****

**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 | - |
| Team11\_SRS\_FR1 | Microsoft 365 | 2 | 2 | 09.11.2024 |
| Team11\_SRS\_FR1.1 | Microsoft 365 | 2 | 2.1 | - |
| Team11\_SRS\_FR2 | Microsoft 365 | 2 | 3 | 04.12.2024 |

Change List

* Extend the Executive Summary with critical details such as functional requirements, system modal, requirements prototype.
* Fix formatting for the "List of Tables" section.
* Extend the Scope section to include back-end components.
* Correct the placement of AWS details in Software Interfaces.
* Add server-side constraints in the Constraints section.
* Update assumptions to reflect potential changes in mobile OS
* Expand Functional Requirements for more comprehensive coverage.
* Change the name of UC1 to "Login to System" in the use case table.
* Add a paragraph in System Model explaining why certain use cases were selected.
* Use "Earn Achievement" and "Get Recipe Instructions" as relevant use cases in System Model.
* Redo diagrams to align with analysis rather than design in system modal.
* Place diagrams side by side for better visualization.
* Make the suggestion feature in Requirements Prototype more complex.
* Use multiple screenshots for Requirements Prototype where needed.
* Remove immeasurable non-functional requirements or make them measurable.
* Specify measurable criteria for success, such as user feedback metrics.
* Justify all paragraphs for formatting.
* Create an ER Diagram for Logical Database Requirements.
* Specify the database technology to be used in Logical Database Requirements.
* Update Verification section to emphasize the manual testing is what we selected. Explain why automated testing was not included in the Verification section.

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 Clerk for user authentication, and OpenAI and Claude for natural language processing. Functional requirements are explained in detail including key functionalities such as, enabling users to interact with the AI assistant meal suggestions and providing step-by-step cooking instructions. Also, non-functional requirements such as usability, performance, reliability, and security are cautiously defined to support high standards of operation. The System Model and Requirements Prototypes sections provides a detailed analysis and visual representation of the selected use cases "Get Meal Suggestions" and "Earn Achievements”. We chose these two use cases because they exemplify the system's ability to handle complex workflows, integrate diverse technologies, and deliver personalized solutions. The selected two use cases are also showed in detail in the requirements prototype section with mock-up designs and scenarios along with visual analysis. 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.[[1]](#footnote-1)

Table of Contents

**Page Number**

[Executive Summary i](#_Toc184180832)

[Table of Contents ii](#_Toc184180833)

[1. Scope 1](#_Toc184180834)

[2. Product Perspective 4](#_Toc184180835)

[3. User Characteristics 15](#_Toc184180836)

[4. Assumptions and Dependencies 17](#_Toc184180837)

[5. Functional Requirements 19](#_Toc184180838)

[6. System Model 32](#_Toc184180839)

[7. Requirements Prototypes 38](#_Toc184180840)

[8. Non-functional Requirements 48](#_Toc184180841)

[1. Usability requirements 48](#_Toc184180842)

[2. Performance requirements 48](#_Toc184180843)

[3. Software system attributes 49](#_Toc184180844)

[4. Constraints 49](#_Toc184180845)

[5. Error Handling Requirements. 51](#_Toc184180846)

[9. Logical Database Requirements 53](#_Toc184180847)

[10. Verification 61](#_Toc184180848)

[11. Discussions 63](#_Toc184180849)

[1. Limitations and Constraints 63](#_Toc184180850)

[2. Health and Safety Issues 63](#_Toc184180851)

[3. Legal Issues 63](#_Toc184180852)

[4. Economic Issues and Constraints 63](#_Toc184180853)

[5. Sustainability 65](#_Toc184180854)

[6. Ethical Issues 65](#_Toc184180855)

[7. Multidisciplinary Collaboration 66](#_Toc184180856)

[12. References 67](#_Toc184180857)

List of Tables

**Page Number**

[Table 1. Cross Referencing Functional Requirements to Use Cases 30](#_Toc184178264)

List of Figures

**Page Number**

[Figure 1. UML Use Case Diagram 29](#_Toc184178542)

[Figure 2. Prototype Model of UC5 32](#_Toc184178543)

[Figure 3. Initial screen before suggestions 34](#_Toc184178544)

[Figure 4. User asks for Uhus Kebap 35](#_Toc184178545)

[Figure 5. Specific Preferences 36](#_Toc184178546)

[Figure 6. Earn Kebabster Achievement 37](#_Toc184178547)

[Figure 7. Kebabster Gold Achievement 39](#_Toc184178548)

[Figure 8. Achievement Progress 40](#_Toc184178549)

[Figure 9. ER Diagram of the Logical Database 53](#_Toc184178550)

[Figure 10. MS Office subscription billing 57](#_Toc184178551)

[Figure 11. Alper Çelik’s gas receipt 57](#_Toc184178552)

[Figure 12. Cemal Fırat Dağ’s new phone 58](#_Toc184178553)

[Figure 13. Cemal Fırat Dağ’s Macbook provided by Jotform 59](#_Toc184178554)

Abbreviations

|  |  |
| --- | --- |
| AI | Artificial Intelligence |
| AWS | Amazon Web Services |
| ECR | Elastic Container Registry |
| 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 |
| RDBMS | Relational Database Management System |
| 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.
* **Back-end Server:** A microservice architecture for managing social network, user data, and AI operations.

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 of 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 and above.
* **IOS:** Devices running IOS version 13 and above.

1. **Software Interfaces**
2. **Clerk**

* **Name:** Clerk
* **Mnemonic:** Not applicable
* **Specification Number:**Not applicable
* **Version Number:** Not applicable
* **Source:** <https://clerk.com/>

**Interface Description:**

* **Purpose:** Clerk acts as a user management and authentication service within the Foodo application. It streamlines secure user account creation, login, and session management while offering tools like passwordless authentication and multi-factor authentication for enhanced security. It also enables real-time synchronization of user data across devices.
* **Message Content and Format:** Interfacing is conducted through REST APIs and Clerk SDKs, utilizing JSON for data exchange.

1. **Nginx**

* **Name:** Nginx
* **Mnemonic:** Not applicable
* **Specification Number:** Not applicable
* **Version Number:** 1.26.2
* **Source:** <https://nginx.org/>

**Interface Description:**

* **Purpose:** Nginx is a high-performance web server, reverse proxy, and load balancer. It is commonly used to serve static content, handle HTTP/HTTPS traffic, distribute load among backend servers, and perform SSL/TLS termination. Nginx supports various protocols like HTTP, HTTPS, TCP, and UDP, and is known for its scalability, speed, and low resource usage. It can also be used for caching, serving dynamic content, and as a reverse proxy for applications running on other web servers or frameworks.
* **Message Content and Format:** Nginx handles HTTP(S) requests and responses in a text-based format, using standard web protocols. These requests and responses contain headers and body content, often in formats like HTML, JSON, XML, or other data types. The server efficiently processes incoming traffic and routes it based on its configuration, which can involve load balancing, proxying requests to backend servers, or serving static content directly

1. **Expo**

* **Name:** Expo
* **Mnemonic:** Not applicable
* **Specification Number:** Not applicable
* **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:** Not applicable
* **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:** Not applicable
* **Version Number:** Not applicable
* **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. **Docker**

* **Name:** Docker
* **Mnemonic:** Not applicable
* **Specification Number:** Not applicable
* **Version Number:** Not applicable
* **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:** Not applicable
* **Version Number:** Not applicable
* **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:** Not applicable
* **Version Number:** Not applicable
* **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. **PostgreSQL**

* **Name:** PostgreSQL
* **Mnemonic:** Not applicable
* **Specification Number:** Not applicable
* **Version Number:** 16.6
* **Source:** <https://www.postgresql.org/>

**Interface Description:**

* **Purpose:** PostgreSQL is an advanced relational database management system (RDBMS) that provides a robust, open-source solution for managing and querying structured data. It serves as the core data store in our system, offering powerful SQL querying capabilities, support for complex data types, and high reliability. PostgreSQL interfaces with backend services to store, retrieve, and manipulate data, ensuring data consistency, integrity, and scalability across applications.
* **Message Content and Format:** Data is exchanged in the form of SQL queries and responses, typically using the PostgreSQL wire protocol. This includes SELECT, INSERT, UPDATE, DELETE commands, and more complex queries with joins, transactions, and stored procedures. Data formats include standard SQL syntax for structured tables, JSONB, and other PostgreSQL-supported formats for flexible data storage and retrieval.

1. **AWS Lambda**

* **Name:** AWS Lambda
* **Mnemonic:** Not applicable
* **Specification Number:** Not applicable
* **Version Number:** Not applicable
* **Source:** <https://aws.amazon.com/lambda>

**Interface Description:**

* **Purpose:** AWS Lambda enables foodo to execute backend code in response to events without the need to manage servers. It will be used for serverless functions such as processing AI requests, managing chat handlers, generating recipes 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. **Amazon Elastic Container Registry**

* **Name:** Amazon Elastic Container Registry (ECR)
* **Mnemonic:** ECR
* **Specification Number:** Not applicable
* **Version Number:** Not applicable
* **Source:** <https://aws.amazon.com/ecr>

**Interface Description:**

* **Purpose:** Amazon ECR provides a fully managed container image registry, enabling secure storage, sharing, and deployment of container images. It serves as the interface for managing containerized applications.
* **Message Content and Format:** Handles the push, pull, and management of container images using standard Docker CLI commands and AWS SDKs. The API supports secure data exchange through HTTPS and authentication via AWS Identity and Access Management (IAM). Configuration and management are performed through the AWS Management Console, AWS CLI, or SDKs.

1. **Google Vertex AI**

* **Name:** Google Vertex AI
* **Mnemonic:** not applicable
* **Specification Number:** Not applicable
* **Version Number:** Not applicable
* **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 back-end server powering the foodo application is designed to support a microservice architecture, ensuring scalability, reliability, and efficient resource management. To maintain optimal performance and responsiveness, the following memory constraints have been established based on the typical memory consumption patterns of modern server-side applications:

**Estimated Memory Consumption**

* **Active Usage:** During peak load scenarios, such as handling concurrent user requests, running AI algorithms for recipe recommendations, and managing real-time data synchronization, each microservice should aim to consume approximately **400-1024 MB** of memory.
* **Idle/Low-Traffic Usage:** When operating under minimal load or during non-peak periods, microservices should ideally consume no more than **100-300 MB** of memory. Services should leverage idle state optimizations, caching, and resource pooling to reduce memory footprint while maintaining readiness to handle incoming requests efficiently.

The foodo mobile 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, 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. It is assumed that during the development of foodo, new versions or updates of iOS and Android will not introduce changes that disrupt the core functionalities and features specified in this SRS document. No immediate updates to the application are planned to address potential OS changes during this period. The core features of the app, as outlined in this document (see the Functional Requirement section), are expected to remain unaffected by OS updates or deprecations.
* **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 API, such as AWS S3, 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 Clerk 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 Clerk Authentication to send a password reset link. If Clerk 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 Clerk 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 the 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 record this information to be used in personalization of recipes to be given to the user and 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 (**FReq3.1.2.1**). 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**).

**FReq3.1.2.1**: When the **FReq3.1.1** is completed, the system shall get the user input and check if the user’s request is valid or not by prompting it to the LLM Provider. If the user request is not a valid request or wants something that will abuse the system, the user shall be warned with an error message (EH-AI1). Also, a function call check should be done within the same LLM request to check whether the user wants suggestions or detailed recipes. If the user wants detailed instructions the system shall continue with instructions (FReq3.1.2.3) else should continue with the suggestion process **(FReq3.1.2.2)**.

**FReq3.1.2.2**: When the **FReq3.1.2.1** is completed with a valid user request and suggestion request, the system shall vectorize the user request and search for the request in the Vector Database. The system shall filter results based on similarity scores which are greater than 0.5. If no results are present the system shall prompt the user to specify more details. If there are more than one results the system shall get details from results like name, ingredients, cooking time, the system shall get user preferences like dietary restrictions, cooking level and favorite meals from database, and embed all these inside the LLM prompt’s meal context part and make an LLM call to LLM Provider. The system shall show the response back to the user and also make meal names highlighted and clickable. After clicking on these linked meal names, the mobile app shall send a predefined message as “I want to prepare X” (X as the meal name) and reinitialize.

**FReq3.1.2.3**: When the **FReq3.1.2.1** is completed with a valid user request and detailed recipe instructions request, the system shall l make an LLM call to check user input whether it has a meal name or not. If no meal name is present, the system shall warn users (EH-AI1) about missing meal names. If the meal name is present, then the system shall vectorize the meal name and query Vector Database. The system shall filter results based on similarity scores which are higher than 0.8. If no result is present the system shall warn users about no similar recipe found and prompt users to give more details about the meal. If there are results, then the recipe instructions shall be sent to the LLM provider for a more structured output to use in instruction cards (**FReq3.1.3**).

**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.

**FReq3.2.1:** Upon completion of a recipe guided by the AI assistant, the mobile app shall show the user "Share Your Dish" button 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 button to "Delete Chat" (**FReq3.3.2**). When a user selects a chat entry, the system shall redirect the user to the AI chat page **(FReq3.1.1)**.

**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" icon button 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.
* Count of friends which is clickable if the user is viewing his/her own profile page and should redirect the user to the friends list page **(FReq4.2.3)**.
* List of recent posts (**FReq4.4.1**).
* 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 allows users to upload or change their profile picture. Users can either select an image from their device gallery or capture a new one using the device camera (if supported). The system shall validate the image file format such as JPEG, PNG, and file size (maximum 5 MB), rejecting unsupported or excessively large files bigger than 5 MB. A preview of the selected image shall be displayed before saving.
* "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 **(FReq5.1)**.

**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 with accepting minimum of 2 and maximum of 64 characters within the "Social Feed", allowing users to search for other users by name. As the user types in the search field, a drop-down list shall display matching profiles from the foodo. Each result shall show the profile photo and name side by side. 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
* Image upload field accepting only image files such as JPEG, PNG etc. and not bigger than 10MB in size.
* "Post" button that remains disabled until at least caption or recipe field is filled, and image is uploaded.

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 caption and recipe (if not immutable recipe from AI) 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.

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

**FReq4.6.1:** Users shall be able to add comments to posts by using the comment icon button. The system shall display a comment input field, when pressed on the comment icon button, 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 clicking the "Edit" icon button on the comment 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 clicking the "Delete" icon button 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, earned achievements, 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 as a notification item in a list with a “Delete” icon button **(FReq6.3)**.

**FReq6.3:** Users shall have the option to delete individual notifications by clicking a "Delete" icon on the right side of the notification item.

**FReq6.4:** In the upper right corner of the notifications page there shall be a trash can icon for deleting all notifications, and when the user clicks on this button, the system shall give the user a confirmation dialog asking if they are sure they want to delete all notifications in the notification box. If the user chooses to delete all notifications, all the notifications stored for the user shall be deleted. If the user decides not to delete the notifications, there shall be no change in the notifications.

**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 1. UML Use Case Diagram

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

Table 1. Cross Referencing Functional Requirements to Use Cases

|  |  |  |
| --- | --- | --- |
| **Functional Requirement ID** | **Use Case ID** | **Initial Plan Functional Requirement ID** |
| 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, FReq3.1.2.1, FReq3.1.2.2, FReq3.1.2.3 | 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

The selected use cases, **Get Meal Suggestions (UC5)** and **Earn Achievements (UC12)**, demonstrate complexity and effective integration with multiple services. In Get Meal Suggestions, a vector database retrieves contextual information based on user input, while LLM-driven input validation and verification ensure accuracy, and user preferences guide personalized recommendations. In Earn Achievements, message brokers track user actions to dynamically generate achievement badges, fostering engagement and recognizing milestones. These use cases highlight the capability to manage intricate workflows, coordinate diverse technologies, and deliver impactful solutions.

1. **UC 5 – Get Meal Suggestions**

**A diagram of a flowchart

Description automatically generatedA black and white drawing of a building

Description automatically generatedA diagram of a company

Description automatically generated**

1. **UC 12 – Earn Achievement**

**A diagram of a computer

Description automatically generatedA diagram of a diagram

Description automatically generated with medium confidenceA diagram of a flowchart

Description automatically generated**

# Requirements Prototypes

The section provides a detailed visualization of the system's core functionalities, focusing on the selected use cases**: Get Meal Suggestions (UC5)** and **Earn Achievements (UC12)**. These prototypes illustrate the interaction between users and the system, showcasing the seamless integration of advanced technologies and user-centric design principles. These prototypes emphasize the system's ability to handle complex workflows and deliver tailored solutions, ensuring an intuitive and engaging user experience.[[4]](#footnote-4)

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

**A screenshot of a chat

Description automatically generated**

Figure 2. Prototype Model of UC5

This use case demonstrates how the system enables users to interact with the AI assistant for personalized meal suggestions. The interaction involves handling user input, generating recommendations, and responding to further requests or clarifications. To ensure realistic behavior, alternative scenarios are also modeled to reflect real-world complexities.

* 1. **User Request for Meal Suggestions:**

The user initiates a request for meal suggestions by providing a preference (e.g., "I'm in the mood for pasta. Any suggestions?").

* 1. **AI Suggestion and User Engagement:**

The AI assistant responds with a personalized suggestion, such as:

"I have a great recipe for creamy mushroom pasta. Would you like to try it?"

Below the AI's response, a "Let's Cook This!" button is displayed, enabling users to proceed with detailed cooking instructions for the suggested meal.

* 1. **Transition to Cooking Instructions:**

By clicking the "Let's Cook This!" button, users are guided to step-by-step cooking instructions, aligning with FReq3.1.3. The entire conversation flow reflects FReq3.1.1 and FReq3.1.2, showcasing the AI's ability to provide actionable meal suggestions.

This prototype highlights the app's seamless transition from meal suggestion to guided cooking, leveraging third-party LLM services to deliver contextual and engaging responses. It effectively demonstrates the application's core functionality of enhancing user engagement and providing value through personalized assistance.

**A person cooking in a kitchen

Description automatically generated**

Figure 3. Initial screen before suggestions

**Detailed Scenarios**

* 1. **User Interaction: Initial Request**

**Explanation:**

In this scenario, the chatbot begins the interaction by welcoming the user and prompting them for their preferences. When the user requests a specific meal that the assistant cannot match ("Uhus Kebab"), the system handles the limitation by offering a relevant alternative suggestion ("Adana Kebap"). This ensures the conversation continues smoothly without leaving the user without options.

* **Cooking Assistant:** “Hello! What would you like to cook today?"
* **User:** "I want to cook Uhus Kebab. Do you have any suggestions?"
* **Cooking Assistant:** Unable to find an exact match, the assistant suggests an alternative: "Sorry, I cannot help you with that. Would you like to try Adana Kebap?"

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Figure 4. User asks for Uhus Kebap

* 1. **Clarification Request for Specific Preferences**

**Explanation:**

This scenario demonstrates the AI's ability to refine suggestions based on user preferences. The assistant first narrows down meal options by asking if the user prefers a creamy or tomato-based recipe, tailoring the recommendations accordingly. When the user selects "creamy," the assistant provides a list of specific recipes and offers clarification upon request. This detailed exchange showcases the AI's flexibility and user-centric design, making the interaction feel personalized and responsive.

**Chatbot:** "Here are some delicious pasta recipes. Would you like them creamy or with a tomato base?"

The user prefers creamy options:

**User:** "I would prefer creamy."

The assistant lists relevant choices:

**Chatbot:** "Great choice! Here are a few creamy pasta recipes: Vegan Alfredo, Creamy Mushroom Pasta, and Creamy Spinach Penne. Do you need clarification on any of these?"

The user requests more details about "Vegan Alfredo."

**Chatbot:** "The Vegan Alfredo is made with cashew cream, garlic, and nutritional yeast for a rich flavor. Would you like to try this recipe?"

**User:** The user confirms, "Yes, this sounds perfect!"

**Chatbot:** The chatbot replies, "Awesome! Are you ready to start cooking?"

A screenshot of a chat

Description automatically generated

Figure 5. Specific Preferences

**Prototype Use Case 2: UC12 - Earn Achievements**

Earning achievements is integral to creating a fun and engaging user experience. In UC12, the system tracks user progress in specific cooking activities and awards badges when certain milestones are achieved. This feature not only motivates users to continue using the application but also enhances their sense of accomplishment. It leverages real-time feedback and visual indicators to celebrate success.

**Detailed Scenarios**

* 1. **Unlocking the "Kebabster" Badge**

**Initial Trigger:** The system monitors user activity and identifies when the user has cooked kebab 10 times.

**Notification**: A congratulatory message appears: "Congratulations! You've earned the 'Kebabster' badge for cooking kebab 10 times. Keep up the great work!"

A screenshot of a cellphone

Description automatically generated

Figure 6. Earn Kebabster Achievement

**Design Elements:**

* The notification features the badge with a vibrant design labeled "Level 1".
* A "Close" button allows users to dismiss the notification.

This scenario illustrates the gamification feature where the system immediately acknowledges milestones. It helps establish user satisfaction and reinforces engagement.

* 1. **Upgrading to "Kebabster Gold"**

This scenario focuses on progression within the achievement system, encouraging users to continue engaging with the application. The visual upgrade and personalized feedback highlight the user's progress and reward their efforts.

**Initial Trigger:**

Upon cooking kebab 20 times, the user qualifies for a badge upgrade to the "Kebabster Gold" level.

**Notification:**

A more elaborate message is displayed: "Congratulations! You've earned the Kebabster Gold badge by cooking kebab 20 times! Welcome to Level 2, Kebabster!"

**Design Elements:**

* The upgraded badge design reflects the higher level and includes a gold motif.
* The congratulatory message is celebratory, motivating users to aim for further milestones.
* A "Close" button remains consistent for usability.

**metin, çizgi film, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Figure 7. Kebabster Gold Achievement

* 1. **Progress Update Towards "Kebabster Gold"**

This scenario demonstrates the system's ability to provide real-time progress updates, keeping the user motivated by showing tangible steps toward the next achievement. It encourages consistent engagement by celebrating incremental progress and creating a sense of anticipation for the upcoming reward. By keeping users informed about their achievements, this scenario exemplifies the gamification aspect of the application, creating a rewarding and engaging user experience.

**Initial Trigger:**

The system tracks the user's progress in cooking kebabs and identifies that they have cooked 15 kebabs, nearing the "Kebabster Gold" achievement.

**Notification:**

A progress update is presented: "Congratulations! You've cooked 15 kebabs. Just 5 more to reach Kebabster Gold Level 2!"

**metin, ekran görüntüsü, iş kartı, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu**

Figure 8. Achievement Progress

**Design Elements:**

* A dynamic visual progress indicator displays the user's current milestone (15/20 kebabs), clearly showing how many more are needed to unlock the next level.
* The notification prominently features the "Kebabster Gold" badge and highlights the user's progress toward achieving Level 2.

# Non-functional Requirements

## Usability requirements

**UR1:** Notifications, including those for timed cooking steps, alerts, and achievements, must be displayed within **3 seconds** of the **%80** of the events to ensure clarity and ease of navigation for the user.

**UR2:** Essential tasks, such as interacting with the AI assistant and accessing social features, should be accessible within **five taps** from the home screen.

**UR3:** During usability testing with **10 users**, **at least 75%** shall rate the ease of navigation as "**easy**" or higher on a 5-point Likert scale.

**UR4:** Animations and transitions (e.g., starting an AI conversation, navigating to the social feed) shall be completed in no more than **3.5 seconds** for **75%** of occurrences, verified through performance testing on standard devices.

## Performance requirements

**PR1:**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 **25** requests, maintaining a high level of responsiveness.

**PR2:** In low-connectivity scenarios (e.g., 3G networks), response times for core interactions shall not exceed **20 seconds** for **80%** of cases, tested using network simulation tools.

**PR3:** 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.

**PR4:**User-generated content, such as images in the social feed, shall load within **5 seconds** for **90%** of **20 requests**, ensuring smooth content interaction.

**PR5:** Push notifications (cooking reminders, social alerts) shall be delivered within **10 seconds** of the triggered event for **95%** of cases, verified through real-world testing with **10 devices**.

**PR6:** Image uploads up to **5 MB** shall complete within **20 seconds**, and images shall display in the feed within **20 seconds**, tested over **30 uploads** under average network conditions.

## Software system attributes

**a) Reliability**:

**SA-R1:** The system shall maintain **95%** uptime for essential functions (user authentication, AI assistance) over a **3-day** period, monitored using uptime tracking tools.

**b) Availability:**

**SA-A1:** The system shall maintain an availability level of **95.55%** in **2 days**, minimizing the risk of downtime.

**c) Security:**

**SA-S1:** The application shall implement a valid SSL/TLS certificate for HTTPS connection for **%100** of the endpoints that transmit data between the client and server.

**SA-S2:** In security testing, the system shall withstand common attacks (e.g., SQL injection, XSS) with **zero successful breaches** during penetration testing sessions.

**d) Maintainability:**

**SA-M1:** The codebase shall be organized into modules (AI processing, user management, etc.), with each module documented. Code reviews shall confirm adherence to modular design.

**e) Portability**

**SA-P1:** The app shall function on devices running **iOS 13+** and **Android 12+**, verified through testing on **at least 3 devices** for each platform.

## 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 Clerk 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."

**EH-AI1: AI Interaction Errors**

**Description**: Errors encountered during interactions with the AI system, including invalid or unclear requests, unsupported inputs, or failures in data processing (e.g., missing critical details like meal names or unrecognized commands).

* **Response**: The system shall display a user-friendly error message within 2 seconds, clearly explaining the issue and providing actionable guidance for correction or refinement of the input.
* **Mechanism**: The system shall validate user inputs through the LLM Provider and internal function checks. Invalid or incomplete requests shall trigger an error response and be logged for analytics and monitoring. In case of unrecognized inputs, the system shall suggest possible corrections or clarifications.
* **Example**: "Your input could not be processed. Please provide more details or clarify your request."

# Logical Database Requirements

**Relational Database Management System:**

PostgreSQL version 16.6 will be used as RDBMS. It will have only one user named “foodo” with root privileges to manage everything on the RDBMS. RDBMS will not be accessed from the public, can only be accessed by services in the same network.

1. **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.

1. **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.

1. **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.

1. **Data Entities and Their Relationships**

**1. User Entity**

* **Fields**:
  + **ID** (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)
  + **FavoriteMeals** (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**:
  + **ID** (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**:
  + **ID** (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**:
  + **PostID** (Primary Key and Foreign Key to Post)
  + **UserID** (Primary Key and 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**:
  + **UserID1** (Primary Key and Foreign Key to User)
  + **UserID2** (Primary Key and Foreign Key to User)
  + **Status** (Enum: pending, accepted, rejected)
  + **RequestDate** (Datetime)
  + **ResponseDate** (Datetime)
* **Relationships**:
  + **Many-to-Many** with **User** (Users can form multiple friendships, with various statuses)

**6. Notification Entity**

* **Fields**:
  + **ID** (Primary Key)
  + **UserID** (Foreign Key to User)
  + **Type** (Enum: Friend Request, Like, Comment, etc.)
  + **Title** (String, max 100 characters)
  + **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)
  + **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).

**Soft Deletion:** When a record is deleted (e.g., user account or post), it is marked as deleted (e.g., via DeletedAt field of the table) rather than being permanently removed from the database. This ensures the data remains available for historical reference while excluding it from active queries. It is applied to every entity with a DeletedAt field.

**Hard Deletion:** In cases where a DeletedAt field is not present, the record is permanently deleted from the database. For any related entities that reference the deleted record, the foreign key references are set to NULL to maintain referential integrity and prevent orphaned records. This ensures that dependent data is not left in an inconsistent state, and the deletion does not violate foreign key constraints.

**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.

**A diagram of a computer

Description automatically generated**

Figure 9. ER Diagram of the Logical Database

# Verification

Our verification strategy for the foodo application prioritizes manual testing methodologies, as they offer a thorough and hands-on system evaluation. Given our time constraints and the need to focus on development, we decided not to include automated testing tools or test suites. This decision reflects our belief that the manual testing approaches described below will provide sufficient coverage to ensure the application's reliability and user experience.

To clarify the testing process and avoid confusion, we have grouped all the manual testing methods into distinct categories. This structure ensures clarity and emphasizes our comprehensive approach to verifying the application's functionality and quality. The following are the testing methodologies we will employ:

**Exploratory Testing:** We will conduct exploratory testing without predefined scripts to uncover unexpected issues and evaluate the user experience. This approach allows testers to mimic real-world user behavior, identify hidden problems, and suggest usability improvements.

**Fault Attack Testing:** We will test how well the system handles errors by intentionally introducing incorrect inputs and edge cases. This ensures the application remains robust even when users provide unexpected or invalid data.

**Checklist-Based Testing:** We will maintain a dynamic checklist to track the verification of specific functional and non-functional requirements. This method ensures comprehensive coverage while remaining organized and systematic.

**Compatibility Testing:** Testing will focus on the application's behavior across different platforms (iOS and Android) to ensure consistent performance and user experience.

**API/Endpoint Testing:** We will test API endpoints using manual tools like Postman to validate data consistency and response accuracy. This ensures that the back-end services work seamlessly with the front-end application.

**User Interface Testing:** We will manually test all UI elements to ensure they function correctly and provide a positive user experience. This includes verifying that buttons, forms, and navigation elements work as intended.

This approach lets us focus on the development process while ensuring that the foodo application meets all functional and non-functional requirements. By structuring the manual testing methods into clear categories and outlining their scope, we provide a comprehensive and adequate verification process tailored to the project's needs.

# 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 delivery, 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 10. 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 11. 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 12. Cemal Fırat Dağ’s new phone

## Sustainability

To promote sustainability, we used digital tools like Google Docs and Discord for notetaking 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.[[5]](#footnote-5)

## 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 13. Cemal Fırat Dağ’s Macbook provided by Jotform

## Multidisciplinary Collaboration

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

# References

**There are no sources in the current 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: “Based on the selected use cases, make an introduction to the requirements prototype section.”

   Rationale: To make an introduction [↑](#footnote-ref-4)
5. GenAI tool: ChatGPT 4

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

   Rationale: To get potential ideas. [↑](#footnote-ref-5)