Software Design Document

Glucose Genie

Francisco Cruz-Urbanc, Krisi Hristova, Carson Ford, Thomas Capro, Jared Jackson

CI 492 – Senior Project

Filippos Vokolos

February 5, 2025

Version 1

Table of Contents

1 Introduction 3
1.1 Purpose 3
1.2 Scope 3
2 Design Overview 3
2.1 Problem Description 3
2.2 Technologies Used 3
2.3 System Architecture 3
2.4 System Sequence Diagrams 4
2.4.1 User Login and Registration 5
2.4.2 Recipe Search 5
2.4.3 Save Recipe 6
2.4.4 Creating Weekly Meal Plan 6
2.4.5 Generate Grocery List 7
2.4.6 View Grocery Stores 7
2.4.7 Update Settings 8
3 Requirements Traceability 8
4 User Interface 9
5 Data Model and Storage 10
5.1 Data Storage 11
5. 2 Data Model 12
6 References 15

1 Introduction

1.1 Purpose

The purpose of this document is to describe the implementation of the Glucose Genie application software based on the requirements detailed in the Software Requirements Specification [1]. The Glucose Genie application software is designed to assist individuals with diabetes in managing their nutritional intake.

1.2 Scope

This document describes the implementation details of the Glucose Genie application software. The software will consist of three main functions. The first is to find and save recipes, second is to create and track weekly meal plans, and third is to locate nearby grocery stores based on price and grocery lists. This document will not specify the testing of the software.

2 Design Overview

2.1 Problem Description

Diabetic patients need a simple and effective way to track their daily nutrient consumption and set up diabetic friendly meal plans. Glucose Genie will provide a searchable database of diabetes friendly recipes displaying nutrient contents that can be saved and added to weekly meal plans, automatically generate weekly grocery lists.

2.2 Technologies Used

Glucose Genie is a mobile application built targeted for devices running iOS 12-17.2. Development technologies require a minimum of macOS13.5 to be able to run Xcode 15.2 which is compatible with Swift 4, 4.2, 5.9. Devices will communicate with the application through HTTP requests.

2.3 System Architecture

The general principle for iOS app architecture is to have three layers:

- **Presentation Layer:** What the user sees (UI, UX) and how they maneuver around the app.
    - PresentationLogic: all classes used to present data and UI to the user.
    - SwiftUI: Apple-created framework used for UI.

- **Business Logic Layer:** Processes data and prepares it to be presented.
    - BusinessLogic: parent class for subsequent children logic classes.

- AccountLogic: validates login input and prepares feedback to be presented.
- MapLogic: processes data from our map API and prepares it to be presented.
- RecipeLogic: processes data from our recipe API and prepares it to be presented.

- **Data Access Layer:** Accesses external data sources to retrieve information.
  - DataAccess: routes access to respective location.
  - Security: all security measures for user login.
  - AccountDatabase: database with all user logins (username and password).
  - MapAPI: API to be used for our map feature.
  - LocalData: data owned by the user's device (such as location).
  - RecipeAPI: API to be used for our recipe feature.

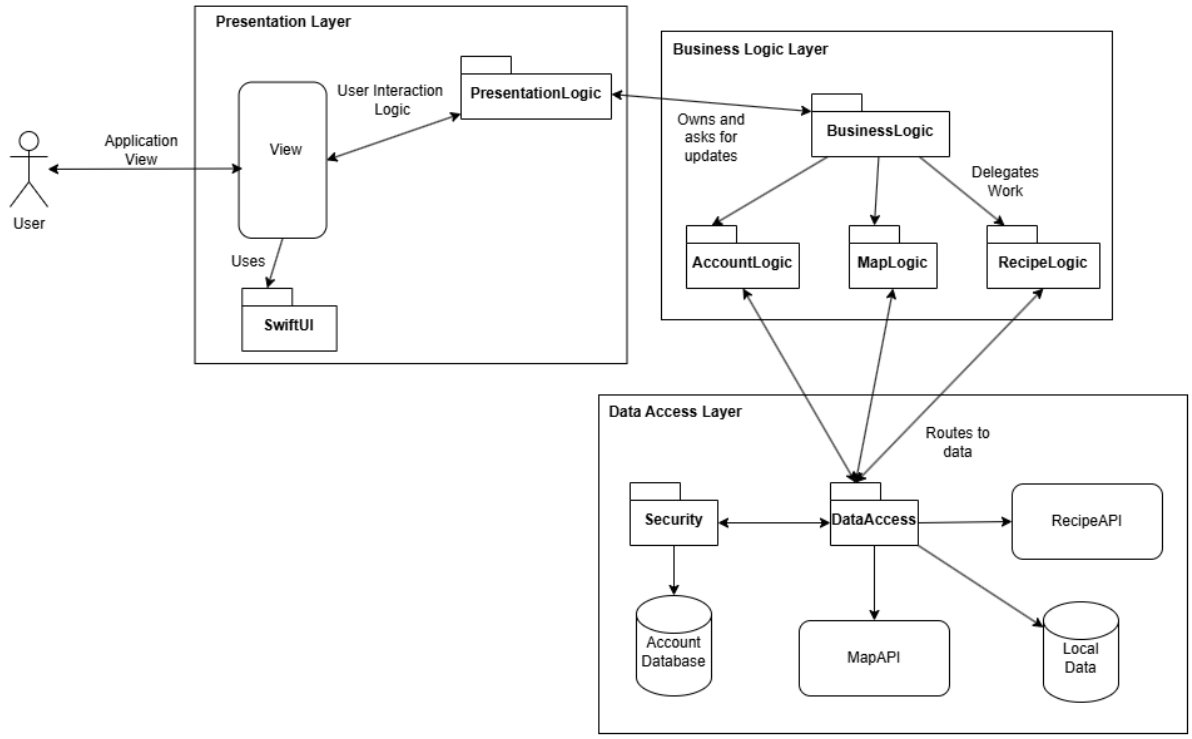

Figure 2.3.1: Glucose Genie Architecture Diagram

2.4 System Sequence Diagrams

The system sequence diagrams (Figures 2.4.1-2.4.7) present the interactions between the user and the software system for different scenarios.

2.4.1 User Login and Registration

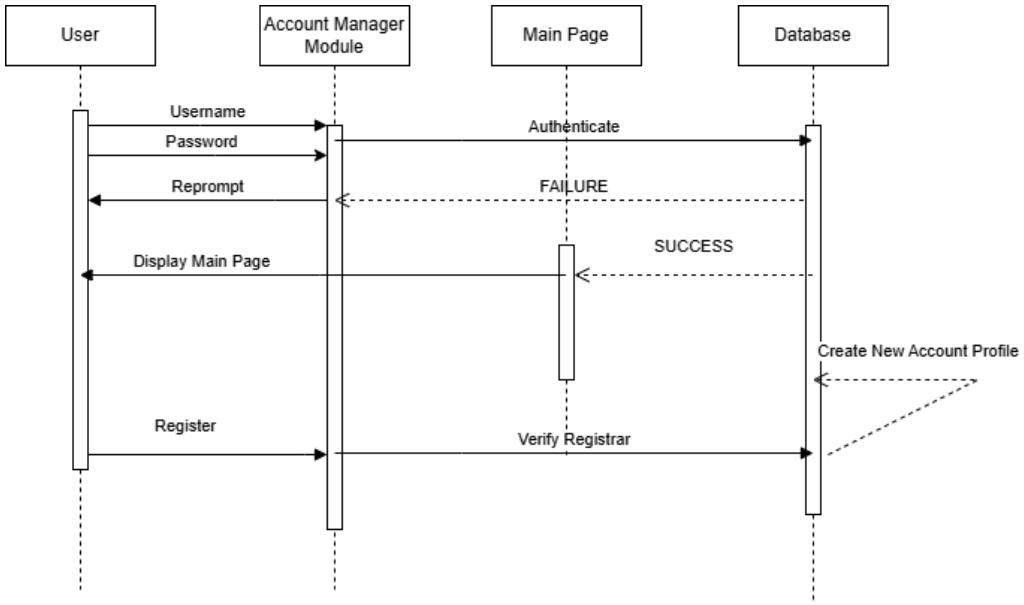

Figure 2.4.1 - Sequence Diagram for User Login and Account Registration

2.4.2 Recipe Search

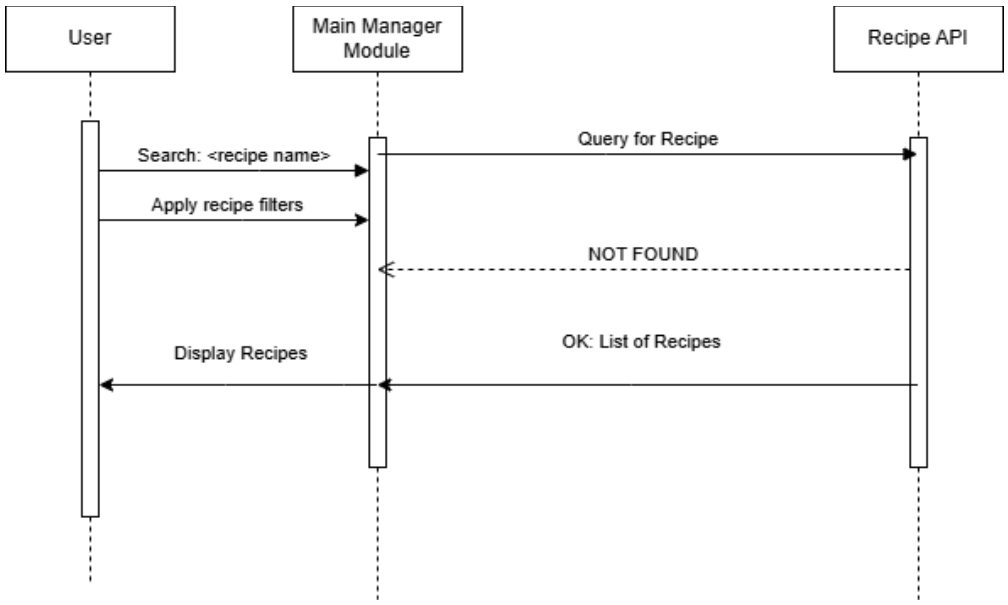

Figure 2.4.2 - Sequence Diagram for Recipe Search

2.4.3 Save Recipe

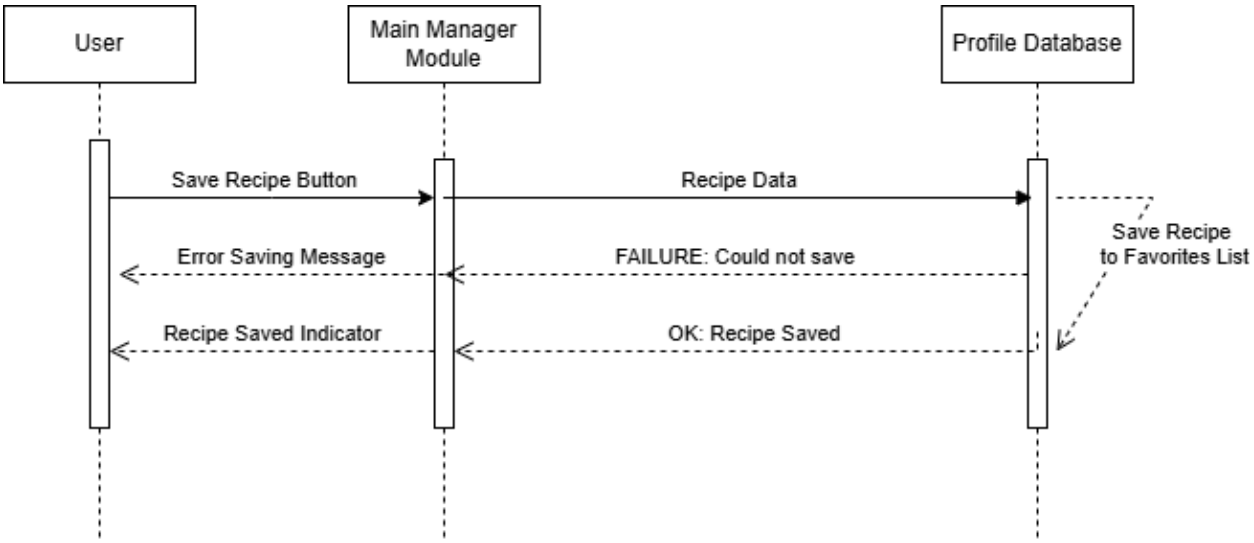

Figure 2.4.3 - Sequence Diagram for Recipe Saving

2.4.4 Creating Weekly Meal Plan

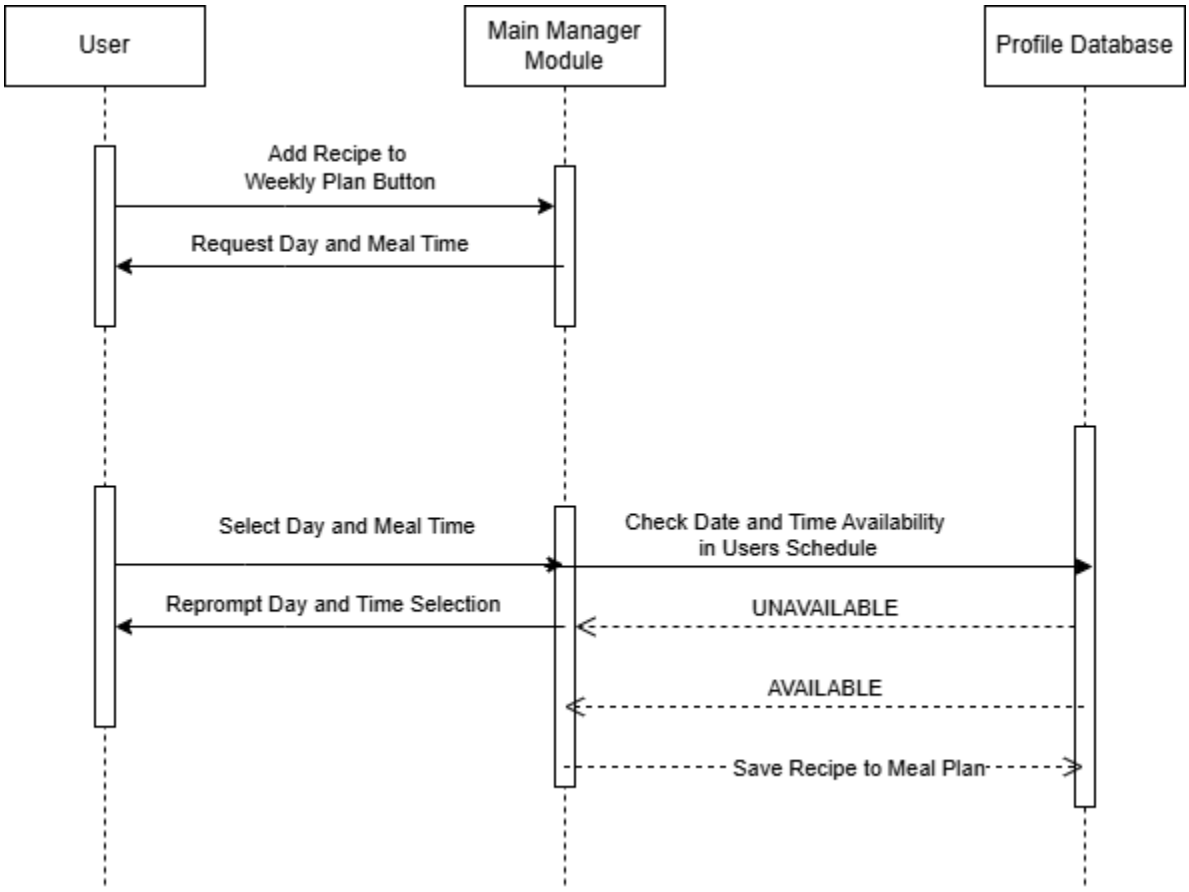

Figure 2.4.4 - Sequence Diagram for User Creating a Weekly Meal Plan

2.4.5 Generate Grocery List

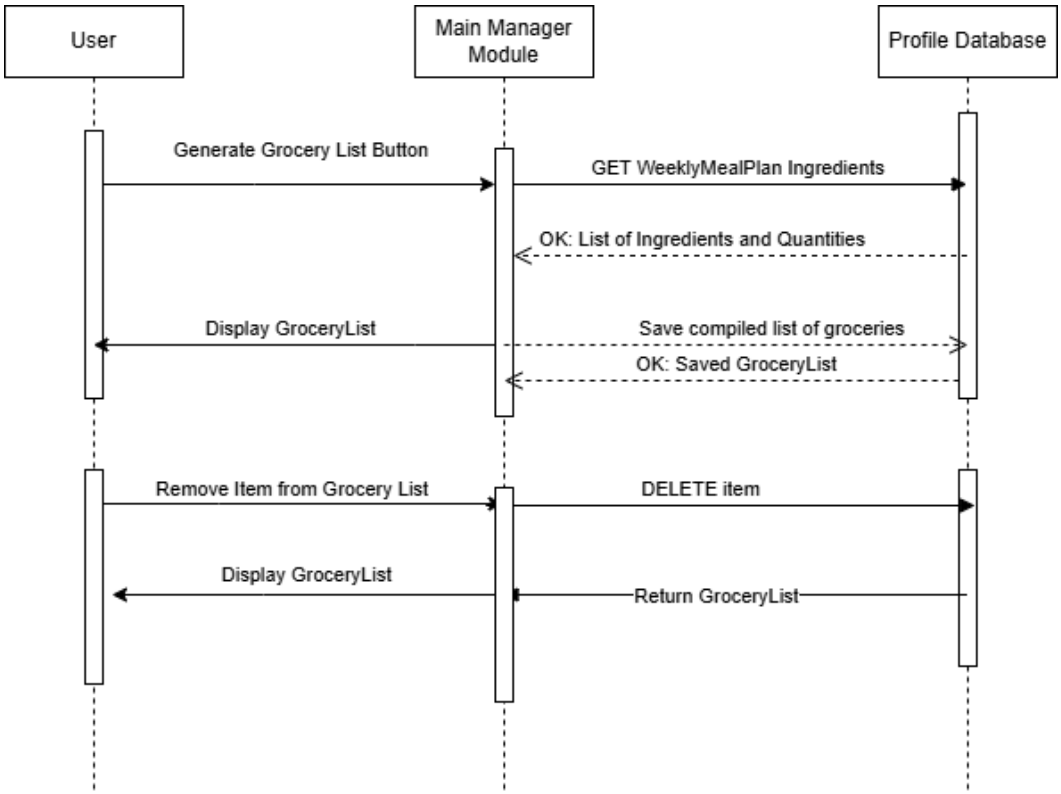

Figure 2.4.5 - Sequence Diagram for User Generating a Grocery List

2.4.6 View Grocery Stores

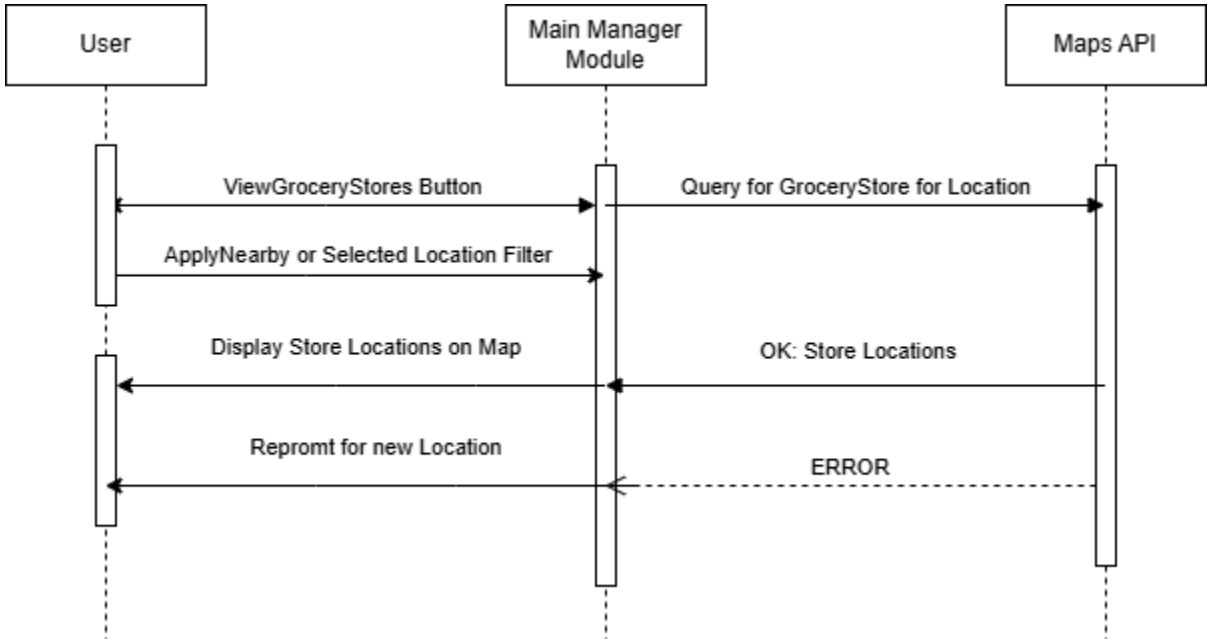

Figure 2.4.6 - Sequence Diagram for Viewing Grocery Stores

2.4.7 Update Settings

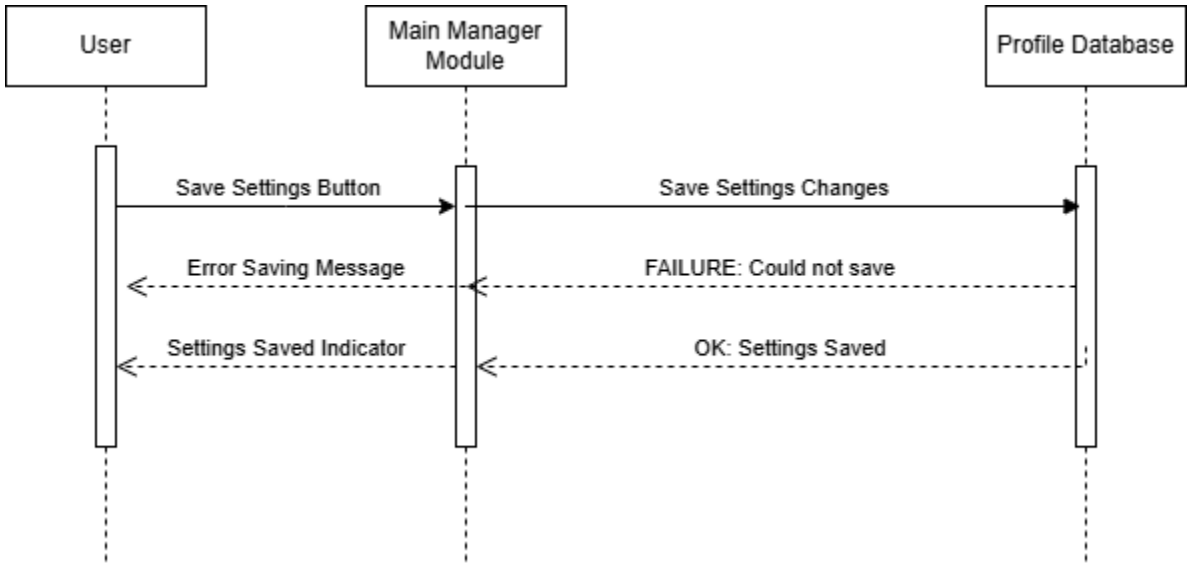

Figure 2.4.7 - Sequence Diagram for User Updating their Account Settings

3 Requirements Traceability

The Requirements Traceability Matrix (RTM) maps functional and non-functional requirements from the Software Requirements Specification (SRS) [1] to their corresponding design components in the system. This ensures that all features and constraints are addressed in the software implementation.

Requirement ID	Requirement Description	Design Component
REQ1	User can create an account	User authentication via Amazon Cognito
REQ2	User can receive information from APIs	Integration with Edamam and Map APIs
REQ3	User can toggle Spanish mode	UI language settings with localization
REQ4	User can save meal recipes to account	User data storage in PostgreSQL
REQ5	User can plan meals for the week	Weekly meal planner module
REQ6	User can track daily nutrient count	Nutrient tracking system
REQ7	User can set price range for suggested meal recipes	Recipe filtering based on cost
REQ8	User can find grocery stores on an interactive map	Map API for store search and pricing

REQ9	User can input allergens that will be excluded in meal recipes	Allergy-based filtering using API parameters
REQ10	User can opt into getting notifications for meal recipes	Notification system for reminders
NFR1	System must handle 99% crash-free user sessions	Robustness and stability mechanism
NFR2	Application should load in < 2 sec	Performance optimization
NFR3	User data should be encrypted and stored securely	Amazon Cognito & PostgreSQL Encryption
NFR4	System should be scalable to handle increasing workload and user growth without having the performance degrade	AWS-based infrastructure
NFR5	App should run on iOS17	IOS compatibility testing
NFR6	Home screen should display meal plans	User Interface – Home Screen
NFR7	Users should be able to search and filter recipes	Recipe Search & Filters
NFR8	Interactive map should provide store addresses/locations	Grocery Store Map Module

4 User Interface

The User Interface UML (Figure 4.0) illustrates how the user interface works as it relates to the main program and the data model.

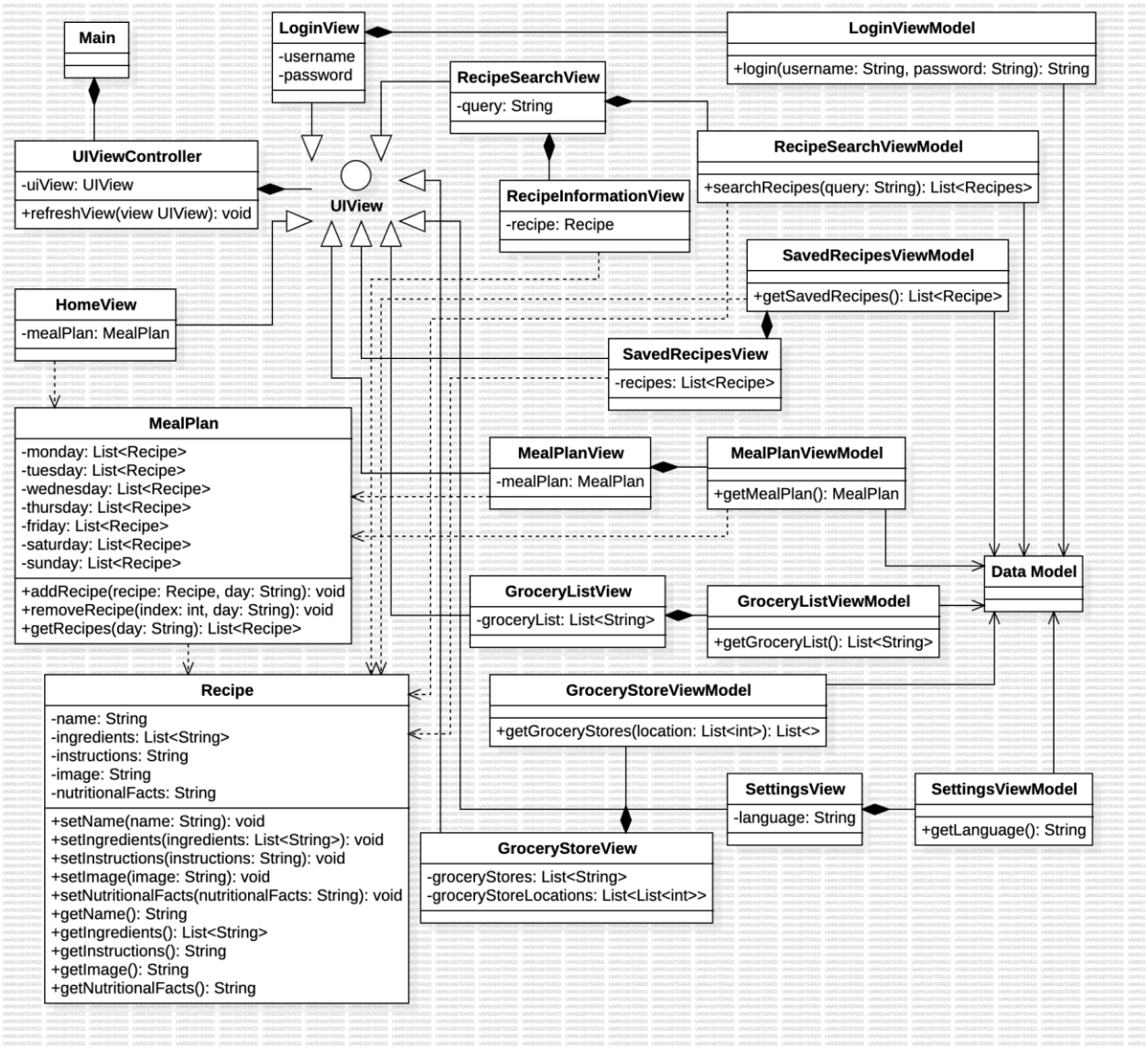

Figure 4.0: User Interface UML

5 Data Model and Storage

The Data Model and Data Storage outlines how app data is created, accessed, updated, and deleted from a user's local device and their account database entry.

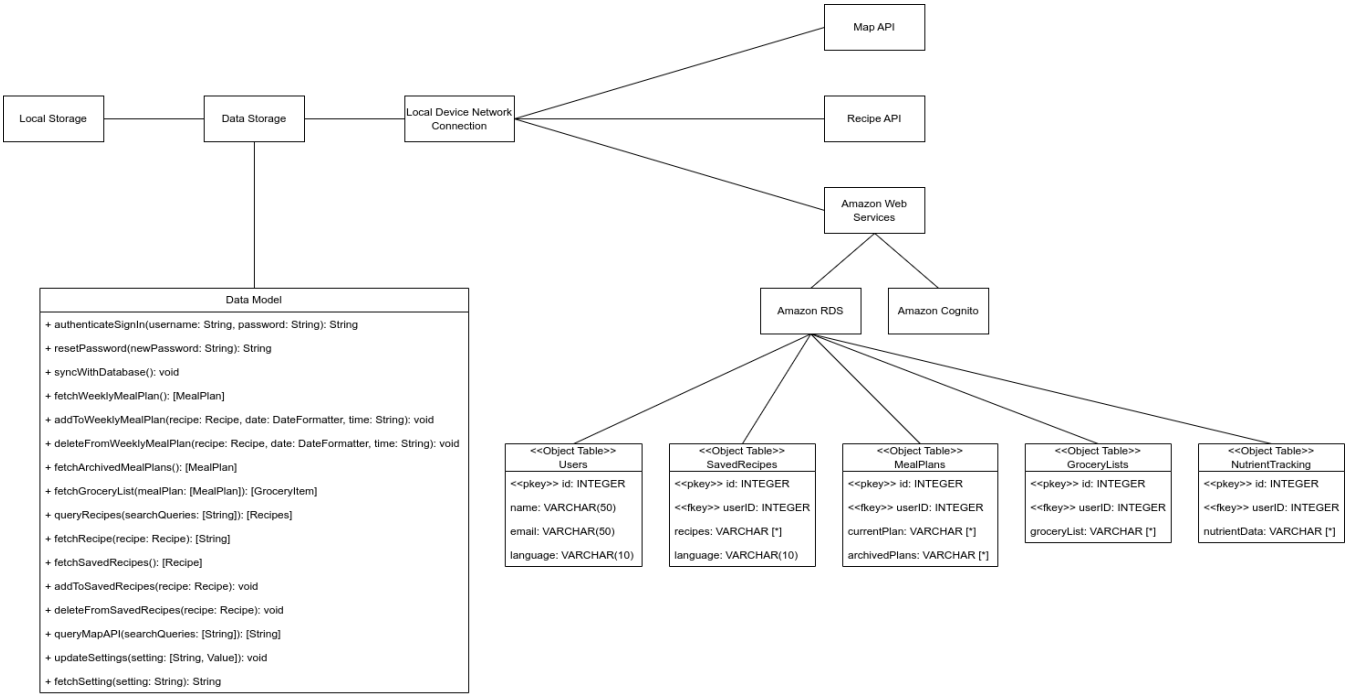

Figure 5.0: Data Model and Storage

5.1 Data Storage

The Glucose Genie application uses a combination of cloud-based and local storage solutions to efficiently manage user accounts, meal plans, grocery lists, and API interactions. This system ensures data consistency, security, and accessibility across different devices.

Primary Storage Solutions

1. Amazon RDS (PostgreSQL)
    a. **Purpose**: Stores user accounts, saved recipes, weekly meal plans, grocery lists, and preferences.
    b. **Justification**: Provides scalable, relational data storage with support for structured queries.
2. Amazon Cognito
    a. **Purpose**: Manages user authentication and secure access to user data.
    b. **Justification**: Ensures secure identity management and allows seamless sign-in across devices.
3. Local Device Storage (iOS Secure Storage)
    a. **Purpose**: Stores temporary data such as offline meal plans and cached API responses.
    b. **Justification**: Enables offline functionality and minimizes redundant API calls.

Database Schema

Table Name	Description
Users	Stores user profile information (ID, name, email, language preference)
SavedRecipes	Stores user-saved recipes with references to API-fetched data
MealPlans	Weekly meal schedules, linking saved recipes to specific days
GroceryLists	User-generated shopping lists based on planned meals
NutrientTracking	Logs user-input nutrient counts for daily monitoring

Data Synchronization Strategy

- **Sync with Cloud:** Updates (e.g., saving a recipe, modifying a meal plan) are pushed to Amazon RDS.
- **Local Caching:** Frequently accessed data (e.g., saved meal plans) is stored on the device for quick retrieval.
- **Conflict Resolution:** If local and cloud data are mismatched, the latest timestamp determines the authoritative version.

This hybrid storage approach allows for both online and offline functionality, ensuring a seamless user experience.

5. 2 Data Model

Method Name	authenticateSignIn
Inputs	username: String, password: String
Outputs	Boolean, error: String
Description	Method attempts to log a user into the app using a provided username and password. If successful, return *true* and proceed to the home screen. If unsuccessful, display an error message.

Method Name	resetPassword
Inputs	newPassword: String
Outputs	result: String
Description	Method will read in a new password requested by the user. If the new password is valid, update the user's account information in the database and display a success message. Otherwise, display an error message and prompt the user to input a different password.

Method Name	syncWithDatabase()
Inputs	None
Outputs	Void
Description	Method syncs user's local data with their database data. Data includes weekly meal plan, saved recipes, etc. If database is behind, update database. If local device is behind, update local device.

Method Name	fetchWeeklyMealPlan
Inputs	None
Outputs	[MealPlan] (meal plan array)
Description	Method fetches a user's weekly meal from their local device to display on their screen.

Method Name	addToWeeklyMealPlan
Inputs	recipe: Recipe, day: DateFormatter, time: String
Outputs	Void
Description	Method adds a recipe object to a user's weekly meal plan. A recipe is added to a day of the week at a specific time.

Method Name	deleteFromWeeklyMealPlan
Inputs	recipe: Recipe, day: DateFormatter, time: String
Outputs	Void
Description	Method deletes a recipe from a user's weekly meal plan.

Method Name	fetchArchivedMealPlans
Inputs	None
Outputs	[MealPlan] (meal plan array)
Description	Method fetches all archived user meal plans from the database to display to the user.

Method Name	fetchGroceryList
Inputs	mealPlan: [MealPlan] (meal plan array)
Outputs	[GroceryItem] (grocery item array)
Description	Method parses a user's weekly meal plan for needed ingredients. It then generates a list of groceries the user needs to acquire to make the recipes for the remainder of the week.

Method Name	queryRecipes
Inputs	searchQueries: [String]
Outputs	[Recipe] (recipe array)
Description	Method queries the database using the user's requested search parameters. After the database returns the resulting list of recipes, display the entries to the user.

Method Name	fetchRecipe
Inputs	recipe: Recipe
Outputs	[String]
Description	Method fetches a requested recipe from the database. After the database returns the recipe, display the information. Recipe information includes name, image, ingredients, instructions, etc.

Method Name	fetchSavedRecipes
Inputs	None
Outputs	[Recipe] (recipe array)
Description	Method fetches a user's saved recipes from their local device to display on screen.

Method Name	addToSavedRecipes
Inputs	recipe: Recipe
Outputs	Void
Description	Method adds a recipe to the user's saved recipe list.

Method Name	deleteFromSavedRecipes
Inputs	recipe: Recipe
Outputs	Void
Description	Method deletes a specified saved recipe from the user's saved recipe list.

Method Name	queryMapAPI
Inputs	searchQueries: [String]
Outputs	[String]
Description	Method queries a map API using the user's requested search parameters. After receiving a result, display the resulting list of grocery store information. Information includes location, operating hours, distance from user, etc.

Method Name	updateSettings
Inputs	setting: [String, Value]
Outputs	Void
Description	Method changes a user setting and updates the user's settings options on their local device. Settings include allergens, language, notifications, price range, etc.

Method Name	fetchSetting
Inputs	setting: String
Outputs	String
Description	Method returns a setting value given a valid setting name. Used when querying the different app APIs.

6 References

[1] Cruz-Urbanc, F., et al. (2025) Software Requirements Specification for Glucose Genie https://github.com/fjcu256/glucose-genie/tree/main/SoftwareRequirementsSpec