TED UNIVERSITY

TED UNIVERSITY

SENG 491

Project Analysis Report

Project Name: Fridge Genius

Team Name: BIS

Team Members:

İrem Güngör – 61375493682

Burcu Gül – 38113171904

Sude Sarı – 24482515562

Advisor:

Elif Kurtaran Özbudak

Jury Members:

Tansel Dökeroğlu

Eren Ulu

Table of Contents

1 Introduction

1.1 Purpose of the Report

The purpose of this report is to provide a comprehensive overview of the Fridge Genius application, which aims to improve users' fridge food, recipe suggestion and shopping experience using image recognition and data analytics. The report describes the system's capabilities, requirements and proposed solutions to address current issues related tofood management and meal preparation. The development process provides information to provide clarity on the scope, goals and functionality of the proposed system.

1.2 Scope of the System

1.2.1.1 Analysis

The "Scope of the System" section is fundamental to defining the boundaries, objectives, and key functionalities of the Fridge Genius application. This part provides clarity on what the system will achieve, who the stakeholders are, and what constraints will be considered during the developmentprocess. The scope serves as a roadmap, ensuring alignment between the development team, stakeholders, and users.

1.2.1.2 Detailed Scope

1. **System Overview**:
    a. Fridge Genius is a mobile application designed to revolutionize how users managetheir food inventories, minimize food waste, and streamline meal planning.
    b. Leveraging **image recognition** and **AI-powered analytics**, the system aims to offer aseamless experience in recognizing food items, suggesting recipes, and generating personalized shopping lists.
2. **Primary Objectives**:
    a. **Enhancing Food Management**: Help users keep track of what's in their fridge by digitizing inventory using image recognition.
    b. **Minimizing Food Waste**: By tracking expiration dates and offering recipes based on available items, the app ensures that food items are utilized effectively before theyspoil.
    c. **Streamlining Shopping**: Automatically generate shopping lists based on userpreferences, frequently used items, and items running low in inventory.
    d. **Personalized Experience**: Offer recommendations tailored to user habits, allergens,and favorite ingredients.
3. **Secondary Objectives**:
    a. **Data-Driven Insights**:
        i. Provide users with analytics about their food consumption habits, includingmost used items and waste patterns.
    b. **Time and Effort Saving**:

i. Eliminate the need for manual inventory management or brainstorming meal ideas, especially for busy individuals or families.

1.2.1.3 Functional Boundaries

The scope defines what functionalities the system will include and excludes non-core functionalities to maintain focus:

1. **Inclusions**:
    a. **Food Recognition**:
        i. Users can scan items through their device's camera or select images from the gallery to add to their fridge inventory.
    b. **Recipe Suggestions**:
        i. Based on the items in the inventory, the app will suggest recipes tailored to user preferences, allergens, and past recipe ratings.
    c. **Shopping List Creation**:
        i. The app will automatically generate shopping lists based on missing items, user consumption patterns, and frequently used ingredients.
    d. **Inventory Management**:
        i. Users can update item quantities, remove consumed items, and manually adjust expiration dates for items not recognized by the app.
    e. **Notifications**:
        i. Reminders for items nearing their expiration dates and suggestions to utilize these items in recipes.
    f. **User Profiles**:
        i. Allow users to set preferences such as allergens, disliked ingredients, and favorite cuisines.
2. **Exclusions**:
    a. **Nutritional Analysis**:
        i. The app does not provide detailed nutritional information about recipes or items.
    b. **Real-Time Health Monitoring**:
        i. The app does not integrate with wearable devices or track users' dietary health.
    c. **Cross-Platform Compatibility**:
        i. Initial development is limited to **iOS**, with potential Android support in future versions.
    d. **Integration with Smart Appliances**:
        i. The app does not sync with physical fridges or other kitchen appliances at this stage.

1.2.1.4 Target Audience

The Fridge Genius app is designed for diverse user groups, each with unique needs:

1. **Primary Users**:
    a. **Households**:
        i. Families or individuals who want to manage their food inventory efficientlyand minimize waste.
    b. **Busy Professionals**:
        i. Users with limited time for grocery planning and meal preparation who seekconvenience.
    c. **Food Enthusiasts**:
        i. Users interested in trying new recipes based on available ingredients.
2. **Secondary Stakeholders**:
    a. **Developers and Designers**:
        i. Those responsible for ensuring the app meets functional and nonfunctionalrequirements.
    b. **Regulatory Bodies**:
        i. Ensuring the app complies with GDPR and other privacy regulations.
    c. **Potential Business Partners**:
        i. Grocery stores or delivery services for future integration opportunities.

1.2.1.5 Constraints and Challenges

1. **Technical Constraints**:
    a. **Hardware Dependency**:
        i. The app requires a mobile device with a camera for food recognition. Performance may vary based on the device's processing power and camera quality.
    b. **External API Dependency**:
        i. The app relies on the **Google Gemini API** for food recognition and recipesuggestions. Downtime or rate limits of the API can affect functionality.
2. **Regulatory Constraints**:
    a. The system must comply with the **General Data Protection Regulation (GDPR)**,ensuring that user data is stored securely and handled transparently.
    b. Users must have control over their data, including the ability to delete their accountand associated information.
3. **Data Constraints**:
    a. The effectiveness of recipe suggestions and shopping list generation depends on thevolume and accuracy of user-generated data.

b. **Image Recognition Accuracy**:
    i. The app must accurately recognize a wide variety of food items underdifferent lighting and angles to provide reliable results.

4. **Scalability:**
    a. As the user base grows, the system must scale to handle increased API requests,database queries, and real-time inventory updates.

1.2.1.2 Outcomes

The desired outcomes of the Fridge Genius application include:

1. **User Benefits**:
    a. **Efficiency**: Users save time by quickly managing their inventory and receiving recipeand shopping suggestions tailored to their preferences.
    b. **Cost Savings**: By reducing food waste and overspending, users can manage theirgrocery budgets effectively.
    c. **Convenience**: The app simplifies the decision-making process for meal preparation.
2. **Development Goals**:
    a. Deliver a scalable, secure, and user-friendly application that meets both functionaland nonfunctional requirements.
    b. Lay the groundwork for future enhancements, such as Android support andintegration with external services.

1.3 Objectives

1.3.1 Improving User Experience:

- Helping users manage products in their fridge more effectively.
- Minimizing food waste and making meal preparation easier.

1.3.2 Use of Artificial Intelligence and Image Recognition:

- Providing automatic recognition of foods with image recognition and artificial intelligence-supported tools.
- Suggesting recipes that suit user habits and preferences.

1.3.3 Understanding User Habits with Data Analytics:

- Providing data analytics on users' food consumption habits, waste trends and most used products, recipes.

1.3.4 Efficient Shopping Planning:

- Creating personalized shopping lists by identifying the products users need.

1.3.5 Long Term Development:

- Creating a solid infrastructure for future improvements (for example, Android support and external service integration or adaptation of the camera to the fridge).

1.4 Audience

1.4.1 Primary Users:

- **Houses:** Families or individuals who want to manage their fridge inventory more effectively and minimize food waste.
- **Busy Workers:** Professionals who are short on time and looking for convenience in shopping planning and meal preparation.
- **Food Enthusiasts:** Users who want to try new recipes based on the ingredients they have on hand.

1.4.2 Secondary Users:

- **Developers and Designers:** People who will ensure that the application meets its functional and non-functional requirements.
- **Regulatory Bodies:** Ensuring the app complies with privacy regulations such as GDPR.
- **Potential Business Partners:** Potential collaborations for future integration opportunities, for example, grocery stores or delivery services.

2 Current System

2.1 Overview of the Current System

The current system is a mobile application developed with Flutter designed at choosing image, identifying food, offering recipe ideas. It allows users to choose an image or capture a photo of food products. It utilizes the Google Gemini API to analyze the image and save the identified food items in a local SQLite database. The application additionally offers recipe recommendations based on the identified food items. The system leverages the following features:

Image Selection: Users have the option to choose a picture from the gallery or take a photo with the camera.

Image Processing: The image undergoes compression and transformation into a smaller version prior to being transmitted to Google Gemini API for analysis. Then Google Gemini API is processing image.

Food Detection: The Gemini API analyzes the image and provides a list of recognized food items.

Data Storage: The selected food items that come from food detection are kept in a local SQLite database.

Recipe Suggestion: The application creates a recipe recommendation through the Gemini API based on identified food items.

Retry Mechanism: In case of API request failures, the application retries the request up to three times with an exponential backoff strategy.

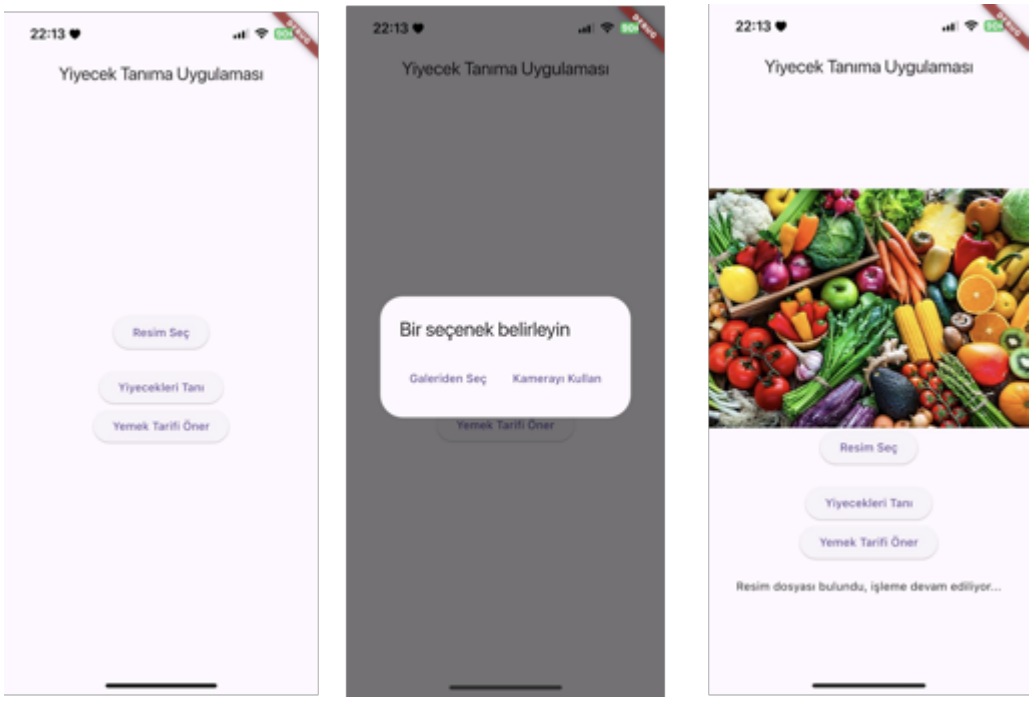

Figure 1: These images show current system screenshots.

The current system demonstrates a prototype that includes basic capabilities to identify foods and provide recipe suggestions. However, there are significant areas for improvement, particularly in terms of performance, offline functionality, error handling, and scalability. Addressing these challenges will be essential to improving the user experience andexpanding the application's reach.

2.2 Problems with the Current System

Many problems and restrictions exist in the present system that impact functionality, performance, and user experience:

1. **Large Image Files:** The application must reduce the size of the whole image file, which can be quite large, to handle it. This can lead to slow performance, increased data usage, and inaccurate response, especially on mobile networks. Large images can cause problems in image processing.

2. **Dependency on External API:** The recipe suggestion and food recognition features depend on the Google Gemini API. If the API service is unavailable, the application cannot provide the intended functionality. An error is received regarding the API connection.

3. **Error Handling and Providing User Feedback:** Even though the system has a retry feature, the error messages shown to the user are restricted and may lack enough details for effective troubleshooting.

4. **Mobile Device Performance Problems:** Image compression and API calls can be demanding, resulting in possible lag or slow, inaccurate functionality, particularly on older or less capable devices.

5. **Limited Offline Functionality:** The application needs a working internet connection for processing images and providing recipe ideas, rendering it ineffective for offline use. Server connection is inconsistent.

2.3 Challenges and Constraints

The current system faces several challenges and limitations that affect its scalability and usability:

1. Rate Limits for APIs: The Google Gemini API imposes rate limits that restrict the number of requests allowed within a specific timeframe. This may hinder users from executing several queries promptly, just a limited time.

2. Network Dependency: The dependency of the system on a constant internet connection may pose difficulties for users. Network disruptions can lead to requests timing out or the connection being reset.

3. Data Privacy Concerns: The application analyzes user images that could include sensitive or personal data. It is essential to guarantee data privacy and safe transmission of these images.

4. Compatibility for Devices: The application must accommodate a diverse array of mobile devices featuring varying processing capabilities, screen dimensions, and operating system releases. Optimizing performance is necessary to guarantee smooth operation on various devices.

5. Scalability: As the user base expands, the rising number of API requests and database activities may result in a decline in performance. The system requires a scalable structure to efficiently manage an increased user base.

3 Proposed System

3.1 Overview

Fridge Genius is a mobile application developed to facilitate users' fridge management, minimize food waste and make meal preparation practical. The app allows users to digitally track their food with image recognition technology and AI-powered data analytics. Fridge Genius makes meal planning easier by offering recipe suggestions based on ingredients in users' fridge sand optimizes users' shopping processes by creating personalized shopping lists. In addition to these features, it provides data-driven insights by analyzing users' consumption habits and waste trends. The app offers a user-friendly interface and a highly accurate system, making meal management more accessible and effective.

3.2 Functional Requirements

User Authentication and Profile Management:
Users can create an account by providing their personal information such as name, email, and preferences. This supports individual food differences in same fridge. Even same home two user with different profile can have different shopping list and recipes. Users who return can log in with their credentials to access their inventory and preferences.

Adding and Managing Fridge Items:
When user open the app with account for adding items in fridge, he must click “Add Item” button. This button activates the mobile device's camera, the user scans each item he places in the fridge. The app identifies the item and automatically adds it to the digital inventory. Users can update or remove items from the inventory at any time. If an item is not recognized, users can manually enter details such as quantity and expiration date.

Image Recognition:
The app uses image recognition technology to identify items and automatically add them to the fridge inventory. When an item is not recognized, the app should provide feedback and offer manual entry as an alternative.

Viewing and Managing Inventory:
The user must access all items in the fridge as a list with quantity and expiration date. 5The user can adjust quantities, remove items they’ve consumed, or update expiration dates manually. Expiration date automatically filled with image processing as default value when user scan the item. However, in foods that have not a clear expiration date, user must give it. When an item consumed by user, he should remove item manually by clicking it. Inventory updates should be made in real time as users consume or edit items.

Recipe Suggestion System:
The app suggests recipes based on the available ingredients in the fridge and the user's favorite foods. The user can view a list of suggested recipes and choose one that matches their preferences. Each recipe includes instructions, ingredients, and the estimated cooking time and type in detailed way. The user can mark recipes as "Tried" or "Favorite" to improve future recommendations with that shopping list and later recipes can made based on the user’s consumption.

Create a Weekly Shopping List:
Based on the user's consumption and favorite recipes, app must make a shopping list. The user can add or remove items from this list, adjusting based on personal needs.

Usage Tracking and Insights:
The app tracks user behavior, such as frequently used ingredients and waste levels, and provides actionable insights. Users can access reports on their consumption habits and get recommendations for more productive habits.

Warning Notifications:
The app sends reminder notifications about items approaching their expiration date, existing recipes, or shopping list updates. Notification settings can be customized based on user preferences.

Adjusting Settings and Preferences:
User can also make changes for allergens. In profile, user can always made changes according to taste and the frequency of recipe suggestions and shopping list updates can be configured by the user. App gives a flexible recipe that always meets with users' preferences.

Cross-Platform Compatibility:
The app supports iOS devices, providing a seamless user experience across platforms. These functional requirements ensure that the application meets the needs of users and provides a high level of availability, security, and efficiency.

3.3 Nonfunctional Requirements

3.3.1.1 Analysis

Nonfunctional requirements are essential for defining the quality attributes of the Fridge Genius system. These requirements ensure the application not only performs its intended functions but also provides a seamless, secure, and reliable user experience. Unlike functionalrequirements, which focus on what the system does, nonfunctional requirements focus on howthe system behaves under various conditions.

3.3.1.2 Detailed Nonfunctional Requirements

1. **Performance**:
    a. **Image Processing Time**:
        i. The application must process images within **1–2 seconds**, ensuring minimal wait time for users.
    b. **Response Time**:
        i. Recipe suggestions and shopping list generation must complete within **3 seconds** of the user's request.
    c. **Concurrent Users**:
        i. The system should support at least **500 concurrent users** without significant degradation in performance.

    d. **Low Latency**:
        i. API calls to the Google Gemini API must maintain an average round-trip time of **less than 200 ms** under standard conditions.
2. **Usability**:
    a. **User Interface Design**:
        i. The interface must be intuitive, requiring no more than **two actions** to scan a food item and add it to the inventory.
    b. **Accessibility**:
        i. The application should comply with **WCAG 2.1 guidelines**, ensuring usability for individuals with visual, auditory, or motor impairments.
    c. **Error Feedback**:
        i. Users must receive clear and actionable error messages in the event of issues such as failed image recognition or API timeouts.
3. **Security**:
    a. **Data Encryption**:
        i. All sensitive data, including user credentials and inventory data, must be encrypted using **AES-256**.
    b. **Secure Communication**:
        i. Use **TLS/SSL protocols** for all communications between the app, APIs, and databases.
    c. **Authentication**:
        i. The system should implement a **secure login mechanism**, including options for multi-factor authentication (MFA).
    d. **Data Privacy**:
        i. Ensure compliance with **GDPR**, allowing users to delete their data permanently upon request.

4. **Reliability**:
    a. **Uptime**:
        i. The application should maintain a **99.9% uptime** for critical functionalities, such as login, food recognition, and recipe suggestions.
    b. **Error Handling**:
        i. Implement retry mechanisms for failed API calls with an exponential backoff strategy.
    c. **Backup and Recovery**:
        i. Ensure periodic backups of critical data (e.g., user profiles, inventory data) with recovery mechanisms in place for unexpected outages.

5. **Scalability**:
    a. The architecture should scale horizontally to handle:
        i. Increased user registrations and simultaneous logins.
        ii. Growing volumes of food recognition data and API requests.
        iii. Larger inventories and user-generated content (e.g., favorites, shopping lists).
    b. Utilize **cloud-based storage and computing** for scalable resource allocation.

6. **Compatibility**:
    a. **Platform Support**:
        i. The application must run smoothly on iOS 14 or later devices.
        ii. It should support real-time connectivity with an integrated fridge camera.
    b. **Device Support**:
        i. The app must be optimized for devices with various screen sizes, from smaller iPhones to larger iPads.
        ii. High-performance devices, such as **iPhone 14 or 15**, are recommended for advanced image processing.
    c. **Backward Compatibility**:
        i. Ensure compatibility with older device models (e.g., iPhone 8) without compromising performance.
    d. Hardware Compatibility:
        i. The camera should have at least **12 MP resolution** with a wide-angle lens to cover all compartments.
        ii. **Nvidia Jetson Nano** or **Raspberry Pi/STM32** can be used for local image processing.
        iii. Connectivity should be ensured via **Wi-Fi** or **Bluetooth**.

7. **Maintainability**:
    a. **Modular Code Structure**:
        i. The application should adopt a modular architecture, allowing independent updates to specific components without disrupting the entire system.
    b. **Documentation**:
        i. Provide comprehensive documentation for developers, including code comments, API usage guidelines, and system architecture diagrams.
    c. **Automated Testing**:
        i. Implement unit tests, integration tests, and performance tests to ensure continuous delivery and maintainability.

8. **Accuracy**:
    a. **Image Recognition**:
        i. Achieve a food recognition accuracy rate of **95% or higher**, even under challenging conditions like low light or partial visibility.
    b. **Recipe Recommendations**:
        i. Ensure recipe suggestions are contextually relevant, incorporating user preferences, allergens, and available inventory.

9. **Energy Efficiency**:
    a. Optimize the app to minimize energy consumption on mobile devices by:
        i. Reducing background processes when the app is idle.
        ii. Efficiently handling image compression and API calls to prevent excessive CPU or battery usage.

10. **Localization and Internationalization**:
    a. **Language Support**:
        i. The application should support multiple languages to cater to a diverse user base.
    b. **Measurement Units**:
        i. Allow users to choose between metric (grams, liters) and imperial (ounces, cups) units for recipes and inventory items.

11. **User Feedback and Updates**:
    c. **Feedback Mechanism**:
        i. Provide an in-app mechanism for users to report issues, suggest features, or provide general feedback.
    d. **Update Notifications**:
        i. Notify users of new features or critical updates without disrupting their usage.

3.3.1.3 Examples in Context

1. **Scenario: Image Processing**:
    a. A user scans a food item using their mobile camera. The app must compress the image, send it to the Google Gemini API, and display recognized items within **2 seconds**.
    b. If the API call fails, the system retries up to **three times** before displaying an error message with suggestions (e.g., check network connection).
2. **Scenario: Recipe Suggestion**:
    a. A user requests a recipe based on the available inventory. The app retrieves relevant data from the local SQLite database and the AI API, ensuring suggestions are generated within **3 seconds**.
3. **Scenario: Error Handling**:
    a. If the user enters invalid credentials during login, the system displays a clear error message: "Incorrect username or password. Please try again."

3.3.1.4 Impact of Nonfunctional Requirements

1. **User Satisfaction**:
    a. Fast response times and clear error messages enhance user trust and engagement.
2. **Data Security**:
    a. Ensuring secure data handling builds credibility and compliance with regulations like GDPR.
3. **Scalability and Growth**:
    a. A scalable architecture prepares the app for an increasing user base and future enhancements.
4. **Competitive Edge**:
    a. High accuracy in image recognition and seamless usability distinguish the application from competitors.

3.4 Pseudo Requirements

Pseudo requirements outline the external constraints, dependencies, and guidelines that influence the design and functionality of the system. These constraints ensure that the development process aligns with regulatory standards, hardware capabilities, and business objectives.

3.4.1.1 Regulatory Compliance

1. **Data Privacy and Security**:
    - **a.** The system must adhere to **General Data Protection Regulation (GDPR)** standards, ensuring:
        i. Users are informed about how their data is collected, processed, and stored.
        ii. Users can request the deletion of their data and account at any time.
        iii. Data transmission is encrypted using **TLS/SSL protocols**, and sensitive information is stored using **AES-256 encryption**.
2. **Ethical AI Use**:
    a. The application must avoid biases in food recognition by training AI models on diverse datasets.
    b. Any recommendations provided (e.g., recipes or shopping lists) must empower users rather than creating unnecessary dependencies.

3.4.1.2 Platform Constraints

1. **Operating System**:
    a. The application is designed exclusively for **iOS devices** and must be compatible with **iOS 14 or later**.
2. **App Store Guidelines**:
    a. The app must comply with **Apple App Store Review Guidelines**, particularly:
        i. Clear disclosures about data collection and permissions.
        ii. Avoidance of excessive background data usage or battery drain.
3. **Device Support**:
    a. The application must support a wide range of iOS devices, including:
        i. **Older models**, such as iPhone 8, to ensure broad accessibility.
        ii. **Newer models**, such as iPhone 14 and iPhone 15, leveraging advanced features like high-resolution cameras for improved performance.

3.4.1.3 Hardware Constraints

1. **Fridge Camera Integration**:
    a. **Camera Specifications**:
        i. A **minimum 12 MP camera** with a wide-angle lens will be integrated into the fridge to capture images of shelves and compartments.
    b. **Processing Unit**:
        i. The system may use an **Nvidia Jetson Nano** or similar GPU for local image processing, enabling faster and more accurate recognition.
        ii. As an alternative, a **microcontroller** such as a **Raspberry Pi 4** or **STM32** can handle basic image processing tasks with reduced power consumption.
2. **Connectivity**:
    a. The fridge camera will communicate with the application through **Wi-Fi** or **Bluetooth**, allowing real-time data transmission to the app.

3.4.1.4 External API Dependency

1. **Google Gemini API**:
    a. **Rate Limits**:
        i. The API imposes rate limits (e.g., **1000 requests per day**) that restrict the frequency of image recognition operations. The system must optimize requests to stay within these limits.

    b. **Availability**:
        i. The system must handle API downtime gracefully by:
            1. Retrying requests with an **exponential backoff strategy**.
            2. Informing users about the issue with clear error messages, such as “Service temporarily unavailable. Please try again later.”
    c. **Cost Management**:
        i. API usage must remain within the allocated budget. The app may encourage users to optimize their scanning behavior or limit non-essential API calls.

3.4.1.5 Business Constraints

1. **Budget**:
    a. The project must stay within a fixed budget, including:
        i. Development costs (e.g., salaries, tools, and licenses).
        ii. Hardware expenses, such as test devices and integrated camera components.
        iii. API usage fees.
2. **Development Timeline**:

a. The system must be fully developed, tested, and deployed within **6 months**, ensuring all functional and nonfunctional requirements are met.

3. **Licensing**:
    a. Only open-source or appropriately licensed libraries and tools should be used to avoid legal conflicts.

3.4.1.6 Data Storage and Scalability

1. **Local Storage**:
    a. User credentials, inventory data, and preferences will be securely stored in a local SQLite database. Sensitive data must be encrypted to prevent unauthorized access.
2. **Cloud Storage**:
    a. Backup and synchronization services will use **cloud-based storage**, ensuring data availability and recovery options.
3. **Scalability**:
    a. The system must scale to handle:
        i. A growing number of users and API requests.
        ii. Larger inventories and frequent app interactions without significant performance degradation.

3.4.1.7 Ethical and Social Constraints

1. **Bias in AI**:
    a. The AI model must recognize a wide range of food items, including culturally specific ingredients, to ensure inclusivity.
2. **User Empowerment**:
    a. Recommendations (e.g., recipes or shopping lists) should be designed to assist users in making informed decisions rather than fostering dependency.

3.4.1.8 Deployment Constraints

1. **Testing and Compatibility**:
    a. The app must be thoroughly tested on various iOS devices to ensure compatibility across screen sizes, hardware capabilities, and software versions.
    b. Testing phases:
        i. **Alpha testing** with internal developers.
        ii. **Beta testing** with a select group of real-world users.
2. **Release Schedule**:
    a. The app must be ready for deployment on the Apple App Store by the project deadline, following iterative testing and debugging.

3.4.1.9 Examples in Context

1. **Camera Integration**:
    a. The fridge camera captures an image, processes it locally using Nvidia Jetson Nano, and sends recognized food items to the mobile app via Wi-Fi.
2. **API Rate Limits**:
    a. If the API request limit is reached, the app prioritizes essential operations like food recognition and defers non-essential requests (e.g., recommendations).
3. **User Data Privacy**:
    a. A user requests to delete their account, triggering the app to erase all personal data, including local and cloud-stored backups, within **24 hours**.

3.4.1.10 Impact of Pseudo Requirements

1. **Development Process**:
    a. These requirements shape the technical and design decisions, ensuring a realistic, compliant, and user-friendly system.
2. **User Trust**:
    a. Addressing privacy and security builds trust and credibility with users.
3. **System Longevity**:
    a. Scalability and modularity prepare the system for future growth and enhancements.

3.5 System Models

The system is a mobile application designed for food recognition using image processing and AI-powered content generation.

It consists of the following main components:

User Interface (UI):

Provides an interface where users can sign up and create a new profile with their own username and password.

Provides an interface where users can login, log out and enter with their own username and password.

Provides an interface for users to select or capture an image and view the results.

Shows the food items recognized by processing the uploaded image and can favorite these foods.

Displays suggested recipes from the food in the user's fridge.

Recommends a shopping list based on the analysis obtained from the user's purchases.

Provides an interface where the user can set their profile.

Image Processing Module:

Receives the image from the user.

Compresses and converts the image if it is a large image.

Sends the image data to the Google Gemini API for analysis.

Google Gemini API:

Processes the image data and creates a list of recognized food items.

Provides content-based suggestions for recipes using the identified ingredients.

Provides content-based suggestions for recipes using favorite ingredients.

Local Database (SQLite):

Stores username, password for user login.

Stores recognized food items locally on the device.

Stores favorite recognized food items.

Allows to retrieve food items for recipe suggestions.

Allows to retrieve food items for shopping list suggestions.

Network Module:

Manages HTTP requests and processes API responses.

Implements retry logic for network failures.

3.5.1 Scenarios

Scenario 1: User Login to Access Application

Actors: User

System: The mobile application performs user authentication by verifying credentials with a remote server.

Description: User navigates to the login interface, enters the username and password, and selects the "Login" button. The system checks the credentials by passing them to the server. If the credentials are validated, the user gains access and is redirected to the home screen. Conversely, if the credentials are invalid, an error notification is presented.

Preconditions: User must have a registered account with valid credentials.

Postconditions: User successfully logs in and if the login attempt fails, the user is redirected to the home screen or receives an error notification.

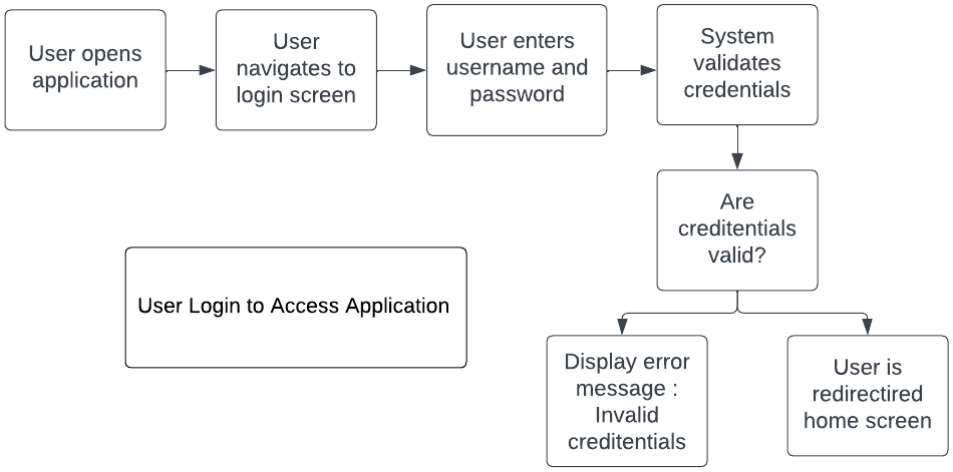

Figure 2: This image shows User Login to Access Application flowchart.

Scenario 2: Food Recognition Using Gallery Image

Actors: User

System: The application analyzes the selected image, passes it to the AI API for identification, and saves the recognized items locally.

Description: User selects an image from his gallery. The system compresses the image and passes it to the Google Gemini API. The API then responds with a list of identified food items, which is displayed to the user and saved in the local database.

Preconditions: User must select an image from the gallery.

Postconditions: The identified food items are displayed, and the data is saved in the local database.

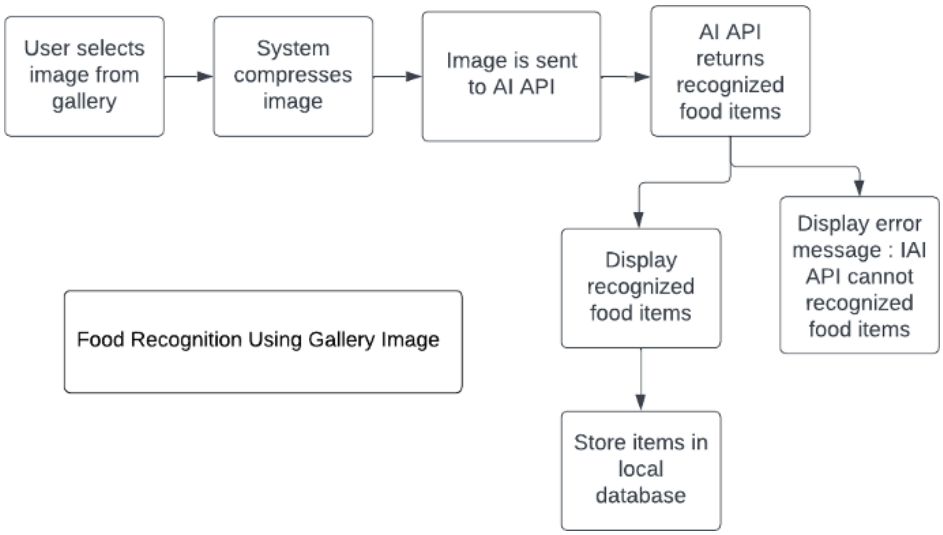

Figure 3: This image shows Food Recognition Using Gallery Image flowchart.

Scenario 3: Food Recognition Using Camera Image

Actors: User

System: The application processes the captured image, sends it to the AI API for identification, and stores the recognized items locally.

Description: User takes a picture using the camera. The system processes the image and sends it to the API for identification. The recognized food items are then displayed and stored in the database

Preconditions: User must take a picture using the camera.

Postconditions: The recognized food items are displayed and stored.

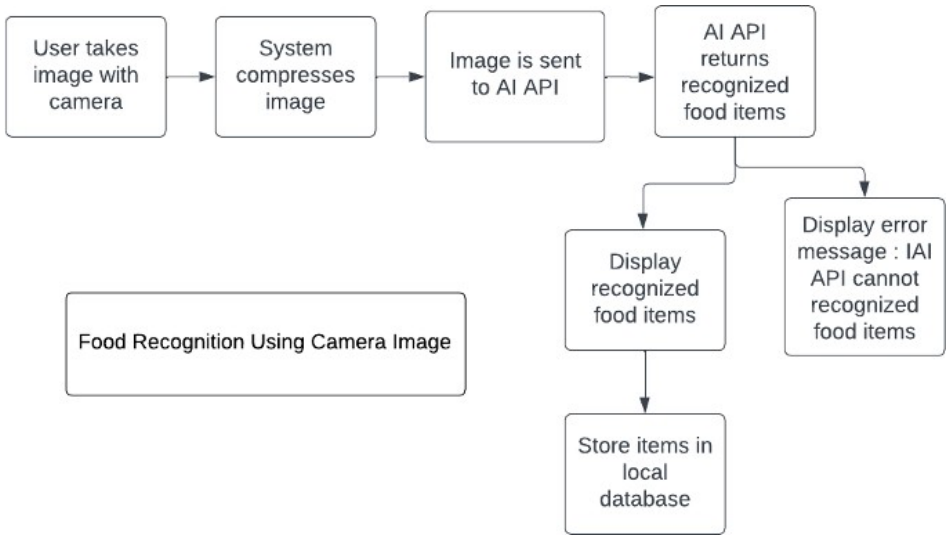

Figure 4: This image shows Food Recognition Using Camera Image flowchart.

Scenario 4: Recipe Suggestion Based on Recognized Food Items

Actors: User

System: The application retrieves the identified food items from the local database and generates recipe suggestions using the AI API.

Description: User requests a recipe suggestion. The system retrieves recognized food items from the database and sends a prompt to the Google Gemini API. The API returns a recipe suggestion, which is displayed to the user.

Preconditions: Recognized food items exist in the local database.

Postconditions: A recipe suggestion is displayed to the user.

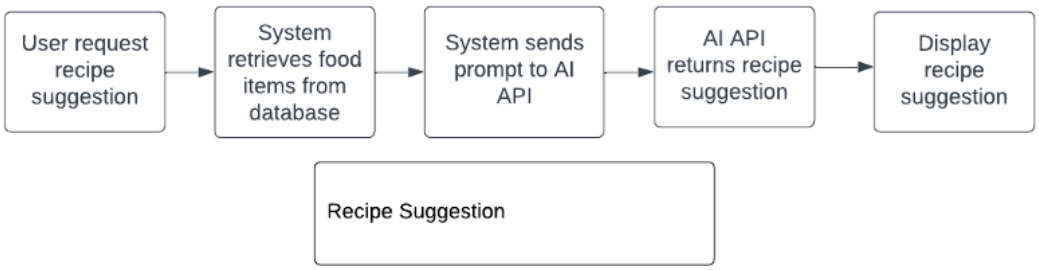

Figure 5: This image shows Recipe Suggestion flowchart.

Scenario 5: Personalized Shopping Suggestions Using AI

Actors: User

System: The application analyzes the user's purchase history with AI to generate tailored shopping recommendations.

Description: User navigates to the "Shopping Suggestions" section. The system retrieves the user's purchase history and sends it to the AI service for analysis. The AI service identifies patterns and generates personalized shopping suggestions, which are displayed to the user for review and updating the shopping list.

Preconditions: User has a recorded purchase history in the system.

Postconditions: User receives tailored shopping suggestions and updates their shopping list accordingly.

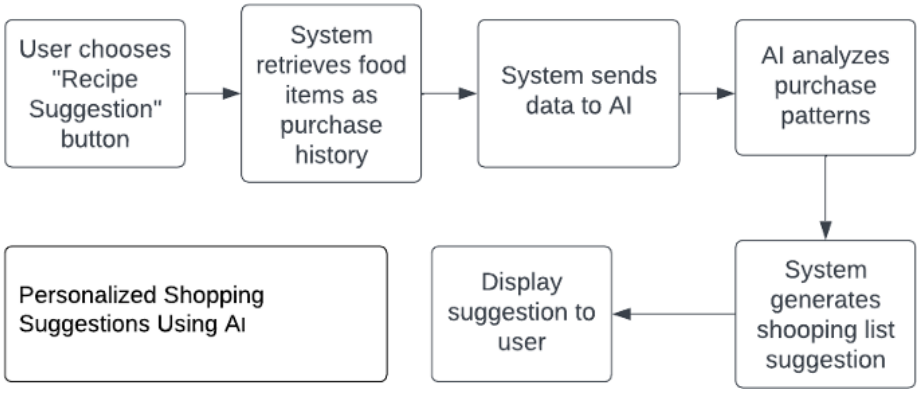

Figure 6: This image shows Personalized Shopping Suggestions Using AI flowchart.

3.5.2 Use Case Models

Login System Use Case Diagram

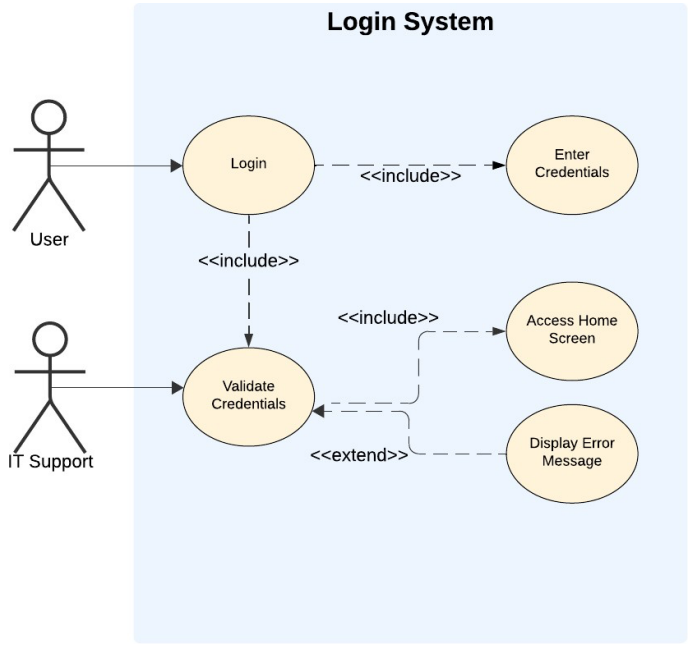

Figure 7: This image shows Login System Use Case Diagram.

Use Case Name	**Enter Credentials**
Actors	User
Preconditions	The user is on the login screen.
Normal Flow	User inputs their username and password.
Alternative Flows or Exceptions	If fields are left empty, the system prompts the user to fill them.
Non-functional Requirements	Usability, Responsiveness

Use Case Name	**Validate Credentials**
Actors	System
Preconditions	User has entered their credentials.

Normal Flow	System checks the entered credentials against the database.
Alternative Flows or Exceptions	If credentials do not match, the system displays an error message.
Non-functional Requirements	Reliability, Security

Use Case Name	**Access Home Screen**
Actors	User
Preconditions	User credentials have been validated successfully.
Normal Flow	System redirects the user to the home screen.
Alternative Flows or Exceptions	If validation fails, the user remains on the login screen.
Non-functional Requirements	Performance, User Experience

Use Case Name	**Display Error Message**
Actors	System
Preconditions	Credentials validation has failed.
Normal Flow	The system shows an error message indicating that the login attempt was unsuccessful.
Alternative Flows or Exceptions	User is prompted to re-enter credentials or choose "Forgot Password."
Non-functional Requirements	Clarity, Feedback

Food Recognition from Gallery and Camera by AI Use Case Diagram

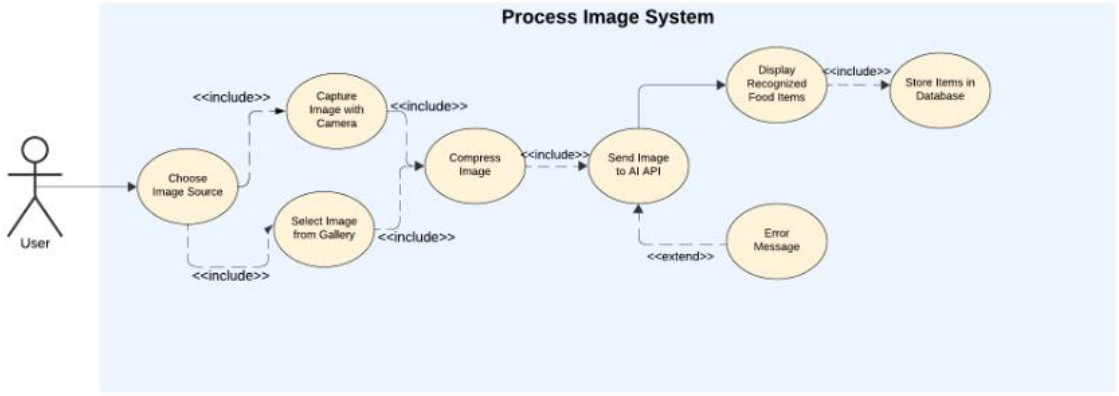

Figure 8: This image shows Process Image System Use Case Diagram.

Use Case Descriptions

Use Case Name	**Choose Image Source**
Actors	User
Preconditions	User is ready to select an image for processing.
Normal Flow	User chooses either "Select Image from Gallery" or "Capture Image with Camera."
Alternative Flows or Exceptions	If the user does not choose an option, the system prompts the user again.
Non-functional Requirements	Usability, Responsiveness

Use Case Name	**Select Image from Gallery**
Actors	User
Preconditions	User has chosen the gallery as the image source.
Normal Flow	User selects an image from the device's gallery.
Alternative Flows or Exceptions	If no image is selected, the user is notified to try again.
Non-functional Requirements	User Experience, Efficiency

Use Case Name	**Capture Image with Camera**
Actors	User
Preconditions	User has chosen the camera as the image source.
Normal Flow	User captures an image using the device’s camera.
Alternative Flows or Exceptions	If the image capture fails, the system prompts the user to retry.
Non-functional Requirements	Performance, Usability

Use Case Name	**Compress Image**
Actors	System
Preconditions	An image has been selected or captured.
Normal Flow	The system compresses the image to reduce its size for faster processing.
Alternative Flows or Exceptions	If compression fails, an error message is displayed.
Non-functional Requirements	Efficiency, Speed

Use Case Name	Send Image to AI API
Actors	System
Preconditions	The image has been compressed successfully.
Normal Flow	The system sends the compressed image data to the AI API for analysis.
Alternative Flows or Exceptions	If the API request fails, an error message is shown.
Non-functional Requirements	Reliability, Network Efficiency

Use Case Name	Display Recognized Food Items
Actors	User
Preconditions	The AI API has returned a list of recognized food items.
Normal Flow	The system displays the recognized food items to the user.
Alternative Flows or Exceptions	If no items are recognized, the system displays "No items found."
Non-functional Requirements	Clarity, User Experience

Use Case Name	Store Items in Database
Actors	System
Preconditions	The recognized food items are available.
Normal Flow	The system stores the recognized food items in the local database.
Alternative Flows or Exceptions	If the database operation fails, an error message is shown.
Non-functional Requirements	Data Integrity, Reliability

Use Case Name	Error Message
Actors	System
Preconditions	An error has occurred during one of the previous steps.
Normal Flow	The system displays an appropriate error message to the user.
Alternative Flows or Exceptions	The user can retry the operation or choose a differentoption.
Non-functional Requirements	Clarity, Feedback

Recipe Suggestion System Use Case Diagram

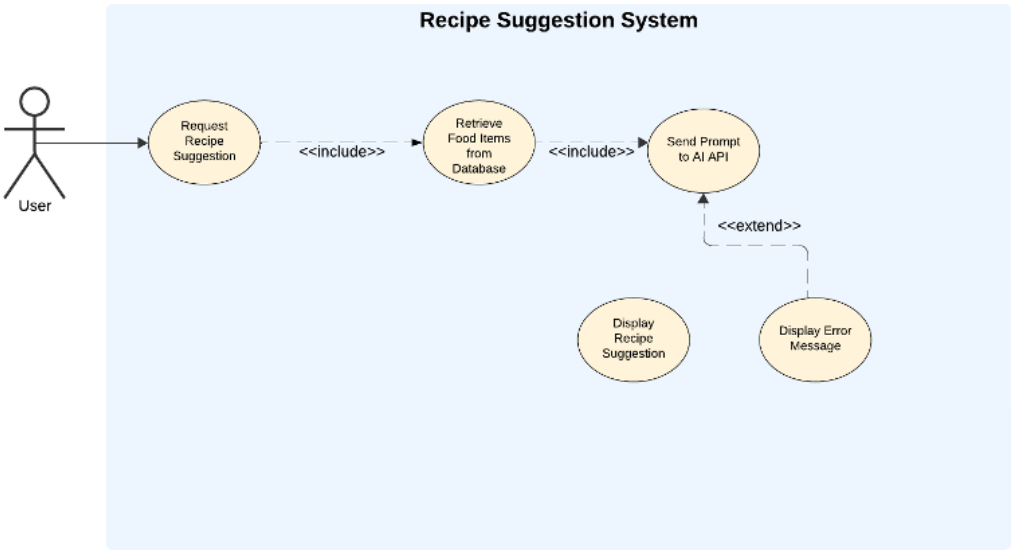

Figure 9: This image shows Recipe Suggestion System Use Case Diagram.

Use Case Descriptions

Use Case Name	**Request Recipe Suggestion**
Actors	User
Preconditions	Recognized food items exist in the local database.
Normal Flow	The user requests a recipe suggestion based on the recognized food items.
Alternative Flows or Exceptions	If no food items are found, the user is informed that no items are available for a recipe.
Non-functional Requirements	Usability, User Experience

Use Case Name	**Retrieve Food Items from Database**
Actors	System
Preconditions	The user has requested a recipe suggestion.
Normal Flow	The system retrieves the recognized food items from the local database.
Alternative Flows or Exceptions	If the database query fails, an error message is shown.
Non-functional Requirements	Data Integrity, Reliability

Use Case Name	Send Prompt to AI API
Actors	System
Preconditions	Food items have been successfully retrieved from the database.
Normal Flow	The system sends a prompt with the food items to the AI API for recipe generation.
Alternative Flows or Exceptions	If the API request fails, an error message is displayed.
Non-functional Requirements	Network Efficiency, Response Time

Use Case Name	Display Recipe Suggestion
Actors	User
Preconditions	The AI API has returned a recipe suggestion.
Normal Flow	The system displays the generated recipe suggestion to the user.
Alternative Flows or Exceptions	If no recipe is generated, the user is notified that no suggestion is available.
Non-functional Requirements	Clarity, User Experience

Use Case Name	Display Error Message
Actors	System
Preconditions	An error has occurred during one of the previous steps.
Normal Flow	The system displays an appropriate error message to the user.
Alternative Flows or Exceptions	User can retry the request or choose a different option.
Non-functional Requirements	Clarity, Feedback

Shopping List Suggestion Use Case Diagram

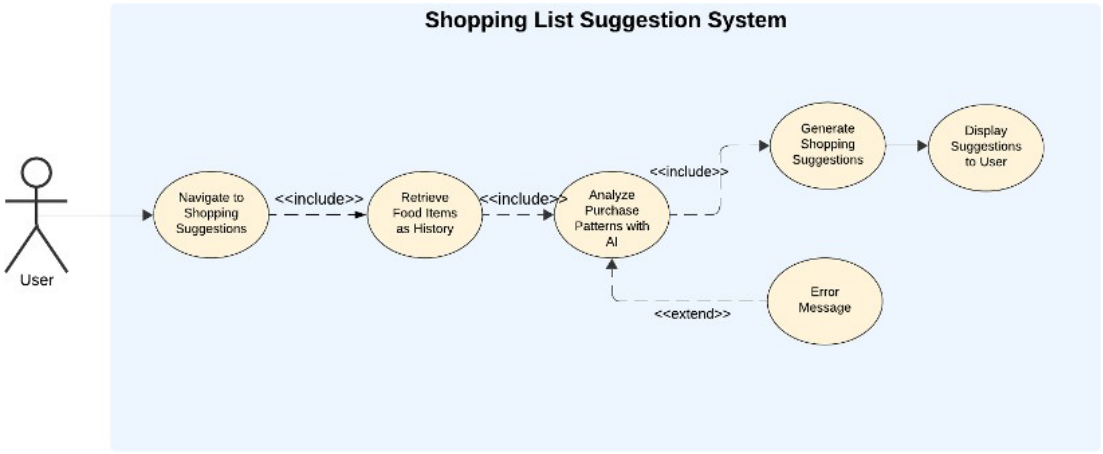

Figure 10: This image shows Shopping List Suggestion System Use Case Diagram.

Use Case Descriptions

Use Case Name	**Navigate to Shopping Suggestions**
Actors	User
Preconditions	User has logged in and is on the main screen of the application.
Normal Flow	User navigates to the "Shopping Suggestions" section of the app.
Alternative Flows or Exceptions	If the user cannot access the section, the system displays an error message.
Non-functional Requirements	Usability, Accessibility

Use Case Name	**Retrieve Food Items as History**
Actors	System
Preconditions	User has previously recognized food items stored in the database.
Normal Flow	The system retrieves the stored food items as the user's purchase history.
Alternative Flows or Exceptions	If no purchase history is found, the system displays a message indicating the absence of data.
Non-functional Requirements	Data Integrity, Reliability

Use Case Name	**Analyze Purchase Patterns with AI**
Actors	System
Preconditions	Purchase history data has been successfully retrieved.

Normal Flow	The system analyzes the purchase patterns using AI algorithms to identify trends and preferences.
Alternative Flows or Exceptions	If the analysis fails, an error message is displayed.
Non-functional Requirements	Scalability, Accuracy

Use Case Name	**Generate Shopping Suggestions**
Actors	System
Preconditions	AI analysis of purchase patterns has been completed.
Normal Flow	The system generates personalized shopping suggestions based on the analyzed data.
Alternative Flows or Exceptions	If no suggestions can be generated, the system informs the user that no recommendations are available.
Non-functional Requirements	Personalization, Efficiency

Use Case Name	**Display Suggestions to User**
Actors	User
Preconditions	Shopping suggestions have been successfully generated.
Normal Flow	The system displays the generated shopping suggestions to the user.
Alternative Flows or Exceptions	If there are no suggestions, the system displays a message indicating the lack of recommendations.
Non-functional Requirements	Clarity, User Experience

Use Case Name	**Error Message**
Actors	System
Preconditions	An error has occurred during one of the previous processes.
Normal Flow	The system displays an appropriate error message to the user.
Alternative Flows or Exceptions	The user can retry the action or choose a different option.
Non-functional Requirements	Clarity, Feedback

3.5.3 Object and Class Model

3.5.3.1 Analysis

The Object and Class Model represents the core architecture of the Fridge Genius system. This model provides an organized structure for the system by defining objects, their attributes, methods, and the relationships between them. It ensures modularity, maintainability, and scalability, while facilitating efficient development and testing.

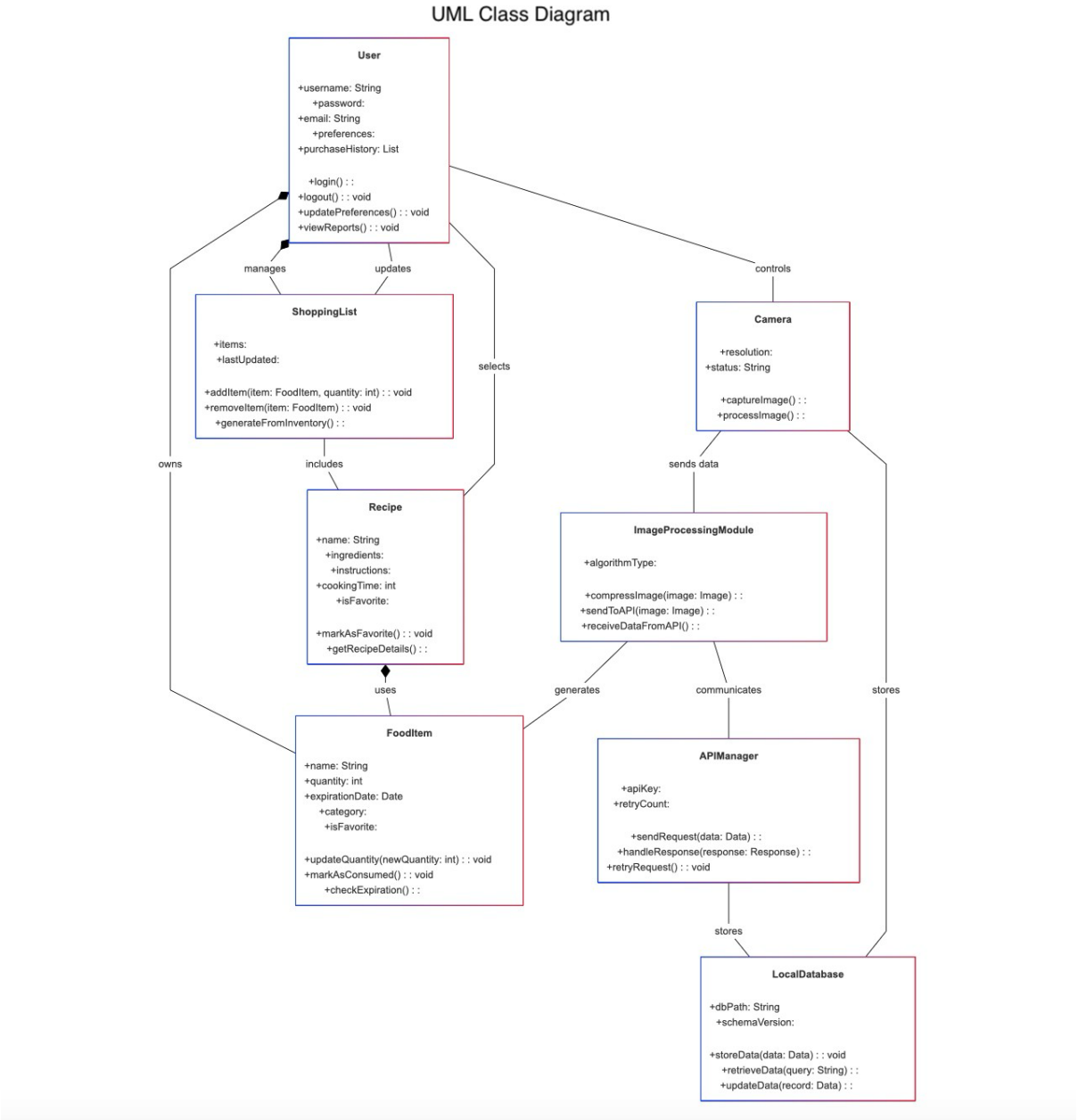

Figure 11: This image shows class diagram of the project.

3.5.3.2 Key Considerations

1. **Modularity**:
    a. Each class handles a specific functionality, making the system easier to maintain and extend.
2. **Scalability**:
    a. The modular design allows for future enhancements, such as adding new AI APIs or supporting Android devices.
3. **Efficiency**:
    a. By using local storage and efficient API calls, the system minimizes latency and dependency on external services.
4. **Fridge Integration**:
    a. Classes like Camera and ImageProcessingModule ensure seamless functionality for integrated hardware.

3.5.4 Dynamic Models

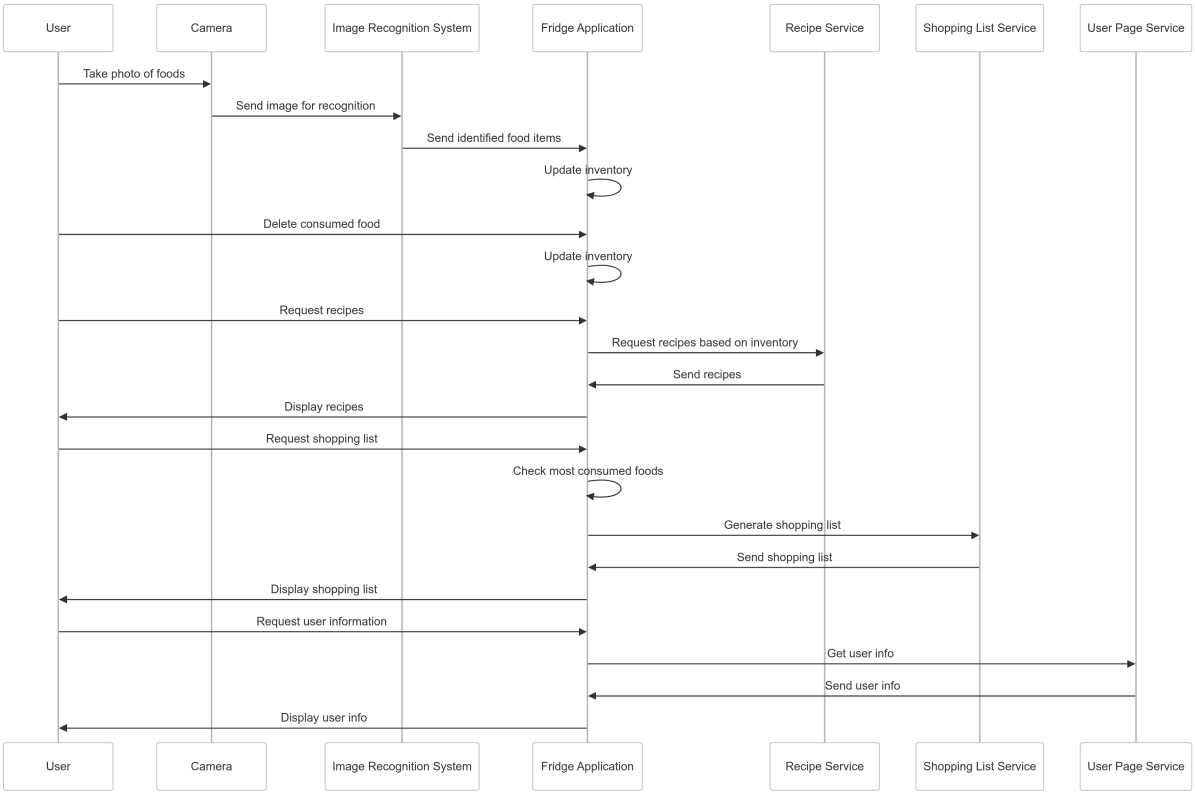

Figure 12: This image shows the sequence diagram of the project.

3.5.5 User Interface – Navigational Paths and Screen Mock-ups

The following will provide flows of the basic tasks that users can perform in the application and sketches of the screens associated with these tasks. These screens will detail how users will access key functions such as browsing for food, viewing recipe suggestions, creating a shopping list, and managing inventory. Additionally, it will be explained how features such as user preferences and notification management are presented in the interface.

Figure 13: These images show mockup design of the project.

4 Glossary

AI (Artificial Intelligence): A branch of computer science that simulates human intelligence through machine learning, natural language processing, and computer vision techniques.
API (Application Programming Interface): A set of protocols and tools for building software applications that allow different programs to communicate with one another.
Compression: The process of reducing the size of a data file without significantly affecting its quality.
Data Privacy: The practice of ensuring that user data is securely handled, stored, and protected from unauthorized access.
GDPR (General Data Protection Regulation): A regulation in EU law on data protection and privacy for individuals within the European Union.

Image Recognition: The process of identifying and categorizing objects or elements within a digital image using AI.

Inventory Management: The systematic approach to managing and tracking the quantity, status, and location of items.
Local Database: A database stored on a user's device, such as SQLite, used for offline storage and retrieval of data.
Scalability: The capacity of a system to handle increased load or expand in size while maintaining performance.

5 References

Bishop, C. M. (2006). *Pattern recognition and machine learning.* Springer.

Google Cloud. (n.d.). *Google Gemini API documentation.* Retrieved from https://cloud.google.com

ISO/IEC 25010:2011. (2011). *Systems and software engineering – Systems and software quality requirements and evaluation (SQuaRE).*

JetBrains. (n.d.). *Datagrip tool documentation.* Retrieved from https://www.jetbrains.com/datagrip/

Nielsen, J. (1993). *Usability engineering.* Morgan Kaufmann.

SQLite Consortium. (n.d.). *SQLite database documentation.* Retrieved from https://sqlite.org

The European Parliament and the Council of the European Union. (2016). *General Data Protection Regulation (GDPR).* Official Journal of the European Union.

Apple Inc. (n.d.). *iOS human interface guidelines.* Retrieved from https://developer.apple.com/design/human-interface-guidelines/