****

**Software Requirements Specification**

**foodo**

**Team 11**

**Alper Çelik, 21903022**

**Batuhan Duras, 22002854**

**Baturalp Sönmez, 22103933**

**Cemal Fırat Dağ, 22003819**

**Davut Durmaz, 21903291**

Bilkent University

Department of Information Systems and Technologies

09.11.2024

Change History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **File Name** | **Document Type** | **Deliverable** | **Version** | **Submission Date** |
| Deliverable\_2.docx | Microsoft 365 | 2 | 0.1 | - |
| Deliverable\_2\_2.docx | Microsoft 365 | 2 | 1 | - |
| Deliverable\_2\_3.docx | Microsoft 365 | 2 | 1.1 | - |
| Deliverable\_2\_4.docx | Microsoft 365 | 2 | 1.2 | - |
| Team1\_SRS\_FR1 | Microsoft 365 | 2 | 2 | 09.11.2024 |

Project Team

|  |  |  |
| --- | --- | --- |
| **Name, Surname** | **Student Id Number** | **e-mail** |
| Alper Çelik | 21903022 | alper.celik@ug.bilkent.edu.tr |
| Batuhan Duras | 22002854 | batuhan.duras@ug.bilkent.edu.tr |
| Baturalp Sönmez | 22103933 | baturalp.sonmez@ug.bilkent.edu.tr |
| Cemal Fırat Dağ | 22003819 | firat.dag@ug.bilkent.edu.tr |
| Davut Durmaz | 21903291 | davut.durmaz@ug.bilkent.edu.tr |

Project Details

|  |  |
| --- | --- |
| **Project Name** | **foodo** |
| **Software Name** | **foodo** |
| **Company Name** | **No sponsor company** |
| **Academic Advisor** | **Dr. Serkan Genç** |
| **GitHub URL** | **https://github.com/foodoHub** |
| **WEB page** | **https://foodo.cfd** |

Individual Contributions Overview

|  |  |
| --- | --- |
| **Name, Surname** | **Summary of Contributions to the Initial Plan Document** |
| Alper Çelik | Contributed to the whole document on equal terms as his teammates. |
| Batuhan Duras | Contributed to the whole document on equal terms as his teammates. |
| Baturalp Sönmez | Contributed to the whole document on equal terms as his teammates. |
| Cemal Fırat Dağ | Contributed to the whole document on equal terms as his teammates. |
| Davut Durmaz | Contributed to the whole document on equal terms as his teammates. |

Executive Summary

The Software Requirements Specification (SRS) for the mobile application "foodo" outlines the comprehensive approach to developing an AI-powered cooking assistant and social network for food enthusiasts. This document covers key aspects, including the scope, functional and non-functional requirements, system architecture, and database structure essential for delivering a robust and user-friendly application. The primary goal of foodo is to provide personalized meal suggestions, adaptive cooking instructions, and interactive social features that engage users of varying cooking skill levels. Detailed user interfaces are specified for processes such as registration, login, AI interaction, and social feed navigation, ensuring an intuitive and accessible user experience. The document emphasizes integration with reliable third-party services such as Firebase for user authentication and cloud storage, and OpenAI and Claude for natural language processing, to facilitate seamless interactions and data handling. Non-functional requirements such as usability, performance, reliability, and security are cautiously defined to support high standards of operation, including real-time user interactions and scalable architecture capable of handling multiple simultaneous users. Constraints related to data privacy, resource availability, and budget limitations are identified, along with assumptions that could affect the implementation of core features. The verification strategy includes manual, exploratory, and compatibility testing to ensure the application meets both functional and non-functional criteria. The SRS also discusses potential issues encountered during its preparation, such as team coordination challenges and economic constraints, ensuring transparency in the development process. This executive overview provides a snapshot of the vital elements and guiding principles for delivering a reliable, engaging, and adaptive cooking experience through the foodo application.[[1]](#footnote-1)

Table of Contents

**Page Number**

[Executive Summary i](#_Toc182046044)

[Table of Contents ii](#_Toc182046045)

[1. Scope 1](#_Toc182046046)

[2. Product Perspective 4](#_Toc182046047)

[3. User characteristics 13](#_Toc182046048)

[4. Assumptions and dependencies 15](#_Toc182046049)

[5. Functional requirements 17](#_Toc182046050)

[6. System Model 29](#_Toc182046051)

[7. Requirements Prototypes 33](#_Toc182046052)

[8. Non-functional Requirements 36](#_Toc182046053)

[1. Usability requirements 36](#_Toc182046054)

[2. Performance requirements 37](#_Toc182046055)

[3. Software system attributes 38](#_Toc182046056)

[4. Constraints 41](#_Toc182046057)

[5. Error Handling Requirements. 42](#_Toc182046058)

[6. Other Non-Functional Requirements 43](#_Toc182046059)

[9. Logical Database Requirements 44](#_Toc182046060)

[10. Verification 51](#_Toc182046061)

[11. Discussions 53](#_Toc182046062)

[1. Limitations and Constraints 53](#_Toc182046063)

[2. Health and Safety Issues 53](#_Toc182046064)

[3. Legal Issues 53](#_Toc182046065)

[4. Economic Issues and Constraints 53](#_Toc182046066)

[5. Sustainability 55](#_Toc182046067)

[6. Ethical Issues 55](#_Toc182046068)

[7. Multidisciplinary Collaboration 56](#_Toc182046069)

List of Tables

**Page Number**

[Table 1. Cross Referencing Functional Requirements to Use Cases 28](#_Toc182045444)

List of Figures

**Page Number**

[Figure 1. UML Use Case Diagram 27](#_Toc182045445)

[Figure 2. Activity Diagram of UC5 29](#_Toc182045446)

[Figure 3. System Sequence Diagram of UC5 30](#_Toc182045447)

[Figure 4. Class Diagram of UC5 30](#_Toc182045448)

[Figure 5. Activity Diagram of UC3 31](#_Toc182045449)

[Figure 6. System Sequence Diagram of UC3 32](#_Toc182045450)

[Figure 7. Class Diagram of UC3 32](#_Toc182045451)

[Figure 8. Prototype Model of UC5 33](#_Toc182045452)

[Figure 9. Prototype Model of UC3 34](#_Toc182045453)

[Figure 10. MS Office subscription billing 54](#_Toc182045454)

[Figure 11. Alper Çelik’s gas receipt 54](#_Toc182045455)

[Figure 12. Cemal Fırat Dağ’s new phone 55](#_Toc182045456)

[Figure 13. Cemal Fırat Dağ’s Macbook provided by Jotform 56](#_Toc182045457)

Abbreviations

|  |  |
| --- | --- |
| AI | Artificial Intelligence |
| AWS | Amazon Web Services |
| GDPR | General Data Protection Regulation |
| HTTP | Hypertext Transfer Protocol |
| HTTPS | Hypertext Transfer Protocol Secure |
| ID | Identifier |
| IOS | iPhone Operating System |
| JSON | JavaScript Object Notation |
| LLM | Large Language Model |
| NLP | Natural Language Processing |
| OS | Operating System |
| OTA | Over-The-Air |
| SA | Software Attribute |
| SDK | Software Development Kit |
| SRS | Software Requirements Specification |
| TLS | Transport Layer Security |
| UML | Unified Modeling Language |
| URL | Uniform Resource Locator |

# Scope

The product is a mobile application named foodo, available on both IOS and Android. The product includes two main components:

* **AI Cooking Assistant:** A smart guide offering personalized meal suggestions and step-by-step cooking instructions based on user preferences and input.
* **Social Network for Food Enthusiasts:** A social platform that creates an environment for food enthusiasts where they can share meals, interact with others, and earn achievements.

foodo will provide the following functionalities to its users:

* Receive tailored meal suggestions based on ingredients, dietary preferences, available time, and special occasions.
* Follow a step-by-step cooking guide with instructions that match the user’s cooking experience and skill level.
* Engage with a community of food lovers, sharing images of meals, and liking, and commenting on posts.
* Earn achievements related to culinary milestones reached by completing cooking activities within the app.

foodo aims to make the cooking experience accessible and enjoyable for a broad range of users from beginners to experienced home cooks. foodo will have the following goals:

**Application of the Software and Relevant Goals:** foodo is designed to make cooking highly accessible, intuitive, and enjoyable for users with diverse culinary skills, from beginners to experienced home cooks. The application's primary objective is to ensure every user can confidently prepare the meal they want with step-by-step, customizable instructions and tailored guidance.

* **Detailed, Adaptive Cooking Guidance**: foodo’s cooking assistant will offer detailed instructions, breaking down complex recipes into manageable steps and adapting based on user feedback. The app will tailor its guidance according to the user's skill level, which is initially recorded during the onboarding process and continually updated as the user gains experience through regular app usage. For beginner users, foodo will provide expanded, detailed instructions to ensure they receive all the necessary guidance to complete each step confidently. For more experienced users, the app will simplify instructions by skipping foundational explanations, while still offering specific details when complexity increases or when certain culinary techniques are involved. This dynamic approach enables foodo to adjust to the changing skill levels and preferences of each user, ensuring they get the right level of support at every stage.
* **Customized Meal Suggestions**: Users can specify preferences such as dietary restrictions, available ingredients, cuisine types, and the time they have to cook. foodo’s AI agent uses these factors to offer highly personalized meal suggestions, ensuring that users can prepare dishes that suit their tastes and circumstances. For instance, if a user has limited time and only a few ingredients, the AI will recommend quick, simple recipes that fit those constraints, allowing for efficient and stress-free meal selection. This adaptability helps users make the most of what they have on hand while ensuring a satisfying cooking experience.
* **Notifications and Timed Reminders**: foodo will provide notifications for key recipe steps like checking the oven, stirring, or flipping items like pancakes or burgers, ensuring users don’t miss critical actions, and complete recipes more efficiently.
* **Building Confidence through Skill-Based Adaptation**: foodo is committed to enhancing user confidence in cooking. For novice cooks, it may start with simple, straightforward recipes, gradually introducing more complex dishes as they become more comfortable. The app monitors user progress, adapting recipes and suggestions to gradually increase cooking complexity as skills improve.
* **Social Network:** foodo aims to make cooking approachable, ensuring every user can prepare meals that meet their expectations. The detailed, interactive guidance and social connection offered through this application differentiate foodo from traditional recipe platforms, providing a supportive and engaging cooking experience.[[2]](#footnote-2)

# Product Perspective

foodo operates as a mobile application available on IOS and Android platforms that is both an AI-powered cooking assistant and a social platform for food enthusiasts. It positions at the forefront of AI-based lifestyle applications, foodo brings a fresh, innovative approach to digital food and cooking experiences.

1. **System Interfaces**

foodo is a standalone mobile app and not part of a larger system. It relies on various external services to operate and deliver its features. These services are essential for foodo to function effectively, but they do not make it part of an integrated system beyond its own application ecosystem.

1. **User Interfaces**

The mobile application delivers a responsive and user-friendly interface across both IOS and Android platforms, utilizing React Native to ensure smooth performance.

1. **Logical Characteristics:** The user interfaces in this application are designed to provide a streamlined, engaging, and intuitive experience for users to interact with various functionalities. Below are the primary interface components, mapped to relevant functional requirements:

* **Minimal Homepage (FReq1.1)**: Upon the first launch, users are greeted with a minimalistic homepage featuring two key buttons: "Login" and "Sign Up." This ensures a clean entry point for both new and returning users, directing them efficiently to the login (FReq1.2) or registration (FReq1.4) processes.
* **Login Interface (FReq1.2)**: The login screen features an "Email Address" input (max 256 characters) validated against standard formats, and a "Password" input field (8-256 characters), hidden by default with an optional toggle for visibility. The "Login" button remains disabled until input fields are valid, ensuring input correctness and usability. Error messages will provide clarity for failed user actions.
* **Forgot Password Interface (FReq1.3)**: Users accessing the "Forgot Password?" functionality are presented with an "Email Address" input and a "Send Reset Link" button, which remains disabled until valid input is provided. This interaction ensures that users can reset their password securely and with clear guidance.
* **Registration Interface (FReq1.4)**: This GUI involves fields for "Name," "Email Address," "Password," and "Re-enter Password," ensuring input validation and guiding users to meet specific criteria for account creation. Feedback mechanisms for invalid inputs and errors improve user comprehension and engagement during the registration process.
* **Main Navigation Interface (FReq2.1)**: The app's footer navigation bar features icon buttons for "Social Feed," "AI Chatbot," and "User Profile." Each icon allows users quick access to primary functions, ensuring seamless transitions and efficient task switching.
* **AI Interaction Interface (FReq3.1.1)**: This screen allows users to input text messages (1-1907 characters) and interact with the AI. The design emphasizes engagement, with relevant responses (like meal suggestions) triggering subsequent actions (e.g., "Let’s cook this!" button for recipe guidance).
* **Cooking Instructions Interface (FReq3.1.3)**: Users follow step-by-step instructions in a card-based format with swiping navigation, providing a visual progress tracker and interaction buttons (e.g., "Next," "Previous," and "Set Timer"). This design focuses on user guidance, progress tracking, and interactive cooking support.
* **Social Feed Interface (FReq2.2)**: After the completion of initial navigation setup (FReq2.1), the social feed screen presents users with an interactive feed where they can view posts in reverse chronological order. A search bar is provided with a “Search” placeholder to allow users to find other users (FReq4.3). A "notifications" icon button is also available, redirecting users to their notifications page (FReq6.1). The interface focuses on user engagement, enabling users to interact with posts, like, comment, and search content seamlessly.
* **Profile Screen (FReq4.1)**: This screen allows users to view their profile (FReq4.1.1), edit profile details (FReq4.1.2), and view achievements and badges (FReq4.1.3). The design emphasizes a clear layout with relevant sections displayed prominently, providing users with easy access to their personal data, profile settings (via a “Settings” icon button on the top right when viewing their own profile), and earned badges. The layout ensures that users can easily manage their personal information and view milestones without confusion or clutter.
* **Achievement Screen (FReq5.1 - FReq5.3)**: The achievements screen categorizes achievements into distinct groups like "Recipe Mastery," "Social Sharing," and "Ingredient Explorer." Badges earned are displayed prominently, with detailed descriptions on how and when they were earned (FReq5.1, FReq5.2). Notifications for newly earned achievements are also displayed in the app’s notification center (FReq5.5). This interface focuses on user engagement by providing a visually appealing and informative display of user milestones, motivating continued app interaction and usage.

1. **Optimization Aspects:**

* **Consistency and Feedback**: The interface maintains consistency in button behaviors, input validation, and error handling, providing predictable and helpful user feedback.
* **Clear Navigation**: Navigation elements, such as the footer bar and buttons help users to quickly understand and move through the app.
* **Error Handling Optimization**: Long and short error messages are provided based on the context of errors, as required in various interfaces. This ensures users are informed of issues in a concise or detailed manner, depending on the situation's complexity.
* **Customizable User Interaction**: For critical tasks, such as input validation during registration or login, dynamic states (e.g., disabling buttons) are used to guide user behavior, improving form completion success rates.

1. **Hardware Interfaces**

**Supported Devices:** The foodo application will be compatible with mobile devices running Android and IOS operating systems. Specifically, it will support devices that meet the following criteria:

* **Android:** Devices running Android version 12.0 (Snow Cone) and above.
* **IOS:** Devices running IOS version 13 and above.

1. **Software Interfaces**
2. **Firebase**

* **Name:** Firebase
* **Mnemonic:** not applicable
* **Specification Number:**
* **Version Number:** 12.4.0
* **Source:** <https://firebase.google.com/>

**Interface Description:**

* **Purpose:** Firebase acts as a backend-as-a-service (BaaS) primarily for user authentication within the foodo application. It simplifies user account management, enabling secure sign-ups and logins while providing real-time synchronization of user data.
* **Message Content and Format:** Interfacing is conducted through REST APIs and Firebase SDKs, utilizing JSON for data interchange.

1. **Expo**

* **Name:** Expo
* **Mnemonic:** not applicable
* **Specification Number:**
* **Version Number:** 51.0.32
* **Source:** <https://expo.dev/>

**Interface Description:**

* **Purpose:** Expo provides an all-in-one toolkit for building, testing, and deploying foodo’s React Native app on IOS and Android. It streamlines development with pre-built native modules, a simple CLI, and supports fast updates through tools like Expo Go for testing and OTA updates for seamless bug fixes and enhancements without app store re-submission. Expo’s managed workflow also simplifies setup for complex features like push notifications and image handling.
* **Message Content and Format:** Uses RESTful APIs and Expo SDKs with JSON data for managing assets, notifications, and OTA updates, ensuring efficient data exchange and integration with the backend.

1. **Kafka**

* **Name:** Kafka
* **Mnemonic:** not applicable
* **Specification Number:**
* **Version Number:** 3.9.0
* **Source:** <https://kafka.apache.org/>

**Interface Description:**

* **Purpose:** Kafka is utilized for managing real-time data streams related to the user's achievements system. It enables the collection and processing of events generated from users' meal shares, allowing the application to analyze user preferences and provide personalized meal suggestions based on their likes and interactions with shared meals.
* **Message Content and Format:** Communicates through a publish-subscribe model, using JSON for message formatting. Each message represents a user action, which can be consumed by different services to update user profiles and generate recommendations.

1. **Qdrant**

* **Name:** Qdrant
* **Mnemonic:** not applicable
* **Specification Number:**
* **Version Number:**
* **Source:** <https://qdrant.tech/>

**Interface Description:**

* **Purpose:** Qdrant is utilized as a vector database to store and retrieve embeddings essential for the training of our AI chatbot. This facilitates the classification of data for effective semantic search and AI-based recommendations, allowing the chatbot to provide relevant responses based on user input and preferences.
* **Message Content and Format:** Communicates through RESTful APIs, using JSON-formatted requests and responses.

1. **Ubuntu**

* **Name:** Ubuntu
* **Mnemonic:** not applicable
* **Specification Number:**
* **Version Number:** 24.04.1
* **Source:** <https://ubuntu.com/>

**Interface Description:**

* **Purpose:** Ubuntu serves as the primary operating system for server-side components of foodo, especially for hosting backend services in a stable, Linux-based environment.
* **Message Content and Format:** Communication happens at the operating system level. No specific message format is required, as Ubuntu is the OS layer for running server applications and Docker containers.

1. **Docker**

* **Name:** Docker
* **Mnemonic:** not applicable
* **Specification Number:**
* **Version Number:**
* **Source:** <https://www.docker.com/>

**Interface Description:**

* **Purpose:** Docker is used to containerize foodo services, ensuring consistency and ease of deployment across different environments. It allows services to be packaged with their dependencies, providing scalability and flexibility.
* **Message Content and Format:** Uses Docker API and Docker Compose files to define and manage containers. Interfacing details can be found in Docker’s official API documentation.

1. **Open AI**

* **Name:** Open AI
* **Mnemonic:** not applicable
* **Specification Number:**
* **Version Number:**
* **Source:** <https://openai.com/>

**Interface Description:**

* **Purpose:** The OpenAI API provides NLP capabilities for personalized assistance, helping users receive meal suggestions and conversational support from the AI-powered cooking assistant.
* **Message Content and Format:** JSON format through HTTPS requests. Detailed documentation on message structure and formats is provided by OpenAI API resources.

1. **Claude**

* **Name:** Claude
* **Mnemonic:** not applicable
* **Specification Number:**
* **Version Number:**
* **Source:** <https://claude.ai/>

**Interface Description:**

* **Purpose:** The Claude API delivers advanced NLP functionalities for interactive conversations, assisting users in receiving tailored meal suggestions and engaging responses from the AI cooking assistant.
* **Message Content and Format:** Communicates through JSON over HTTPS requests. The structured API responses and request formats are detailed in the Claude API documentation.

1. **Amazon Web Services**

* **Name:** Amazon Web Services
* **Mnemonic:** AWS
* **Specification Number:**
* **Version Number:**
* **Source:**<https://aws.amazon.com/>

**Interface Description:**

* **Purpose:** AWS Lambda enables foodo to execute backend code in response to events without managing servers. Lambda will be used for serverless functions such as triggering notifications, processing AI requests, updating user achievements, and performing background tasks related to meal suggestions and cooking analytics.
* **Message Content and Format:** AWS Lambda functions are invoked via HTTP(S) requests. The communication format will typically involve JSON for both requests and responses. AWS SDKs will be used to interact with Lambda.

1. **Google Vertex AI**

* **Name:** Google Vertex AI
* **Mnemonic:** not applicable
* **Specification Number:**
* **Version Number:**
* **Source:** <https://cloud.google.com/vertex-ai>

**Interface Description:**

* **Purpose:** Google Vertex AI assists in model training, deployment, and management for AI-based features within foodo, such as generating embeddings for recipe similarity and categorization.
* **Message Content and Format:** Interfaces via REST APIs and JSON-formatted data, with structured responses per Google’s API guidelines.

1. **Communication Interfaces**

* **HTTPS/REST API:** Serves as the primary mode of interaction between the front-end application and backend services, facilitating secure data transmission and structured interactions with cloud-based resources.
* **SSH (Secure Shell):** Essential for secure remote access to servers and management of backend services, ensuring that system administrators can effectively maintain the infrastructure.

1. **Memory Constraints**

The foodo application is designed to provide users with seamless cooking assistance experience while managing resources efficiently. To ensure optimal performance across supported devices (Android 12.0 and above, IOS 13 and above), the following memory constraints have been established based on the typical memory consumption patterns observed in popular mobile applications:

**Estimated Memory Consumption**

* **Active Usage:** The foodo application should aim to maintain a memory usage of approximately **150-300 MB** during active interactions, such as browsing recipes, receiving AI suggestions, and sharing meals.
* **Background Usage:** When running in the background, the application should ideally consume no more than **100-200 MB** to minimize the impact on overall device performance and battery life.

# User characteristics

**Demographic Characteristics**

**Age**: The app can cater to a wide age range, typically targeting users aged 18-45, who are likely to engage with social media, cooking, and AI-driven assistance.

**Gender**: Gender-inclusive; the app should be appealing and accessible to all genders.

**Geographical Location**: The app could be designed for global accessibility, with a focus on regions where mobile app usage is high and there is interest in cooking, recipes, and social interactions.

**Technological Characteristics**

**Device Ownership:** Users should own or have access to smartphones capable of running mobile apps (iOS or Android).

**Internet Access**: Users should have reliable access to the internet, as many features (e.g., AI interactions, social feeds) rely on network connectivity.

**Familiarity with Mobile Apps:** Users should have basic knowledge of navigating mobile apps, including interacting with buttons, input fields, and notifications.

**Behavioral Characteristics**

**Interest in Cooking and Food:** Users should have a general interest in cooking, trying new recipes, and improving their cooking skills. The app targets users interested in engaging with meal suggestions and guided cooking tutorials.

**Social Interaction Preference:** Users should have a desire to connect with others through sharing posts, liking and commenting on content, and forming social connections through friend requests.

**Openness to AI Interaction:** Users should be comfortable interacting with AI assistants for meal recommendations and cooking guidance.

**Skill Levels**

**Cooking Skills:** The app targets users with various cooking skill levels, from novices to experienced cooks. Customization options and tailored suggestions accommodate all user groups.

**Technology Proficiency:** While users are not expected to be tech experts, they should have basic mobile app navigation skills.[[3]](#footnote-3)

# Assumptions and dependencies

The following assumptions and dependencies are factors that affect the requirements specified in this SRS. While they are not direct constraints on the software design, any changes to these factors may impact the feasibility, functionality, or delivery of foodo.

* **Mobile Operating Systems Availability**: It is assumed that foodo will be compatible with the latest versions of iOS and Android. Any significant changes in mobile OS versions, updates, or deprecations that affect core functionalities (e.g., notifications, screen display, or user permissions) may require adjustments to the app’s design and feature implementation.
* **Access to AI Services:** The personalized meal suggestions and adaptive cooking guidance in foodo depend on reliable access to AI models. It is assumed that these AI services will remain available at a consistent performance level throughout development and deployment. Any change in access, pricing, or service availability of AI resources could require re-evaluation of foodo’s functionality or the consideration of alternative AI implementations.
* **Data Availability and Quality for AI Training**: foodo relies on high-quality data to train and improve its AI models, ensuring accurate meal suggestions and adaptive cooking guidance. It is assumed that the necessary datasets—such as culinary information, ingredient databases, and user preference data—will remain accessible, up-to-date, and licensed for use throughout the project. Any disruption in data availability, occurrence of broken or faulty data, or changes in data licensing could impact the AI’s accuracy and functionality. If data sources become unreliable or unavailable, alternative datasets or sources may need to be considered, which could affect foodo’s performance and require adjustments to the project’s requirements.
* **Cloud Infrastructure Availability:** foodo relies on cloud infrastructure for a smooth user experience. It is assumed that cloud resources such as AWS will remain stable and affordable throughout the project lifecycle. If there are changes in cloud service access, pricing, or reliability, adjustments may be needed in how foodo stores and manages data or handles real-time processing.
* **User Device Capabilities:** The functionality of foodo assumes that users’ mobile devices will have basic features such as a stable internet connection, push notification support, and sufficient processing power to run the app smoothly. If user devices lack these capabilities, it may limit the app’s performance or accessibility.
* **Community Guidelines and User Content Control:** foodo’s community features depend on guidelines for safe interactions and content moderation. It is assumed that the app will have access to tools for user reporting, content filtering, and moderation. If these tools or guidelines change, the social experience may need to be adjusted.
* **Stable Development Environment:** foodo depends on a reliable development environment that includes key tools and services such as IntelliJ IDEA for coding, React Native for mobile development, Spring Boot for backend services, and GitHub for version control. Continued access to essential third-party APIs—such as Firebase for user authentication, and AWS S3 for data storage—is also critical. Any disruption, version deprecation, or loss of support for these tools or services could delay the project, cause compatibility issues, and impact the overall quality of foodo’s final product.

# Functional requirements

1. **Requirements List**

**FReq1.1:** Upon the first launch, the mobile app shall display a minimal homepage with the following elements:

* A **"Login"** button shall direct users to the login screen to complete **FReq1.2.**
* A **"Sign Up"** button shall direct users to the registration screen to complete **FReq1.4.**

**FReq1.2:** Upon completing **FReq1.1**, the mobile app shall display a GUI with the following elements:

* An **"Email Address"** input field, accepting a maximum of 256 characters and validated against standard email format rules.
* A **"Password"** input field, accepting a minimum of 8 and a maximum of 256 characters. The password shall be hidden by default, with an option to display the entered text.
* A **"Login"** button remains disabled until both fields are valid. When pressed, the app shall use Firebase Authentication to log in the user. If the credentials are incorrect or the account does not exist, an error message (**EH-U2**) will be displayed. If there is no error, the system shall redirect the user to the social feed (**FReq4.4.1**).
* A **"Forgot Password?"** link, allowing users to initiate the **FReq1.3.**
* A **"Back"** button which returns the user to homepage, also clears “Email Address” and “Password” fields.

**FReq1.3:** The mobile app shall display a GUI with "Forgot Password?" title and the following elements:

* An **"Email Address"** input field, accepting a maximum of 256 characters and validated against standard email format rules.
* A **"Send Reset Link"** button remains disabled until the “Email Address” field is valid. When pressed, the app shall use Firebase Authentication to send a password reset link. If Firebase Authentication System handles password requests successfully then the system shall notify the user to check his/her email inbox.

**FReq1.4**: Upon completing **FReq1.1**, the mobile app shall display a GUI containing:

* A "Next" button.
* A "Name" input field, accepting a minimum of 2 and a maximum of 64 characters.
* An "Email Address" input field, accepting a maximum of 256 characters, which shall be validated against standard email format rules.
* A "Password" input field, accepting a minimum of 8 and a maximum of 256 characters, which must contain at least one number, one lowercase letter, one uppercase letter, and one special character from the set “!"#$%&'()\*+,-./:;<=>?@[]^\_`{|}~”.
* A "Re-enter Password" input field, which must match the "Password" field.

The "Next" button shall remain disabled until all fields are valid according to the specified criteria. Once all criteria are met, the "Next" button shall become enabled. When the "Next" button is pressed, the app shall send all collected information to the Firebase Authentication Service to create a new user account. If errors are detected in the entered information, the app shall display an error label in accordance with error handling requirements (**EH-U1**). If there is no error, the system shall show user the onboarding GUI to complete **FReq1.5.**

**FReq1.5:** After **FReq1.4** is complete, the mobile app shall display a new GUI with the following elements:

* A list of checkboxes for selecting dietary restrictions.
* A dropdown menu labeled "Cooking Level" with the options "Novice", “Average”, “Expert".
* A list of checkboxes labeled "Favorite Meals" that allows users to select their favorite meals.
* A "Start" button at the bottom of the GUI, initially disabled.

The "Start" button shall become enabled when both a dietary restriction and a cooking level have been selected, once the “Start” button is clicked the system shall redirect the user to the AI Chatbot Page (**FReq3.1.1**).

**FReq2.1**: The mobile app shall provide a navigation bar on the footer including:

* “Social Feed” Icon Button on the left which redirects users to Social Feed Page (**FReq2.2**).
* “AI Chatbot” Icon Button in the middle which redirects users to AI Chatbot Page (**Freq2.3**).
* “User Profile” Icon Button on the right which redirects users to the profile page (**FReq2.4**).

**FReq2.2:** After the completion of **FReq2.1**, the mobile app shall enable users to use the social feed navigation header including:

* Text field with a “Search” placeholder to search users (**FReq4.3**).
* A “notifications” icon button which redirects users to notifications page (**FReq6.1**).

**FReq2.3:** Upon the completion of **FReq2.1**, the mobile app shall show the AI Chat Page (**FReq3.1.1**) with a navigation bar on top including:

* “Show More” Icon Button on the left, which redirects users to the “Chat List” page (**FReq3.3.1**) when pressed.
* “Delete Chat” Icon Button on the right, which enables users to delete chat (**FReq3.3.2**).

**FReq2.4:** The mobile app shall enable users to navigate to the profile page (**FReq4.1.1**) and a “Settings” icon button on top right of the corner, which is only shown when user is viewing his/her profile page, redirects user to profile settings page (**FReq4.1.2**).

**FReq3.1.1**: The mobile app shall provide users with the ability to interact with an AI assistant via text input accepting a minimum of 1 and a maximum of 1907 characters, and a “Send” button. When the “Send” button is clicked all the user input content will be sent to foodo AI to process (**FReq3.1.2**). If there is an error on the server side, an error message (**EH-S1**) will be shown to the user. If not and the response coming from foodo AI is a meal recipe, the system shall show a “Let’s cook this!” button to start the cooking instructions process (**FReq3.1.3**).

**FReq3.1.2**: When the **FReq3.1.1** is completed, the system shall use LLM to process user input and give a meal suggestion or recipe response.

**FReq3.1.3:** The app shall present cooking instructions in a card-based format, allowing users to easily navigate through each step of the process by swiping (swipe left for previous step, swipe right for next step) or previous and next icon buttons. Each card will represent a single cooking step and shall include the following elements:

* **Progress Tracker (FReq3.1.5)** on the top,
* **Step Number and Title** below the Progress Tracker,
* **Instruction Details** below the Step Number and Title,
* **Clarification Button** for requesting further clarification (**FReq3.1.7**) or more details about the current step below the Instruction Details,
* **Set Timer Button (Optional)** enabling the user to set a timer **(FReq3.1.4)** if the current cooking step involves a timed action,
* **Next Button** allowing users to move to the next step on the bottom right of the card,
* **Previous Button** allowing users to move to the previous step on the bottom left of the card.

**Freq3.1.4**: The app shall include a timer feature that users can set for cooking steps (**FReq3.1.3**) requiring specific time durations (e.g., baking, boiling). The system shall provide automatic notifications when the set time has elapsed. The timer settings shall be adjustable, and users shall be able to stop, pause, or reset the timer as needed.

**Freq3.1.5**: The app shall provide users with a visual progress tracker on top of the step cards (**FReq3.1.3**), showing which steps have been completed and which steps remain. The progress tracker shall update automatically as the user moves through each step.

**Freq3.1.6**: If a user navigates away from the cooking instructions screen (**FReq3.1.3**), the app shall save the user’s current position within the recipe and any timer settings (**FReq3.1.4**). Upon returning to the instructions, the app shall display the user’s saved position and prompt them to resume from the last completed step or restart from the beginning.

**FReq3.1.7:** The system shall enable user to get further instructions about the step in the cooking instructions (**FReq3.1.3**).

**FReq3.2.1:** Upon completion of a recipe guided by the AI assistant, the mobile app shall show the user "Share Your Dish" option that redirects them to the post sharing screen (**FReq4.4.2**) for sharing their cooked meal. The recipe field will be immutably prefilled with the recipe to complete **FReq4.4.2**.

**FReq3.3.1:** Upon accessing the "AI Chat" section (from the navigation bar or other designated access points), the mobile app shall display a list of previous AI chat sessions. Each entry in the list shall include the following:

* Chat preview text, displaying the first few characters of the last exchange within the chat.
* Timestamp indicating when the last message was sent/received.
* An icon or button to "Delete Chat" (**FReq3.3.2**).  
  When a user selects a chat entry, the system shall redirect the user to the detailed view of that AI chat session, allowing further interactions.

**FReq3.3.2:** Within the AI Chat list (**FReq3.3.1**), users shall have the option to delete a specific chat session by tapping a "Delete" button/icon next to the chat preview. Upon selection, the system shall display a confirmation dialog box asking, "Are you sure you want to delete this chat session?" If the user confirms, the system shall permanently delete the chat and update the chat list accordingly. If the user cancels, no changes shall occur, and the chat shall remain in the list.

**FReq4.1.1**: When a user navigates to their profile or another user's profile, the mobile app shall display a detailed profile view, including:

* Profile picture (if set by the user).
* Name (if applicable).
* Count of friends (if applicable).
* List of recent posts (**FReq4.4.1** for viewing others' posts).  
  A "Settings" icon shall appear on the top right corner if the user is viewing their own profile, allowing access to profile settings (**FReq4.1.2**).

**FReq4.1.2:** When a user selects the "Settings" icon from their profile view, the system shall display a "Profile Settings" screen containing the following fields:

* "Name" input field, allowing the user to update their name with a minimum of 2 and a maximum of 64 characters.
* "Profile Picture" upload option.
* "Save Changes" button, which remains disabled until at least one field is modified.  
  Upon saving, the system shall validate the input and update the user's profile. If validation fails, the system shall display relevant error (**EH-U1**) messages.

**FReq4.1.3:** The profile page shall display a dedicated section for achievements and badges earned by the user. This section shall include:

* A grid or list view of earned badges, with each badge displaying its name and a brief description.
* Users shall be able to tap on individual badges to view detailed descriptions and requirements.

**FReq4.2.1**: Users shall be able to send friend requests to other users via their profile page or other designated access points (e.g., search results). Upon pressing the "Send Friend Request" button, the system shall validate the request (e.g., check for existing requests or friendships). If valid, the system shall send a friend request notification to the recipient (**FReq6.2**).

**FReq4.2.2**: Users shall receive friend requests via notifications. When a user views a friend request, the system shall present "Accept" and "Reject" buttons. If "Accept" is selected, the system shall update both users' friend lists (**FReq4.2.3**) and send a confirmation notification. If "Reject" is selected, the request shall be removed, and the sender shall not be notified.

**FReq4.2.3**: Users shall be able to view a list of their friends from their profile. The list shall display the friend's profile picture, name, and options to view their profile or remove them (**FReq4.2.4**).

**FReq4.2.4**: Users shall have the option to remove friends from their friend list. Upon selecting "Remove Friend," the system shall display a confirmation dialog box. If confirmed, the system shall update the user's friend list and remove the friend relationship.

**FReq4.3:** The system shall provide a search bar within the "Social Feed", allowing users to search for other users by name. As users’ type, the system shall provide a dropdown with matching results. Selecting a result shall redirect the user to the selected profile.

**FReq4.4.1:** The mobile app shall display posts from other users in a social feed, sorted in reverse chronological order (most recent posts first). Each post shall include:

* User's profile picture and name.
* Post content (text, images, etc.).
* Date and time of posting.
* Interaction buttons (e.g., "Like", "Comment").

**FReq4.4.2:** Users shall be able to create posts by accessing a "Create Post" screen. This screen shall include:

* Caption input field accepting a maximum of 1907 characters.
* Recipe input field accepting a maximum of 1907 characters
* Option to upload images.
* "Post" button that remains disabled until at least one content field is filled.  
  Upon posting, the system shall validate and publish the content to the social feed.

**FReq4.4.3:** Users shall be able to edit their posts by selecting the "Edit" button from a post's options menu. The system shall display the original content in an editable format. Users can modify the text and images and then save the changes. Upon saving, the system shall validate and update the post.

**FReq4.4.4:** Users shall have the option to delete their own posts. Upon selecting "Delete Post," the system shall display a confirmation dialog. If confirmed, the post shall be permanently removed from the feed.

**FReq4.5.1:** Users shall be able to "Like" posts by pressing a "Like" button. The system shall update the like count and display the change in real-time.

**FReq4.5.2:** If a user has liked a post, they shall be able to "Unlike" it by pressing the same button. The system shall update the like count accordingly.

**FReq4.6.1:** Users shall be able to add comments to posts. The system shall display a comment input field allowing a maximum of 256 characters below each post. Upon submission, the system shall validate and add the comment to the post's comment section.

**FReq4.6.2:** Users shall have the option to edit their own comments. Upon selecting "Edit," the system shall display the original comment in an editable format. Once changes are saved, the system shall validate and update the comment.

**FReq4.6.3:** Users shall have the option to delete their own comments. Upon selecting "Delete," the system shall display a confirmation dialog. If confirmed, the comment shall be removed.

**FReq5.1:** The mobile app shall provide a dedicated section within the user's profile to view all earned achievements and badges. This section shall include the following features:

* A grid or list view displaying each earned badge, with a visual icon, badge name, and brief description.
* Clicking on a badge shall display additional details, including when and how the badge was earned, milestones achieved (e.g., "5 Recipes Completed" for Recipe Mastery).
* The app shall notify (**FReq6.2**) users when they earn a new achievement. Users shall be able to view details of the new achievement through the notification.

**FReq5.2:** The app shall automatically award achievements to users based on predefined criteria and milestones. Examples include:

* Recipe Completion Milestones: Users shall earn badges based on the number of completed recipes (e.g., 5, 10, 25, 50 recipes). Upon reaching each milestone, the system shall display a new achievement badge within the "Recipe Mastery" category.
* Post Sharing Milestones: Users shall earn badges for sharing posts within the app. Each milestone (e.g., 10, 20, 50 posts shared) shall grant a unique achievement badge, displayed in the "Social Sharing" category.
* Category-Based Recognition: Achievements shall be grouped by types such as "Recipe Mastery," "Social Sharing," and "Ingredient Explorer," with badges awarded based on activity and proficiency in specific areas.
* Notifications shall be triggered whenever a user earns a new badge, appearing in the app’s notification center and including details of how and when the achievement was earned. Users shall be able to access these details from the achievements section of their profile.

**FReq6.1:** Users shall be able to view notifications related to friend requests, post interactions, etc., on a dedicated "Notifications" page.

**FReq6.2:** The system shall send notifications to users based on interactions, such as friend requests and post likes.

**FReq6.3:** Users shall have the option to delete individual notifications by selecting a "Delete" icon.

**FReq6.4:** Users shall have the option to clear all notifications with a single action, confirmed via a confirmation dialog.

**FReq7.1**: If a user interacts with the AI assistant for help during a recipe, the app shall send a feedback message after the interaction, asking whether the AI response was helpful. Options shall include “Helpful,” “Somewhat Helpful,” and “Not Helpful,” along with an optional comment field for additional feedback. This input shall be used to improve the accuracy and relevance of AI responses.

**FReq7.2**: Upon completing a cooking session, the app shall display a summary feedback pop-up allowing users to rate their overall experience with the recipe. This summary shall prompt users to rate multiple aspects, such as taste, cooking instructions, and ingredient availability, to gather comprehensive feedback on the entire experience.

1. **Use Case Diagram**

**A diagram of a diagram

Description automatically generated**

Figure . UML Use Case Diagram

1. **Mapping Functional Requirements to Use Cases**

Table . Cross Referencing Functional Requirements to Use Cases

|  |  |  |
| --- | --- | --- |
| **Functional Requirement ID** | **Use Case ID** | **Initial Plan Functional Requirement ID** [1] |
| FReq1.1, FReq1.2 | UC1 | FReq1 |
| FReq1.3 | UC2 | FReq1 |
| FReq1.1, FReq1.4, FReq1.5 | UC3 | FReq1 |
| FReq2.1, FReq2.2, FReq2.3, FReq2.4 | UC4 | - |
| FReq3.1.1, FReq3.1.2 | UC5 | FReq2.1 |
| FReq3.1.3, FReq3.1.4, FReq3.1.5, FReq3.1.6 | UC6 | FReq2.2 |
| FReq3.2.1, FReq4.4.2, FReq4.4.3, FReq4.4.4 | UC7 | FReq3.3 |
| FReq4.4.1 | UC8 | FReq3.4 |
| FReq4.1.1 | UC9 | FReq3.4 |
| FReq4.1.2 | UC10 | FReq3.1 |
| FReq4.1.3, FReq5.1 | UC11 | FReq4 |
| FReq5.2 | UC12 | FReq4 |
| FReq4.2.1, FReq4.2.2, FReq4.2.4 | UC13 | FReq3.2 |
| FReq4.2.3 | UC14 | FReq3.2 |
| FReq6.2, Freq3.1.4, FReq4.2.1, FReq4.2.2, FReq4.5.1, FReq4.6.1, FReq5.2 | UC15 | - |
| FReq6.1 | UC16 | - |
| FReq6.3, FReq6.4 | UC17 | - |
| FReq7.1, FReq7.2 | UC18 | FReq5 |
| FReq4.5.1, FReq4.5.2, FReq4.6.1, FReq4.6.2, FReq4.6.3 | UC19 | FReq3.3 |
| FReq4.3 | UC20 | - |
| FReq3.3.1 | UC21 | - |
| FReq3.3.2 | UC22 | - |

# System Model

1. **A diagram of a flowchart

   Description automatically generatedFirst Use Case – Use Case 5 – Get Meal Suggestions**

Figure . Activity Diagram of UC5

A diagram of a diagram

Description automatically generated

Figure . System Sequence Diagram of UC5

A screen shot of a computer screen

Description automatically generated

Figure . Class Diagram of UC5

1. **Second Use Case – UC3 – Registration**

A diagram of a software system

Description automatically generated with medium confidence

Figure . Activity Diagram of UC3

A diagram of a computer

Description automatically generated

Figure . System Sequence Diagram of UC3

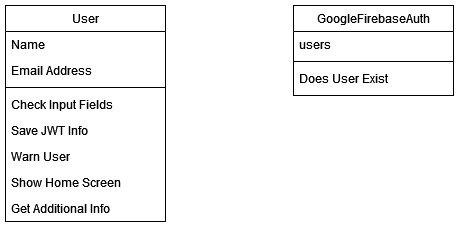


Figure . Class Diagram of UC3

# Requirements Prototypes

For the System Modal section, UC3 (Registration) and UC5 (Meal Suggestions) were selected for detailed analysis because they were the most complicated 2 use cases among all use cases. Both use cases involve multiple steps, conditional logic, and they require seamless integration with third-party services—Firebase for user authentication in the case of UC3, and a large language model (LLM) provider for generating personalized meal suggestions in UC5. This complexity makes these use cases important to the core functionality and user experience of the application.

**Prototype Use Case 1: UC5 - Get Meal Suggestions**

**A screenshot of a chat

Description automatically generated**

Figure . Prototype Model of UC5

The prototype demonstrates the interaction between the user and the AI assistant for meal suggestions in a conversational chat interface.

* The user begins by requesting a meal suggestion (e.g., "I'm in the mood for pasta. Any suggestions?").
* The AI assistant responds with a specific suggestion (e.g., "I have a great recipe for creamy mushroom pasta. Would you like to try it?").
* Below the AI response, a "Let's Cook This!" button is displayed, providing users with a direct option to proceed with detailed cooking instructions for the suggested meal.

This prototype illustrates a complete user flow from the AI-generated meal suggestion to the user's ability to engage further by opting to start the cooking process. By clicking the "Let's Cook This!" button, users are directed to step-by-step cooking instructions as described in FReq3.1.3. The conversation flow aligns with FReq3.1.1 and FReq3.1.2, demonstrating the AI's role in providing meal suggestions and actionable options for users.

The visual representation emphasizes how the app supports user engagement by offering a seamless transition from meal suggestion to guided cooking, leveraging third-party LLM services to deliver personalized and contextualized responses.

**Prototype Use Case 2: UC3 - User Registration**

**A screenshot of a phone

Description automatically generated**

Figure . Prototype Model of UC3

The provided prototype screenshot for the User Registration use case displays a simple yet intuitive registration form for new users of the "foodo" application. The screen contains the following input fields:

* Name
* Email
* Password
* Password Confirmation

Below the fields, there is a prominent "Next" button, which remains disabled until all fields are validated and filled correctly.

This prototype captures the core user registration process, demonstrating the essential interface elements required for new user onboarding. Upon entering all necessary information, the "Next" button allows progression to the subsequent stages of the onboarding journey, such as dietary preferences (per functional requirements). Input validation for each field ensures accurate data entry, meeting security and user experience standards.

# Non-functional Requirements

## Usability requirements

**UR1.1**: The app must provide clear and immediate feedback for user actions, such as confirming successful post shares or issuing alerts when an error occurs. All feedback notifications should display within **2.5 seconds** of the action.

**UR1.2**: Notifications, including those for timed cooking steps, alerts, and achievements, must be easily understandable and dismissible, ensuring clarity and ease of navigation for the user.

**UR2.1**: foodo must ensure consistent design elements, icons, and interactions across both iOS and Android platforms. UI components should behave similarly to maintain a uniform experience for users switching between devices.

**UR3.1**: Essential tasks, such as interacting with the AI assistant and accessing social features, should be accessible within **five taps** from the home screen.

**UR3.2**: The design will prioritize clean visuals, clear icons, and concise language. User testing should indicate that **75% of users rate ease of navigation as "easy"** or higher.

**UR4.1**: The app should provide adaptive guidance based on user preferences and cooking skill levels, configured during onboarding and updated based on user interactions.

**UR4.2**: User feedback should indicate **75% satisfaction** with the relevance of personalized guidance.

**UR5.1**: Delay between the click and the operation performed should be less than **5 seconds on average**.

**UR5.2**: **75% of functional requirements** should be completed in **10 taps or fewer**, ensuring quick access to core features.

**UR6.1**: Animations and transitions within the app, such as starting a conversation with the AI Assistant, or navigating to the social feed, should be completed within **0.5 to 2.5 seconds** for a seamless user experience.

**UR7.1**: foodo must include clearly understandable messages in prompts and confirmation steps to prevent errors in critical actions, such as posting content or deleting items.

**UR7.2**: foodo must provide clear, contextual error messages that explain the issue and guide the user to resolve it. Error messages should display within **5 seconds** and use simple, concise language to help users quickly understand and correct the action.

## Performance requirements

**PR1.1:** The system shall support up to 20 simultaneous users accessing the app during peak times without performance degradation. This includes both active and passive users interacting with social and AI features.

**PR1.2:** The app shall scale to handle up to 100 unique sessions daily. This scalability shall ensure smooth performance during high-demand periods.

**PR2.1:**Core interactions, such as recipe searches, AI responses, and social actions (e.g., likes, comments), shall be processed in under 5 seconds for 90% of requests, maintaining a high level of responsiveness.

**PR2.2:** For low-connectivity scenarios, such as 3G networks, response time shall not exceed 20 seconds for core interactions to ensure usability across different network conditions.

**PR2.3:** The AI cooking assistant shall answer the prompts in under 20 seconds after receiving input, with a maximum time of 60 seconds during peak usage, supporting smooth and prompt user engagement.

**PR3.1:** Updates between the app and cloud database shall synchronize at 25-second intervals, ensuring that actions such as posting or updating profile details are promptly reflected across devices.

**PR3.2:**User-generated content, such as images in the social feed, shall load within 5 seconds for 90% of requests, ensuring smooth content interaction.

**PR4.1:** Push notifications, including cooking reminders and social alerts, shall be delivered to user devices within 10 seconds of the triggered event, ensuring timely updates for important actions.

**PR4.2:** The app shall send cooking step alerts (e.g., flipping, stirring) within 10 seconds of the scheduled time, ensuring accuracy in time-sensitive guidance.

**PR5.1:** Image uploads for recipes and social posts shall be completed within 10 seconds for a standard max 5 MB image. Uploaded images shall display in the user’s feed within 20 seconds to enhance the sharing experience.

**PR6.1:** The app shall back up user data to the cloud every 168 hours to ensure data safety and recovery in case of system failure.

**PR6.2**: In the event of an outage, such as a **server failure, network disruption, or database corruption**, user data shall be fully recoverable within **24 hours**. This will minimize downtime, preserve user content, and ensure continuity of user preferences across sessions.

**PR7.1:** The system shall process up to **500 MB of image data per hour** under normal conditions for transfers to and from AWS S3 buckets, supporting routine uploads and downloads of user-generated content (e.g., recipe images and social posts) without impacting performance or response times.

## Software system attributes

**a) Reliability**:

**SA-R1:** All essential functions, including user authentication and AI assistance, shall have backup systems in place to keep them running smoothly. This ensures that, even if part of the system fails—like a server going down or a network issue—these critical features will continue to work without major interruptions for users.

**SA-R2:** The system shall include fallback options for essential services, such as data storage and AI responses, ensuring that core functions remain accessible even if a primary component experiences downtime. For example, if AWS S3 object storage is unavailable, MinIO will serve as a local alternative to ensure continuous data access and user experience.

**SA-R3:** To ensure reliability at delivery, at least 95% of the foodo’s functions (e.g., posting photo) should be covered by automated tests. This includes unit tests for each module and integration tests to ensure all components work together without issues.

**SA-R4:** Users should get an answer. They shouldn’t wait forever.

**b) Availability**:

**SA-A1:** The system shall maintain an availability level of 89.55% on a weekly basis, minimizing the risk of downtime.

**SA-A2:** The system shall employ checkpoint and recovery mechanisms to enable rapid recovery following service interruptions, with restart processes requiring no more than 1 hour to restore core functions.

**SA-A3:** Scheduled maintenance shall occur during off-peak hours to reduce impact on users, and users shall receive prior notifications for any maintenance that may result in temporary unavailability.

**c) Security**:

**SA-S1**: The application shall implement Transport Layer Security (TLS) 1.2 or higher with a valid SSL/TLS certificate for HTTPS, encrypting all data transmitted between the client and server.

**SA-S2:** Firebase will handle all password management, providing secure access controls and reducing risks associated with credential storage.

**SA-S3:** Database credentials and sensitive encryption keys shall be securely managed using a Key Management Service (e.g., AWS KMS). This setup will prevent unauthorized access, protect sensitive information, and ensure that credentials remain confidential and inaccessible to unauthorized entities.

**SA-S4**: the system shall maintain an audit log of all critical actions performed by users and administrators. Audit logging shall be configured using **rsyslog** on EC2 instances, with regular log rotation managed by **cron jobs** to ensure consistent and secure log management.

**SA-S7**: User data privacy shall comply with data protection regulations (e.g., GDPR) and include consent for data collection and processing.

**SA-S8:** The application shall safeguard the AI interaction flow by limiting prompt complexity, input length, and validating prompt structure, ensuring consistent user experience and preventing misuse.

**d) Maintainability**:

**SA-M1:** The software shall be designed with a modular architecture, organizing separate modules for core functions like AI processing, user management, content storage, and social interactions. This modularity will simplify maintenance by allowing updates and fixes to be applied independently to specific components.

**SA-M2:** All code shall follow clear and consistent coding standards, with comprehensive documentation provided for each module. This will make it easier to understand how different parts of the system work and allow them to get up to speed quickly. Consistent API contracts across modules will ensure smooth interactions between parts of the software, making future updates and maintenance simpler and more efficient.

**SA-M3:** Dependencies on external libraries and frameworks shall be documented with version control specifications, ensuring compatibility and ease of updates. This approach will help prevent dependency conflicts and ensure stability during maintenance.

**SA-M4:** The maintenance process shall include structured manual testing protocols for critical functionalities, focusing on user experience and system stability. Regular checks shall be performed after updates to verify core features, ensuring that performance remains consistent, and no regressions impact essential workflows. This approach will help maintain quality and reliability without relying on automated testing.

**e) Portability**

**SA-P1:** The software shall be compatible with the latest versions of iOS and Android, using the cross-platform framework React Native. This platform-agnostic approach will ensure that foodo can be easily deployed on both operating systems without extensive rework.

**SA-P2:** Host-dependent code shall be minimized, constituting no more than 20% of the overall codebase. This limitation will simplify the porting process to other potential mobile operating systems, reducing dependency on platform-specific components.

## Constraints

**C1: Regulatory Compliance:** foodo shall comply with relevant data protection regulations, including GDPR and KVKK, which mandate data privacy and control standards. These regulations constrain how user data is collected, processed, and stored, requiring consent management and secure data handling practices.

**C2: Platform-Specific Limitations:** The application shall follow the guidelines and limitations set by the iOS App Store and Google Play Store. This includes restrictions on acceptable content, data collection practices, and permissions required for functionality, for data access, notifications, and in-app purchases.

**C3: Network Dependency:** foodo’s core features, such as the AI cooking assistant and social network, require reliable internet access to function fully. Offline access may be limited, and the application is thus constrained by the availability and stability of network connectivity, impacting user experience in low or no connectivity environments.

**C4: Resource Availability:** The project relies on cloud infrastructure (e.g., AWS) and third-party AI models, meaning the application is dependent on the continued availability and performance of these resources. Any service limitations, cost increases, or policy changes in cloud or AI providers can impact foodo’s functionality, requiring potential adjustments in design or feature availability.

**C6: Development Team Availability and Schedule:** Given the part-time status of the development team members and overlapping academic and work commitments, time availability is constrained. This limits development capacity, potentially requiring prioritization of certain features or adjustments in timelines to meet project milestones.

**C7: Language and Localization:** The initial launch will be limited to English, as full multilingual support is out of scope due to time and resource constraints. This limits the immediate target audience to English-speaking users, though additional languages may be considered for future iterations.

**C8: Budget Constraints:** The project operates with an official budget of zero. However, to enhance the project beyond basic functionality, team members are using their own pocket money and salaries. This constraint means we rely heavily on free resources, student versions, and low-cost tools. Limited funding impacts our choices in development tools, cloud infrastructure, and third-party services, potentially restricting initial scalability and access to advanced features. Despite these limitations, we aim to make strategic, cost-effective decisions to ensure quality and functionality within our financial boundaries.

## Error Handling Requirements.

This section outlines how *foodo* shall respond to various error types to ensure reliability and user experience. The following error-handling mechanisms address both user-driven errors and system malfunctions.

**EH-U1: Input Validation Errors**

**Description**: If a user inputs data that does not match expected formats (e.g., special characters in names, invalid email formats, passwords that do not meet complexity requirements).

* **Response**: The system shall display a clear, contextual error message within 2 seconds, guiding the user to correct the input.
* **Mechanism**: The system shall validate input fields in real-time where applicable and provide immediate feedback. All invalid inputs shall be logged for analytics purposes.
* **Example**: "Invalid email address format."

**EH-U2: Authentication Errors**

**Description**: Errors encountered during user authentication, such as incorrect login credentials or non-existent accounts.

* **Response**: The system shall display a clear, contextual error message within 2 seconds, providing appropriate guidance (e.g., re-entering credentials or creating a new account).
* **Mechanism**: The system shall validate login credentials against the Firebase Authentication Service. Unsuccessful authentication attempts shall be logged, with a limit on the number of attempts allowed before a cooldown period.
* **Example**: "Invalid email or password. Please try again."

**EH-S1: System Errors**

**Description**: Errors related to server connectivity, database access, or unexpected internal issues that prevent normal operation.

* **Response**: The system shall display a non-technical, user-friendly error message and attempt to recover gracefully where possible (e.g., retrying connection). The user shall be informed of the issue within 2 seconds.
* **Mechanism**: System errors shall be logged with detailed information for further investigation. Alerts may be triggered for administrators in case of critical issues.
* **Example**: "Unable to connect to the server. Please check your internet connection and try again."

## Other Non-Functional Requirements

**Accessibility Requirements:**

**AR1:** Text content and UI elements shall maintain a minimum contrast ratio of 4.5:1 for normal text and 3:1 for large text to ensure readability for users with visual impairments.

# Logical Database Requirements

**Types of Information Used by Various Functions**

The mobile app utilizes different types of data in various functionalities to enhance the user experience. Here are examples of how data is used within the app's functionalities:

* **User Data**: Used during user registration, login, and profile management. For example:
  + When a user signs up, their name, email, and dietary preferences are stored.
  + User data is accessed during profile viewing and updates (e.g., changing cooking levels).
* **Post Data**: Accessed and manipulated when users create, edit, view, or delete posts. This data is used in the social feed functionality to display posts in reverse chronological order.
* **Comment Data**: Used for viewing, adding, editing, and deleting comments on posts.
* **Like Data**: Used to record and display user interactions on posts through likes. The app updates and tracks counts based on user interactions.
* **Friendship Data**: Facilitates social interactions, such as sending, accepting, or rejecting friend requests. This data is also used to manage and view friend lists.
* **Notification Data**: Triggered by interactions such as friend requests, likes, or comments. Notifications are displayed in the app's notification center to keep users informed of relevant interactions.
* **Achievement Data**: Used to track and display user milestones, such as completed recipes or shared posts. Achievements appear on user profiles and motivate engagement.
* **AI Chat Data**: Accessed whenever users interact with the AI assistant. It includes conversation logs and summaries, enabling contextual and personalized interactions.

**b) Frequency of Use**

* **User Data**: Accessed frequently during user login, profile updates, and other app interactions.
* **Post Data**: Accessed often as users create, view, edit, and delete posts; heavily used during social feed navigation.
* **AI Chat Data**: Accessed whenever users engage in interactions with the AI assistant.
* **Friendship Data**: Accessed when users send, accept, or reject friend requests and view friend lists.
* **Notification Data**: Accessed whenever a user receives, views, or clears notifications.
* **Achievement Data**: Accessed when users earn, view, or check progress on their achievements.

**c) Accessing Capabilities**

* **Read Access**: Required for retrieving user profiles, posts, comments, likes, chats, and notifications.
* **Write Access**: Required for creating posts, comments, and logging interactions with the AI assistant.
* **Update Access**: Needed for editing posts, updating user profiles, marking notifications as read, modifying friend connections, removing posts, comments, chat sessions, and clearing notifications.
* **Delete Access:** Needed to remove likes.

**d) Data Entities and Their Relationships**

**1. User Entity**

* **Fields**:
  + **UserID** (Primary Key)
  + **Name** (String, 2-64 characters)
  + **Email** (String, max 256 characters, unique, validated)
  + **ProfilePicture** (String, URL)
  + **CookingLevel** (Enum: Novice, Average, Expert)
  + **DietaryRestrictions** (Array of Strings)
  + **CreatedAt** (Datetime)
  + **UpdatedAt** (Datetime)
  + **DeletedAt** (Datetime)
* **Relationships**:
  + **One-to-Many** with **Post** (A user can create multiple posts)
  + **One-to-Many** with **Comment** (A user can make multiple comments)
  + **One-to-Many** with **Like** (A user can like multiple posts)
  + **One-to-Many** with **Notification** (A user can receive multiple notifications)
  + **One-to-Many** with **Achievement** (A user can earn multiple achievements)
  + **One-to-Many** with **AI Chat** (A user can have multiple chat sessions)
  + **Many-to-Many** with **User** via **Friendship** (Users can have multiple friends with statuses: pending, accepted, rejected)

**2. Post Entity**

* **Fields**:
  + **PostID** (Primary Key)
  + **UserID** (Foreign Key to User)
  + **Content** (Text, max 1907 characters)
  + **ImageURL** (String, optional)
  + **CreatedAt** (Datetime)
  + **UpdatedAt** (Datetime)
  + **DeletedAt** (Datetime)
* **Relationships**:
  + **Many-to-One** with **User** (Each post belongs to one user)
  + **One-to-Many** with **Comment** (A post can have multiple comments)
  + **One-to-Many** with **Like** (A post can have multiple likes)

**3. Comment Entity**

* **Fields**:
  + **CommentID** (Primary Key)
  + **PostID** (Foreign Key to Post)
  + **UserID** (Foreign Key to User)
  + **Content** (Text, max 256 characters)
  + **CreatedAt** (Datetime)
  + **UpdatedAt** (Datetime)
  + **DeletedAt** (Datetime)
* **Relationships**:
  + **Many-to-One** with **Post** (Each comment is associated with one post)
  + **Many-to-One** with **User** (Each comment is created by one user)

**4. Like Entity**

* **Fields**:
  + **LikeID** (Primary Key)
  + **PostID** (Foreign Key to Post)
  + **UserID** (Foreign Key to User)
  + **LikedAt** (Datetime)
* **Relationships**:
  + **Many-to-One** with **Post** (Each like is associated with one post)
  + **Many-to-One** with **User** (Each like is made by one user)

**5. Friendship Entity**

* **Fields**:
  + **FriendshipID** (Primary Key)
  + **UserID1** (Foreign Key to User)
  + **UserID2** (Foreign Key to User)
  + **Status** (Enum: pending, accepted, rejected)
  + **RequestDate** (Datetime)
  + **ResponseDate** (Datetime)
* **Relationships**:
  + **Many-to-Many** with **User** (Users can for m multiple friendships, with various statuses)

**6. Notification Entity**

* **Fields**:
  + **NotificationID** (Primary Key)
  + **UserID** (Foreign Key to User)
  + **NotificationType** (Enum: Friend Request, Like, Comment, etc.)
  + **Content** (String, max 500 characters)
  + **CreatedAt** (Datetime)
  + **ReadStatus** (Boolean)
  + **DeletedAt** (Datetime)
* **Relationships**:
  + **Many-to-One** with **User** (Each notification belongs to one user)

**7. Achievement Entity**

* **Fields**:
  + **AchievementID** (Primary Key)
  + **UserID** (Foreign Key to User)
  + **AchievementType** (String or Enum)
  + **EarnedDate** (Datetime)
  + **MilestoneDescription** (String)
* **Relationships**:
  + **Many-to-One** with **User** (Each achievement belongs to one user)

**8. AI Chat Entity**

* **Fields**:
  + **ChatID** (Primary Key)
  + **UserID** (Foreign Key to User)
  + **Timestamp** (Datetime)
  + **ChatContent** (Text or JSON for conversation logs)
  + **LastInteractionSummary** (Text)
  + CreatedAt (Datetime)
  + DeletedAt (Datetime)
* **Relationships**:
  + **Many-to-One** with **User** (Each AI chat session belongs to one user)

**e) Integrity Constraints**

* **User Email Constraint**: Unique constraint on user emails.
* **Post Content Constraint**: Maximum length constraint for post text content.
* **Friendship Constraints**: Ensure unique relationships between pairs of users (no duplicates).
* **Referential Integrity**: Foreign keys in child tables (e.g., Posts, Comments) must reference existing entries in parent tables (e.g., Users).
* **Password Constraints**: Must meet security criteria for complexity (hashed and salted).

**f) Data Retention Requirements**

* **User Data**: Retained indefinitely unless the user deletes their account.
* **Posts**: Retained until explicitly deleted by the user.
* **AI Chats**: Retained for a specified period (e.g., 30 days) for user access and review; older chats may be archived or deleted.
* **Friendship Data**: Retained as long as the friendship exists, or a request is pending.
* **Achievements**: Retained indefinitely to track user progress and milestones.

# Verification

Software verification ensures the *foodo* application meets all specified functional and non-functional requirements. Our verification strategy will include straightforward, effective methods to confirm that the core features work as expected. Below are the main approaches we will use:

**Manual Testing:** We will perform manual testing during active development to validate user interface elements, workflows, and interactions function as intended. This testing will follow a checklist based on our functional requirements, ensuring that critical features such as onboarding, AI interactions, and social features operate correctly. Specific scenarios, such as user registration, login flows, and AI-driven recipe interactions, will be prioritized to ensure core functions are reliable and user-friendly.

**Exploratory Testing:** We will conduct exploratory testing without predefined scripts to uncover unexpected issues and evaluate user experience. This allows us to identify any potential usability improvements or hidden problems. Exploratory testing will be iterative, with results informing future testing to improve coverage and overall app quality.

**Fault Attack Testing:** Using fault attack testing, we will apply error-guessing techniques to simulate common input errors and boundary cases, ensuring the application gracefully handles incorrect input. This type of testing helps make the app more robust against potential faults and ensures that unexpected user behaviors do not cause system failures.

**Checklist-Based Testing:** We will maintain a simple checklist to track the verification of specific functional and non-functional requirements. This approach will guide our testing process and ensure every key area is noticed. The checklist will be reviewed and updated periodically based on the outcomes of initial tests to address any evolving requirements.

**Compatibility Testing:** We will run basic compatibility tests for iOS and Android to verify that the app functions consistently across platforms. These tests will ensure core interactions remain reliable on different devices, allowing us to identify compatibility issues early and maintain user experience across diverse hardware configurations.

**API/Endpoint Testing:** Initially, API testing will be performed manually using Postman to validate the reliability of data exchanges and response consistency. This step ensures that backend endpoints respond correctly and support seamless user interaction. The testing will also cover edge cases, such as unexpected input and response time checks, to ensure stability in different conditions.

By using these practical verification methods, we aim to ensure that the foodo application meets its essential requirements and fits within the scope of our senior project. This structured approach helps maintain quality and boosts user confidence in the app’s reliability and performance.

# Discussions

## Limitations and Constraints

Since all of our team members are working part-time jobs It was hard for us to meet. The schedule of each team member was constraining our meeting times. But by looking at our priorities we managed to meet all of us in person multiple times for the initial plan. Also, the deadline for the first version of the SRS document coincided with the midterms. Thus this situation leaves us with a very limited time to complete this deliverable, unfortunately.

## Health and Safety Issues

During the preparation of the initial plan, we experienced health and safety

issues such as poor eating habits like excessive fast-food consumption, overuse

of caffeine, as well as neck, bone, and muscle pain due to prolonged screen time. Additionally, due to the colder weather and seasonal changes increased our risk of catching the flu. For example, our teammate Cemal caught the flu during the SRS deliverable process, requiring him to put in extra effort to maintain his performance Additionally, balancing our midterm studies with the limited time for completing the SRS deliverable led to some sleepless nights, which unfortunately impacted our overall health.

## Legal Issues

We did not encounter any legal issues.

## Economic Issues and Constraints

We had to buy a MS Office subscription to meet the document formatting rules.

A screenshot of a computer

Description automatically generated

Figure . MS Office subscription billing

Also, the face-to-face meetings required our team members to make extra expenses for transportation.

A hand holding a receipt

Description automatically generated

Figure . Alper Çelik’s gas receipt

During the preparation of the SRS document, our teammate Cemal had to buy a new phone after it broke unexpectedly. This added to his expenses, as he relies on his phone for communication and project coordination.

A cellphone with a person covering his eyes

Description automatically generated

Figure . Cemal Fırat Dağ’s new phone

## Sustainability

To promote sustainability, we used digital tools like Google Docs and Discord for note-taking and collaboration instead of paper. Additionally, we opted to walk from dorms to buildings such as A and C for meetings, using campus facilities rather than personal vehicles or other means of commute.[[4]](#footnote-4)

## Ethical Issues

During the preparation of the SRS document, our teammate Cemal used a computer provided by his part-time employer Jotform. Although this allowed him to work more efficiently, it raised ethical considerations regarding using employer-provided resources for academic purposes.



Figure . Cemal Fırat Dağ’s Macbook provided by Jotform

## Multidisciplinary Collaboration

We did not take any multidisciplinary collaboration while preparing the SRS document.

1. GenAI tool: ChatGPT 4

   Prompt: “Can you please shorten and formalise text without changing the context.”

   Rationale: To make sure that the executive summary is not more than one page. [↑](#footnote-ref-1)
2. GenAI tool: ChatGPT 3.5

   Prompt: “What might be foodo’s purpose in for social network?”

   Rationale: To give detailed content [↑](#footnote-ref-2)
3. GenAI tool ChatGPT 4o

   Prompt: “give me ideas about potential user characteristics for our mobile app” [↑](#footnote-ref-3)
4. GenAI tool: ChatGPT 4

   Prompt: “Can you please give sustainability part for our discussion.”

   Rationale: To get potential ideas. [↑](#footnote-ref-4)