendix B: Key Source Code Structure

Listing 1: Main Application Structure

```
lib/
        main.dart                    # Application entry point
        constants.dart               # App constants and styles
        Models/
            recipe_model.dart          # Recipe data model
            user_model.dart            # User data model
        Providers/
            recipe_provider.dart       # Recipe state management
            user_provider.dart         # User state management
        Screens/
            splash_screen.dart         # Splash screen
            login_screen.dart          # Login screen
            register_screen.dart       # Registration screen
            home_screen.dart           # Home screen
            profile_screen.dart        # Profile screen
            recipe_screen.dart         # Recipe list screen
            recipe_detail_screen.dart# Recipe details
            recipe_upload_screen.dart# Recipe upload
            recipe_edit_screen.dart    # Recipe editing
            favorite_screen.dart       # Favorites screen
        Services/
            api_service.dart           # API communication
            database_service.dart      # Local database
        Widgets/
```

```
cached_image.dart          # Cached image widget
ingredient_selector.dart # Ingredient selector
```

6.3 Appendix C: Database Schema

Table	Description
Users	`id, username, email, password_hash, profile_image_path, is_admin, created_at`
Recipes	`id, name, category, cooking_time, calories, image_path, ingredients, steps, emotions, user_id, created_at`
Ingredients	`id, name, created_at`
Favorites	`user_id, recipe_id, created_at`
Recipe_Ingredients	`recipe_id, ingredient_id`

Table 8: Database Schema Overview

6.4 Appendix D: API Endpoints Summary

Endpoint	Description	Method
/api/auth/register	User registration	POST
/api/auth/login	User login	POST
/api/recipes	Get all recipes / Upload recipe	GET, POST
/api/recipes/:id	Get/Update/Delete specific recipe	GET, PUT, DELETE
/api/ingredients	Get all ingredients / Add new	GET, POST
/api/users/favorites	Manage favorite recipes	GET, POST, DELETE
/api/users/profile-picture	Update profile picture	PUT
/api/users/username	Update username	PUT
/api/users/{id}/recipes	Get user's recipes	GET

Table 9: Key API Endpoints

6.5 Appendix E: Dependencies

Listing 2: pubspec.yaml Dependencies

```
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.8
  sqflite: ^2.4.2
  path: ^1.9.1
  dropdown_search: ^5.0.2
  provider: ^6.0.5
```

```
image_picker: ^1.0.4
http: ^1.1.0
intl: ^0.18.1
flutter_cache_manager: ^3.3.0
cached_network_image: ^3.3.0
path_provider: ^2.1.0
shared_preferences: ^2.2.2
```

6.6 Appendix F: Testing Requirements

- **Unit Testing**:
  - Test all model classes (Recipe, User)
  - Test API service methods
  - Test provider state changes
  - Test utility functions and helpers
- **Integration Testing**:
  - Test complete user flows (registration → login → home)
  - Test recipe upload and editing workflows
  - Test favorite management
  - Test error scenarios and recovery
- **UI Testing**:
  - Test widget rendering and interactions
  - Test navigation flows
  - Test responsive design on different screen sizes
  - Test accessibility features
- **Performance Testing**:
  - Measure app startup time
  - Test scroll performance with large recipe lists
  - Test memory usage and leak detection
  - Test battery consumption
- **Security Testing**:
  - Penetration testing of API endpoints
  - Data validation and sanitization tests
  - Authentication and authorization tests
  - Secure storage validation

Document Change History

Version	Date	Author	Changes
1.0	Oct 2024	Development Team	Initial Release