VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Specialized Project (CO4029)

RESEARCH AND DEVELOP SMART PERSONAL ASSISTANT FOR SCHEDULE OPTIMIZATION, HEALTH REMINDERS, AND VOICE-BASED NOTE MANAGEMENT

Major: Computer Science

Thesis Committee:	Software Engineering
Supervisor(s):	Assoc. Prof. Quan Thanh Tho, Ph.D
Student 1:	Dinh Ba Khanh (2252323)
Student 2:	Nguyen Anh Khoa (2252351)
Student 3:	Tran Chi Tai (2252727)

Ho Chi Minh City, December 2025

Instruction's Signature

_____ **Date:** _____

Assoc. Prof. Quan Thanh Tho, Ph.D (Project Instructor)
Associate Professor
Faculty of Computer Science and Engineering

Acknowledgement

First and foremost, we would like to express our deepest gratitude to Assoc. Prof. Quản Thành Thơ for his invaluable guidance. His profound knowledge and dedicated support have been instrumental to the success of this specialized project.

Furthermore, we extend our sincere appreciation to the lecturers of the Faculty of Computer Science and Engineering, Vietnam National University - Ho Chi Minh City University of Technology. The fundamental knowledge and solid foundation provided by the faculty have been essential in enabling us to execute and complete this project effectively.

Finally, we would like to thank our families and friends for their unwavering support, encouragement, and for always being by our side throughout this journey.

Ho Chi Minh City, December 2025
The authors,
Đinh Bá Khánh
Nguyễn Anh Khoa
Trần Chí Tài

Declaration Of Authenticity

My team consists of Đinh Bá Khánh, Nguyễn Anh Khoa, and Trần Chí Tài. We declare that we solely conducted this specialized project under the supervision of Assoc. Prof. Quản Thành Thơ at the Faculty of Computer Science and Engineering, Vietnam National University, Ho Chi Minh City University of Technology.

Throughout the project, our team ensures that all references and external sources utilized are appropriately cited and thoroughly documented.

Ho Chi Minh City, December 2025
The authors,
Đinh Bá Khánh
Nguyễn Anh Khoa
Trần Chí Tài

Abstract

With the rapid development of Artificial Intelligence (AI), the integration of intelligent systems into personal productivity has become increasingly vital. Effective time management is a critical skill, yet traditional methods of manually organizing schedules can be time-consuming and inefficient. To address this challenge, this project presents the design and implementation of a **Smart Schedule AI**, a mobile application designed to optimize personal scheduling through voice interaction.

The proposed system utilizes a suite of modern technologies designed for scalability and performance. For the server-side architecture, we aim to leverage the Python ecosystem, specifically combining Django for core management and FastAPI for high-performance services. regarding the client-side implementation, Flutter has been identified as the primary candidate due to its robust multiplatform capabilities, which allow for unified deployment across different devices. Finally, MongoDB has been selected as the database management system; its semi-structured document model is well-suited to facilitate the system's future scalability.

Key features of the system include a Speech-to-Schedule module, which leverages Natural Language Processing (NLP) to transform voice commands into actionable tasks or schedule entries. The processing pipeline is designed to utilize OpenAI Whisper for robust speech-to-text transcription. Following transcription, the system integrates the Gemini Large Language Model alongside the Duckling library to perform semantic analysis and entity extraction, ensuring precise identification of temporal details and user intent.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In chapter 1, the motivation, goal, and scope of the application are illustrated. The outline of the report is also presented.

1.1 Motivation

Modern life is characterized by rapid development and increasing complexity, making effective personal organization an essential requirement. As individuals face a growing number of distractions, minimalism has emerged as a practical approach to improving productivity and personal well-being. The principles of minimalism—clarity, focus, and the reduction of unnecessary elements—align with the demand for efficient tools that support self-management.

Despite the availability of numerous productivity applications, personal organization remains fragmented. Users commonly rely on separate tools for task management, scheduling, and health monitoring, such as tracking sleep or daily water intake. This fragmentation increases cognitive load, interrupts workflow continuity, and reduces overall efficiency in daily activity management.

This project is motivated by these limitations and aims to design an integrated personal management application that consolidates multiple aspects of daily organization into a single system. The proposed solution supports task management, scheduling, and basic health tracking within a unified platform. To improve accessibility and interaction efficiency, the system incorporates artificial intelligence–based natural language processing (NLP), enabling users to issue voice commands that are automatically converted into structured tasks and organized schedules.

In addition, the application emphasizes a streamlined UI/UX design optimized for mobile interaction. By minimizing operational complexity and reducing the time required to manage daily activities, the proposed system seeks to provide an efficient and coherent solution for personal organization.

1.2 Goal

Based on the identified limitations in existing personal management tools, this project defines the following goals.

The primary goal of this project is to develop an integrated application that supports productivity-oriented users in managing daily activities while maintaining a balanced lifestyle. Instead of relying on multiple standalone tools, the proposed system provides a unified platform for organizing work tasks, managing schedules, and monitoring essential aspects of personal well-being. This approach promotes simplicity and coherence in self-management and aligns with widely recognized principles of effective time and life balance.

A secondary goal of the project is to enhance user interaction efficiency through the integration of artificial intelligence–based natural language processing (NLP). The system enables users to issue voice commands that express objectives, tasks, habits, or scheduled events, which are then automatically transformed into structured tasks and calendar entries. This interaction model reduces manual input, improves accessibility, and supports faster task management.

Furthermore, the system incorporates intelligent notification mechanisms to assist users in managing multiple responsibilities. By including basic health-tracking functions, such as monitoring sleep and daily water intake, the application also promotes awareness of personal well-being, which is often overlooked in work-intensive environments.

1.3 Scope

To achieve the stated goals, the scope of this project is defined as follows.
The scope of this project is limited to the design and development of a cross-platform mobile application for Android and iOS platforms. The application focuses on three core functional modules: schedule management, task management, and basic health management, including daily water intake and sleep quality tracking. Additional functionalities within scope include notification support and limited offline access, enabling users to view existing tasks and schedules without an active internet connection.

The project includes the integration of artificial intelligence–based natural language processing (NLP) to support voice-based interaction. Through this feature, users can create and manage tasks, schedules, and routines using spoken input.

User interface and user experience (UI/UX) design are also within the project scope. The system workflow is refined based on analysis of existing mobile applications in the same domain, with an emphasis on simplifying task and schedule management processes and reducing interaction complexity in daily use.

1.4 Thesis Structure

This thesis is organized into four main chapters, each addressing a specific stage of the system design, development, and evaluation process.

- **Chapter 1 – Introduction**
  This chapter introduces the background and motivation of the project. It outlines the problem statement, research objectives, and scope of the work. In addition, it provides an overview of the thesis structure to guide readers through the subsequent chapters.

- **Chapter 2 – Technology Stacks**
  This chapter presents the technology stack selected for system development. It covers mobile development technologies, server-side frameworks, database management systems, and machine learning tools. Each technology is analyzed and compared with alternatives to justify its suitability for the proposed system.

- **Chapter 3 – System Design and Analysis**
  Chapter 3 focuses on the overall system design and analysis. It reviews related works and existing applications, followed by requirement analysis including functional and non-functional requirements. The chapter then presents use-case design, architectural strategy, logical and physical system views, and database design. In addition, it describes the speech-to-schedule processing pipeline, including automatic speech recognition, entity extraction, intent classification, temporal normalization, schedule creation, and error handling. Wire-frame designs are also introduced to illustrate the user interface structure.

- **Chapter 4 – Work Evaluation and Future Improvement**
  The final chapter evaluates the implemented system based on functionality, usability, and performance. It discusses achieved results, identifies system limitations, and proposes potential improvements and future development directions.

Chapter 2

Technology Stacks

In Chapter 2, the authors present insights into the technologies used in developing the mobile application. These include mobile application development, server-side development, database management systems, and machine learning technologies.

2.1 Mobile Development

2.1.1 Flutter

Flutter, first introduced by Google in 2017, has quickly evolved into one of the most influential frameworks for building cross-platform applications. Its modern architecture and high-performance rendering engine have reshaped how developers create mobile, web, and desktop interfaces.

Flutter applications are constructed using widgets, the fundamental building blocks that define both structure and behavior. Widgets describe their visual configuration and update themselves when their internal state changes. The framework efficiently computes the minimal updates needed to transition between UI states through a diff-based rendering process. This widget-driven approach enables consistent, reactive, and expressive user interfaces across multiple platforms. [1]

2.1.2 React Native

React Native, introduced by Meta in 2015, has become a widely adopted framework for developing cross-platform mobile applications using JavaScript and React principles. Its core idea is to allow developers to write application logic once while rendering platform-specific native UI components, resulting in applications that closely match the performance and behavior of purely native apps.

React Native uses a declarative component model, where UI elements are defined as components that respond to changes in state. Communication between JavaScript logic and native platform APIs is facilitated through a bridge mechanism, enabling access to device features and native modules. This architecture makes React Native well-suited for rapid development cycles and teams already familiar with React and web technologies. [2]

2.1.3 Kotlin Multiplatform

Kotlin Multiplatform, developed by JetBrains, enables the sharing of application logic across multiple platforms while preserving native UI and platform-specific capabilities. Rather than providing a unified rendering system, Kotlin Multiplatform focuses on code reuse at the business-logic level, such as networking, data processing, and state management. This design allows developers to maintain native performance and native interface components on Android, iOS, desktop, and web targets.

A Kotlin Multiplatform project typically separates common code from platform-specific code, with shared modules compiled to different targets such as JVM, JavaScript, or native binaries. Platform-specific modules integrate with the shared logic while implementing UI elements and device APIs. This approach provides flexibility, reduces duplication, and is especially beneficial for large-scale or long-term projects that require consistency across multiple platforms. [3]

2.1.4 Comparison and Mobile Development Technology Selection for the Project

Framework	Pros	Cons
Flutter	**Consistent UI Across All Platforms** Uses Skia engine for uniform rendering across Android, iOS, web, and desktop. **Customizable Widget System** Large widget library supporting expressive, custom UIs with minimal boilerplate.	**Requires Learning** Requires mastering Dart and a widget-centric architecture/mental model. **Self-creating UI** Manual implementation required for platform-specific navigation and accessibility.
React Native	**JavaScript + React Ecosystem** Leverages existing JS tools and knowledge for reduced onboarding time. **Fast Iteration + Modular Architecture** Supports rapid UI testing via fast refresh and flexible native module integration.	**Performance Bottlenecks** JavaScript-Native bridge may introduce latency in complex, real-time updates. **Requires Native Modules for Advanced Features** Advanced tasks often require platform-specific expertise (Kotlin, Swift).
Kotlin Multiplatform	**Shared Business Logic with Fully Native UI** Reduces code duplication while maintaining platform-specific user experiences. **Performance and Platform Integration** Compiles to native binaries with direct access to all platform-native APIs.	**Requires Kotlin and Swift** UIs must be written separately for each platform using native frameworks. **Limited Learning Resources** Smaller ecosystem with fewer ready-made libraries compared to competitors.

Table 2.1: Comparison of Mobile Development Technologies

Based on the comparative analysis of advantages and disadvantages in **Table 2.1**, our team has decided to adopt **Flutter** as the primary framework for mobile development. Flutter's highly customizable widget system enables the creation of rich, interactive user interfaces, making it especially suitable for applications that require enhanced user engagement. In addition, its ability to target multiple platforms through a unified codebase aligns with our long-term objective of expanding to additional platforms in the future. Considering the team's available development time and skill capacity, we conclude that Flutter offers the most balanced solution and is capable of meeting all current and foreseeable project requirements.

2.2 Server Development

2.2.1 Django

Django is a high-level Python web framework created in 2003 by Adrian Holovaty and Simon Willison at the Lawrence Journal-World. It was named after the guitarist Django Reinhardt and officially released to the public in 2005 [4]. The framework encourages rapid development and clean design by managing common web development tasks, freeing developers from "reinventing the wheel." It is also free and open source [5].

Django utilizes the Model-View-Template (MVT) pattern [6], wherein the Model manages the data structure and database queries, the View handles incoming web requests and returns the appropriate response, and the Template controls the layout and presentation of data.

2.2.2 FastAPI

FastAPI is a modern, high-performance web framework for building APIs with Python, originally released in December 2018 by Sebastián Ramírez [14]. It is designed to build APIs using standard Python type hints, ensuring valid data structures and editor support.

According to the official documentation [13], FastAPI distinguishes itself through several key features. First, as its name implies, the framework offers significant speed advantages and is considered one of the fastest Python frameworks available, delivering performance on par with NodeJS and Go. This speed applies not only to execution throughput but also to the rapid pace of development. Second, FastAPI is designed to enhance reliability by reducing developer-induced errors by approximately 40 percent, primarily through its strict typing system. Third, the framework prioritizes robustness and ease of use, featuring an intuitive interface that is easy to learn while minimizing code duplication by allowing multiple parameters per declaration and automatically generating production-ready interactive documentation. Finally, FastAPI is fully compatible with open standards for APIs, specifically OpenAPI (formerly Swagger) and JSON Schema, ensuring standards-based development.

2.2.3 Spring Boot

Spring Boot is an open-source, enterprise-level Java framework built upon the Spring ecosystem. Its primary objective is to streamline the development process by minimizing configuration overhead. Designed to run on the Java Virtual Machine (JVM), Spring Boot facilitates the creation of production-ready, stand-alone applications. The framework is defined by three core features [10]: autoconfiguration, which automatically configures the application based on the dependencies present; opinionated configuration, which provides sensible defaults to reduce the need for manual setup; and stand-alone execution capability, which enables applications to run independently without requiring an external web server.

2.2.4 Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment designed for building scalable network applications. Originally developed by Ryan Dahl in 2009 [11], it allows developers to use JavaScript for server-side scripting. Node.js is supported by a wide range of frameworks, such as Express.js and NestJS, which streamline backend development [12].

The Node.js architecture is characterized by several key features that contribute to its effectiveness in modern web development. Its non-blocking I/O model enables asynchronous execution of operations, preventing the system from stalling during heavy tasks and significantly reducing response latency while improving throughput. The platform offers a rich ecosystem that provides a comprehensive set of built-in modules, such as HTTP and File System, alongside access to a vast library of third-party packages via the Node Package Manager (NPM). Furthermore, Node.js employs an event-driven architecture that utilizes a single-threaded event loop to handle multiple concurrent clients efficiently, making it highly suitable for real-time applications.

2.2.5 Comparison and Server-side Technology Selection for the Project

Framework	Pros	Cons
Django (Python)	**Batteries-Included:** Built-in ORM, Auth, and Admin panel for rapid prototyping. **Mature & Secure:** Robust protection against CSRF and SQL injection.	**Monolithic:** High overhead for simple microservices; rigid built-in behaviors. **Synchronous Roots:** Less efficient for high-concurrency real-time features.
FastAPI (Python)	**High Performance:** Asynchronous architecture comparable to Node.js and Go. **Auto-Docs:** Automatic Swagger UI generation for streamlined API testing.	**Microframework:** Requires manual configuration of external libraries for Auth/DB. **Newer Ecosystem:** Smaller community and fewer niche third-party plugins.
Spring Boot (Java/Kotlin)	**Enterprise Standard:** Robust ecosystem for Cloud/Security and logic sharing with Android. **Scalable:** Designed for high-availability and complex distributed systems.	**Complexity:** Steep learning curve due to DI and AOP concepts. **Resource Heavy:** High memory consumption and slow startup times.
Node.js (JavaScript)	**Unified Language:** Allows JS/TS sharing between backend and mobile. **Real-Time Ready:** Event-driven architecture ideal for chat and I/O tasks.	**Single-Threaded:** CPU-intensive tasks can block the event loop. **Unopinionated:** Lack of enforced structure can lead to unmaintainable codebases.

Table 2.2: Comparison of Mobile Backend Frameworks

Selection of Backend Technologies Based on the comparative analysis of benefits and drawbacks presented in **Table 2.2**, our team has decided to adopt a Python-centric backend architecture utilizing both **Django** and **FastAPI**.

The primary driver for this decision is the unified language ecosystem. By utilizing Python for both the core application logic (Django) and the high-performance AI services (FastAPI), we ensure linguistic consistency across the entire server-side infrastructure. This reduces the cognitive load on developers and eliminates the need for context-switching between different programming languages.

Furthermore, Python was selected for its accessibility and machine learning capabilities. As highlighted in the comparison, Python offers a gentle learning curve, allowing the team to rapidly accelerate development. Crucially, given the project's requirement for a "Speech-to-Schedule" module, Python is the industry standard for Artificial Intelligence and Natural Language Pro-

cessing (NLP). Using Python allows us to integrate machine learning models directly into our FastAPI services without requiring complex bridges to external languages, ensuring seamless performance for the voice processing features.

2.3 Database Management System

2.3.1 MongoDB

MongoDB is a source-available, cross-platform, document-oriented database classified under the NoSQL paradigm. It utilizes BSON (Binary JSON), a binary representation of JSON-like documents that supports optional schemas. Development began in 2007 by the American software company 10gen as part of a platform-as-a-service product. In 2009, 10gen transitioned to an open-source development model and began offering commercial support, eventually rebranding itself as MongoDB Inc. [7].

Architecturally, MongoDB is designed to store semi-structured data. The database is organized into collections, which function similarly to tables in relational databases. Each collection contains multiple documents. A key feature of MongoDB is its schema flexibility: unlike relational rows, two distinct documents within the same collection are not required to share an identical structure [15].

2.3.2 PostgreSQL

PostgreSQL is a robust, open-source object-relational database management system (ORDBMS) known for its proven reliability and advanced architecture since 1996 [16].

About the main points of PostgreSQL, the system distinguishes itself through a unique combination of capabilities: it supports hybrid data models by natively storing JSON/JSONB alongside relational data, ensures high concurrency without locking conflicts via Multi-Version Concurrency Control (MVCC), and offers exceptional extensibility by allowing developers to write stored procedures in languages like Python. This versatility, combined with strict ACID compliance and Write-Ahead Logging for data integrity, makes it a highly adaptable solution for complex, data-driven applications [17].

2.3.3 Comparison and Database Management System Selection for the Project

Database	Pros	Cons
PostgreSQL (Relational/SQL)	**Data Integrity:** Fully ACID compliant with strong constraint enforcement (Foreign Keys). **Hybrid Capabilities:** Native JSONB support for storing unstructured data within relational tables.	**Rigid Schema:** Structural changes require migrations, which can be complex during growth. **Scaling:** Primarily relies on vertical scaling; sharding is more complex than in NoSQL systems.
MongoDB (Document/NoSQL)	**Flexible Schema:** No predefined structure; allows dynamic data format changes during development. **Horizontal Scalability:** Native sharding support for distributing data across multiple servers.	**Weaker Integrity:** Lacks default strict relationship enforcement; relies on application-level logic. **Resource Usage:** Higher RAM and disk requirements due to BSON data denormalization.

Table 2.3: Comparison of PostgreSQL and MongoDB

Based on the comparison in **Table 2.3**, our team decided to choose **MongoDB** as the primary Database Management System for this project. This decision was driven by three key factors aligned with our system's architecture.
First, MongoDB's **document-oriented model** is highly compatible with the data structures used in our application. Since the "Speech-to-Schedule" module utilizes Python and AI libraries that output complex, nested JSON data, storing this information directly in BSON format eliminates the need for complex object-relational mapping (ORM).
Second, the **flexible schema** facilitates rapid prototyping. As the project is in the development phase, the data requirements for user tasks and voice logs are subject to change. MongoDB allows us to iterate on the data model without the overhead of maintaining rigid schema migrations, ensuring that backend development can keep pace with changes in the mobile application.
Finally, the **horizontal scalability** of MongoDB ensures that the system can handle increasing volumes of unstructured voice data and notification logs efficiently as the user base grows.

2.4 Machine Learning

The Smart Schedule AI system employs a multi-layered machine learning pipeline to transform raw user voice commands into structured scheduling data. This pipeline integrates three key components: (1) **Whisper** for robust offline speech recognition, (2) a **Large Language Model (LLM)**, specifically Google's **Gemini**, for semantic understanding and entity extraction, and (3) **Duckling** for rule-based temporal expression normalization. Together, these components enable end-to-end automation of schedule creation from natural, spontaneous user input.

2.4.1 Whisper

Whisper is an open-source speech recognition model developed by OpenAI [24], trained on 680,000+ hours of multilingual and multitask supervised data [25]. Its architecture is based on a **transformer-encoder-decoder** design, allowing it to model long audio sequences and capture fine-grained phonetic features across diverse languages, including Vietnamese. One of Whisper's core strengths lies in its **robustness** to real-world noise conditions, such as background conversations, environmental sounds, microphone quality, and various accents. This robustness is essential for the Smart Schedule AI system, where user commands may be recorded in uncontrolled environments. Additionally, Whisper operates **fully offline**, ensuring data privacy and low latency during inference. In the system pipeline, Whisper serves as the first stage: converting raw voice input into text with high lexical accuracy. Because subsequent modules (LLM-based entity extraction and temporal normalization) rely on the produced text, Whisper's precision directly influences the overall reliability of the schedule generation process.

To select the most appropriate speech recognition engine, four leading solutions were evaluated based on offline capability, accuracy for Vietnamese, processing speed, cost, and suitability for personal assistant applications.

Table 2.4: Comparison of Speech Recognition Engines

Engine	**Offline**	**Accuracy**	**Speed**	**Cost**	**Best Use**
Google Cloud Speech	No	Excellent	High	Paid	Professional apps
Azure Speech	No	Very High	High	Paid	Enterprise
Vosk	Yes	Moderate	Moderate	Free	Offline apps
Whisper (Local)	**Yes**	**Excellent**	**Moderate**	**Free**	**High-accuracy offline**

Google Cloud Speech-to-Text offers excellent accuracy across 125+ languages with real-time streaming capabilities. However, it requires continuous internet connectivity and incurs recurring costs ($0.006 per 15 seconds), making it unsuitable for privacy-focused personal assistants operating in environments with limited connectivity.

Azure Speech Service provides enterprise-grade recognition with strong Vietnamese support and customization options. Like Google Cloud, it depends on internet connectivity and enterprise pricing structures that introduce operational complexity unnecessary for individual user applications.

Vosk is a lightweight offline toolkit supporting 20+ languages with compact 50 MB models. While fully offline and free, Vosk's accuracy significantly lags behind transformer-based models, particularly for noisy environments and code-switching between Vietnamese and English—common in Vietnamese users' speech patterns.

Why Whisper Was Chosen: After comprehensive evaluation, Whisper was selected as the optimal solution [25]. It uniquely combines cloud-level accuracy with full offline operation, trained on diverse multilingual data that handles Vietnamese colloquial expressions and code-switching naturally. All processing occurs locally, ensuring complete privacy for sensitive scheduling data without transmitting voice to external servers. The model operates without usage costs or API quotas, enabling sustainable long-term deployment. Whisper provides multiple model sizes (39M to 1550M parameters) allowing optimal trade-offs between accuracy and computational requirements—the medium model achieves strong accuracy on consumer hardware with 2-5 sec-

ond latency for typical voice commands. While cloud APIs offer marginally faster processing, Whisper's combination of offline operation, accuracy, privacy, zero cost, and Vietnamese language support makes it the optimal foundation for the Smart Schedule AI's speech-to-text pipeline.

2.4.2 Large Language Model (Gemini)

Google's Gemini [26] is a family of **multimodal** Large Language Models designed for advanced reasoning, language understanding, and contextual inference [27]. Unlike rule-based or traditional NLP pipelines, Gemini leverages deep transformer architectures to perform **semantic parsing** on complex, ambiguous user instructions.

In the Smart Schedule AI application, Gemini is used primarily for intent classification and entity extraction, including:

- detecting user intent (e.g., create, update, delete, or query a schedule)
- extracting event titles
- identifying date phrases (e.g., "tomorrow night", "next Monday", "tối mai")
- extracting time ranges or durations
- extracting contextual attributes such as location (optional)

One of the challenges in Vietnamese natural language input is **semantic ambiguity**. Expressions such as "tối nay", "cuối tuần", or "đặt lịch đá banh 7 giờ đến 9 giờ" may vary depending on cultural norms or context. Gemini mitigates this issue by leveraging **probabilistic semantic reasoning** [27], enabling it to infer the most likely interpretation based on linguistic patterns commonly observed in Vietnamese everyday communication. Furthermore, the use of an LLM significantly reduces the need for handcrafted grammar rules, dictionary patterns, or manual feature engineering. Gemini is capable of **generalizing** across unseen phrasing, making the system flexible and scalable.

2.4.3 Duckling

Duckling [28] is a **rule-based** natural language parsing library developed by Facebook, widely used for entity extraction tasks involving numbers, dates, times, durations, quantities, and currency [29]. While Gemini provides high-level semantic understanding, Duckling performs **deterministic normalization** of temporal expressions into standard machine-readable formats.

This component is essential for converting natural expressions such as:

- "tối mai" → `2025-12-01 20:00`
- "hai tiếng nữa" → `duration:  120 minutes`
- "7 giờ đến 9 giờ" → `start_time = 07:00, end_time = 09:00`
- "thứ hai tuần sau" → `2025-12-02`

Duckling ensures that ambiguous or culturally dependent date-time expressions are grounded into precise **ISO-8601 timestamps** that can be inserted directly into a digital calendar.

Because Duckling functions entirely offline [28], it complements Whisper in enabling a privacy-preserving pipeline for sensitive temporal data. The combination of deterministic parsing (Duckling) and probabilistic semantic inference (Gemini) provides both interpretability and flexibility.

In the system architecture, Duckling forms the final ML/NLP stage before persistence, enabling the conversion of raw extracted entities into structured objects ready for database storage and integration with calendar modules.

Chapter 3

System Design and Analysis

In Chapter 3, the authors describe the system design and analysis process. This chapter covers related works that provide motivation for the study, requirement analysis, use-case design, system architecture, database design, the speech-to-schedule module, and the application wireframes.

3.1 Related Works

3.1.1 Google Calendar

Google Calendar is a widely adopted digital scheduling platform offering event creation, reminders, recurring rules, sharing, and cross-device synchronization. Its natural-language event creation [18] is highly convenient but still struggles with ambiguous or culturally dependent expressions, especially in Vietnamese. This limitation motivates the Smart Scheduler AI's use of LLM-based extraction for deeper semantic understanding.

While Google Calendar provides reliable server-driven notifications, it does not adapt to user habits or productivity patterns. In contrast, Smart Scheduler AI aims to incorporate learning-based optimization [23] to make scheduling predictive rather than static. Google Calendar's semi-relational event model and use of metadata inspired the ERD of this project, though it lacks multimodal inputs such as speech commands. Smart Scheduler AI addresses this gap by integrating Whisper [24] for speech-to-text and transformer-based event extraction.

3.1.2 Water Reminder Applications

Water reminder apps such as Plant Nanny and WaterMinder illustrate principles of behavioral nudging and adaptive notifications. Most rely on simple rule-based reminders, but some personalize notification frequency using user compliance data. These design patterns highlight the importance of context-aware reminders—an idea central to the Smart Scheduler AI's adaptive scheduling strategy.

Their minimal interaction design, often allowing one-tap logging or wearable integration, also emphasizes low-friction user experience. This parallels the project's speech-based event creation using Whisper [25] and LLM-driven entity extraction, reducing the cognitive load of manual scheduling.

3.1.3 SleepCycle

SleepCycle applies audio-based detection and machine learning to analyze sleep patterns and deliver personalized recommendations [21]. Although focused on sleep tracking, it demonstrates how individualized behavioral modeling can improve daily routines. Its audio-processing pipeline is analogous to this project's use of Whisper [24] for converting raw audio into structured information.

SleepCycle's adaptive insights and "smart alarm" highlight the value of timing optimization, reinforcing the need for scheduling systems that adjust to user behavior. This aligns with the Smart Scheduler AI's long-term goal of learning from missed events or recurring habits and applying methods such as reinforcement learning or Bayesian optimization [23].

3.1.4 Notion Calendar

Notion Calendar [22] integrates scheduling with task and knowledge management, enabling events to be linked with projects, documents, and customizable metadata. This flexible structure informs the Smart Scheduler AI's potential use of enriched event attributes such as tags or priorities to improve learning and personalization.

Its streamlined interface and strong workflow integration emphasize low-friction interaction, a principle echoed in this project's speech-driven scheduling approach. Although Notion Calendar does not employ AI-driven optimization, its robust event data model and synchronization mechanisms serve as useful references for designing the backend architecture of the Smart Scheduler AI.

3.1.5 Comparative Analysis: Smart Schedule AI

The Smart Schedule AI represents a unified approach to personal time management by integrating calendar scheduling, hydration reminders, and sleep management into a single application. While specialized apps like Google Calendar, water reminder tools, and SleepCycle [21] excel in their respective domains, they operate as isolated systems requiring users to manage multiple applications simultaneously. This fragmentation creates cognitive load from context-switching and prevents holistic optimization—for example, avoiding water reminders during meetings or adjusting sleep times based on next-day calendar commitments.

A key differentiator is the voice-first AI interface powered by Whisper [24] and Gemini [26]. Users can create events and tasks through natural speech such as "Tôi có meeting với team vào 3 giờ chiều mai" or "Nhắc tôi uống nước mỗi 2 tiếng," transforming the application from passive storage into an active virtual assistant that proactively manages daily routines with minimal user effort.

To prevent feature bloat, Smart Schedule AI adopts a minimalist philosophy, implementing only essential functionalities from each domain. Hydration tracking focuses on reminder delivery and basic logging, omitting gamification elements. Sleep management provides recommendations and smart alarms without comprehensive sleep stage analysis. The calendar emphasizes voice-driven event creation and intelligent scheduling suggestions [23] while avoiding enterprise features like room booking or complex permissions. This streamlined approach maintains a lightweight footprint suitable for mobile devices while focusing engineering resources on perfecting cross-domain integration and AI-powered optimization—the system's core value proposition.

3.2 Requirement Analysis

3.2.1 Domain Context and Stakeholders

Domain Context

In today's technology-driven world, the labor market is becoming increasingly competitive, requiring individuals to continuously improve their skills and productivity. As workloads intensify and responsibilities expand, effective self-management has emerged as one of the most essential competencies for personal and professional success. To meet these demands, individuals frequently rely on digital tools to organize their schedules, track tasks, and monitor personal well-being. However, current practices often require users to download and manage multiple separate applications, resulting in fragmented workflows and decreased efficiency.

The growing need for integrated and minimalistic solutions highlights a significant gap in the personal productivity domain. Most users still manage their tasks and schedules through manual interactions—such as tapping on mobile interfaces or drag-and-drop actions on websites or desktop tools. While these methods are functional, they can be time-consuming and cognitively demanding, especially for individuals balancing multiple responsibilities. As a result, there is increasing interest in technologies that streamline these processes and reduce the interaction burden.

Advancements in artificial intelligence, particularly natural language processing (NLP), offer promising opportunities to transform how individuals manage their daily activities. By enabling users to create schedules, manage tasks, and perform organizational actions through natural language commands, NLP can significantly enhance accessibility and reduce the friction associated with traditional manual input methods. This aligns with contemporary trends toward automation, intelligent digital assistants, and voice-driven user experiences.

Positioned within this evolving landscape, our project aims to develop an integrated visual assistant that consolidates health, work, and lifestyle management into a single, unified application. By combining minimalistic design principles with AI-powered interaction, the system seeks to simplify self-management processes, reduce distractions, and support users in maintaining a balanced and productive lifestyle. The value of such an approach is closely aligned with modern productivity theories, emphasizing focus, efficiency, and seamless task execution.

Stakeholders

For this project, stakeholders are categorized into two groups: internal stakeholders and external stakeholders.

- **Internal Stakeholders:**
  - **Business Owners / Development Team:** This group includes all project members responsible for planning, designing, and implementing the application. They define requirements, make technical decisions, and ensure the system meets its objectives.
  - **Project Instructor / Supervisor:** The instructor acts as both advisor and evaluator. They provide guidance during development, ensure academic standards are met, and assess the final project outcome.

- **External Stakeholders:**

  – **Customers / End Users:** These stakeholders include students, productivity enthusiasts, and professionals who will use the application to manage schedules, tasks, and health routines. Their experience and feedback determine system effectiveness.

  – **Technology Providers:** These include third-party tools, platforms, and frameworks such as Flutter, Django, Dart, Python, NLP models, hosting services, and databases. Their stability and performance directly impact the product quality.

3.2.2 Functional Requirements

Authentication and User Information – Functional Requirements

- **FR-Auth-1: User Registration**
  The system shall allow a new user to create an account using an email address and password.
  The system shall validate that the email address is in a correct format and is not already registered.

- **FR-Auth-2: User Login (Email and Password)**
  The system shall allow a registered user to log in using their email address and password.
  The system shall display an appropriate error message if the credentials are invalid.

- **FR-Auth-3: Login with Google Account**
  The system shall allow a user to log in using a Google account via OAuth or a similar authentication mechanism.
  The system shall create a new user account automatically if a Google login is successful and no corresponding local account exists.

- **FR-Auth-4: Password Reset (Forgot Password)**
  The system shall provide a "Forgot Password" option on the login screen.
  The system shall send a password reset email to the user's registered email address upon request.
  The system shall allow the user to set a new password via a secure link contained in the password reset email.

- **FR-Auth-5: Update Avatar**
  The system shall allow an authenticated user to upload or change their avatar image.
  The system shall display the updated avatar in the user profile and other relevant screens.

- **FR-Auth-6: Initial Personal Information Input**
  The system shall require a user, on first login, to provide personal information including full name, date of birth, weight, height, and sex.
  The system shall store this personal information in the user's profile.

- **FR-Auth-7: Update Personal Information**
  The system shall allow an authenticated user to view and update their personal information from a profile screen.

- **FR-Auth-8: Change Password**
  The system shall allow an authenticated user to change their password from within the application.
  The system shall require the user to enter their current password before setting a new password.

- **FR-Auth-9: Theme Mode (Dark/Light Mode)**
  The system shall allow an authenticated user to switch between dark mode and light mode.
  The system shall apply the selected theme immediately after the user changes the setting.
  The system shall retain the user's selected theme for subsequent sessions.

Schedule Management – Functional Requirements

- **FR-Schedule-1: Add Schedule**
  The system shall allow a user to create a new schedule by providing a title, selecting a date, choosing a repeat option, and assigning a category.
  The system shall validate all required fields before saving a new schedule.
- **FR-Schedule-2: Manual Schedule Input**
  The system shall provide a manual input form that allows users to create schedules without using automated or AI-based features.
- **FR-Schedule-3: AI Voice Input for Schedule Creation**
  The system shall allow users to create schedules using voice input.
  The system shall process voice input using natural language processing (NLP) to extract relevant schedule information.
  The system shall automatically generate a schedule based on the interpreted voice command.
- **FR-Schedule-4: User-Created Categories During Schedule Creation**
  The system shall allow users to create custom categories while creating a schedule.
- **FR-Schedule-5: Modify Schedule**
  The system shall allow a user to edit an existing schedule.
  The system shall display a confirmation prompt before saving any modifications.
- **FR-Schedule-6: Delete Schedule**
  The system shall allow a user to delete an existing schedule.
  The system shall display a confirmation prompt before deleting a schedule.
- **FR-Schedule-7: Category Assignment**
  The system shall allow users to assign a category to each schedule.
  The system shall allow users to define a color for each category.
- **FR-Schedule-8: User-Created Categories from Category View**
  The system shall allow users to create custom categories while viewing the category list.
- **FR-Schedule-9: Category List Viewing**
  The system shall allow users to view all available schedule categories.
- **FR-Schedule-10: Category Visibility Control**
  The system shall allow users to select which categories are visible or hidden in the schedule view.
- **FR-Schedule-11: Category-Based Reminders**
  The system shall allow users to configure reminders based on the category assigned to a schedule.
  The system shall trigger reminders according to predefined reminder rules.

- **FR-Schedule-12: Schedule Views**
  The system shall provide multiple schedule views, including a Today View and a Week View.
- **FR-Schedule-13: Navigation Between Days and Weeks**
  The system shall allow users to navigate to the next or previous day.
  The system shall allow users to navigate to the next or previous week.

Task Management – Functional Requirements

- **FR-Task-1: Create Goal**
  The system shall allow a user to create a goal by specifying a title and a deadline.
  The system shall validate all required fields before saving a goal.
- **FR-Task-2: Create Task**
  The system shall allow a user to create a task by entering a task title.
  The system shall create tasks independently unless they are explicitly assigned to a goal.
- **FR-Task-3: Assign Task to Goal (Drag and Drop)**
  The system shall allow a user to assign a task to a goal using a drag-and-drop interaction.
  The system shall update the task-to-goal assignment immediately after the action.
- **FR-Task-4: Move Task Between Goals**
  The system shall allow a user to move a task between different goals using drag-and-drop.
- **FR-Task-5: Edit Task**
  The system shall allow a user to edit an existing task.
  The system shall require confirmation before saving task modifications.
- **FR-Task-6: Mark Task as Completed**
  The system shall allow a user to mark a task as completed.
  The system shall visually indicate the completion status of the task.
- **FR-Task-7: Goal Progress Calculation**
  The system shall update the progress of a goal when associated tasks are marked as completed or uncompleted.
  The system shall calculate goal progress as a percentage of completed tasks.
- **FR-Task-8: Remove Task**
  The system shall allow a user to delete a task.
  The system shall display a confirmation prompt before task deletion.
- **FR-Task-9: Remove Goal**
  The system shall allow a user to delete a goal.
  The system shall handle associated tasks according to predefined system rules (e.g., unassigning or deleting tasks).
- **FR-Task-10: View Goal Progress**
  The system shall allow a user to view goal progress in percentage format.
  The system shall update goal progress in real time as task status changes.

Health Management – Functional Requirements

- **FR-Health-1: Input Body Measurements**
  The system shall allow a user to input body measurements, including weight and height.
  The system shall validate that all input values fall within reasonable human ranges.
- **FR-Health-2: Calculate Daily Water Intake**
  The system shall calculate a recommended daily water intake based on the user's body measurements.
  The system shall update the recommended value whenever body measurements are modified.
- **FR-Health-3: Update Water Intake**
  The system shall allow a user to update consumed water intake using increment and decrement controls.
  The system shall display water intake updates in real time.
- **FR-Health-4: Input and Track Sleep Duration**
  The system shall allow a user to input daily sleep duration.
  The system shall allow a user to configure a sleep reminder time.
- **FR-Health-5: Sleep Reminder Notifications**
  The system shall send sleep reminder notifications according to user-defined settings.
- **FR-Health-6: Water Intake Notifications**
  The system shall send reminder notifications to prompt users to drink water.
  The system shall adjust notification timing based on the user's daily water intake progress.

3.2.3 Non-Functional Requirements

Performance Requirements

- **NFR-Perf-1: Concurrent User Support**
  The system shall support at least 1,000 concurrent users without significant performance degradation.
- **NFR-Perf-2: Response Time**
  The system shall maintain an average response time of no more than 2 seconds for common operations, including viewing schedules, updating tasks, and accessing health data.
- **NFR-Perf-3: AI Scheduling Processing Time**
  The AI-powered quick scheduling feature shall process and generate schedules within 5 seconds under normal system load.
- **NFR-Perf-4: Calendar Data Loading**
  Calendar data for the previous 12 months shall be loaded within 3 seconds under normal operating conditions.
- **NFR-Perf-5: Notification Timing Accuracy**
  The system shall trigger water intake and sleep reminder notifications within 1 minute of their scheduled trigger time.

Reliability and Availability Requirements

- **NFR-Reli-1: System Availability**
  The system shall maintain a minimum uptime of 99.5% per month, excluding scheduled maintenance periods.

- **NFR-Reli-2: Data Retention**
  The system shall retain user data, including schedules and personal information, for a minimum of 365 days after the user's last recorded activity unless deletion is explicitly requested.
  Upon a user-initiated deletion request, all associated data shall be permanently removed within 7 days.

- **NFR-Reli-3: Fault Recovery**
  The system shall provide automatic recovery mechanisms capable of restoring core system functionalities within 5 minutes after an unexpected failure.
  The system shall ensure that no committed transactions are lost during the recovery process.

- **NFR-Reli-4: Notification Reliability**
  The notification service shall successfully deliver at least 99% of all scheduled reminders during normal operation.
  The system shall ensure that reminder notifications are neither duplicated nor omitted, with delivery occurring within ± 1 minute of the scheduled trigger time.

- **NFR-Reli-5: Timezone Support**
  The system shall support scheduling across multiple timezones, covering at least UTC-12 to UTC$+12$.
  Events, reminders, and health notifications shall be displayed and triggered at the correct local time with zero offset error after timezone conversion.

Security and Privacy Requirements

- **NFR-Sec-1: Secure Communication**
  All communication between client and server shall be encrypted using industry-standard protocols such as HTTPS/TLS.

- **NFR-Sec-2: Data Protection**
  Sensitive user data, including schedules, health metrics, and personal information, shall be securely stored using appropriate encryption mechanisms.

- **NFR-Sec-3: Access Control**
  The system shall enforce access control to ensure that users can access only their own data.

- **NFR-Sec-4: Credential Storage**
  Authentication credentials shall be stored using secure hashing algorithms consistent with industry best practices.

- **NFR-Sec-5: Privacy Compliance**
  The system shall adhere to data protection principles, including user consent, data minimization, and secure data deletion upon user request.

- **NFR-Sec-6: Notification Privacy**
  The system shall suppress notifications during protected periods, including: user-defined sleeping hours, focus sessions and scheduled meetings.

Usability and User Experience Requirements

- **NFR-Usab-1: Interface Consistency**
  The system shall provide a consistent and uniform user interface across all functional modules.
  At least 90% of UI components (e.g., buttons, icons, navigation elements) shall follow a unified design style and interaction behavior across modules.

- **NFR-Usab-2: Learnability**
  A new user shall be able to perform core operations, including creating schedules, updating water intake, and managing tasks, within 10 minutes without external guidance.
  This criterion shall be satisfied by at least 80% of first-time users during usability testing.

- **NFR-Usab-3: Theme Support**
  The system shall support both light mode and dark mode and apply the selected theme consistently across all screens.
  Theme changes shall be applied within 1 second and persist across application restarts.

- **NFR-Usab-4: Accessibility**
  The user interface shall comply with basic accessibility standards, including readable font sizes, sufficient color contrast, and screen reader compatibility.
  At least 90% of UI screens shall meet WCAG 2.1 Level AA contrast and readability guidelines.

- **NFR-Usab-5: Schedule Visualization**
  The system shall provide multiple schedule views, including daily and weekly views, to support user navigation.
  Users shall be able to switch between schedule views within 2 seconds.

- **NFR-Usab-6: Voice Interaction Usability**
  The AI-based voice scheduling feature shall be usable without requiring technical knowledge or training.
  At least 80% of voice input commands issued by first-time users shall be correctly interpreted and converted into schedules on the first attempt.

3.3 Use-case Design

3.3.1 General Use-case view

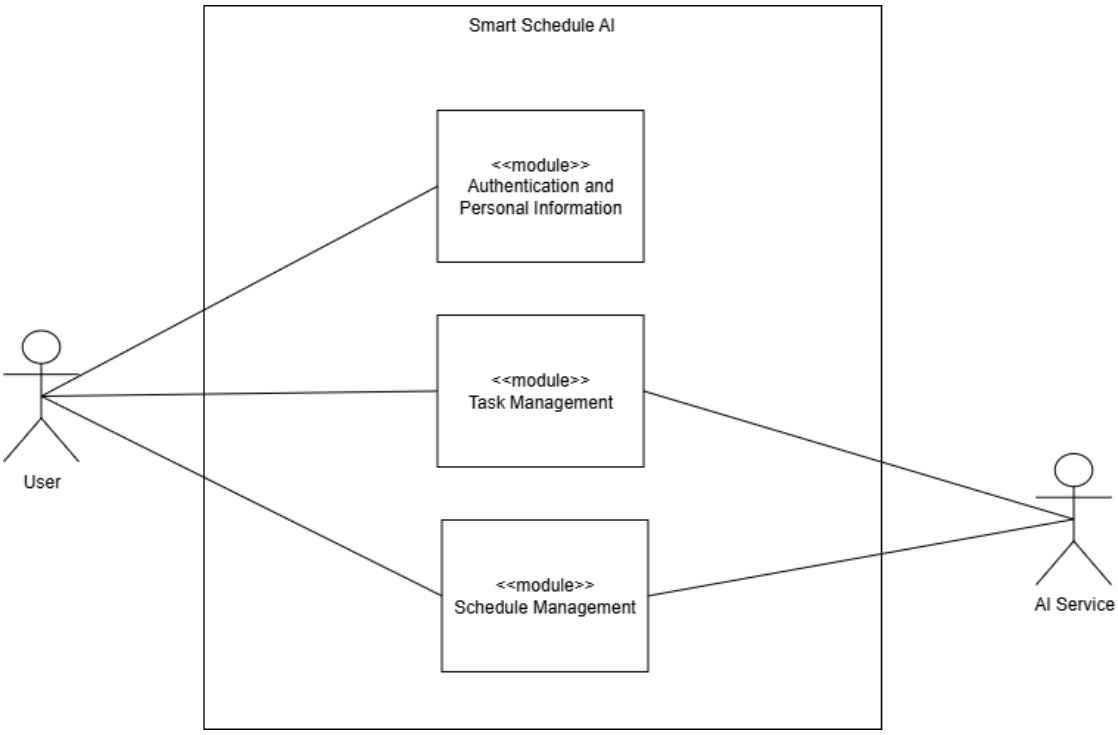

Figure 3.1: General Use-case view

Figure 3.1 illustrates the general use case of the entire system. The system involves two main actors: the user and the AI Service (or Speech-to-Text Service). In addition, the system is composed of three main modules:

- **Authentication and Personal Information**: This module handles user authentication and the management of personal information.
- **Task Management**: This module provides task-related services, such as creating and editing tasks and goals.
- **Schedule Management**: This module offers scheduling services, including creating and editing schedules.

3.3.2 Authentication and Personal Information

Use-case Diagram

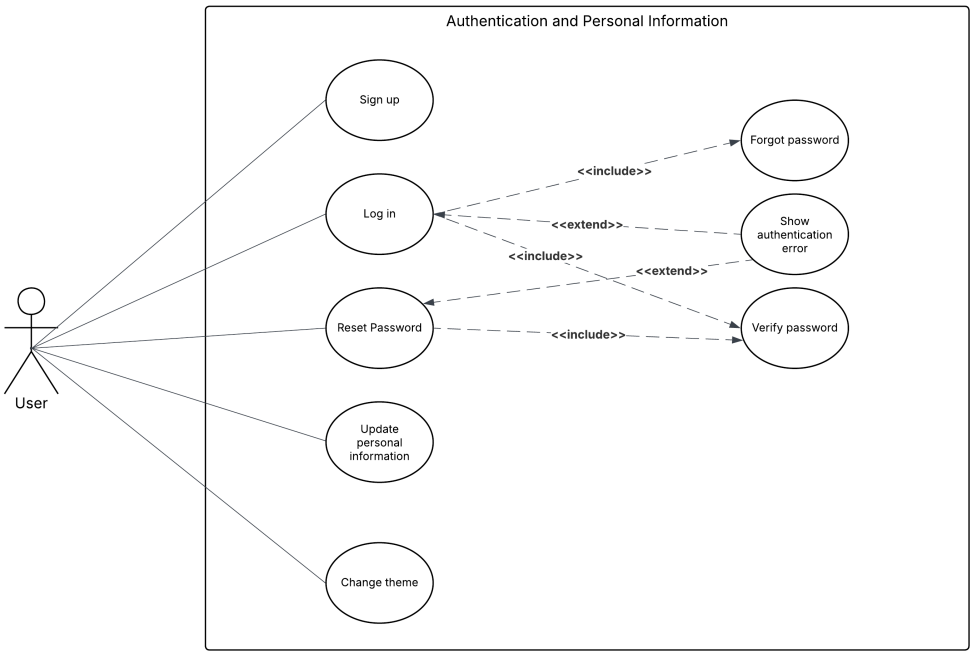

Figure 3.2: Authentication and Personal Information Use-case Diagram

Figure 3.2 illustrates the use-case of Authentication and Personal Information, which includes 2 main sections: Authentication and Personal Information.

The user can create a new account via the sign up feature, after that, they can log into the system to use our system. Besides, in the worst case, if they forget their password, they can reset their password.

Personal Information supports 2 main purposes. First, users can update their information if something changes. For example, users who have an error while signing up (e.g. first name, last name, sex, etc.), can change this. Next, they can change the theme of the application. Our app has 2 versions of theme including Light Mode and Dark Mode, users can change to the theme that they want.

Use-case Description

Use Case ID	UC-01
Use Case Name	Sign Up
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User register a new account in the system.
Trigger	The user select "Sign Up" option.
Preconditions	PRE-1. User is not already registered.
Postconditions	POST-1. A new account is created and stored into database.
Normal Flow	1. User provides personal information. (username, password, email, etc.) 2. System validates the inputs. 3. System creates an account. 4. A confirmation message is displayed to the user.
Alternative Flow	Alternative at step 2: the username is already existed. 2.a.1 Display the notification to user. Continue at step 1 of basic flow. Alternative at step 2: the email is already existed. 2.b.1 System display the notification to user Continue at step 1
Exceptions	Exception at step 3: Database connection error 3.a.1 Display error message to the user. Use-case ends here. Exception at step 2: System validation service unavailable 2.c.1 Inform the user to try again later. Use-case ends here.

Table 3.1: Use Case UC-01: Sign Up

Table 3.1 is a description table of use case Sign Up. Through this use case, user can sign up for their account with their provided details. If everything goes correctly, there will be a message notifying to the user. Nevertheless, if the username or email has already been taken, they have to choose another username or email. If there is an error while signing up, users can try again later.

Use Case ID	UC-02
Use Case Name	Log In
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User logs in to their account via their username and password.
Trigger	The user select "Log In" option.
Preconditions	PRE-1. User has already registered their account.
Postconditions	POST-1. User is authenticated and redirected to the main page.
Normal Flow	1. User provides username and password. 2. System checks credentials. 3. System verifies password. 4. System grants access to user.
Alternative Flow	Alternative at step 2: The username or password is incorrect. 2.a.1 Display notification to user. Continue at step 1.
Exceptions	Exception at step 2: Database connection error 2.b.1 Display error message to the user. Use-case ends here. Exception at step 2: System validation service unavailable 2.c.1 Inform the user to try again later. Use-case ends here.

Table 3.2: Use Case UC-02: Log In

Table 3.2 of the use case Log In. Through this use case, after signing up, users can log into their accounts. If the credentials are correct, the user will be redirected to the home page. However, if the credentials are wrong, the users are required to re-enter again. If any error occurs, the system will display to the user and ask them to try again later.

Use Case ID	UC-03
Use Case Name	Reset Password
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User resets their account password.
Trigger	The user select "Reset Password" option.
Preconditions	PRE-1. User has already registered their account.
Postconditions	POST-1. The password of the user account is changed.
Normal Flow	1. User provides username, old password, and new password. 2. System validates current credentials. 3. System validates new password requirements. 4. System updates the password in the database. 5. Confirmation message is displayed to the user.
Alternative Flow	Alternative at step 2: Old password verification failed. 2.a.1 Display authentication error to user. Continue at step 1. Alternative at step 3: New password does not meet security requirements. 3.a.1 System rejects and asks for a stronger password. Continue at step 1.
Exceptions	Exception at step 4: Password update fails due to database error. 4.a.1 System informs the user to try again later. Use-case ends here. Exception at step 2: System validation service unavailable. 2.b.1 System displays a service unavailable message. Use-case ends here.

Table 3.3: Use Case UC-03: Reset Password

Table 3.3 is the description table of the use case Reset Password. Through this use case, user can reset their password. If all provided details are correct, the user's password will be changed. However, if any incorrect information is provided, the user's password will not be changed, and the system will display a notification to the user, and they can retry later.

Use Case ID	UC-15
Use Case Name	Update Personal Information
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User updates their personal information.
Trigger	User selects the "Edit" option on the Personal Information page.
Preconditions	PRE-1. User must be logged in with a registered account.
Postconditions	POST-1. The system saves the updated personal information and displays the updated details.
Normal Flow	1. User navigates to Personal Information page. 2. User selects "Edit" button. 3. User edits information (name, date of birth, sex, address, etc.). 4. User clicks "Save" button. 5. System validates inputs. 6. System updates information in the database. 7. System redirects user to the Personal Information page.
Alternative Flow	Alternative at step 3: User clicks "Cancel" button. 3.a.1 System discards changes. 3.a.2 System redirects user to Personal Information page (step 7). Use-case ends here. Alternative at step 5: Invalid data entered (e.g., invalid phone format). 5.a.1 System displays error message to user. Continue at step 3.
Exceptions	Exception at step 6: Database error while saving. 6.a.1 System shows "Update Failed" message. Use-case ends here.

Table 3.4: Use Case UC-15: Update Personal Information

Table 3.4 is the description table of the use case Update Personal Information. Through this use case, user can update their personal details. If everything goes correctly, the user's information will be updated. However, if any information is wrong (e.g. letter in a phone number, etc.), users cannot change the information.

Use Case ID	UC-16
Use Case Name	Change Theme
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User changes the visual theme of the application (Light/Dark mode).
Trigger	User toggles the "Change Theme" switch.
Preconditions	PRE-1. User must be logged in.
Postconditions	POST-1. The application interface updates to the selected theme.
Normal Flow	1. User toggles the theme switch. 2. System updates the UI appearance to the target theme. 3. System saves the user's preference.
Alternative Flow	None.
Exceptions	Exception at step 3: Theme setting fails to save due to system error. 3.a.1 System displays an error message. 3.a.2 System keeps the current theme. Exception at step 2: Client-side rendering issue. 2.a.1 System reverts to the default theme (light mode).

Table 3.5: Use Case UC-16: Change Theme

Table 3.5 is the description table of the use case Change Theme. With this use case, user can change the theme of the application. Conventionally, the default theme of the application is the Light Mode. However, if the users love Dark Mode, they can change to Dark Mode to have a better experience.

3.3.3 Task Management

Use-case Diagram

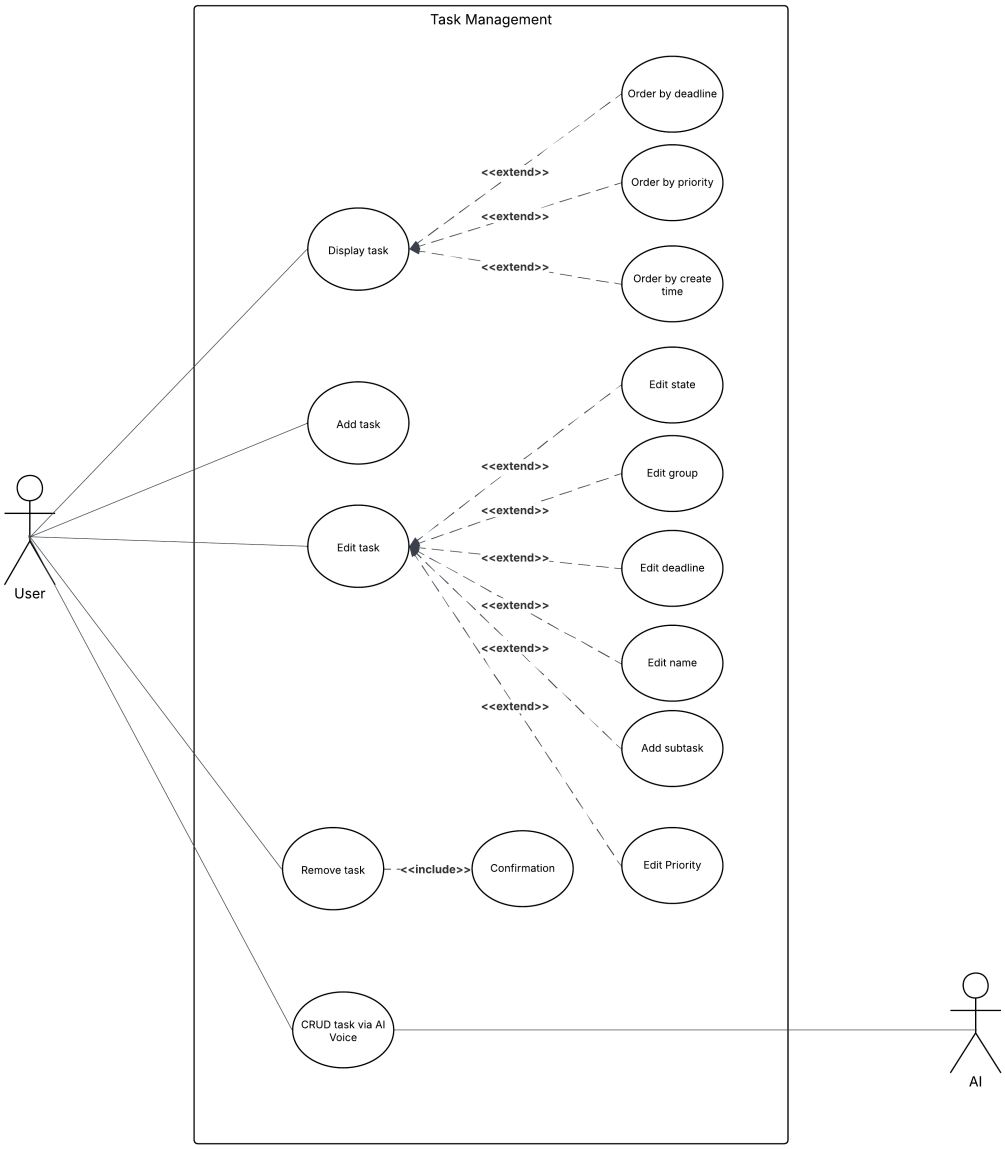

Figure 3.3: Task Management Use-case Diagram

Figure 3.3 illustrates the use case diagram of Task Management. This component includes one of the application's significant parts: Task Management. Tasks are managed using conventional approaches, but users can use their voice via the AI Voice option to set tasks quickly.

Users can create, update, delete, or retrieve a list of tasks. For instance, if you have an assignment to do, you can set a task name 'Assignment', and after finishing this assignment, you can update the task status to 'Done'. Moreover, user can use their voice to set a task by saying after pressing the voice icon, like "Create task Assignment".

Use-case Description

Use Case ID	UC-05
Use Case Name	Add Task
Primary Actor(s)	User **Secondary Actor(s)** None
Description	The user creates a new task to add to the schedule or task list.
Trigger	The user selects the "Add Task" function.
Preconditions	PRE-1. The user has logged into the system.
Postconditions	POST-1. The new task is saved in the database and displayed in the task list.
Normal Flow	1. User clicks the "Add Task" button. 2. System displays a form for entering task information (name, description, deadline, priority). 3. User enters the required information. 4. User clicks "Save". 5. System validates input and saves the task to the database. 6. System displays the new task in the list.
Alternative Flow	Alternative at step 3: User cancels the operation. 3.a.1 System discards data. 3.a.2 System returns to the previous screen. Alternative at step 5: Missing required information (e.g., task name). 5.a.1 System shows an error and prompts the user to re-enter data. Continue at step 3.
Exceptions	Exception at step 5: Database connection error. 5.b.1 System displays error message: "Unable to save task. Please try again later." Use-case ends here.

Table 3.6: Use Case UC-05: Add Task

Table 3.6 is a description table of the use case Add Task. With this use case, user can add a

new task to the list of tasks. If everything goes correctly, a new task will be added to the list. Nevertheless, if something goes wrong, no task will be added to the list, and the system will notify the user.

Use Case ID	UC-04
Use Case Name	Display Task
Primary Actor(s)	User **Secondary Actor(s)** None
Description	The user views the list of tasks. The system displays the list with details (title, description, start time, deadline, status). The user can sort the list by deadline, priority, or creation time.
Trigger	The user selects the "View Task List" function.
Preconditions	PRE-1. The user has successfully logged in.
Postconditions	POST-1. The task list is displayed according to the selected sorting criteria.
Normal Flow	1. User selects the "Display Task" feature. 2. System retrieves the user's task list from the database. 3. System displays the task list (title, start/end time, deadline, priority, status). 4. User scrolls and views details of each task.
Alternative Flow	Alternative at step 2: No task exists in the database. 2.a.1 System displays message: "You don't have any tasks yet. Please create a new task." Use-case ends here. Alternative at step 3: User changes sort order. 3.a.1 User selects option to sort by Deadline, Priority, or Time. 3.a.2 System re-sorts and displays the task list according to the chosen criteria.
Exceptions	Exception at step 2: Database connection error. 2.b.1 System displays message: "Unable to load data. Please try again later." Use-case ends here.

Table 3.7: Use Case UC-04: Display Task

Table 3.7 is the description table of the use case Display Task. With this use case, user can retrieve a list of tasks. If everything is OK, the list of tasks will be displayed to the user. However, if something goes wrong (e.g. Database connection error, etc.), the system will display the error to the user.

Use Case ID	UC-06
Use Case Name	Edit Task
Primary Actor(s)	User **Secondary Actor(s)** None
Description	The user edits the information of an existing task (status, group, deadline, name, or priority).
Trigger	The user selects the "Edit Task" function on an existing task.
Preconditions	PRE-1. The user has logged into the system. PRE-2. The task to be edited exists in the database.
Postconditions	POST-1. The task is updated with the new information in the database.
Normal Flow	1. User selects "Edit Task". 2. System displays the current task information form. 3. User modifies the desired fields (Status, Group, Deadline, Name, Priority). 4. User clicks "Save". 5. System validates and updates the task in the database. 6. System displays the task with updated information.
Alternative Flow	Alternative at step 4: User cancels the operation. 4.a.1 System discards changes. 4.a.2 System returns to the task detail view without updating.
Exceptions	Exception at step 2: Task does not exist (e.g., deleted by another session). 2.a.1 System displays error message: "Task not found." Use-case ends here. Exception at step 5: Data saving error. 5.a.1 System displays error message: "Unable to update. Please try again later." Use-case ends here.

Table 3.8: Use Case UC-06: Edit Task

Table 3.8 is the description table of the use case Edit Task. With this use case, user can edit the existing tasks. If everything goes correctly, the task's details will be updated. On the one hand, if everything goes wrong (e.g. edit a deleted task, etc.), the error will be displayed to the user.

Use Case ID	UC-07
Use Case Name	Remove Task
Primary Actor(s)	User **Secondary Actor(s)** None
Description	The user deletes a task from the list. The system requires confirmation before deletion.
Trigger	The user selects the "Remove Task" button from the task list.
Preconditions	PRE-1. The user has logged in. PRE-2. The task to be deleted exists in the database.
Postconditions	POST-1. The task is deleted from the database and no longer displayed in the list.
Normal Flow	1. User selects "Remove Task". 2. System displays a confirmation window ("Are you sure you want to delete this task?"). 3. User selects "Yes". 4. System deletes the task from the database. 5. System updates the task list.
Alternative Flow	Alternative at step 3: User selects "No" (Cancels). 3.a.1 System cancels the deletion process. 3.a.2 System closes the confirmation window and retains the task.
Exceptions	Exception at step 1: Task does not exist. 1.a.1 System displays error message: "Task not found." Use-case ends here. Exception at step 4: Database connection error. 4.a.1 System displays error message: "Unable to delete. Please try again later." Use-case ends here.

Table 3.9: Use Case UC-07: Remove Task

Table 3.9 is the description table of the use case Remove Task. With this use case, user can delete the existing tasks. If tasks are deleted successfully, the success message will be displayed to the user. In the opposite case, if there is an error occurs, the error message will be displayed to the user.

Use Case ID	UC-18
Use Case Name	Add Task via Voice
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User creates a task using voice commands interpreted by AI.
Trigger	User taps the "Voice" button.
Preconditions	PRE-1. App has internet access.
Postconditions	POST-1. Task is saved and displayed with recognized details.
Normal Flow	1. User taps "Voice" button and speaks request. 2. System processes audio and presents details for confirmation. 3. User agrees to the details. 4. System saves the task.
Alternative Flow	Alternative at step 1: Permission missing. 1.a.1 System requests microphone access. 1.a.2 If granted, proceed to step 1. Alternative at step 4: Category does not exist. 4.a.1 System auto-creates the new category. 4.a.2 System saves the task.
Exceptions	Exception at step 2: AI Service unavailable/Network error. 2.a.1 System displays error message. Use-case ends here. Exception at step 2: Timezone mismatch in extracted data. 2.b.1 System auto-converts time to device local time. Continue at step 3.

Table 3.10: Use Case UC-18: Add Task via Voice

Table 3.10 is a description table of the use case Add Task via Voice. Through this use case, user can add a new task with the voice button. For the success cases, after clicking the voice button and saying the request, the request will display the information of the task and ask the user to confirm. If the user confirms the information, a new task will be added to the user's list of tasks. On the opposite side, the error message will be illustrated to the user.

Use Case ID	UC-21
Use Case Name	Display Task via Voice
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User retrieves and views their schedule using voice commands.
Trigger	User taps the "Voice" button.
Preconditions	PRE-1. User is logged into the app. PRE-2. Device internet connection is active.
Postconditions	POST-1. The schedule/task list is displayed to the user.
Normal Flow	1. User taps "Voice" button. 2. User speaks display command (e.g., "Show my tasks today"). 3. System processes command and queries the database. 4. System displays the requested schedule.
Alternative Flow	Alternative at step 3: No tasks found. 3.a.1 System notifies user that the schedule is empty. Use-case ends here.
Exceptions	Exception at step 3: AI Service/Connection failure. 3.b.1 System displays a "Service Unavailable" pop-up. Use-case ends here. Exception at step 3: API fails to process request. 3.c.1 System returns an error response asking user to retry. Use-case ends here. Exception at step 4: Schedule displayed with wrong Time Zone. 4.a.1 System auto-converts time to device's local time zone.

Table 3.11: Use Case UC-21: Display Task via Voice

Table 3.11 is the description table of the use case Display Task via Voice. Through this use case, user can retrieve a list of tasks with their voice. If there is no error, the list of tasks will be displayed to the user. Nevertheless, if the errors appear, the error message will be demonstrated to the user.

Use Case ID	UC-20
Use Case Name	Edit Task via Voice
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User updates an existing task using voice commands.
Trigger	User taps the "Voice" button.
Preconditions	PRE-1. User is logged in. PRE-2. Internet connection is active.
Postconditions	POST-1. Task is updated with confirmed details.
Normal Flow	1. User taps "Voice" and speaks task details to edit. 2. System processes input and requests confirmation. 3. User confirms changes. 4. System updates the task.
Alternative Flow	Alternative at step 1: Permission missing. 1.a.1 System requests microphone access. 1.a.2 If granted, proceed to step 1. Alternative at step 4: New category detected. 4.a.1 System auto-creates category. 4.a.2 System completes task update.
Exceptions	Exception at step 2: AI Service/Connection failure. 2.a.1 System displays error message. Use-case ends here. Exception at step 2: Timezone mismatch. 2.b.1 System auto-converts to local time. Continue at step 2.

Table 3.12: Use Case UC-20: Edit Task via Voice

Table 3.12 is the description table of the use case Edit Task via Voice. With this use case, user can edit the task with their voice. After the user's voice is recorded and analyzed, the confirmation message will be displayed to the user. If the user agrees with the confirmation, the details will be updated. In other cases, the error message will be illustrated to the user.

Use Case ID	UC-19
Use Case Name	Remove Task via Voice
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User deletes a task using voice commands.
Trigger	User taps the "Voice" button.
Preconditions	PRE-1. User is logged in. PRE-2. Internet connection is active.
Postconditions	POST-1. The specified task is removed from the database.
Normal Flow	1. User taps "Voice" and speaks task ID to delete. 2. System identifies task and requests confirmation. 3. User confirms deletion. 4. System deletes the task.
Alternative Flow	Alternative at step 1: Permission missing. 1.a.1 System requests microphone access. 1.a.2 If granted, proceed to step 1. Alternative at step 3: User cancels/denies. 3.a.1 System cancels the operation.
Exceptions	Exception at step 2: AI Service/Connection failure. 2.a.1 System displays error message. Use-case ends here. Exception at step 2: Task not found. 2.b.1 AI informs user that the task ID is invalid. Use-case ends here.

Table 3.13: Use Case UC-19: Remove Task via Voice

Table 3.13 is the description table of the use case Remove Task via Voice. With this use case, user can remove the existing tasks. If everything goes correctly, the tasks will be deleted. However, if something goes wrong, the error message will be displayed.

3.3.4 Schedule Management

Use-case Diagram

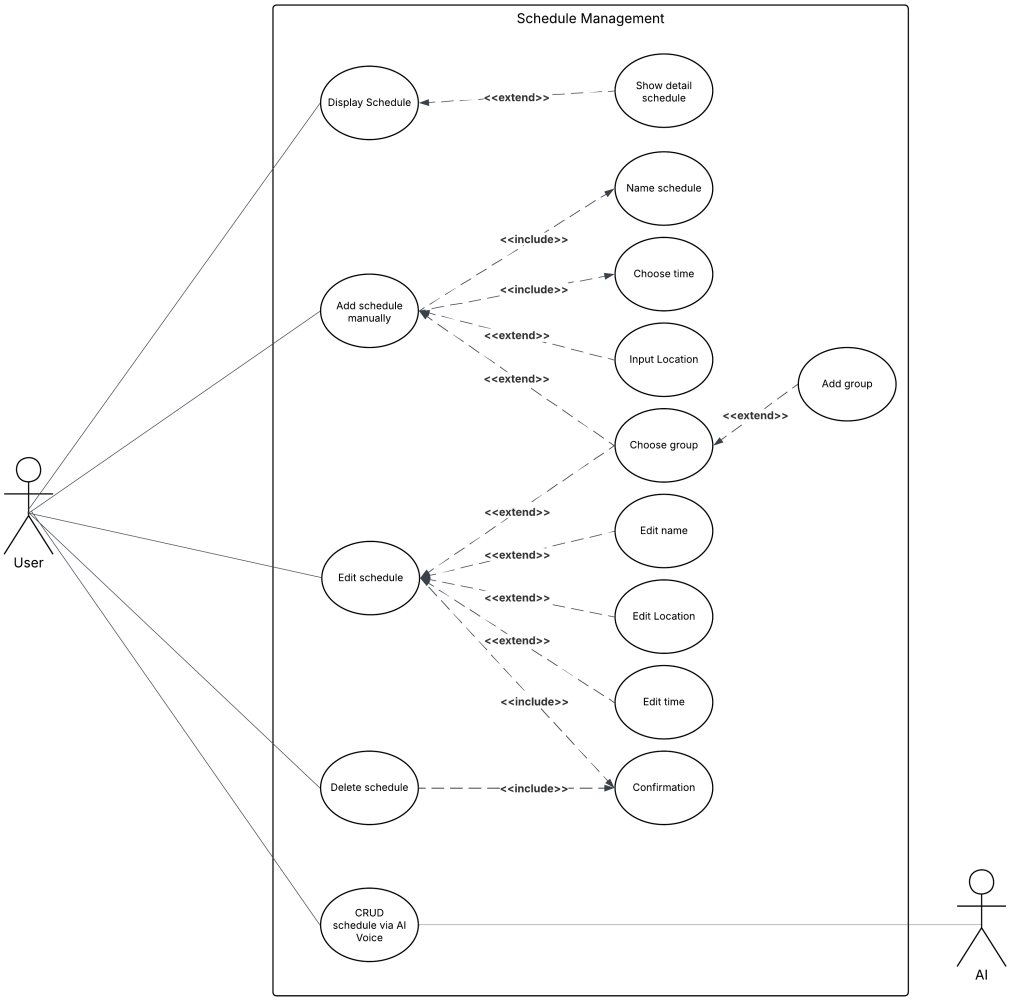

Figure 3.4: Schedule Management Use-case Diagram

Figure 3.4 illustrates the use case diagram of Schedule Management. This component is one of the most important features of the application. A schedule is a part of our life, we use a schedule on a daily basis to manage what works. Moreover, with this feature, users can create, edit, delete, or retrieve schedules.

Users can also rapidly set, delete, get, or edit the schedules with the voice button. For example, a user can just say to the voice button, "Set the schedule for my football game from 7 pm to 9 pm tonight". The system will set the schedule based on the given information and display the

notification back to the user.

Use-case Description

Use Case ID	UC-09
Use Case Name	Add Schedule
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User adds a new schedule event to the system.
Trigger	User clicks the "Add Schedule" button (+ icon).
Preconditions	PRE-1. User is logged in. PRE-2. User is on the Schedule tab.
Postconditions	POST-1. New schedule is saved and displayed on the timetable.
Normal Flow	1. User clicks the "+" button. 2. System displays entry form (overlay). 3. User enters details (Name, Start/End Time, Location, Category). 4. User clicks "Save/Confirm". 5. System validates data and saves schedule. 6. System closes form and updates the timetable view.
Alternative Flow	Alternative at step 3: Category does not exist. 3.a.1 User selects "Add New Category". 3.a.2 System creates category and assigns it. Continue at step 3. Alternative at step 6: Schedule conflict detected. 6.a.1 System saves the schedule. 6.a.2 System renders the new event overlapping the existing one. Use-case ends here.
Exceptions	Exception at step 5: API/Save Error. 5.a.1 System displays error message. 5.a.2 System keeps form open with user data retained. Continue at step 4. Exception at step 6: Timezone mismatch. 6.b.1 System auto-converts time to local device time before displaying. Use-case ends here.

Table 3.14: Use Case UC-09: Add Schedule

Table 3.14 is the description table of the use case Add Schedule. Through this use case, user can add a new schedule. If everything is OK, a new schedule will be added. On the opposite side, there is no schedule added, and the error messages will be illustrated.

Use Case ID	UC-08
Use Case Name	Display Schedule
Primary Actor(s)	User **Secondary Actor(s)** None
Description	System displays the user's schedule and allows view navigation.
Trigger	User selects the "Schedule" tab.
Preconditions	PRE-1. User is logged in.
Postconditions	POST-1. Schedule is displayed correctly in the user's local time.
Normal Flow	1. User selects the "Schedule" tab. 2. System retrieves data and displays schedule (Default: Week view). 3. User navigates (swipes left/right) or changes view mode (Day/-Month). 4. System updates the display accordingly.
Alternative Flow	Alternative at step 2: No schedule data exists. 2.a.1 System displays an empty timetable grid. Use-case ends here. Alternative at step 3: User clicks a specific event. 3.a.1 System opens a detailed pop-up for the event. 3.a.2 User views or selects option to edit. Continue at step 4.
Exceptions	Exception at step 2: Data retrieval error. 2.b.1 System displays error: "Unable to load schedule." Use-case ends here. Exception at step 2: Timezone mismatch detected. 2.c.1 System auto-converts event times to device's local time. Continue at step 2.

Table 3.15: Use Case UC-08: Display Schedule

Table 3.15 is the description table of the use case Display Schedule. With this use case, user can retrieve a list of schedules. If the list of schedules is not empty, it will be illustrated to the user. However, if anything goes wrong, the error message will be displayed to the user.

Use Case ID	UC-10
Use Case Name	Edit Schedule
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User updates an existing schedule event.
Trigger	User taps a specific schedule event.
Preconditions	PRE-1. User is logged in and viewing the Schedule tab.
Postconditions	POST-1. Schedule is updated and the view is refreshed.
Normal Flow	1. User taps the event to edit. 2. System opens form with current details. 3. User modifies details (Time, Location, etc.). 4. User taps "Save". 5. System updates database and refreshes view.
Alternative Flow	Alternative at step 4: Event is recurring. 4.a.1 System prompts: "Edit this instance" or "Edit all"? 4.a.2 User selects option; System updates accordingly. Continue at step 5. Alternative at step 3: Category missing. 3.a.1 User creates new category in-place. Continue at step 3.
Exceptions	Exception at step 5: Save failed. 5.a.1 System alerts error; form remains open. Continue at step 4. Exception at step 2: Timezone mismatch. 2.a.1 System auto-converts to local time. Continue at step 3.

Table 3.16: Use Case UC-10: Edit Schedule

Table 3.16 is the description table of the use case Edit Schedule. Within this use case, user can edit the existing schedules. If nothing goes wrong, the edited schedules will be updated. Nevertheless, if nothing goes correctly, the error message will be displayed.

Use Case ID	UC-11
Use Case Name	Remove Schedule
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User deletes a schedule event from the system.
Trigger	User taps a specific schedule event.
Preconditions	PRE-1. User is logged in and viewing the Schedule tab.
Postconditions	POST-1. Schedule is removed and the timetable is updated.
Normal Flow	1. User taps the event to delete. 2. User taps "Delete" button. 3. User confirms deletion. 4. System removes event and updates view.
Alternative Flow	Alternative at step 2: Event is recurring. 2.a.1 System prompts: "Delete this instance" or "Delete all"? 2.a.2 User selects option; System deletes accordingly. Use-case ends here. Alternative at step 3: User cancels. 3.a.1 System closes confirmation dialog. Use-case ends here.
Exceptions	Exception at step 4: Deletion failed (API Error). 4.a.1 System displays error; popup remains open. Continue at step 2. Exception at step 1: Timezone mismatch. 1.a.1 System auto-converts to local time before display. Continue at step 2.

Table 3.17: Use Case UC-11: Remove Schedule

Table 3.17 is the description table of the use case Remove Schedule. Within this use case, user can remove the existing schedules. If the chosen schedules exist, user may remove them successfully. However, if something goes wrong, the error will be demonstrated.

Use Case ID	UC-12
Use Case Name	Add Schedule via Voice
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User creates a schedule event using voice commands.
Trigger	User taps the "Voice" button.
Preconditions	PRE-1. User is logged in. PRE-2. Internet connection is active.
Postconditions	POST-1. Schedule is saved and displayed.
Normal Flow	1. User taps "Voice" and speaks schedule details. 2. System processes input and requests confirmation. 3. User confirms details. 4. System saves the schedule.
Alternative Flow	Alternative at step 1: Permission missing. 1.a.1 System requests microphone access. 1.a.2 If granted, proceed to step 1. Alternative at step 2: Exact duplicate detected. 2.a.1 AI warns of duplicate; asks for explicit confirmation. 2.a.2 User confirms. Continue at step 4. Alternative at step 4: Time overlap or Missing Category. 4.a.1 System allows overlap or auto-creates category. 4.a.2 System saves schedule.
Exceptions	Exception at step 2: AI/Connection failure. 2.b.1 System displays error message. Use-case ends here. Exception at step 2: Timezone mismatch. 2.c.1 System auto-converts to local time. Continue at step 2.

Table 3.18: Use Case UC-12: Add Schedule via Voice

Table 3.18 is the description table of the use case Add Schedule via Voice. Through this use case, user can add a new schedule with their voice. If the system records user's voice and analyzes it successfully, the confirmation will be delivered to the user. If the user confirms, a new schedule will be added. On the opposite side, the error message will be displayed.

Use Case ID	UC-17
Use Case Name	Display Schedule via Voice
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User retrieves their schedule using voice commands.
Trigger	User taps the "Voice" button.
Preconditions	PRE-1. User is logged in. PRE-2. Internet connection is active.
Postconditions	POST-1. The requested schedule is displayed.
Normal Flow	1. User taps "Voice" button. 2. User speaks display command. 3. System verifies command and queries database. 4. System displays the schedule.
Alternative Flow	Alternative at step 3: Schedule is empty. 3.a.1 System notifies user of empty schedule. Use-case ends here.
Exceptions	Exception at step 3: AI Service/API Error. 3.b.1 System displays error response. Use-case ends here. Exception at step 4: Timezone mismatch. 4.a.1 System auto-converts to local time. Continue at step 4.

Table 3.19: Use Case UC-17: Display Schedule via Voice

Table 3.19 is the description table of the use case Display Schedule via Voice. Within this use case, user can retrieve a list of schedules by using their voice. If nothing goes wrong, the list of schedules will be displayed. Nevertheless, if anything goes wrong, the error will be displayed.

Use Case ID	UC-14
Use Case Name	Edit Schedule via Voice
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User updates an existing schedule event using voice commands.
Trigger	User taps the "Voice" button.
Preconditions	PRE-1. User is logged in. PRE-2. Internet connection is active.
Postconditions	POST-1. Schedule is updated with confirmed details.
Normal Flow	1. User taps "Voice" and speaks schedule details to edit. 2. System processes input and requests confirmation. 3. User confirms changes. 4. System updates the schedule.
Alternative Flow	Alternative at step 1: Permission missing. 1.a.1 System requests microphone access. 1.a.2 If granted, proceed to step 1. Alternative at step 2: Duplicate detected. 2.a.1 AI warns of duplicate; asks for confirmation. 2.a.2 User confirms. Continue at step 4. Alternative at step 4: Overlap or Missing Category. 4.a.1 System allows overlap or auto-creates category. 4.a.2 System saves schedule.
Exceptions	Exception at step 2: AI Service/API Error. 2.b.1 System displays error message. Use-case ends here. Exception at step 2: Timezone mismatch. 2.c.1 System auto-converts to local time. Continue at step 2.

Table 3.20: Use Case UC-14: Edit Schedule via Voice

Table 3.20 is the description table of the use case Edit Schedule via Voice. Through this use case, user can update schedules' details using their voice. If the system records the voice and analyzes it successfully, the system will send a confirmation message to the user. If the user agrees, the schedules will be updated. However, in unexpected situations, the error message will be demonstrated.

Use Case ID	UC-13
Use Case Name	Remove Schedule via Voice
Primary Actor(s)	User **Secondary Actor(s)** None
Description	User deletes a schedule event using voice commands.
Trigger	User taps the "Voice" button.
Preconditions	PRE-1. User is logged in. PRE-2. Internet connection is active.
Postconditions	POST-1. Schedule is removed from the database.
Normal Flow	1. User taps "Voice" and speaks request to delete schedule. 2. System identifies event and requests confirmation. 3. User confirms deletion. 4. System removes the schedule.
Alternative Flow	Alternative at step 1: Permission missing. 1.a.1 System requests microphone access. 1.a.2 If granted, proceed to step 1. Alternative at step 2: Event is recurring. 2.a.1 AI asks: "Delete this instance or all occurrences?" 2.a.2 User answers; System processes deletion accordingly. Use-case ends here.
Exceptions	Exception at step 2: AI Service/API Error. 2.b.1 System displays error message. Use-case ends here. Exception at step 2: Timezone mismatch during identification. 2.c.1 System auto-converts to local time for confirmation. Continue at step 2.

Table 3.21: Use Case UC-13: Remove Schedule via Voice

Table 3.21 is the description table of the use case Remove Task via Voice. With this use case, user can delete the existing schedules. If nothing goes wrong, the schedules will be deleted. Nevertheless, if anything goes wrong, the schedules will not be deleted, and the error message will be displayed.

3.4 Architecture Design

3.4.1 Architecture Strategy and Pattern Selection

Software Architecture delineates the high-level structure of a system, defining its components, their interrelationships, and the governing rules for their interaction. Several notable architectural styles are relevant to this project:

Client-Server Architecture: This fundamental style segregates system functionality into two distinct partitions: resource providers (servers) and service requesters (clients) [8].

This architecture offers several significant advantages. First, it enables centralization of control, security policies, and data storage, thereby ensuring data integrity and simplified updates. Second, it provides enhanced scalability, as server resources can be scaled vertically or horizontally independently of the client hardware. Third, it ensures broad accessibility, allowing clients to access server resources from diverse locations and platforms without platform-specific constraints.

However, the architecture also presents notable disadvantages. The most critical concern is the single point of failure, where server downtime renders the entire application unavailable to clients. Additionally, the architecture exhibits strong network dependency, with performance strictly bound by network latency and bandwidth, meaning poor connectivity directly degrades user experience. Furthermore, managing centralized servers requires dedicated infrastructure and maintenance efforts, resulting in considerable maintenance overhead.

Layered Architecture: Also known as N-tier architecture, this pattern organizes the system into horizontal layers, where each layer has a specific role (e.g., presentation, business logic, data access). It is widely adopted for its clear separation of concerns [8].

The primary advantages of this approach include effective separation of concerns, wherein each layer focuses on a specific aspect, making the system easier to develop and maintain. The architecture also enhances testability, as individual layers can be mocked and tested in isolation. Moreover, it provides abstraction benefits, ensuring that changes in one layer, such as switching databases, rarely impact other layers like the user interface.

Despite these benefits, layered architecture introduces certain drawbacks. Performance overhead becomes apparent as data must travel through multiple layers to reach the database, which can introduce latency and lead to the "architectural sinkhole" anti-pattern. Additionally, tight coupling often occurs in practice, as changes to business rules frequently require cascading changes across both the database and UI layers.

Event-Driven Architecture (EDA): EDA is a distributed, asynchronous style used to build highly scalable applications. It relies on the production, detection, and consumption of events, allowing components to react to state changes in real-time without direct coupling [8].

This architectural style provides several key advantages. It enables strong decoupling, as producers and consumers do not need to know about each other, allowing independent development and evolution. The architecture supports asynchronous processing, ensuring that heavy tasks such as speech processing do not block the main execution flow, thereby improving perceived performance. Furthermore, it offers excellent scalability, as event consumers can be scaled up

dynamically to handle bursts in traffic.

However, EDA introduces its own challenges. The complexity of debugging and tracing flows through an asynchronous system is significantly harder than in linear systems. Additionally, ensuring data consistency across services becomes challenging due to eventual consistency, as updates may not happen instantly everywhere.

Microservices Architecture: This style structures an application as a collection of loosely coupled, independently deployable services. Each service is organized around a specific business capability (Bounded Context) and communicates via lightweight protocols [9].

The advantages of microservices architecture are substantial. It provides fault isolation, ensuring that a failure in one service, such as Task Management, does not necessarily crash the entire system. The architecture enables technology heterogeneity, allowing different services to be written in different languages or use different databases best suited for their specific task. Moreover, it promotes agility through smaller codebases that allow for faster deployment cycles and easier understanding for new developers.

Conversely, microservices architecture presents significant challenges. It requires robust DevOps culture to manage the operational complexity of deployment, monitoring, and orchestration of many services. Additionally, implementing transactions that span multiple services, known as distributed transactions, is complex and difficult to manage, creating data consistency challenges.

Hybrid Architecture Strategy: **Figure 3.5** illustrates the architecture of this project. To optimize system performance and scalability, this project adopts a **Hybrid Architecture** that combines multiple architectural styles. At the highest level, the system employs Client-Server architecture to separate the user interface (Mobile App) from the backend logic, ensuring clear boundaries between the client and server. The server-side is decomposed into independent modules using a Microservices architecture, allowing for autonomous deployment and scaling. Internally, each microservice utilizes a Layered Architecture to organize the codebase, ensuring separation of concerns between business logic and data access. Finally, an Event-Driven architecture is employed to handle asynchronous processes and integrate with third-party services, such as AI models, without blocking the main system flow.

The three core modules—**Authentication and Personal Information**, **Task Management**, and **Schedule Management**—adhere to a uniform internal architecture. To ensure separation of concerns, each module is structured into four distinct layers. The Presentation Layer manages the interface for client interactions, including API endpoints, URL routing, and Views. The Service Layer encapsulates the core business logic and functional requirements provided by the module. The Data Access Layer handles data persistence logic through abstractions such as Models and Repositories. Finally, the Database Layer consists of a dedicated, isolated database for each module to ensure data autonomy.

The **Speech-to-Schedule** service operates using an Event-Driven Architecture (EDA) to ensure efficiency. When a user utilizes this service, the system analyzes the input and extracts the necessary entities, such as time and intent. Subsequently, the data is routed to either **Schedule Management** or **Task Management** for processing. Finally, the system triggers a notification to confirm the action to the user.

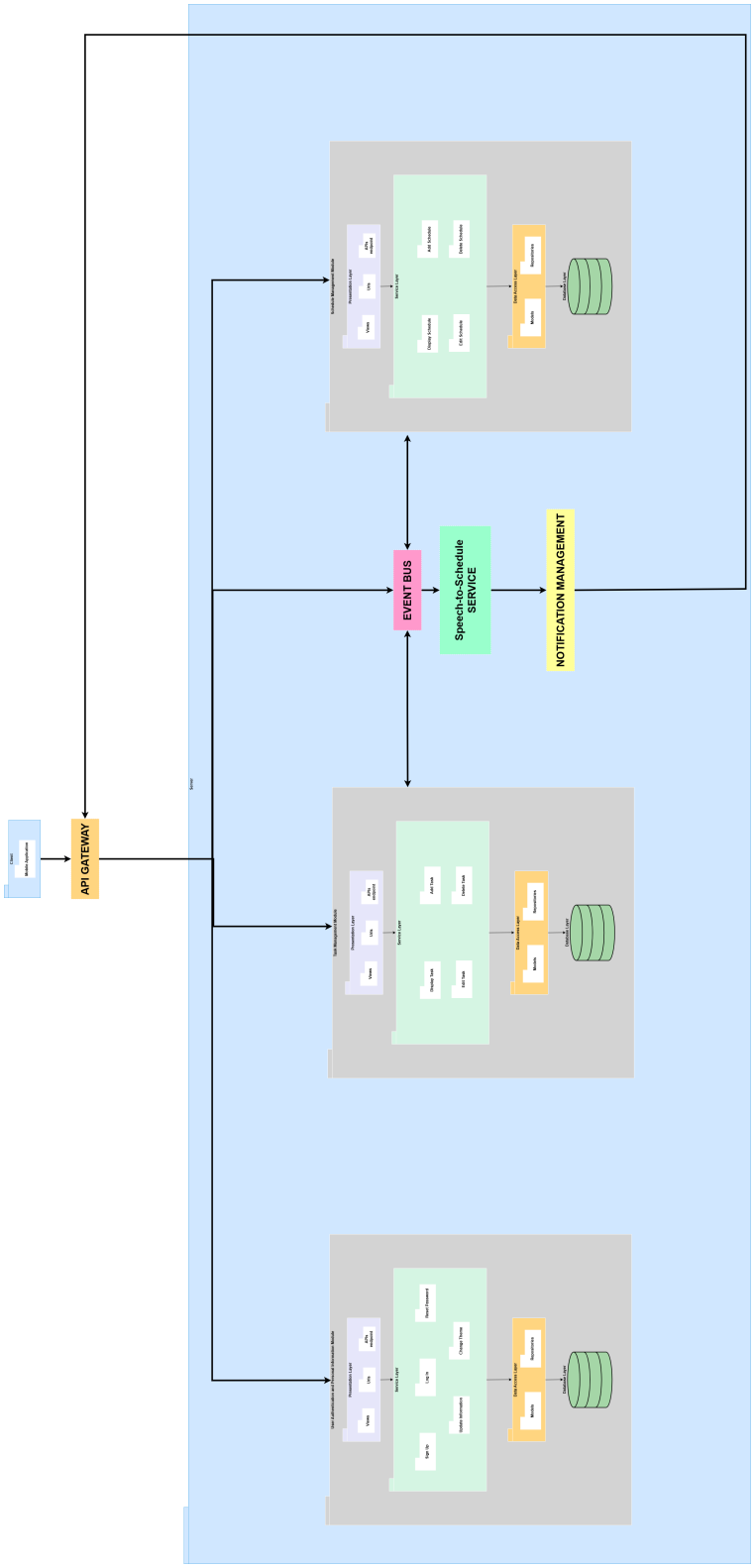

Figure 3.5: Architecture Diagram

3.4.2 Logical View

3.4.2.1 Sequence Diagram

Authentication and Personal Information

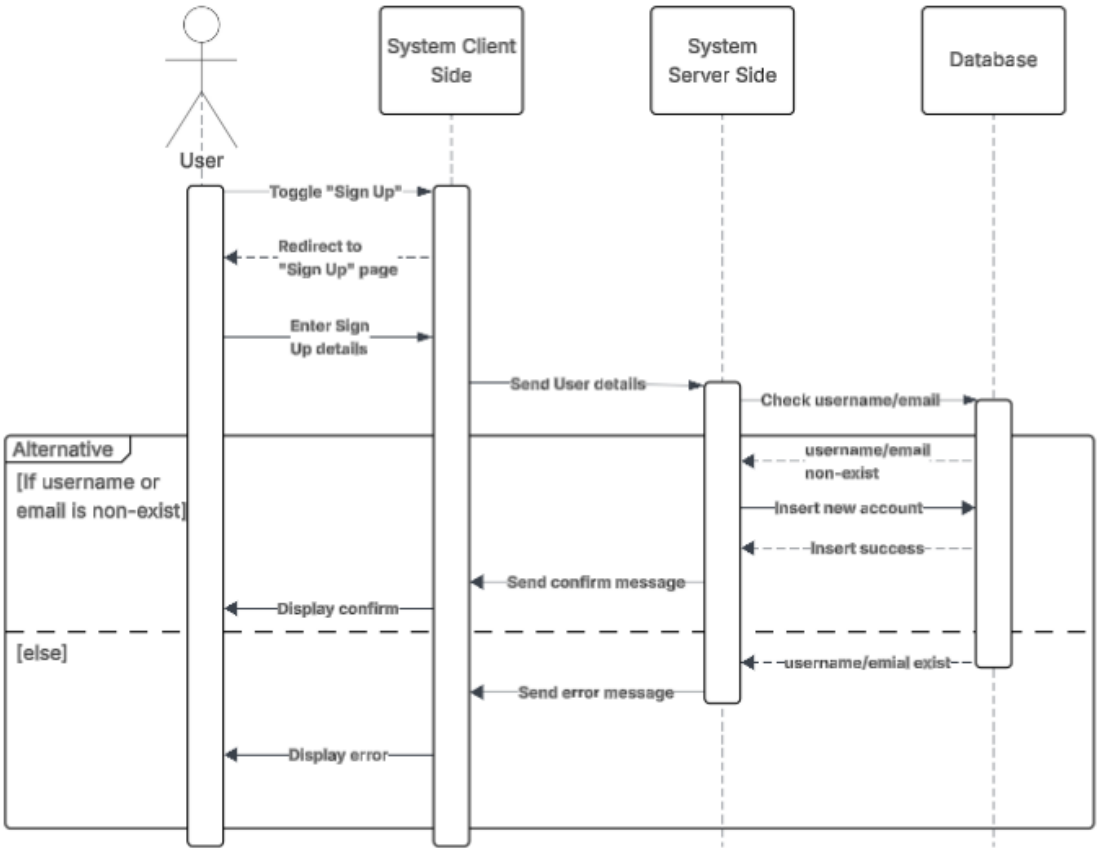

Figure 3.6: Sign Up Sequence Diagram

Figure 3.6 illustrates the interactions between the user and the system. It starts from the user choosing the Sign Up option and is redirected to the Sign Up page. The user is then prompted to enter credentials. After checking credentials, if everything goes correctly, the new account will be created and display a successful message to the user. On the opposite side, the error will be displayed to the user.

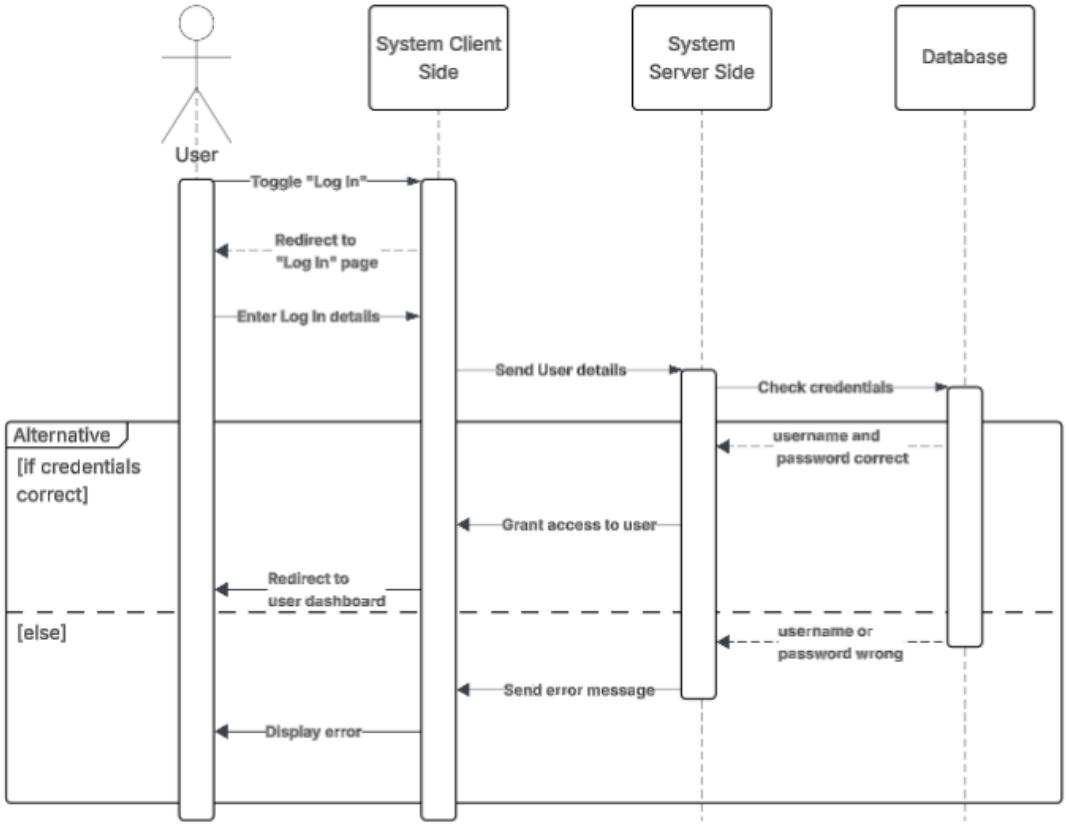

Figure 3.7: Log In Sequence Diagram

Figure 3.7 illustrates the interactions between the user and the system. It starts from the user choosing the Log In option and is redirected to the Log In page. The user is asked to enter the needed details. If the credentials are correct, the user will be redirected to the home page. Otherwise, the user will be notified about the error.

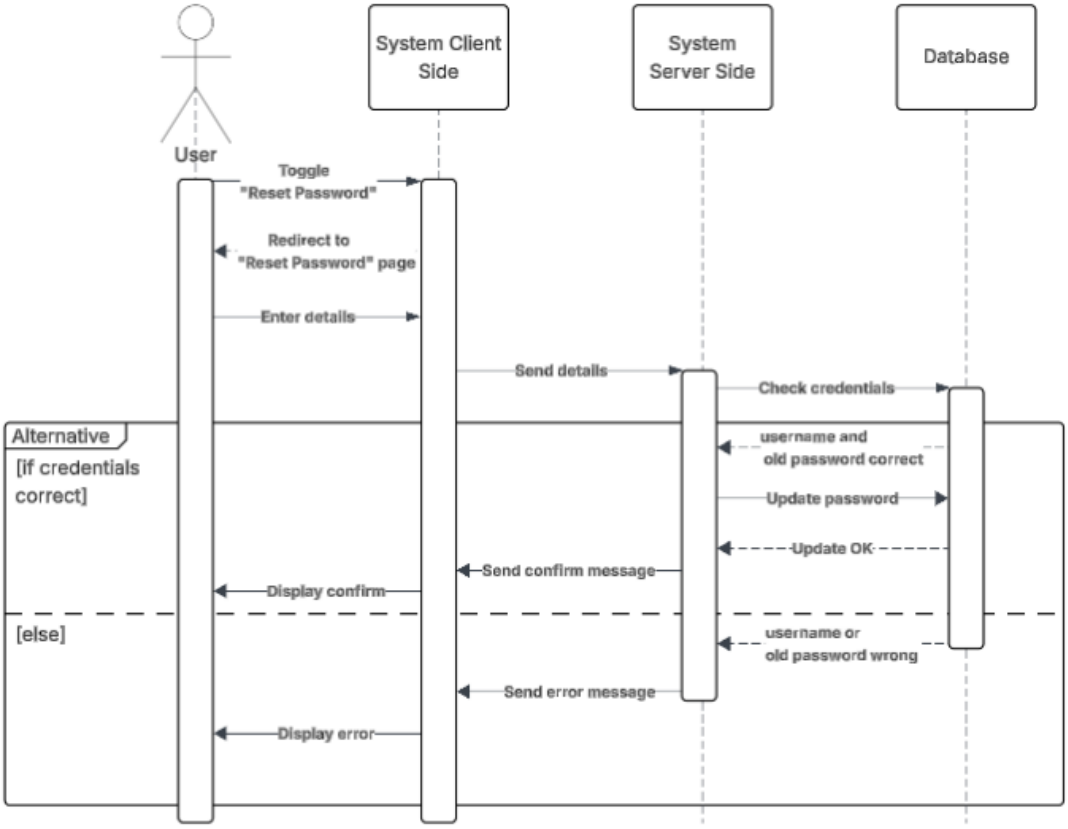

Figure 3.8: Reset Password Sequence Diagram

Figure 3.8 illustrates the interactions between the user and the system. Starting from when the user toggles the Reset Password option, they will be redirected to the reset password page. The user will be asked to enter the needed details. If everything goes correctly, the user's password will be updated; otherwise, an error will be displayed.

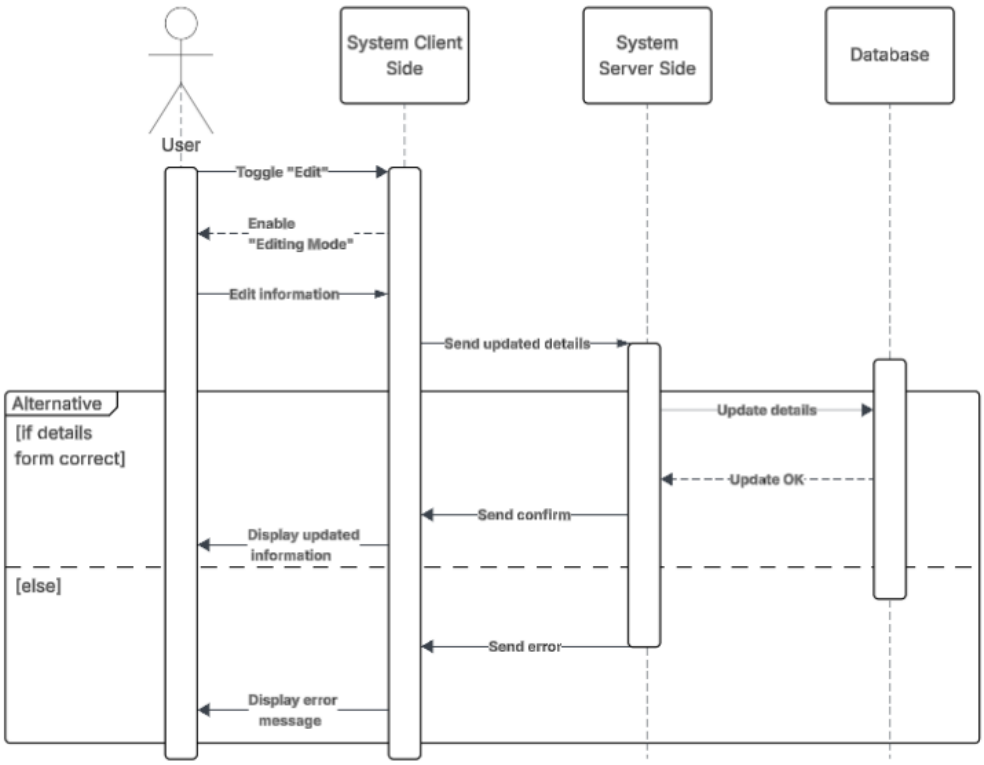

Figure 3.9: Update Personal Information Sequence Diagram

Figure 3.9 depicts the interactions between the user and the system. The user who chooses the Edit option will be redirected to the page edit. The user updates the new details. If nothing is wrong, the user details will be updated. On the opposite side, the error will be displayed.

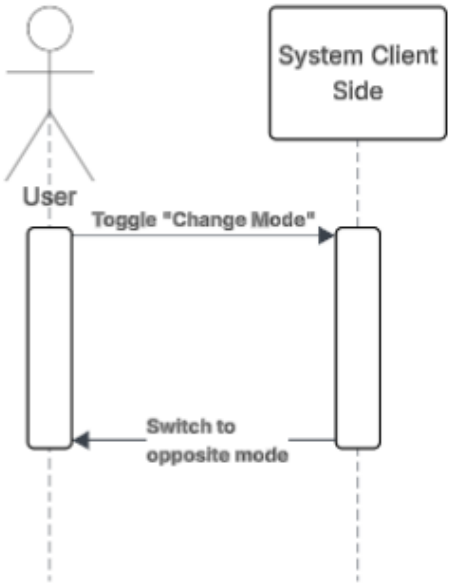

Figure 3.10: Change Theme Sequence Diagram

Figure 3.10 illustrates how the user interacts with the client side of the system to change the application theme. The user merely needs to choose the Change Mode button, and the theme will be changed to the opposite theme of the current one.

Task Management

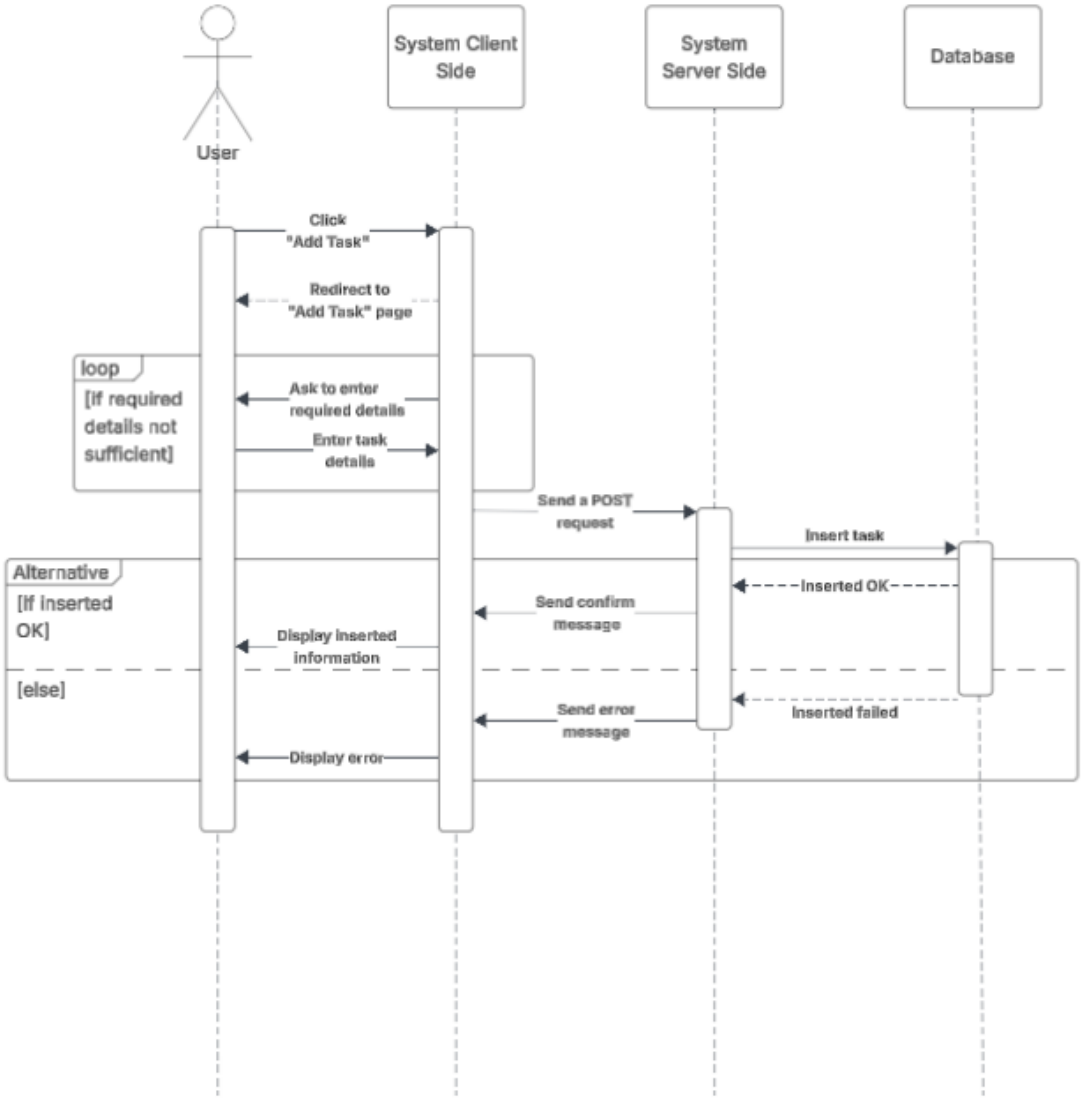

Figure 3.11: Add Task Sequence Diagram

Figure 3.11 depicts the flow of interactions between the user and the system to add a new task. The user chooses the Add Task option, and the Add Task page will appear. The user enters all details. If required details are missed, the system will ask the user to input the needed information. Then, the system will insert a new task. If nothing interrupts, the task is inserted successfully. Otherwise, the error will be displayed.

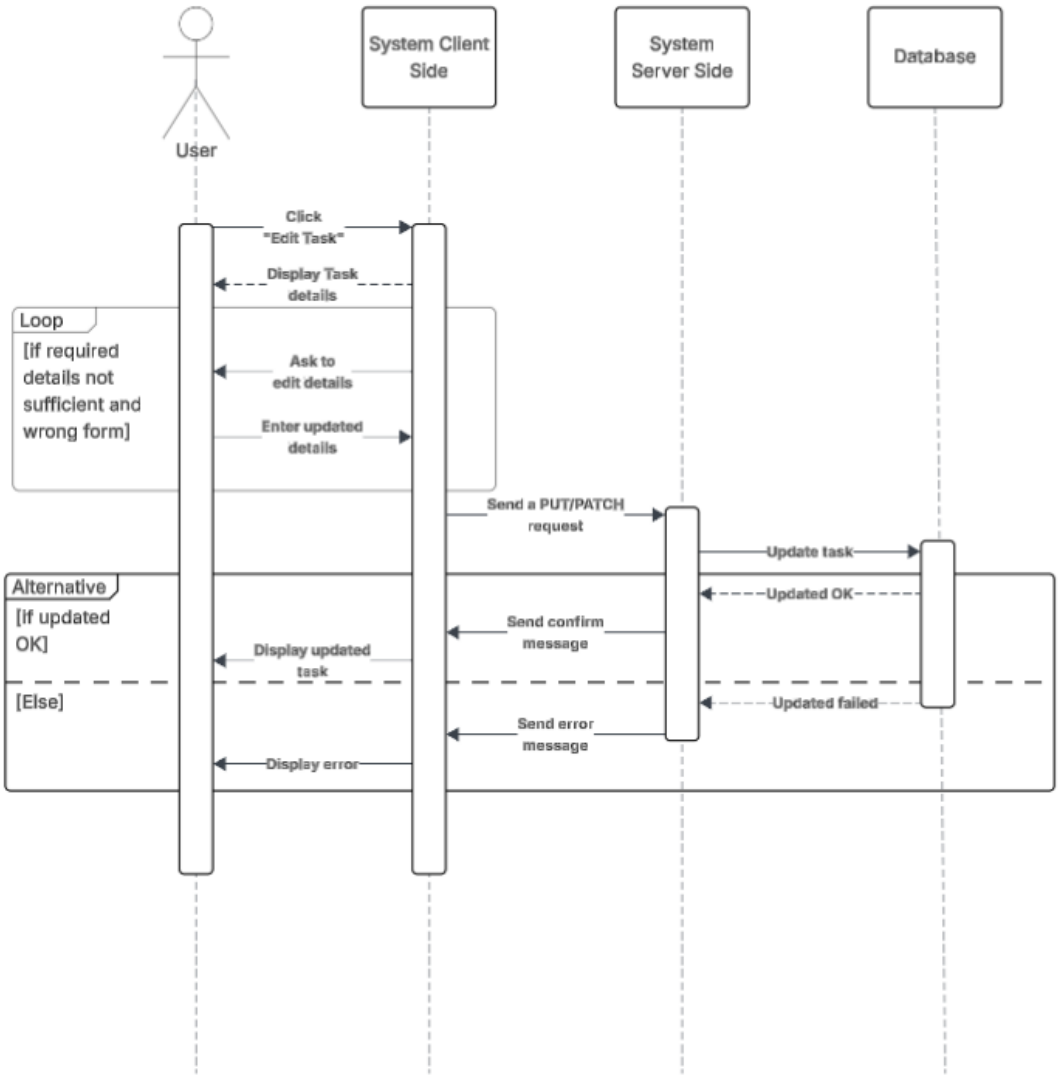

Figure 3.12: Edit Task Sequence Diagram

Figure 3.12 illustrates the interactions between the user and the system to edit a task. Starting since the user chooses the Edit Task option and is redirected to the Edit page. Then, the user is asked to enter the edited details. If the required details are missed, they will need to add the required information. If no errors, the task details are updated. Otherwise, the error will be displayed.

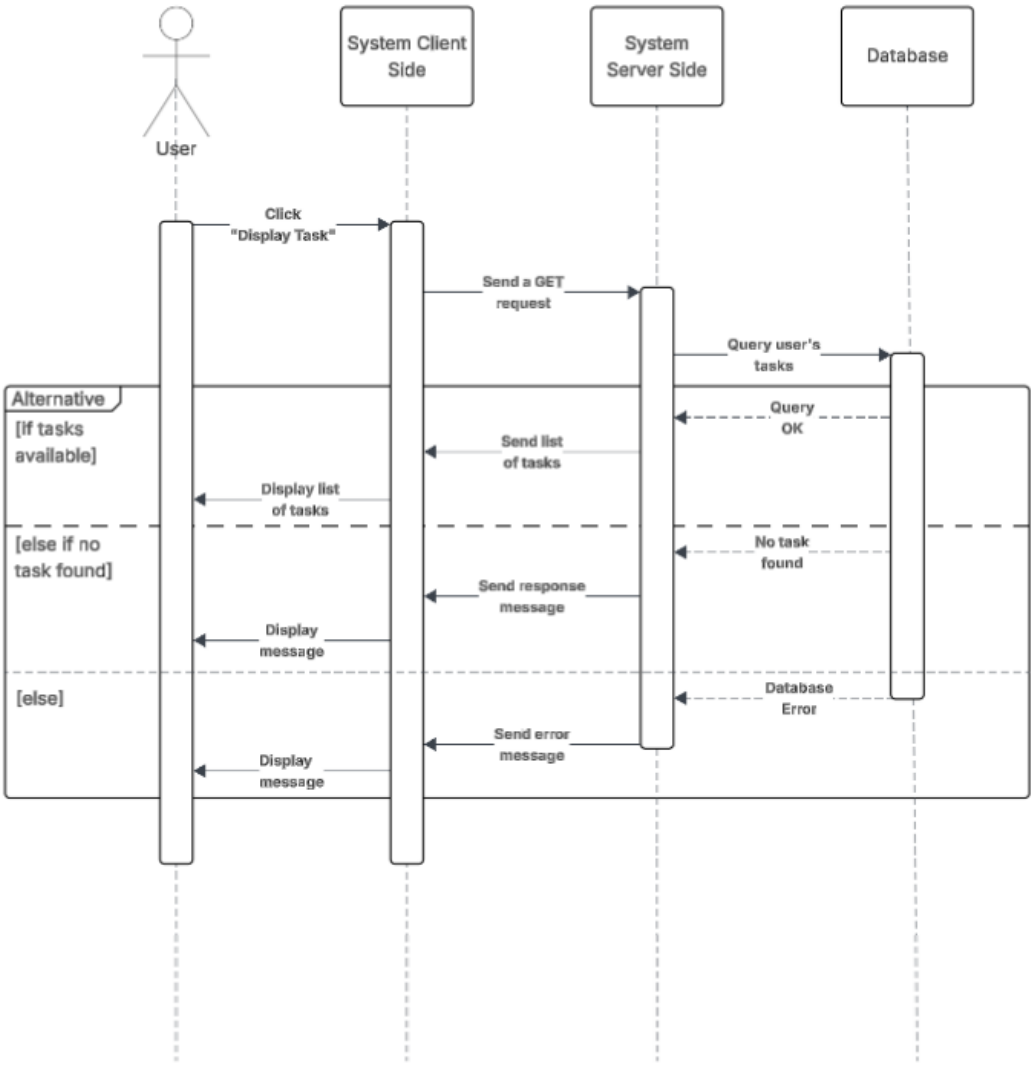

Figure 3.13: Display Task Sequence Diagram

Figure 3.13 illustrates the process of retrieving the list of tasks. The user chooses the Display Task option, and the system asks the database to retrieve the list of tasks. If the list of tasks is not empty and no errors, the list of tasks will be retrieved and displayed to the user. Otherwise, the errors will be displayed.

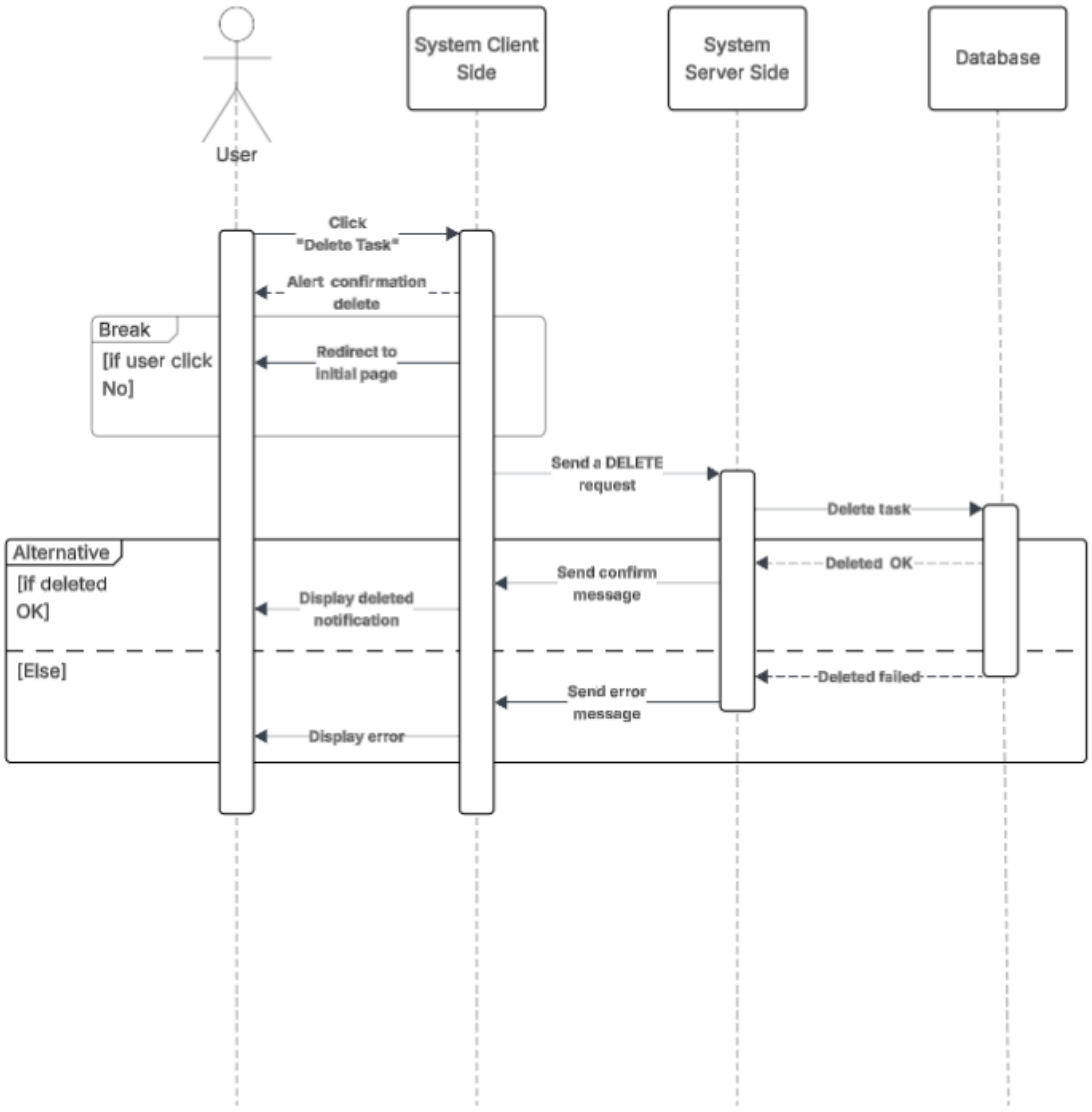

Figure 3.14: Remove Task Sequence Diagram

Figure 3.14 depicts the flow of interactions between the user and the system to remove a task. Starting from the user choosing the delete option, the system alerts the user confirmation message. If the user chooses the cancel or reject option, the flow will be terminated. However, if the user chooses the remaining option, this task will be eliminated. If there is an error, the message will be displayed.

Using Voice button

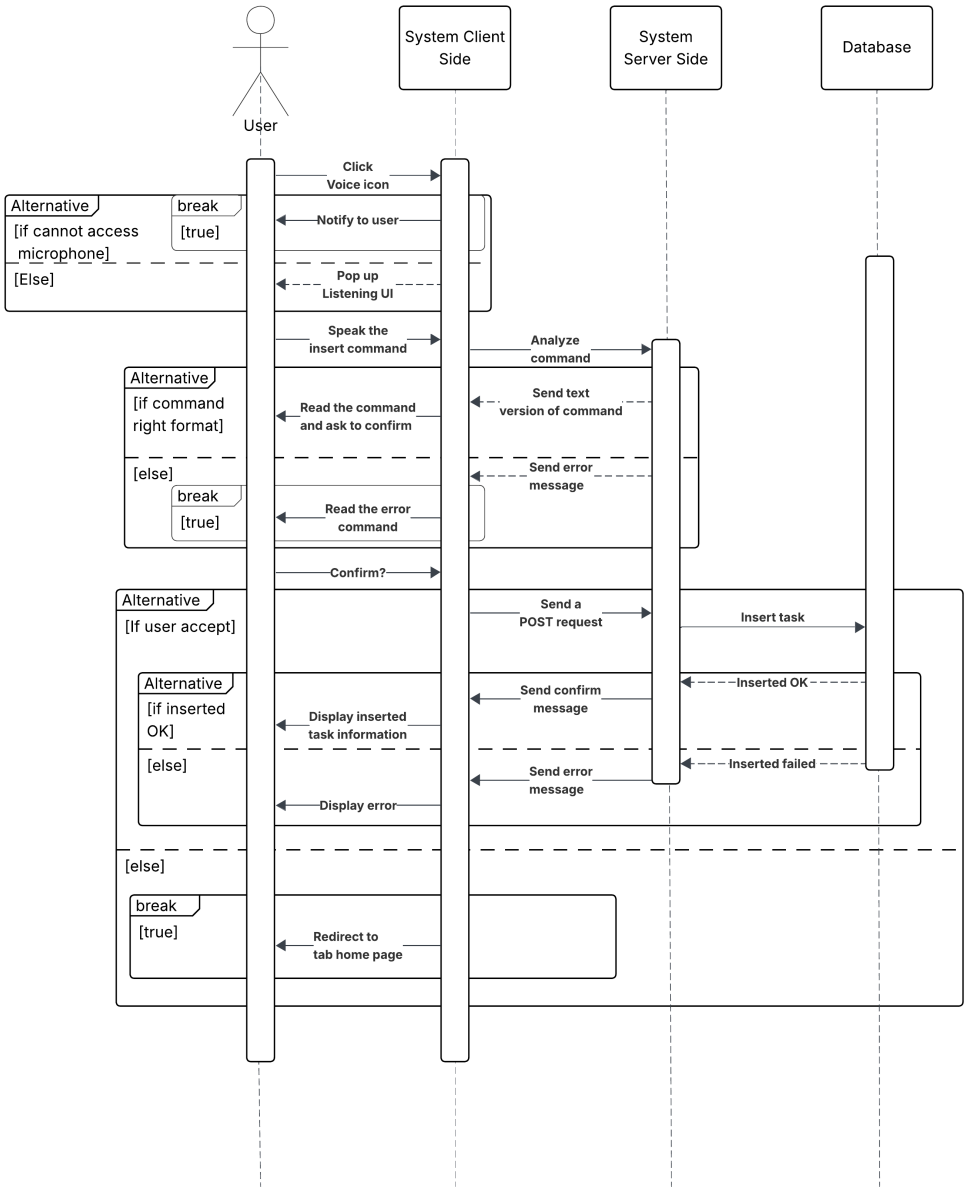

Figure 3.15: Add Task via Voice Sequence Diagram

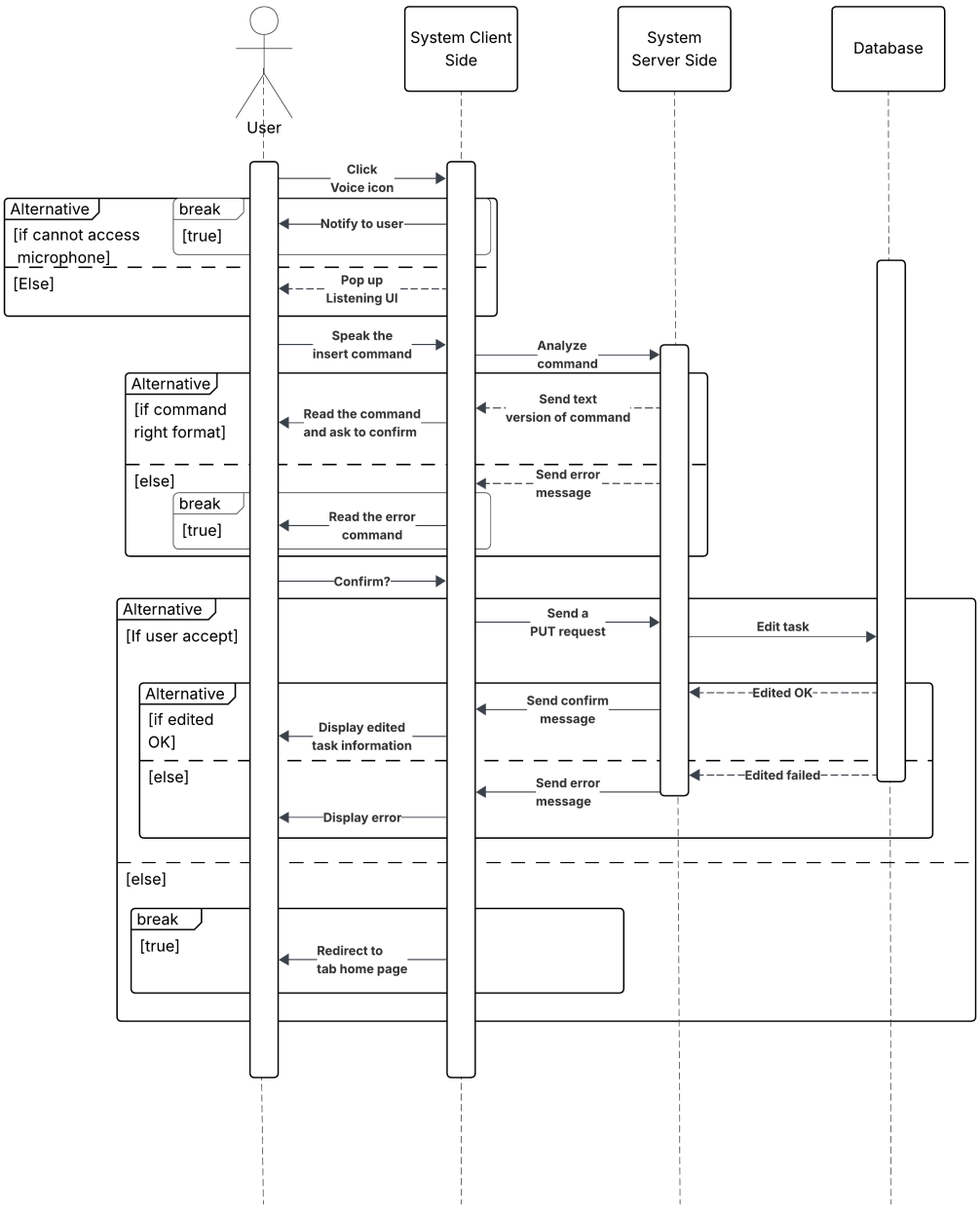

Figure 3.16: Edit Task via Voice Sequence Diagram

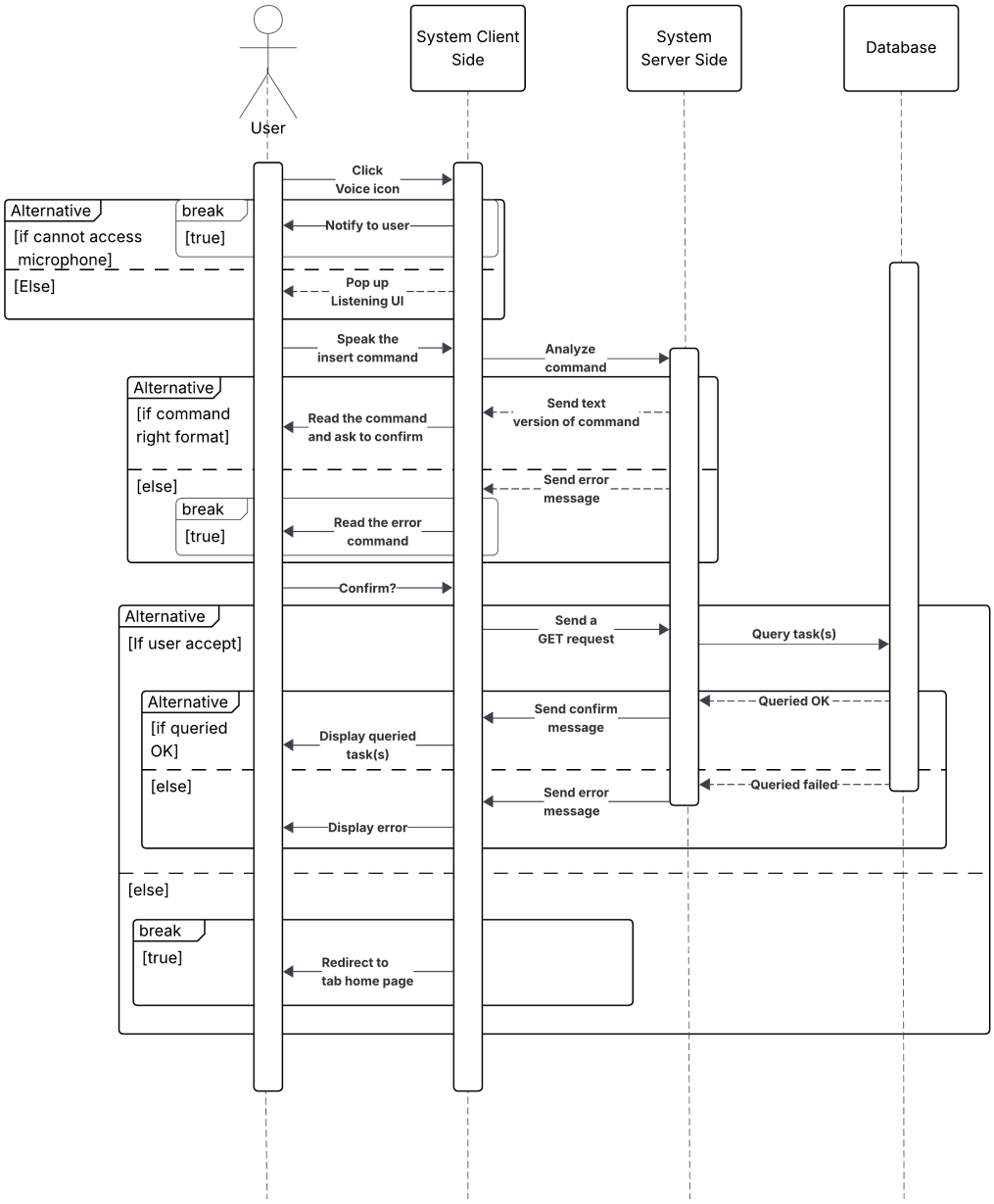

Figure 3.17: Display Task via Voice Sequence Diagram

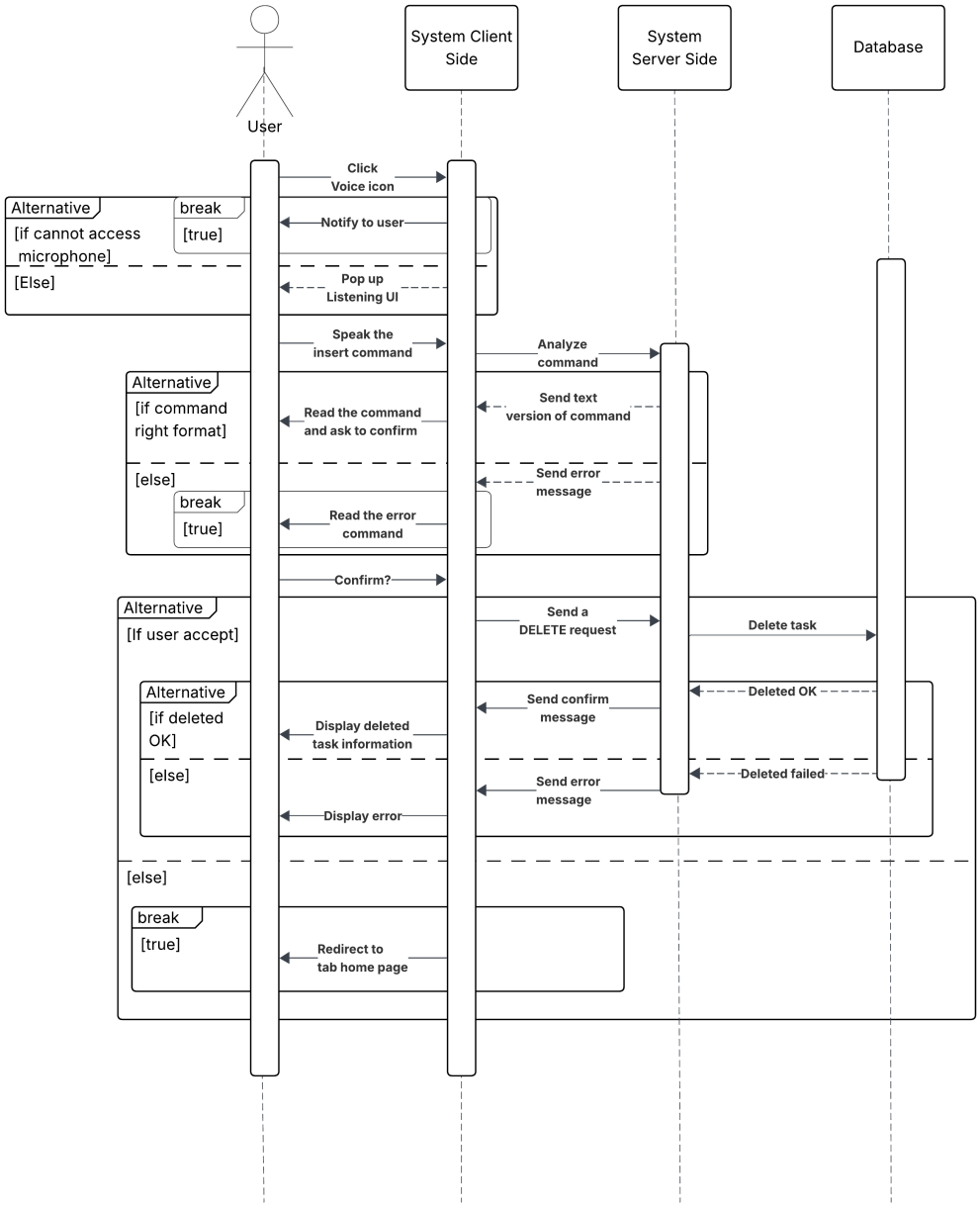

Figure 3.18: Remove Task via Voice Sequence Diagram

These sequence diagrams depict the interactions between the user and the system to use the voice option in task management. After clicking the voice button, if the system cannot access the device's microphone, the process will be terminated. Then, the user will speak the request to the system. The system analyzes the speech. If the system analyzes the speech successfully,

based on the purpose of the speech, the actions will be executed. If an unexpected error occurs, the system will display. Otherwise, the system executes the actions.

Figure 3.15 displays the add task using voice button. **Figure 3.16** displays the edit task using voice button. **Figure 3.17** illustrates the display task with voice button. Finally, **Figure 3.18** demonstrates the remove task with voice button.

Schedule Management

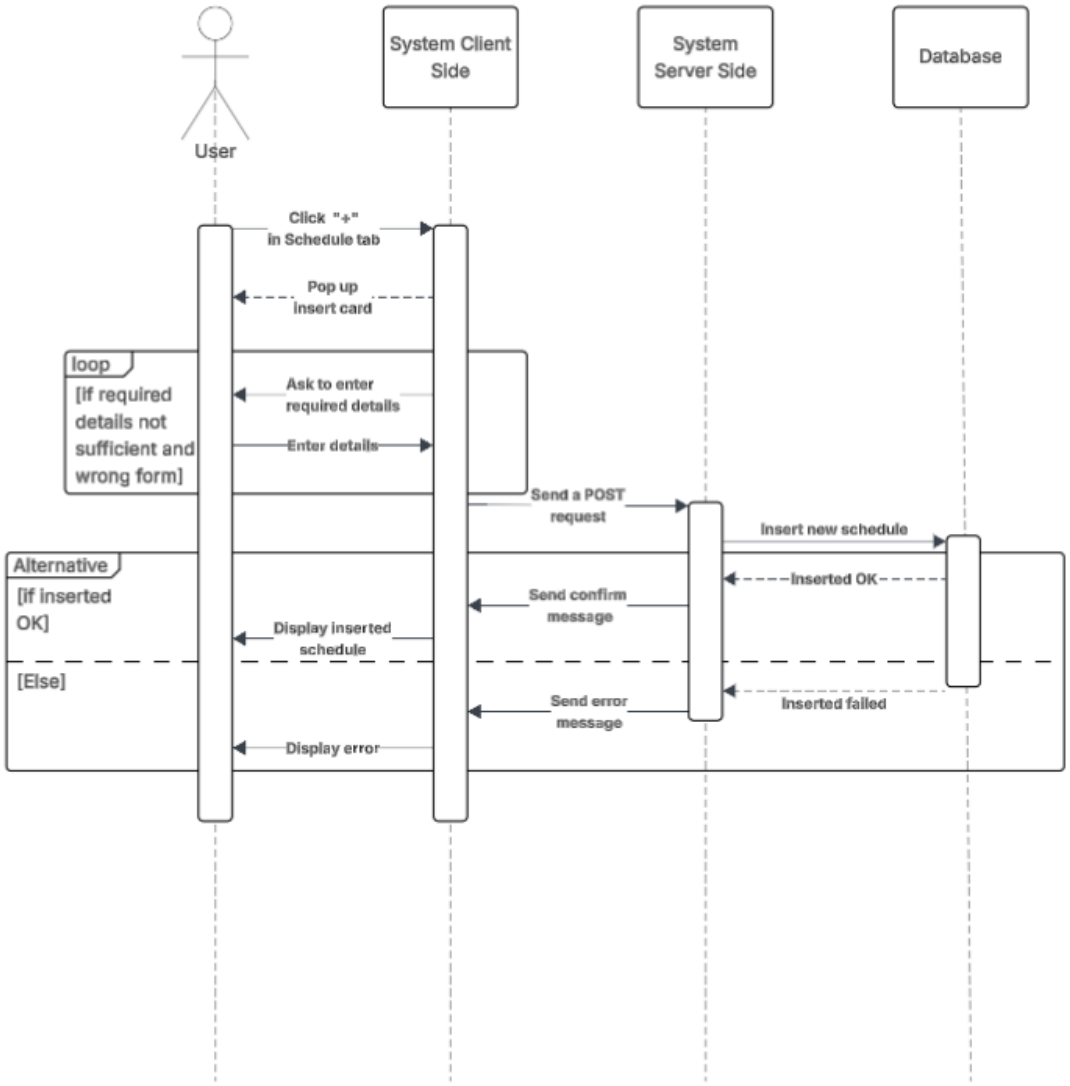

Figure 3.19: Add Schedule Sequence Diagram

Figure 3.19 illustrates the interaction between the user and the system to add a new schedule. First, the user chooses the option add new schedule, then the insert UI will appear. Then, the user enters the needed details. If the required details are empty or the details are wrong format, the user needs to enter them again. If no errors, a new schedule will be added. Otherwise, the error will be illustrated.

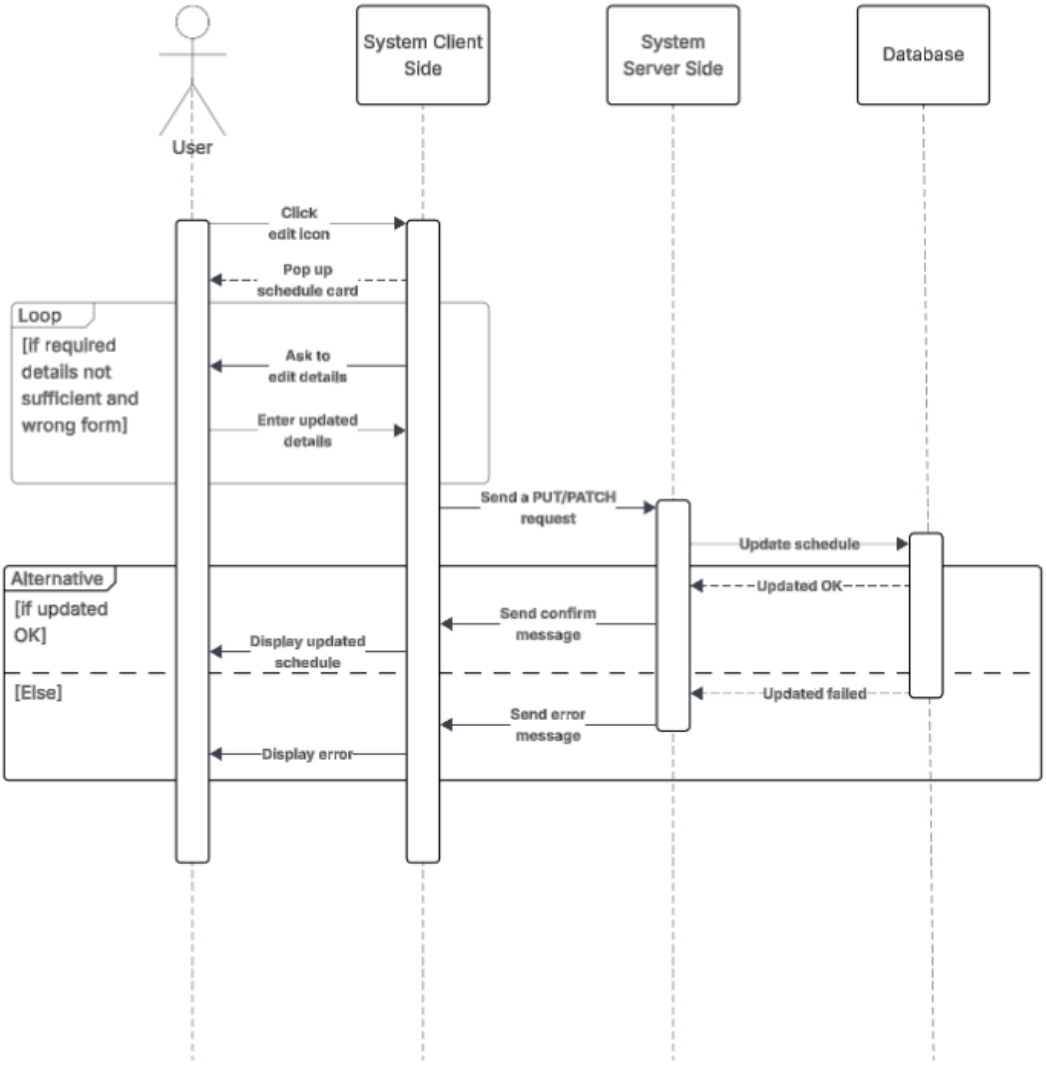

Figure 3.20: Edit Schedule Sequence Diagram

Figure 3.20 depicts the interaction between the user and the system to edit a schedule. After choosing the edit option, the user is asked to enter the edited details. If the required details are empty or the details are wrong format, the user needs to enter them again. If no errors, a schedule's details will be updated. Otherwise, the error will be illustrated.

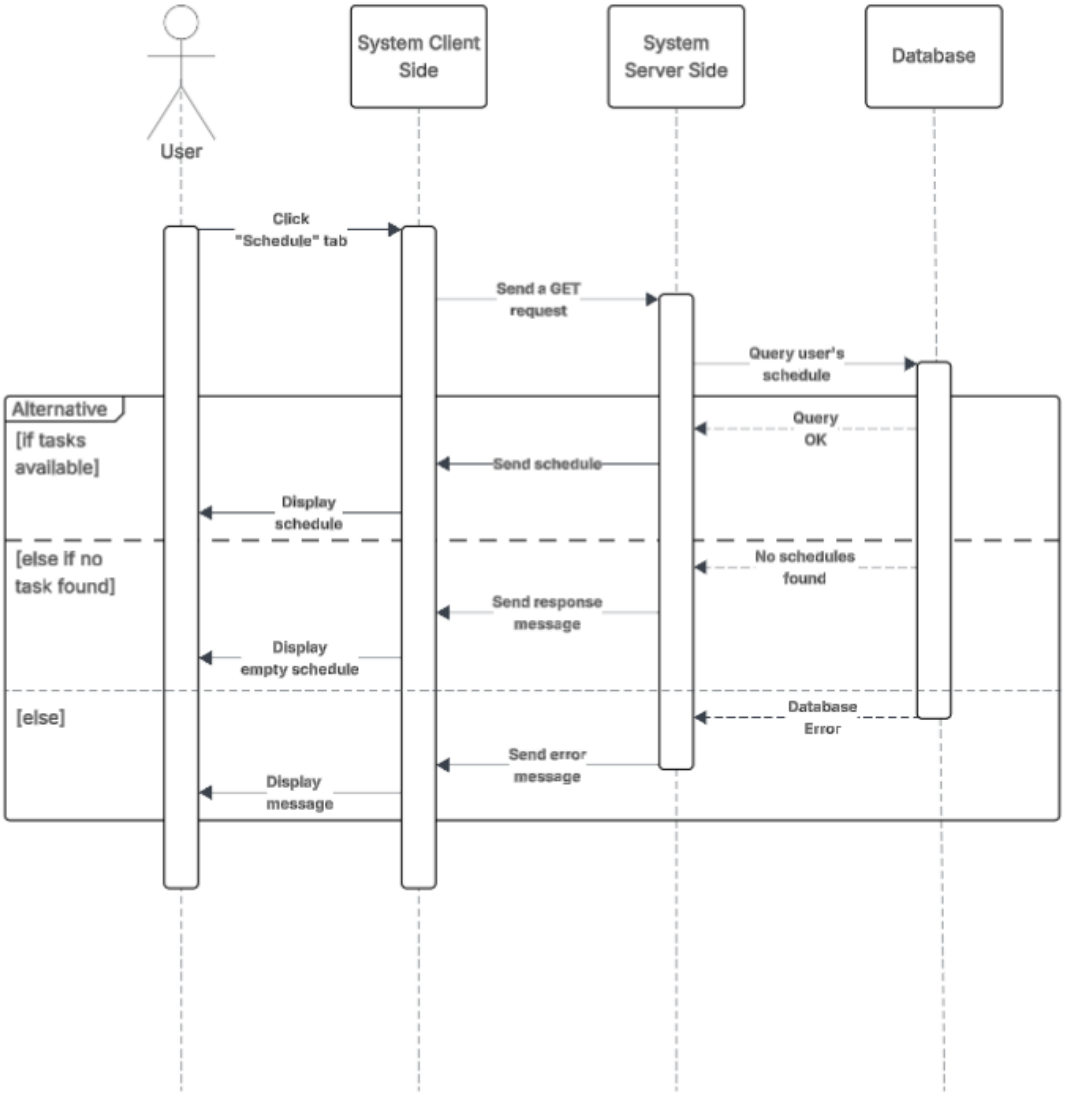

Figure 3.21: Display Schedule Sequence Diagram

Figure 3.21 demonstrates the interaction between the user and the system to retrieve a list of schedules. After choosing the display option, the system will ask the database for the list of schedules. If the query is successful, the system will display the results. However, if anything goes wrong, the system will display the errors.

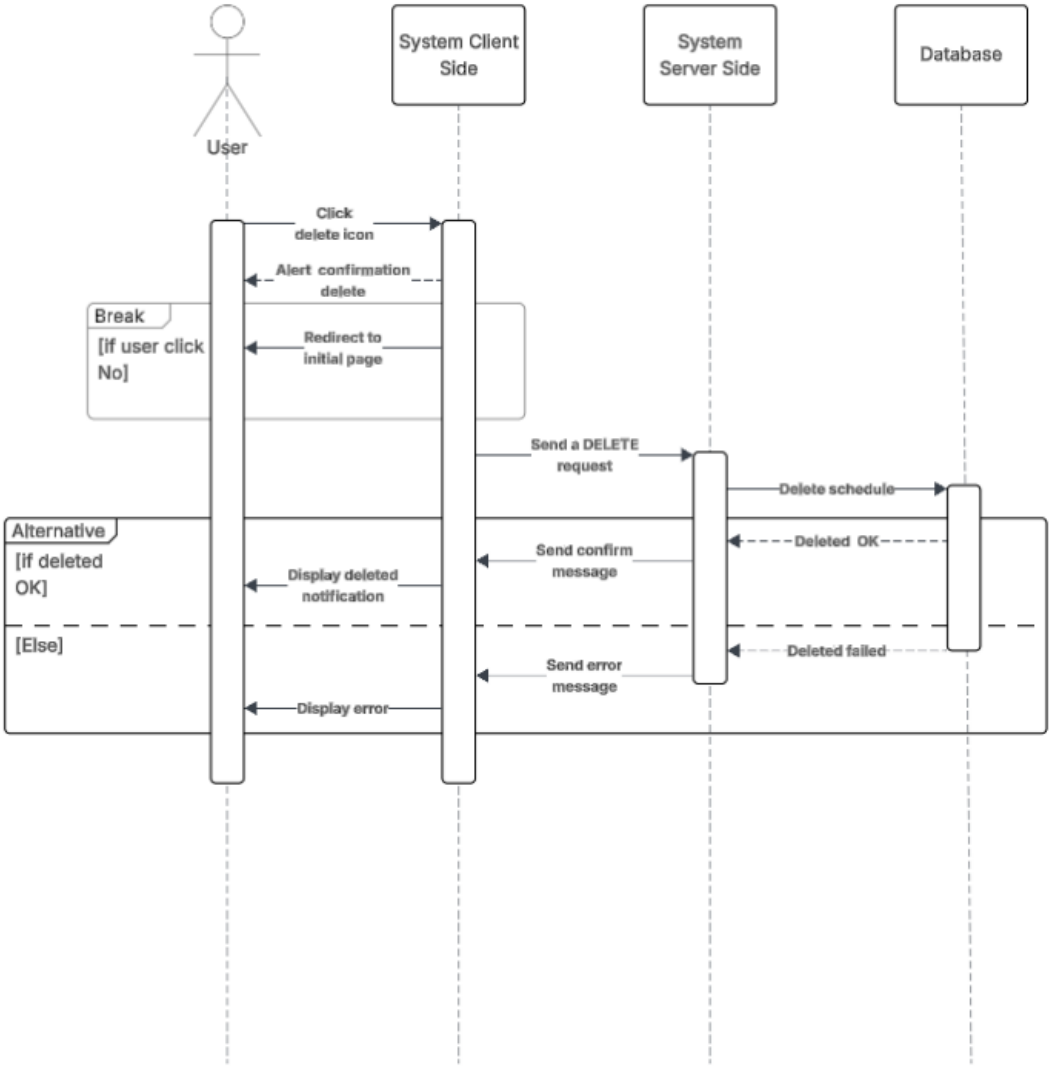

Figure 3.22: Remove Schedule Sequence Diagram

Figure 3.22 illustrates the interaction between the user and the system to remove an existing schedule. After choosing a to-delete schedule and clicking the delete icon, there will be an alert to confirm the action. If the user chooses the reject or cancel option, the process will be terminated. In the opposite cases, the system will send the request to the database to eliminate the schedule. If nothing goes wrong, this schedule will be deleted. However, if anything goes wrong, the error will be displayed.

Using Voice Button

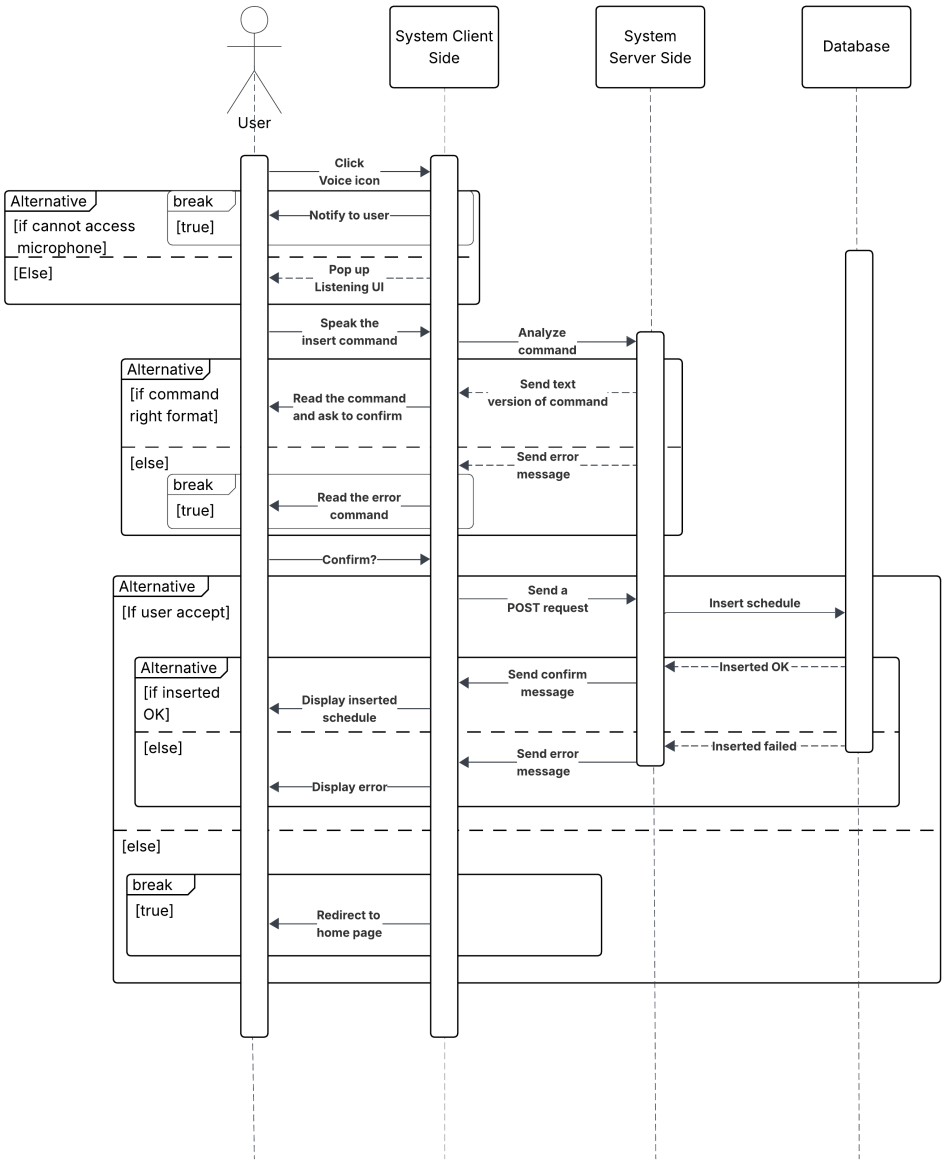

Figure 3.23: Add Schedule via Voice Sequence Diagram

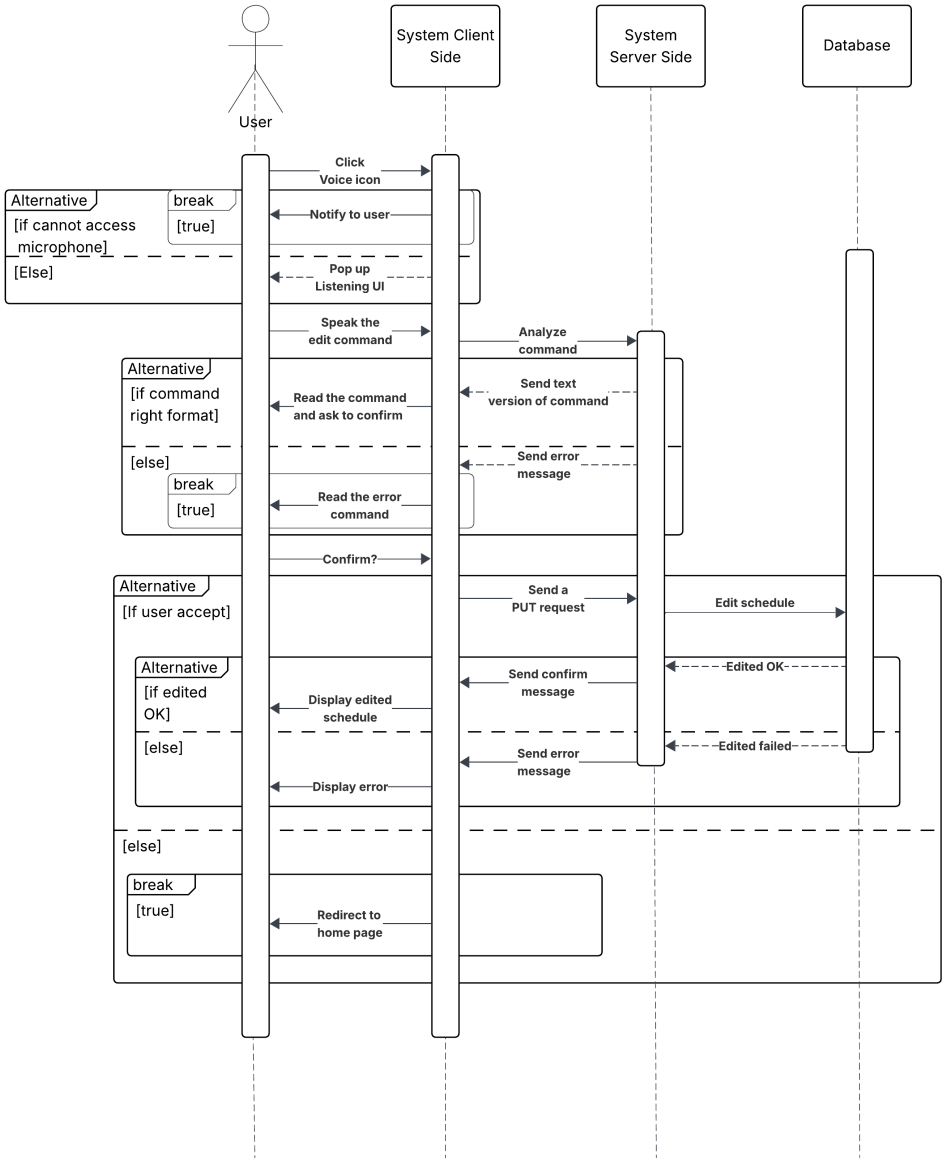

Figure 3.24: Edit Schedule via Voice

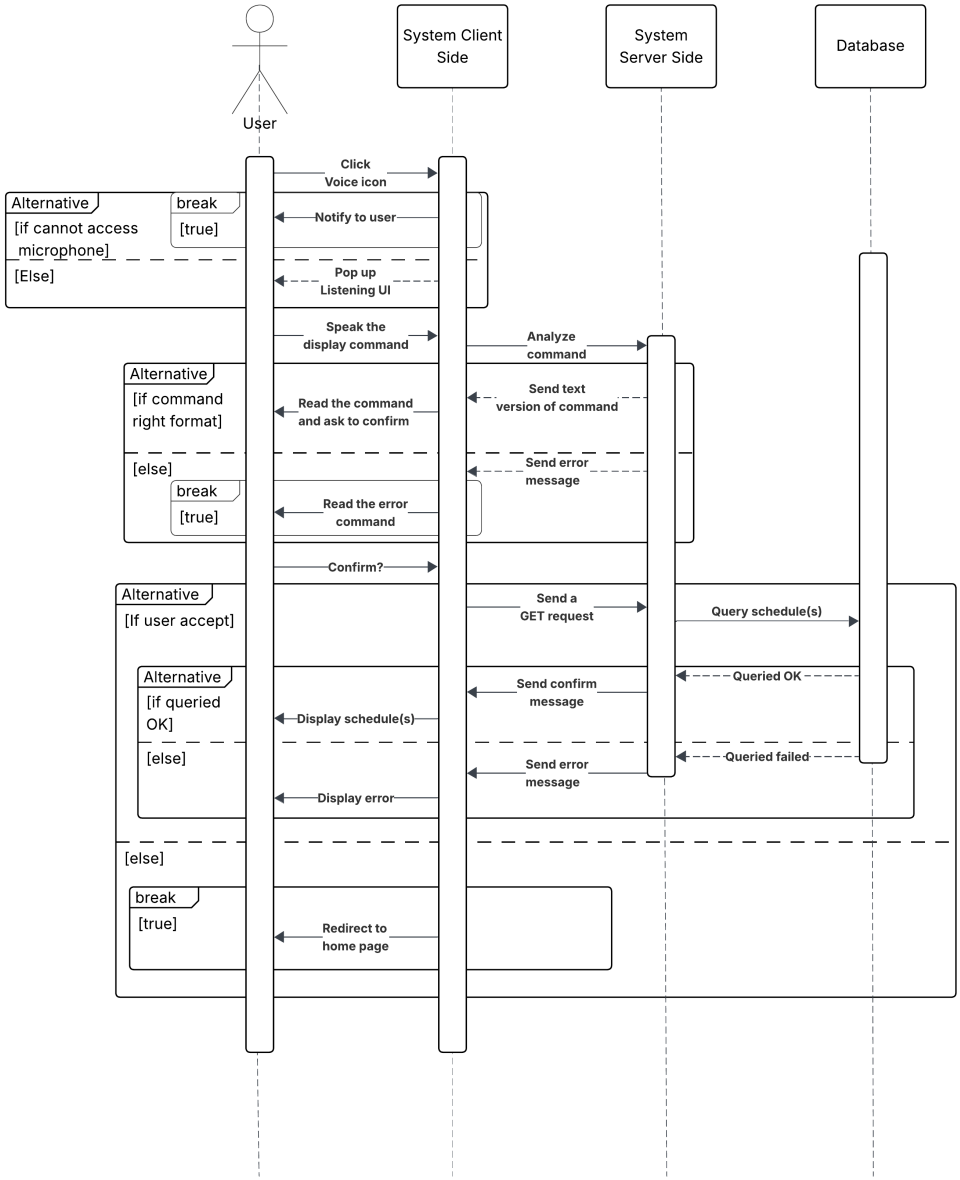

Figure 3.25: Display Schedule via Voice Sequence Diagram

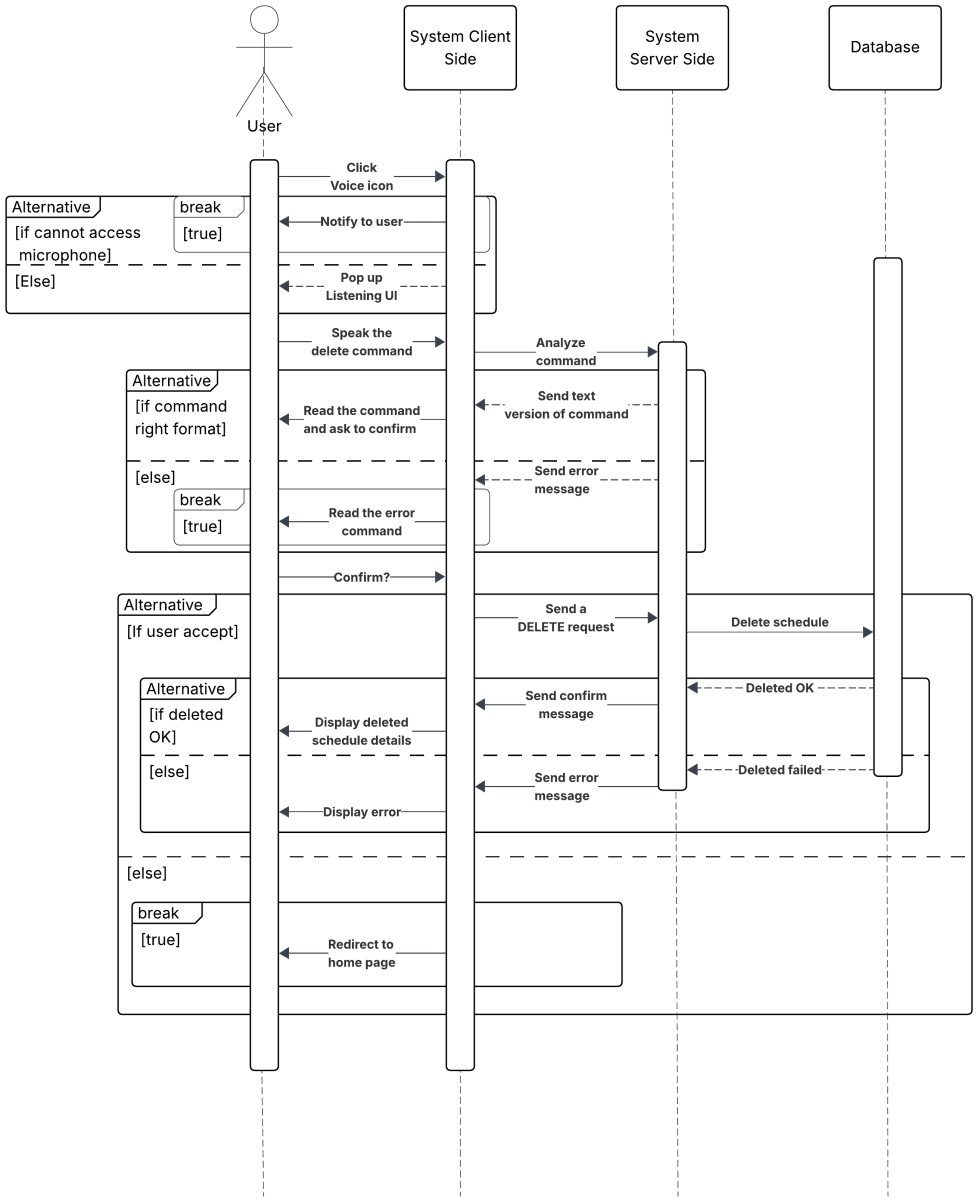

Figure 3.26: Remove Schedule via Voice Sequence Diagram

These sequence diagrams depict the interaction between the user and the system using voice to manage the schedule. After clicking the voice button, if the system is not allowed to use the microphone, the process will be terminated. In the opposite case, the system will start analyzing the recorded speech. If the speech is analyzed successfully and adapts to the requirements, the system will execute the actions. If errors occur, these errors will be displayed.

Figure 3.23 displays the add schedule using voice button. **Figure 3.24** illustrates the edit schedule with voice button. **Figure 3.25** illustrates the display schedules with voice button. Finally, **Figure 3.26** demonstrates the remove schedule using voice button.

3.4.2.2 Class Diagram

This comprehensive class diagram **Figure 3.27** illustrates a layered architecture for a scheduling and task management system, primarily partitioned into distinct functional modules such as Scheduling, Task Management, Authentication, and Core Services. Key components follow the Repository pattern, as evidenced by classes like ScheduleRepository and UserRepository, promoting a clear separation between the business logic and data access layers. Furthermore, the inclusion of NLPProcessor and SpeechToTextService suggests the system is designed to handle advanced natural language and voice inputs for creating and managing schedules.

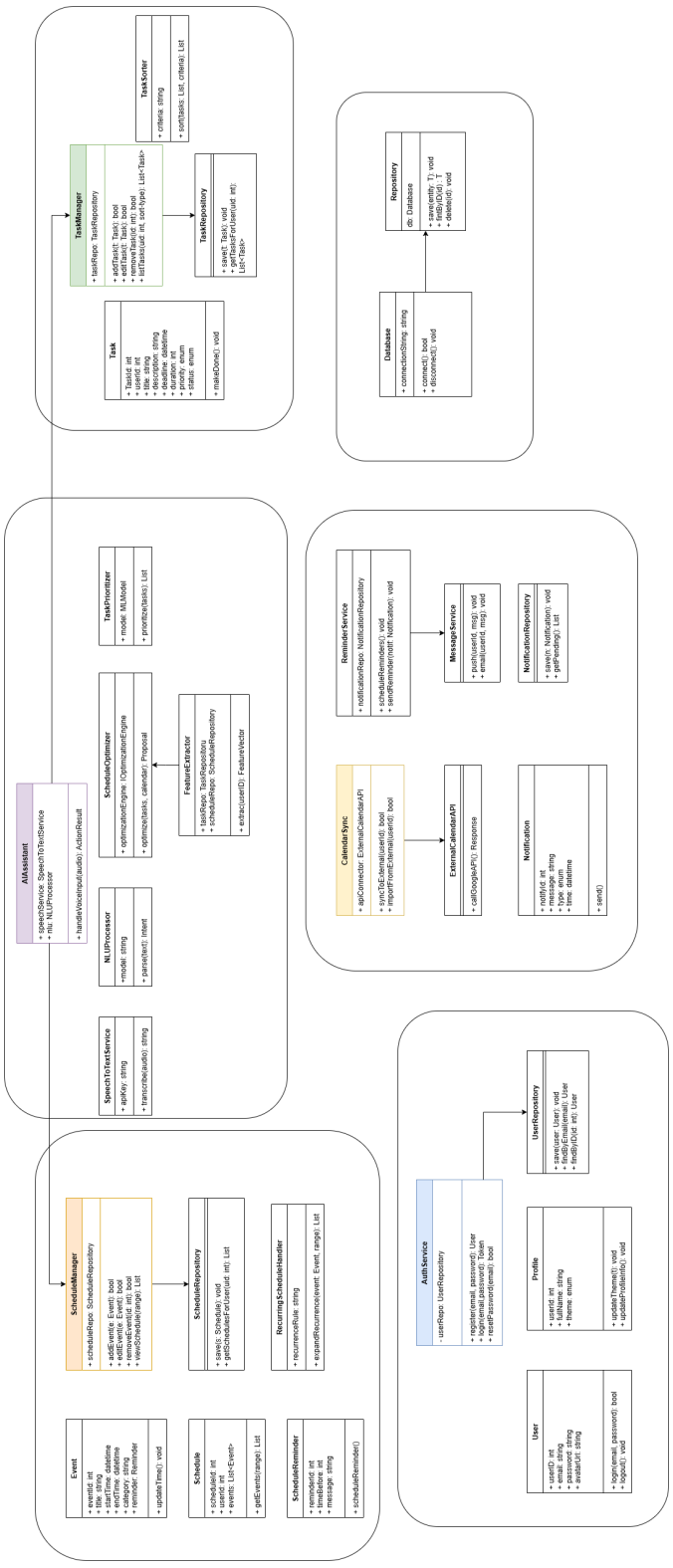

Figure 3.27: View Class Diagram

3.4.3 Process View

Activity Diagram

Sign Up

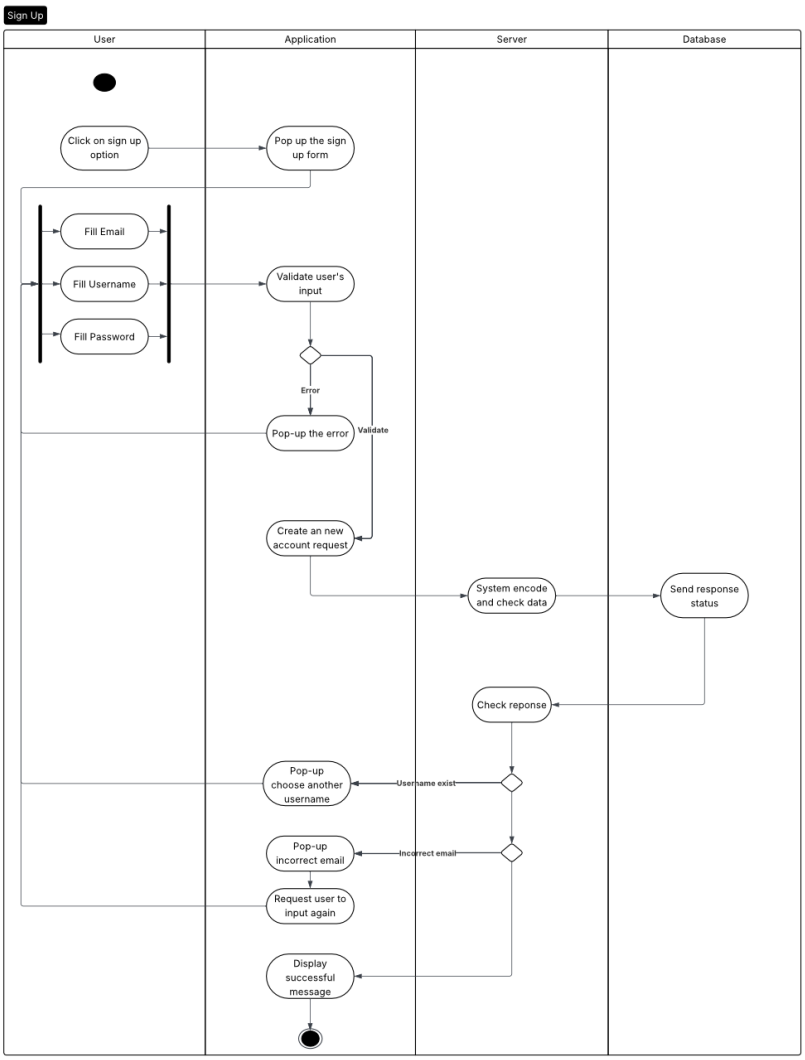

Figure 3.28: Sign-Up Activity Diagram

Figure 3.28 outlines the user sign-up workflow. The process starts when the user selects the sign-up option and enters the required information. The application validates the inputs and prompts the user to correct any errors. Once the data is valid, the system sends a registration request to the server, which checks for issues such as existing usernames or invalid email formats. If inconsistencies are found, the user is asked to revise the information. When all validations pass, the server creates the new account, and the application confirms successful registration to the user.

Login

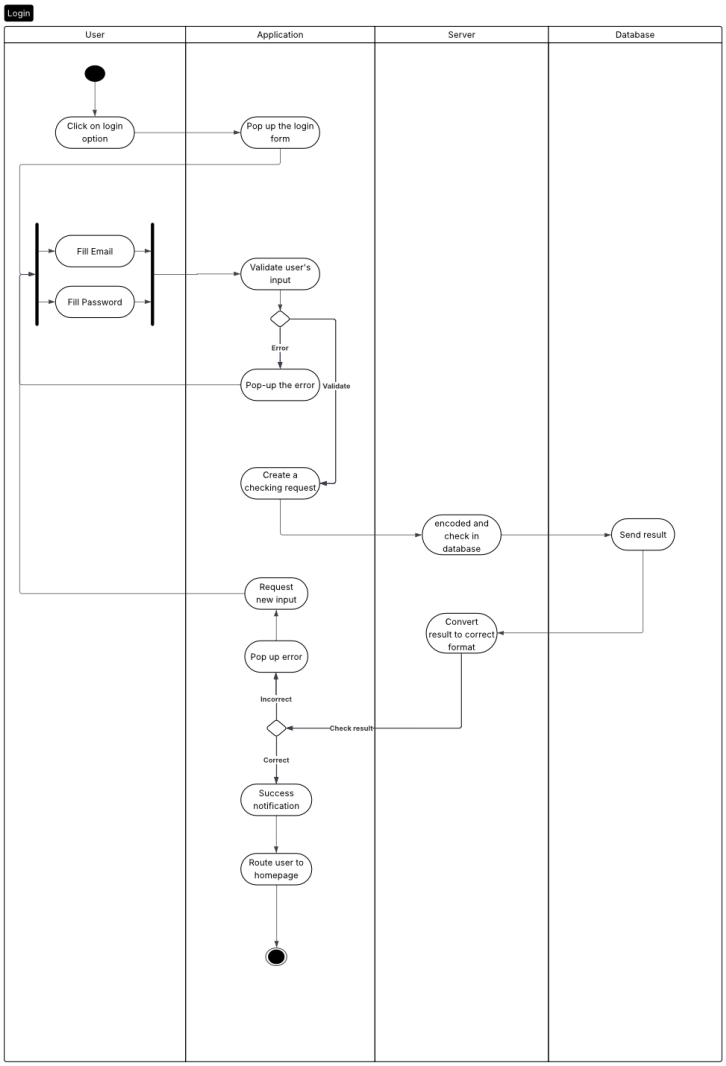

Figure 3.29: Login Activity Diagram

Figure 3.29 represents the user login process. The workflow begins when the user selects the login option and enters their email and password. The application validates the provided credentials and prompts the user to correct any invalid or incomplete inputs. Once the input passes validation, the system sends a login verification request to the server, which checks the credentials against stored records. If the credentials are incorrect, the application displays an error and requests the user to re-enter their information. When the server confirms valid credentials, the application notifies the user of a successful login and redirects them to the homepage.

Reset Password

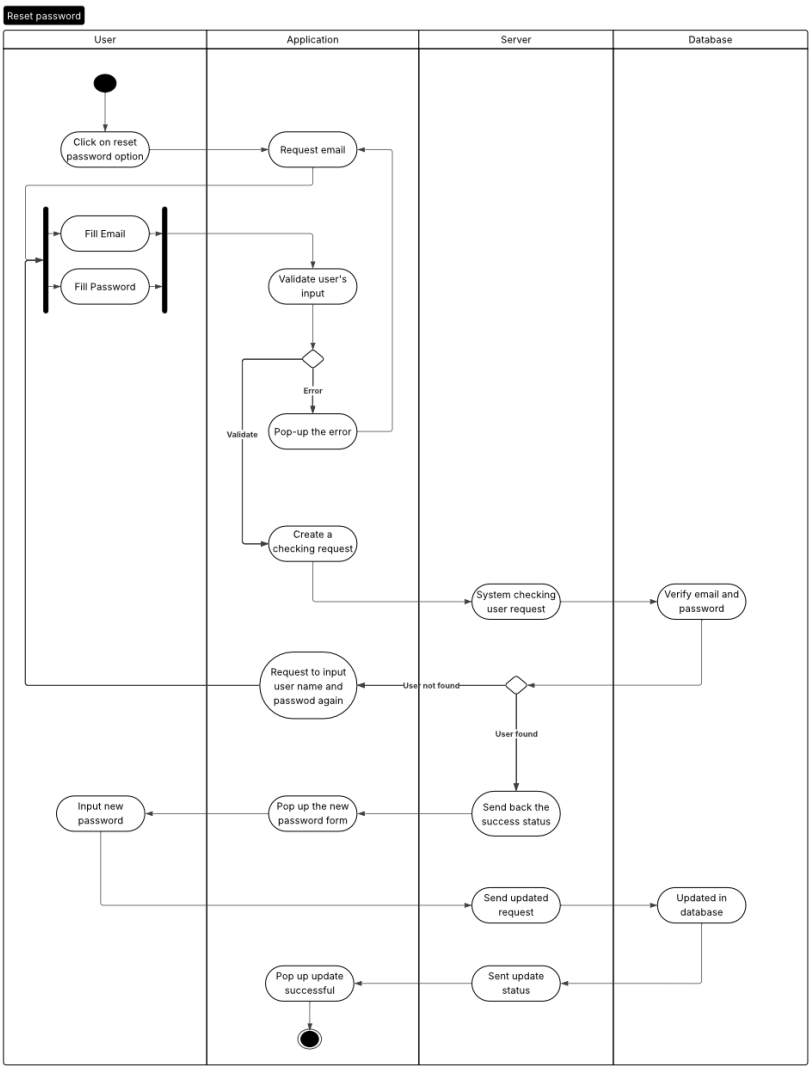

Figure 3.30: Reset Password Activity Diagram

Figure 3.30 illustrates the reset password process. The workflow begins when the user selects the reset password option and enters the required information. The application validates the input and prompts the user to correct any errors. Once the data is valid, the system sends a verification request to the server, which checks whether the user exists in the database. If the user is not found, the application requests the user to re-enter their information. If the user is identified, the server returns a success status and the application displays the new password form. After the user enters a new password, the system sends an update request to the server, which updates the record in the database. When the update is successful, the application notifies the user of a successful password reset, completing the process.

Update User Information

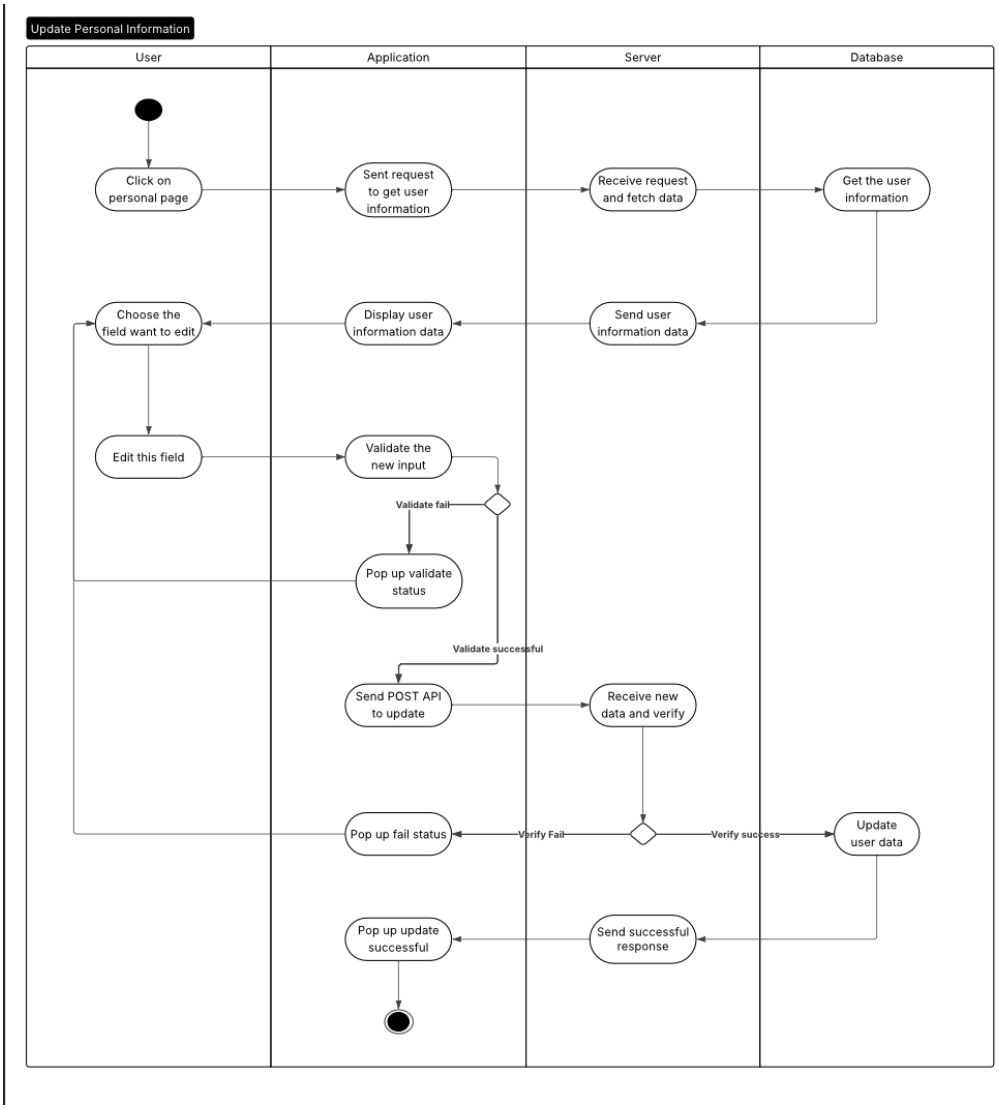

Figure 3.31: Update Personal Information Activity Diagram

Figure 3.31 illustrates the process of updating personal information. The workflow begins when the user navigates to their personal page, prompting the application to request and display the user's current information. The user selects a field to edit and submits new input, which the application validates. If validation fails, the system displays an error and requests correction. When the input is valid, the updated data is sent to the server for verification. If verification fails, an error status is returned; if successful, the server updates the data in the database and sends a confirmation response. The application then displays a success notification, completing the process.

Display User Task

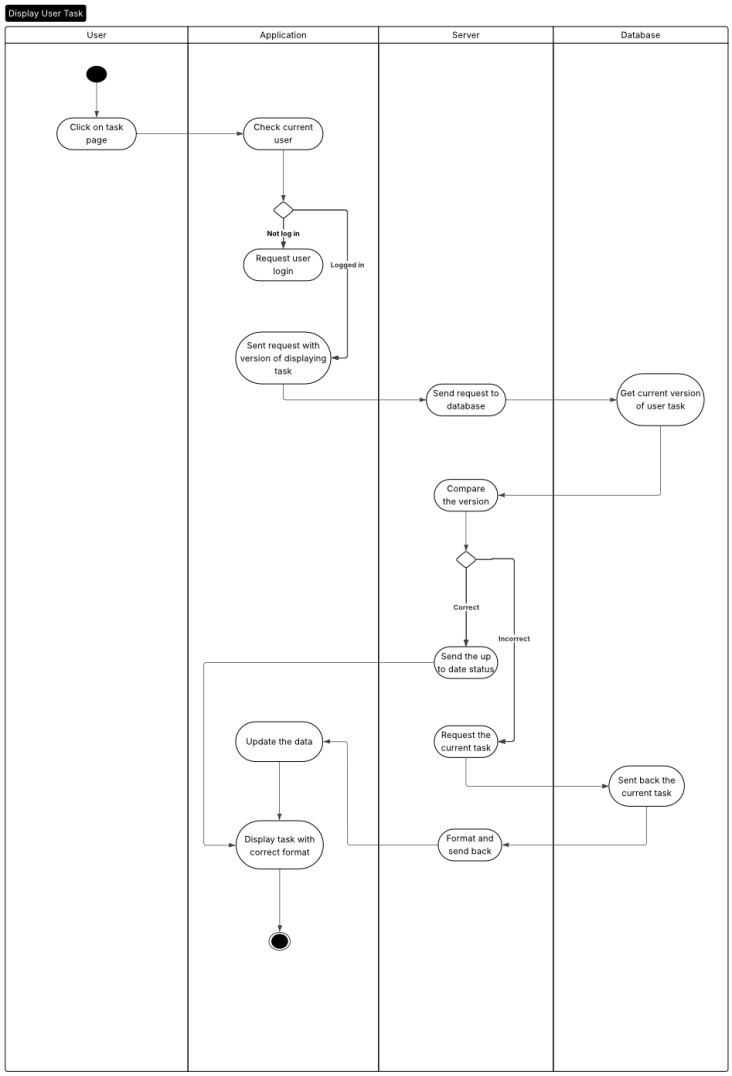

Figure 3.32: Display User Task Activity Diagram

Figure 3.32 describes the process of displaying a user's task list. The workflow begins when the user accesses the task page, prompting the application to check whether the user is currently logged in. If authentication is required, the user is asked to log in before proceeding. Once authenticated, the application sends the task version information to the server, which retrieves the current version from the database and compares it with the client's version. If the versions do not match, the server returns the updated task data; if they match, an up-to-date status is sent. The application then updates the task data if needed and displays the task information in the correct format, completing the process.

Add User Task

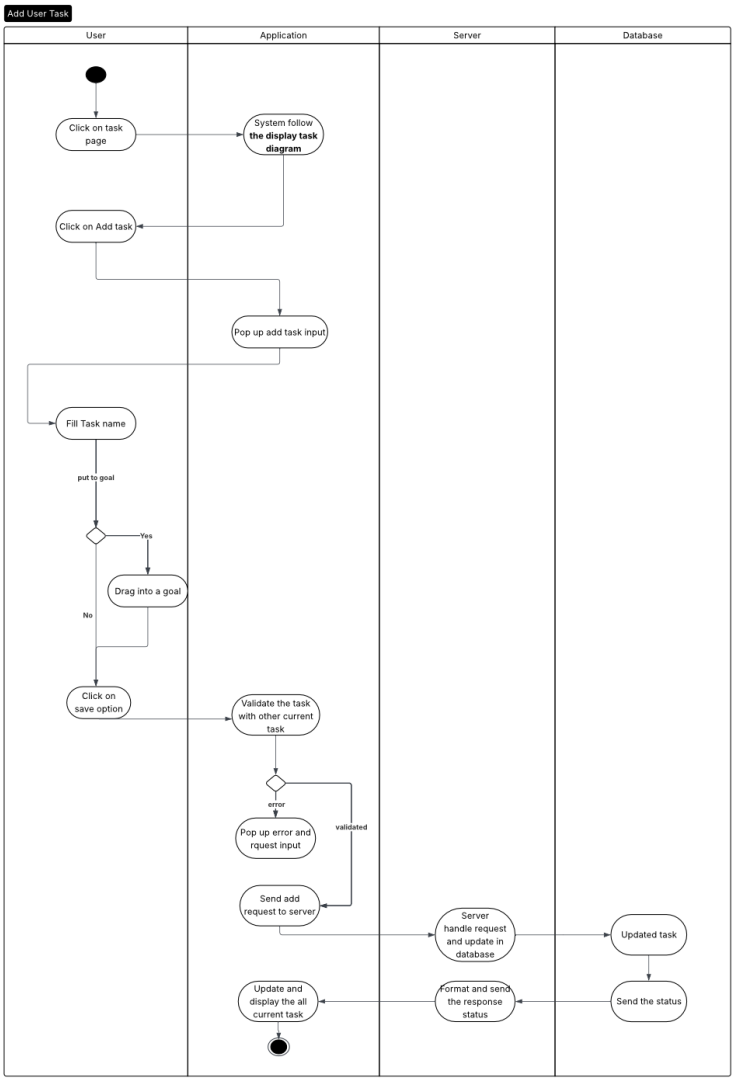

Figure 3.33: Add User Task Activity Diagram

Figure 3.33 describes the process of adding a new user task. The workflow begins when the user navigates to the task page, where the system retrieves and displays the current task information. The user selects the option to add a new task, prompting the application to display the task input form. After entering the task name, the user may assign it to a goal before saving. The application validates the new task against existing tasks and requests correction if any conflict or error is detected. When the input is valid, the application sends an add-task request to the server, which updates the task data in the database and returns a response status. The application then updates the displayed task list and shows the newly added task, completing the process.

Edit User Task

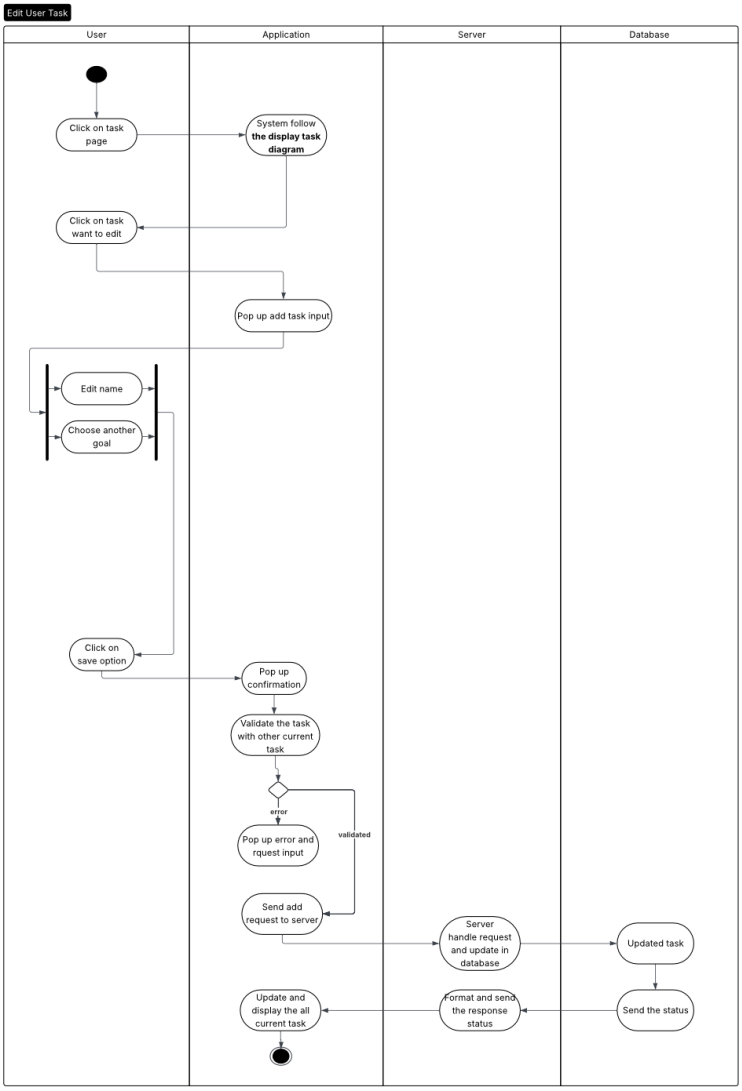

Figure 3.34: Edit User Task Activity Diagram

Figure 3.34 describes the process of editing an existing user task. The workflow begins when the user navigates to the task page, where the system loads and displays the current tasks. The user selects a task to edit, prompting the application to show the task input form. The user can modify the task name, assign it to another goal, and then save the changes. The application validates the updated task against existing tasks to prevent conflicts, requesting correction if validation fails. When the task is valid, the system sends an update request to the server, which processes the change and updates the task in the database. After receiving a successful response, the application refreshes and displays the updated task list, completing the process.

Remove User Task

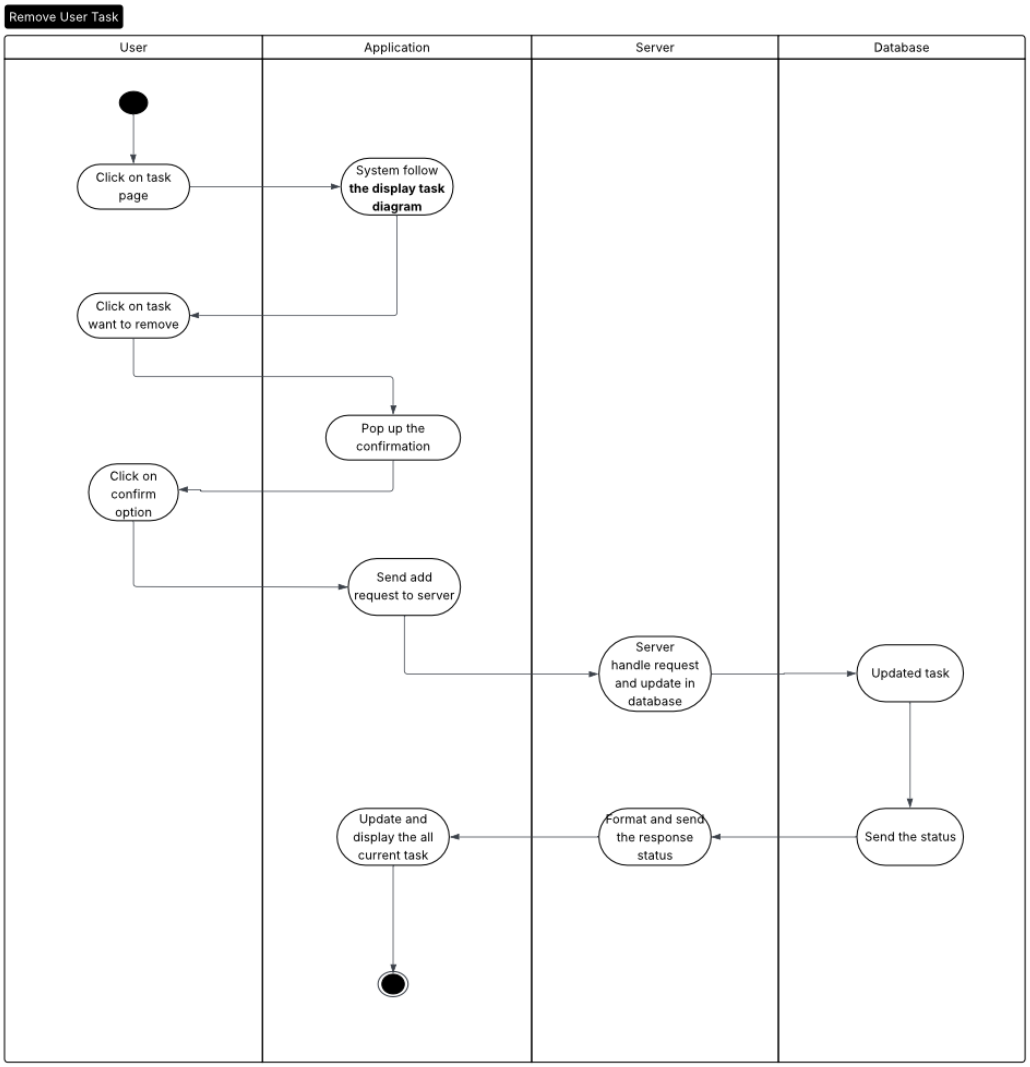

Figure 3.35: Remove User Task Activity Diagram

Figure 3.35 describes the process of removing an existing user task. The workflow begins when the user navigates to the task page, where the system loads and displays the current tasks. The user selects the task they want to remove, prompting the application to display a confirmation dialog. After the user confirms the action, the application sends a removal request to the server. The server processes the request, updates the task data in the database, and returns a status response. The application then refreshes and displays the updated task list, completing the removal process.

Add User Goal

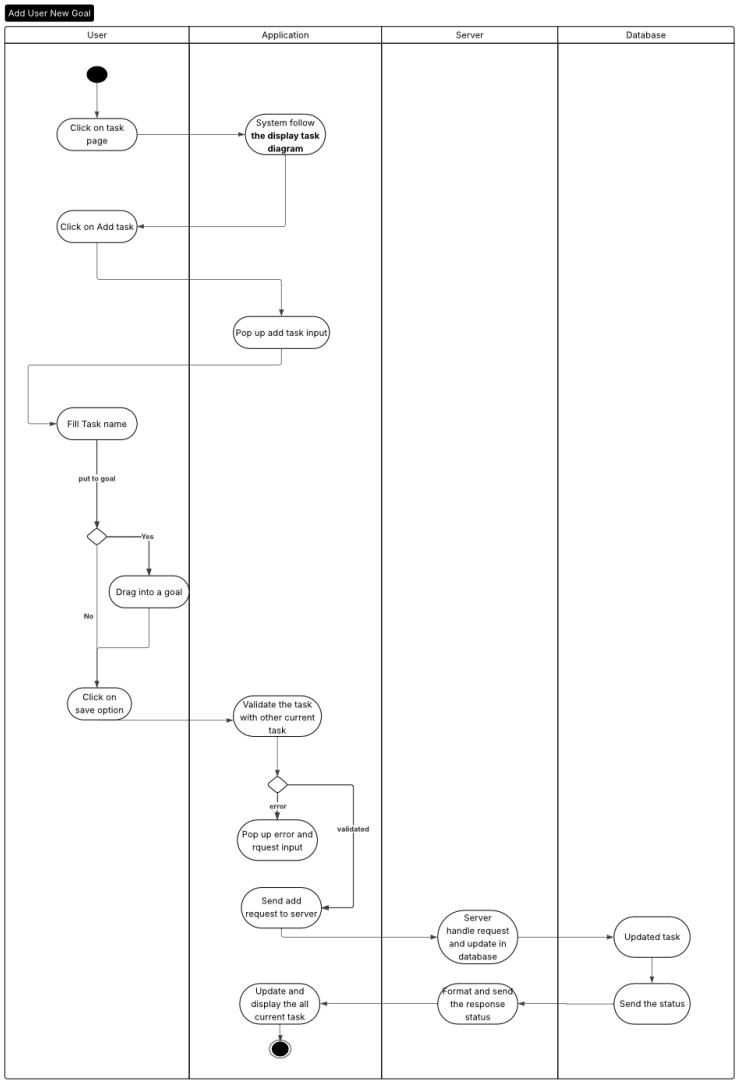

Figure 3.36: Add User New Goal Activity Diagram

Figure 3.36 illustrates the process of adding a new goal for the user. The workflow begins when the user navigates to the task page, where the system retrieves and displays the current tasks. The user selects the option to add a new goal, prompting the application to show the input form. After entering the goal name, the user may assign it to an existing category before saving. The application validates the new goal to ensure no conflicts with current tasks or goals, requesting correction if any issues arise. Once the input is valid, the system sends a request to the server to create the new goal. The server processes the request, updates the data in the database, and returns the status response. The application then refreshes and displays the updated task list with the newly added goal, completing the process.

Display User Schedule

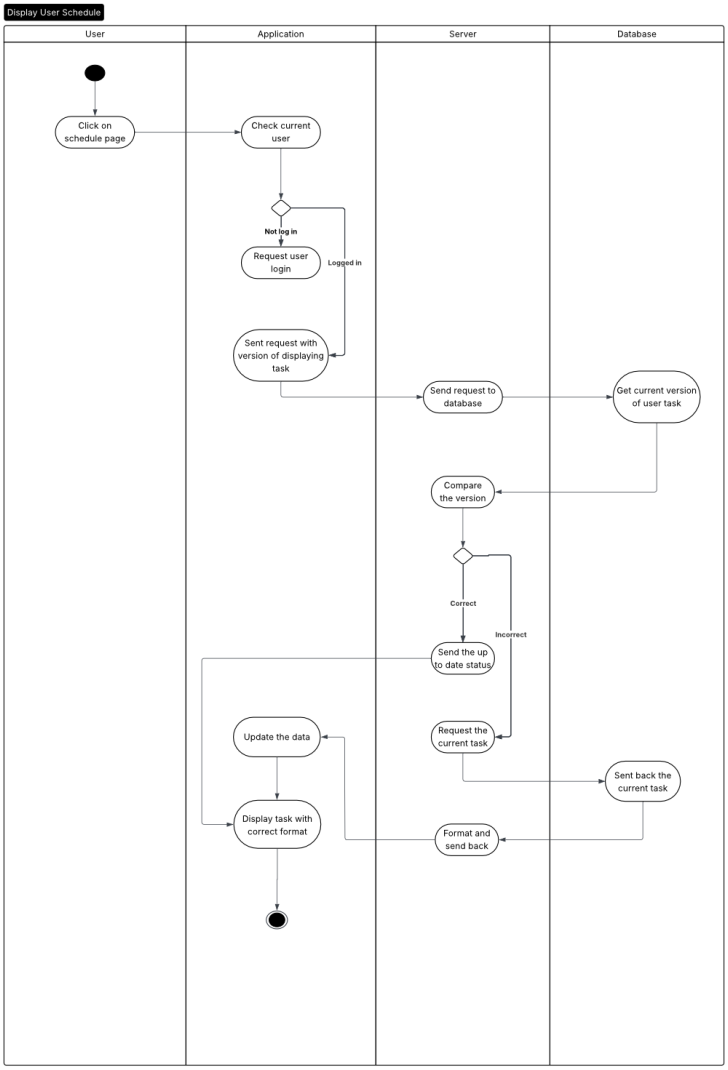

Figure 3.37: Display User Schedule Activity Diagram

Figure 3.37 illustrates the process of displaying the user's schedule. The workflow begins when the user opens the schedule page, prompting the application to verify the user's login status. If the user is not authenticated, a login request is displayed; otherwise, the system proceeds by sending the current schedule version information to the server. The server retrieves the latest schedule data from the database and compares the version with the client's version. If the versions differ, the server sends the updated schedule; if they match, an up-to-date status is returned. The application then updates the schedule data when necessary and displays it in the correct format, completing the process.

Add User Schedule

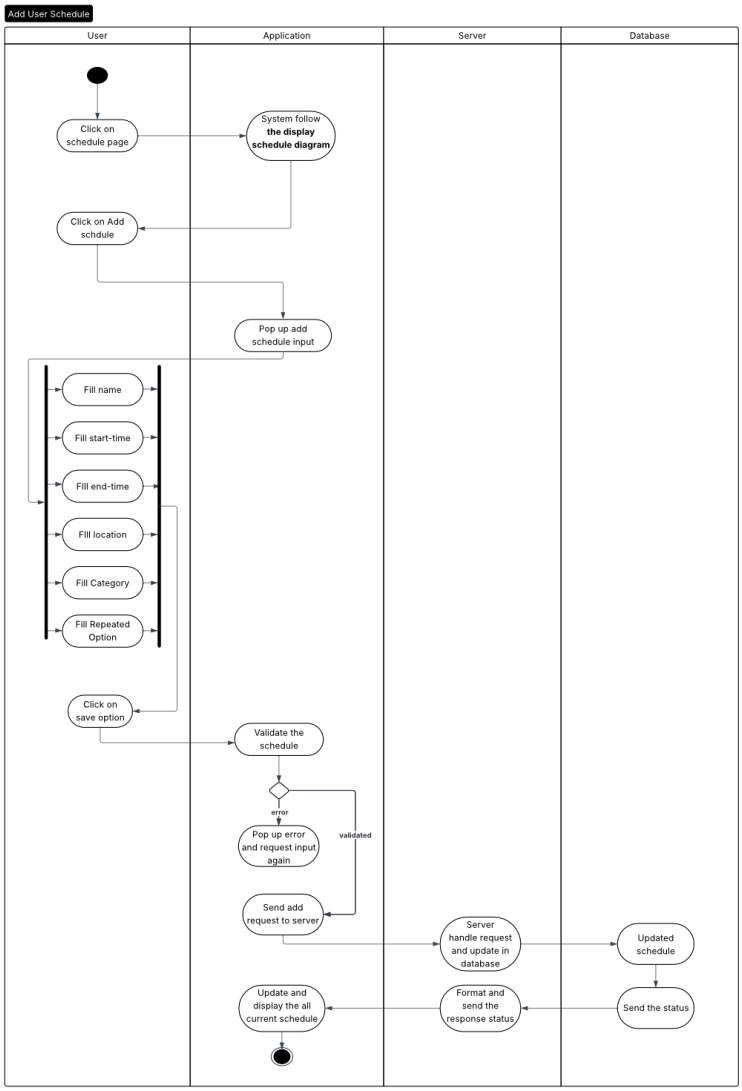

Figure 3.38: Add User Schedule Activity Diagram

Figure 3.38 illustrates the process of adding a new schedule entry. The workflow begins when the user opens the schedule page, where the system loads and displays the current schedule data. The user selects the option to add a new schedule, prompting the application to display the schedule input form. The user then fills in the required details, including name, start time, end time, location, category, and repeat options. Upon saving, the application validates the input and requests correction if any errors are found. Once the schedule information is valid, the application sends an add-schedule request to the server. The server processes the request, updates the schedule in the database, and returns a status response. The application then updates and displays the latest schedule list, completing the process.

Edit User Schedule

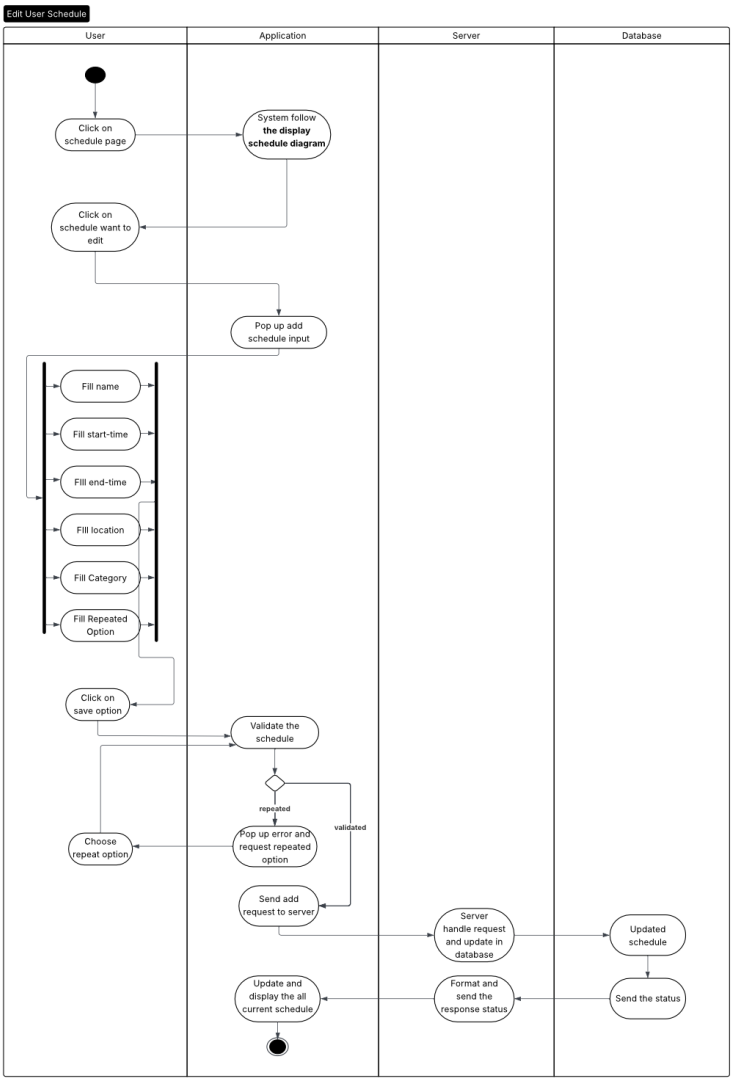

Figure 3.39: Edit User Schedule Activity Diagram

Figure 3.39 shows the process of editing an existing schedule entry. The workflow begins when the user accesses the schedule page, where the system loads and displays the current schedule information. The user selects a schedule item to edit, prompting the application to display the input form populated with the existing details. The user can modify fields such as name, start time, end time, location, category, and repeat options before saving. The application validates the updated information and requests correction if any issues are detected, particularly with repeated scheduling conflicts. When the input is valid, an update request is sent to the server. The server processes the request, updates the schedule in the database, and returns a status response. The application then refreshes and displays the updated schedule list, completing the editing process.

Remove Schedule

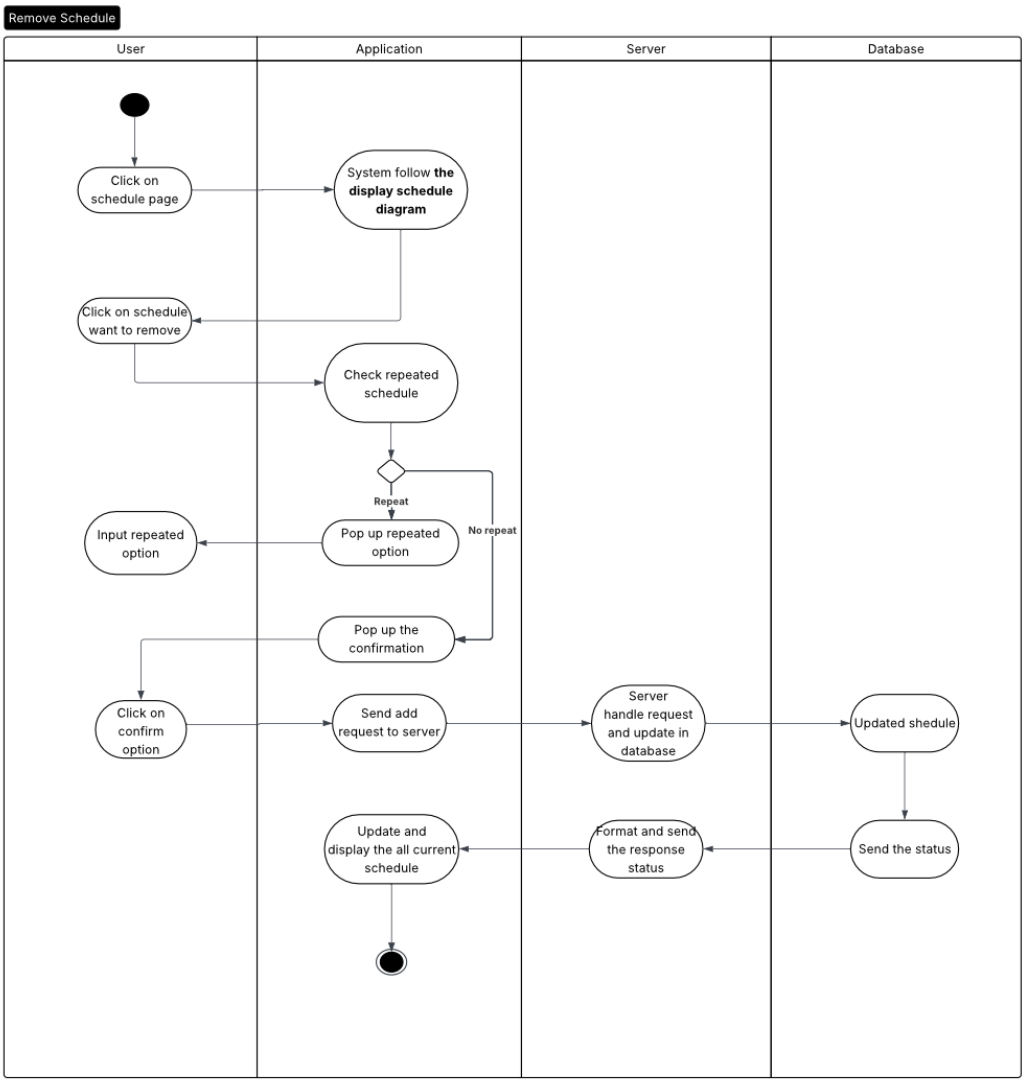

Figure 3.40: Remove Schedule Activity Diagram

Figure 3.40 illustrates the process of removing a schedule entry. The workflow begins when the user navigates to the schedule page, where the system loads and displays the current schedule data. The user selects the schedule item they want to remove, prompting the application to check whether the schedule is part of a repeated sequence. If it is repeated, the user is asked to specify whether they want to delete a single occurrence or the entire series. After the selection is made, the system displays a confirmation dialog. Once the user confirms the removal, the application sends a delete request to the server. The server processes the request, updates the schedule data in the database, and returns a response status. The application then refreshes and displays the updated schedule list, completing the removal process.

3.4.4 Development View

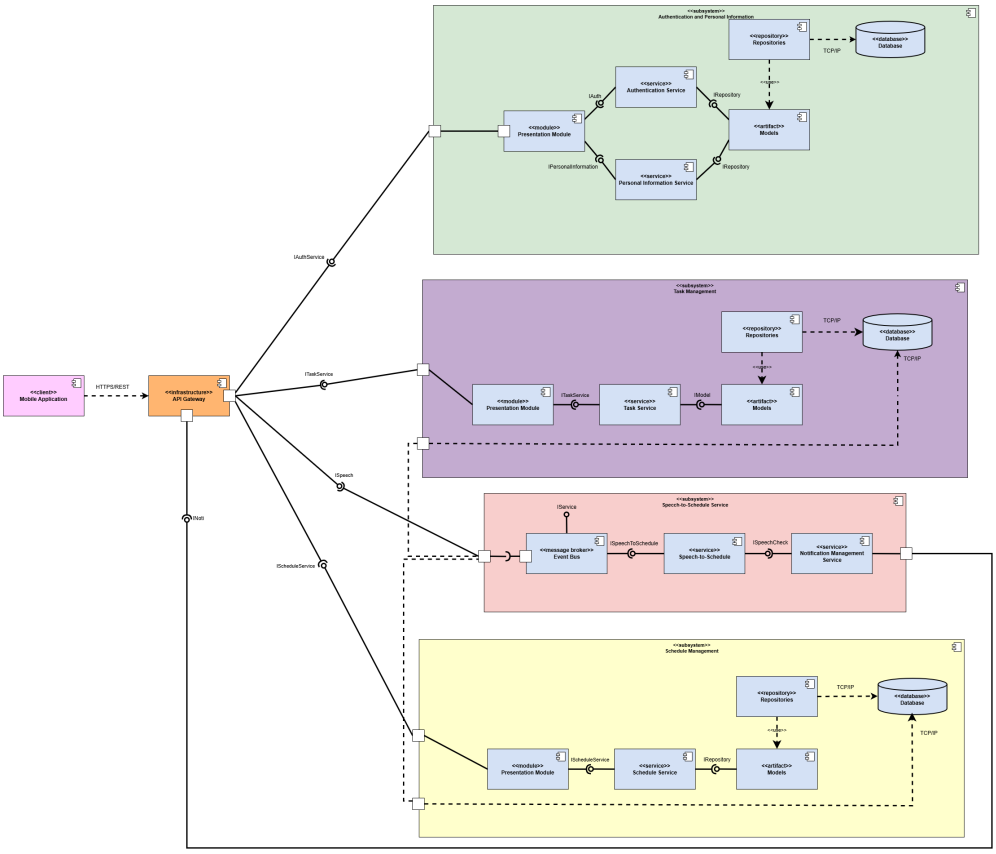

Figure 3.41: Component Diagram

Figure 3.41 illustrates the component diagram of the system's architecture. The system adopts a modular design where the Mobile Application serves as the client-side entry point, connecting the user to the backend services. An API Gateway acts as an intermediary, routing incoming requests from the application to the appropriate server-side modules.

The backend is divided into distinct, encapsulated subsystems. The Authentication and Personal Information module manages user identity and profile data, utilizing a dedicated repository and database for persistence. The Task Management module handles the lifecycle of user tasks, maintaining its own separate database. Similarly, the Schedule Management module organizes the user's calendar events using a layered architecture comparable to the Task module. Finally, the Voice Processing module facilitates event-driven features; it employs an Event Bus for asynchronous communication, a Speech-to-Schedule component to convert voice commands into tasks, and a Notification Service to manage outgoing alerts.

3.5 Database Design

3.5.1 Entity-Relationship Model

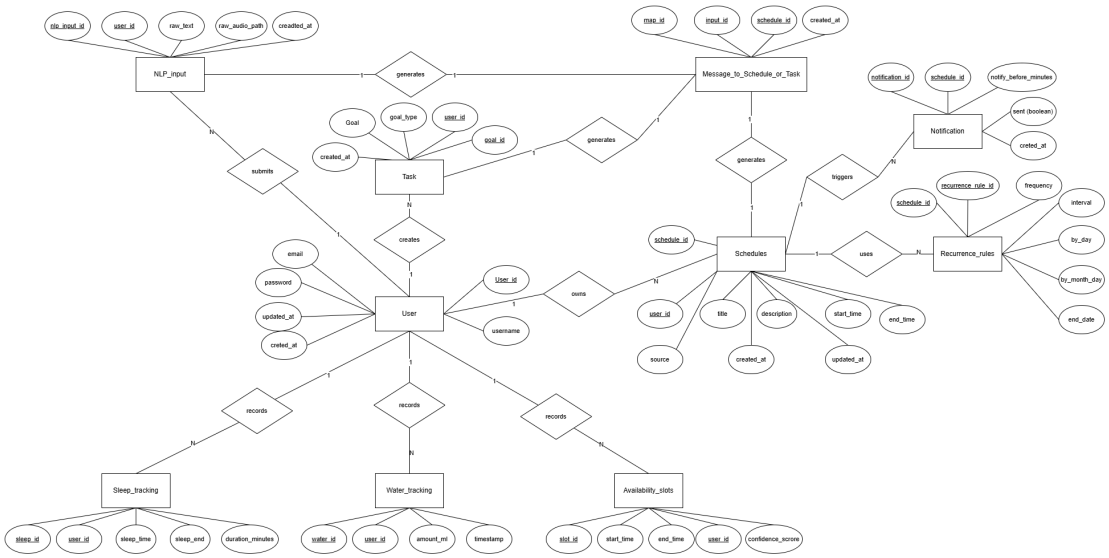

Figure 3.42: Entity Relationship Diagram

Figure 3.42 shows the system's Entity-Relationship Diagram. The database design for the Smart Schedule AI platform allows for different input methods (speech to text to calendar events and speech to text to work items), automatic task creation through artificial intelligence, and monitoring in various areas including sleep patterns, water intake, and long-term goals. The ERD illustrates a modular yet well-integrated design that meets the needs of a smart and responsive personal scheduling solution.

User-Centered Data Foundation:

The `User` entity acts as the main part of the schema, holding authentication details and basic profile information. Each personalized element—including calendar management, work items, goals, sleep records, hydration logs, and calculated availability windows—links to `User` through one-to-many relationships. This design ensures appropriate data separation while allowing for fully personalized analytics and AI-driven suggestions.

Natural Language Processing Input Layer:

The `NLP_input` entity captures raw user communications from both spoken and written sources. Each entry includes transcribed text (`raw_text`) along with optional audio file links (`raw_audio_path`). The one-to-many relationship with `User` enables complete tracking of historical data for system review, model improvement, and behavioral pattern discovery. Notably, this component supports two different processing routes:

$$\text{speech} \rightarrow \text{text} \rightarrow \text{calendar}, \qquad \text{speech} \rightarrow \text{text} \rightarrow \text{work item}$$

This dual pathway allows the language model pipeline to derive either time-specific appointments or actionable tasks from a single natural language input.

AI Translation and Linkage Layer:

The `Message_to_Schedule_or_Task` entity links unstructured language input to organized system outputs. It records the conversion process where each `NLP_input` instance creates one or more calendar entries or work assignments. The one-to-one or one-to-many relationship with `NLP_input` handles different situations, such as a single message producing several calendar events or multiple related messages contributing to one overall task. This setup improves pipeline clarity and helps troubleshoot language model outputs.

Calendar System and Repetition Specifications:

The `Schedules` entity keeps all calendar appointments, including details like title, description, time limits, and source indicator (whether user-entered or AI-generated). To manage recurring appointments, the `Recurrence_rules` entity uses iCalendar-compliant repetition rules, such as frequency settings, interval options, weekday guidelines, and optional end dates. A one-to-many relationship between `Schedules` and `Recurrence_rules` provides flexible yet standardized management of recurring events.

Alert Delivery Infrastructure:

The `Notification` entity connects reminders with calendar entries and sets alert timing (`notify_before_minutes`). The boolean `sent` attribute ensures single-delivery semantics, allowing for asynchronous operations and preventing duplicate reminders. The one-to-many relationship from `Schedules` enables multiple alerts per individual appointment.

Wellness and Behavioral Monitoring:

The data model includes wellness data to support broader behavioral analysis. The `Sleep_tracking` component records sleep periods for pattern assessment and sleep-aware scheduling. The `Water_tracking` module tracks hydration events, aiding in lifestyle profiling. The `Availability_slots` entity shows AI-calculated free time periods, each marked with a confidence level to support smart scheduling suggestions.

Work Item and Objective Administration:

The `Task` entity represents tasks that come from either user commands or the AI planning system. It connects to both `User` and possibly to `Goal`. The `Goal` entity captures long-term aspirations in areas like academics, health, and productivity, allowing the platform to align daily actions with broader user goals. The introduction of the new processing pathway (speech to text to work item) means that spontaneous verbal commands can create structured tasks immediately, making the system feel more natural and responsive.

3.5.2 Mapping ERD to Relational Model

The transition from the ERD representation to the relational structure follows standard database design practices. Each independent entity becomes a separate table with its own primary key. Entity properties become table columns. For one-to-many relationships, the primary key from the "one" side appears as a foreign key in the table on the "many" side. One-to-one relationships

are handled by transferring the primary key from one entity to another while ensuring uniqueness. Multi-valued properties or dependent entities are placed into additional tables when needed.

In our setup, the `User` entity is central, serving as the main point for most connections. From this core entity, links extend to operational modules including task management (`Task`), calendar operations (`Schedules`), sleep tracking (`Sleep_tracking`), water logging (`Water_tracking`), and natural language input processing (`NLP_input`). Each functional component acts as a standalone table linked to the `User` table through the `user_id` foreign key reference.

The `Schedules` entity supports recurring appointments. Thus, a one-to-many mapping principle applies: a single repetition specification (`Recurrence_rules`) may control multiple calendar entries, so the `recurrence_rule_id` primary key appears as a foreign key in the `Schedules` table. In the same way, individual calendar entries can generate multiple reminders, forming a one-to-many relationship between `Schedules` and `Notification`.

An important aspect of the architecture is the intermediary entity `Message_to_Schedule_or_Task`, which represents the link between natural language input and created entities (`Task` or `Schedule`). This connection is realized as a separate table containing the `input_id` foreign key linking to `NLP_input` and the `schedule_id` foreign key linking to `Schedules`. This approach allows tracking of input data origins and maintains referential integrity between the NLP component and the scheduling system.

In summary, the transformation process ensures that the resulting relational structure fully meets design principles (first, second, and third normal forms), eliminates data duplication, and maintains all restrictions defined in the ERD. The final relational model clearly represents the structure while enabling easy implementation on database platforms like PostgreSQL.

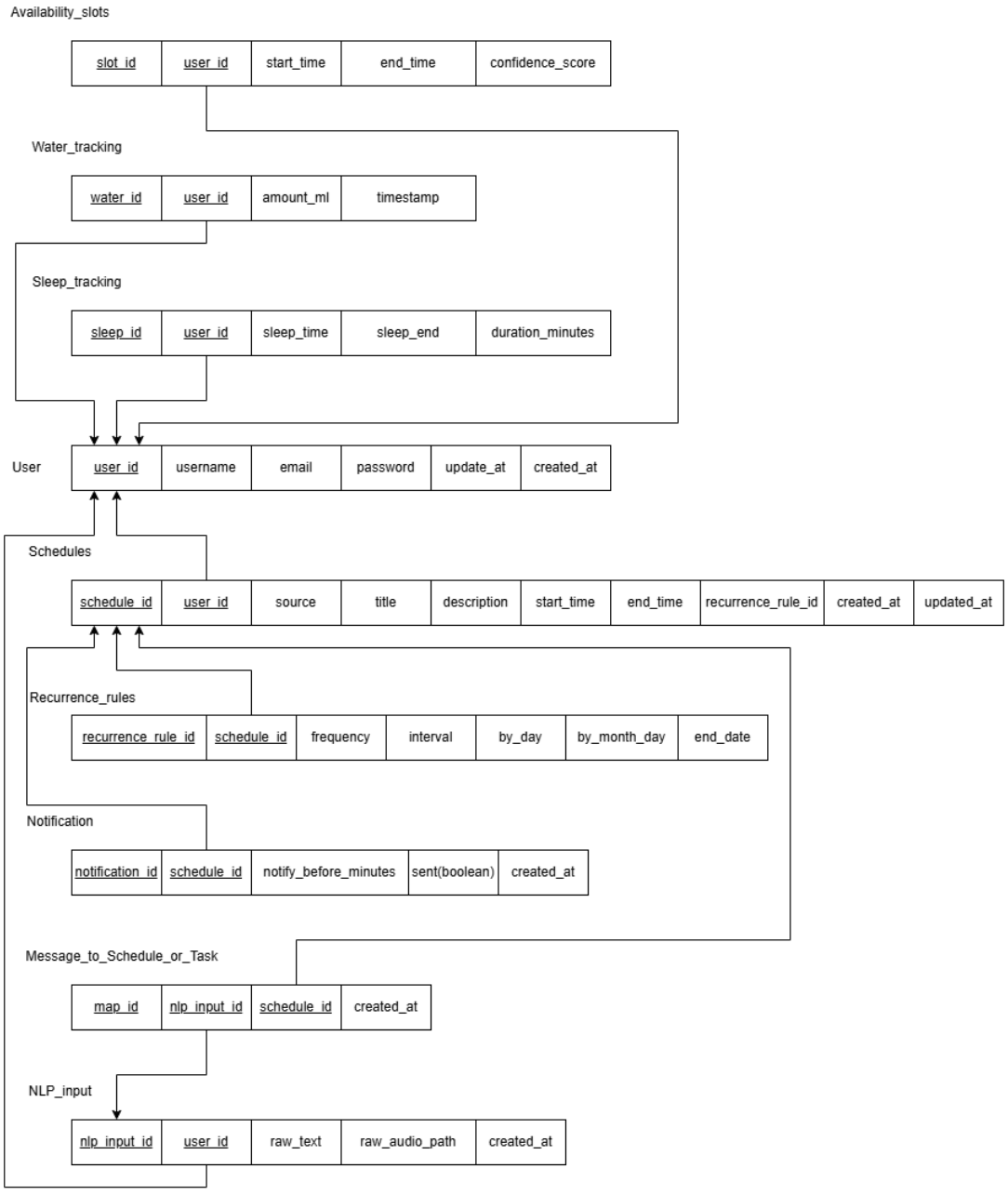

Figure 3.43: Mapping ERD

According to **Figure 3.43**, which illustrates the ERD transformation for our project, the complete relational schema is provided below. This includes all tables, primary keys, foreign keys, and constraints.

1. User

The User table includes `user_id` as the primary key, along with `email`, `username`, `password`, `created_at`, and `updated_at` fields. It has one-to-many relationships with Task, Schedules, Sleep_tracking, Water_tracking, Availability_slots, and NLP_input entities.

2. Task

The Task table uses `task_id` as its primary key and has `user_id` as a foreign key referencing `User.user_id`. Other fields include `goal_type`, `target_value`, `current_value`, and `created_at`. This table has a one-to-one relationship with Message_to_Schedule_or_Task and a many-to-one relationship with User.

3. NLP_input

The NLP_input table uses `nlp_input_id` as its primary key and includes `user_id` as a foreign key pointing to `User.user_id`. Additional fields include `raw_text`, `raw_audio_path`, and `created_at`. It establishes either one-to-one or one-to-many relationships with Message_to_Schedule_or_Task and a many-to-one relationship with User.

4. Message_to_Schedule_or_Task

This intermediary table has `map_id` as the primary key and includes `input_id` as a foreign key linking to `NLP_input.nlp_input_id` and `schedule_id` as a nullable foreign key linking to `Schedules.schedule_id`. The `created_at` timestamp is also included. It forms a one-to-one relationship with Schedules through the `schedule_id` reference.

5. Schedules

The Schedules table uses `schedule_id` as its primary key and includes `user_id` as a foreign key to `User.user_id`. Fields include `title`, `description`, `source`, `start_time`, `end_time`, a nullable `recurrence_rule_id` foreign key to `Recurrence_rules.recurrence_rule_id`, plus `created_at` and `updated_at` timestamps. This table has a one-to-many relationship with Notification.

6. Recurrence_rules

The Recurrence_rules table uses `recurrence_rule_id` as its primary key. Fields include `frequency`, `interval`, `by_day`, `by_month_day`, and `end_date`. It has a one-to-many relationship with Schedules.

7. Notification

The Notification table is identified by `notification_id` and includes `schedule_id` as a foreign key to `Schedules.schedule_id`. Other fields include `notify_before_minutes`, a boolean `sent` flag, and `created_at` timestamp.

8. Sleep_tracking

The Sleep_tracking table has `sleep_id` as its primary key and includes `user_id` as a foreign key referencing `User.user_id`. Additional fields include `sleep_start`, `sleep_end`, and `duration_minutes`.

9. Water_tracking

The Water_tracking table has `water_id` as its primary key and includes `user_id` as a foreign key to `User.user_id`. Other fields include `amount_ml` and `timestamp`.

10. Availability_slots

The Availability_slots table has `slot_id` as its primary key and includes `user_id` as a foreign key referencing `User.user_id`. Other fields include `start_time`, `end_time`, and `confidence_score`.

3.6 Speech-to-Schedule

The Smart Schedule AI platform's main intelligent automation mechanism is the Speech-to-Schedule pipeline. This multi-phase processing framework uses a number of specialized AI models and standardization engines to transform unstructured vocal inputs into structured calendar records. The architecture tackles issues related to implicit intention detection, cultural subtleties, and temporal vagueness in natural language scheduling queries.

3.6.1 Speech-to-Schedule Architecture Overview

The entire pipeline comprises three fundamental processing phases: (1) Automatic Speech Recognition (ASR) for audio transcription, (2) Large Language Model-driven entity extraction and intention categorization, and (3) temporal standardization and database integration. Figure 3.44 depicts the comprehensive operational workflow.

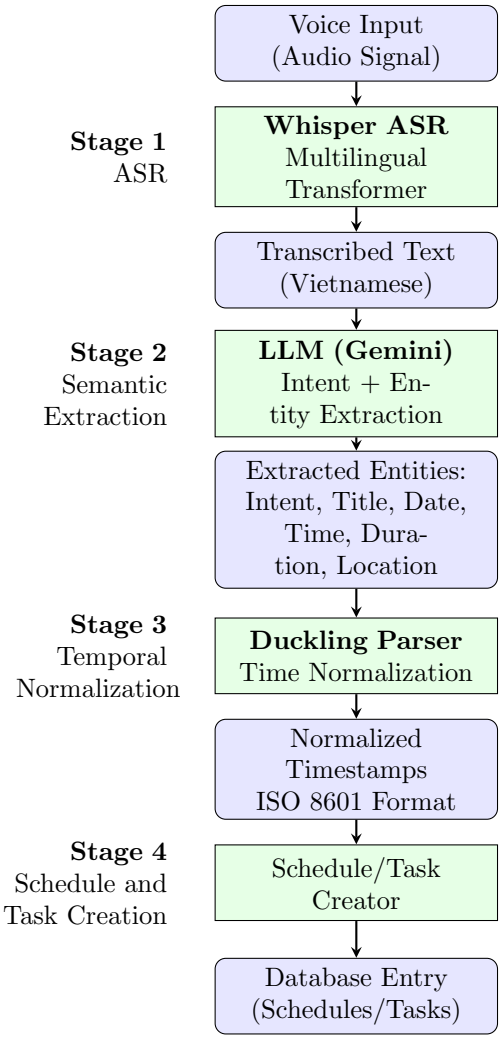

Figure 3.44: Speech-to-Schedule Processing Pipeline

Figure 3.44 presents the Speech-to-Schedule processing workflow. The constituent operations are elaborated in the subsequent sections.

3.6.2 Stage 1: Automatic Speech Recognition with Whisper

The initial phase utilizes OpenAI's Whisper model, an advanced transformer-based ASR platform trained on 680,000 hours of multilingual audio data. Whisper's design incorporates an encoder-decoder transformer configuration specifically engineered for reliable speech recognition across varied acoustic environments and languages.

Whisper Architecture and Processing

Whisper uses the following order to process audio input:

The system starts with audio preprocessing, in which an 80-channel log-Mel spectrogram computed on 25 ms windows with a 10 ms stride is created from the input audio waveform. This time-frequency representation maintains computational efficiency while capturing the spectral and temporal characteristics of speech.

Several transformer blocks make up the encoder processing phase, which uses multi-head self-attention mechanisms to analyze the spectrogram. Even in noisy environments, robust feature extraction is made possible by individual attention heads learning to focus on specific acoustic features, such as prosody, speaker characteristics, and phonetic boundaries.

The decoder generates text tokens in an autoregressive fashion during decoder processing with language modeling, conditioned on both the encoder output and previously generated tokens. Whisper is particularly suitable for Vietnamese speech recognition without explicit language designation because of its multilingual training, which enables automatic language identification.

In terms of output generation, the model produces Vietnamese text transcription with capitalization and punctuation, achieving word error rates (WER) that are on par with specialized Vietnamese ASR platforms while maintaining zero-shot generalization capabilities.

Whisper was chosen because of its multilingual dependability, ability to handle code-switching, which is common in Vietnamese conversational speech, and superior performance on domain-specific vocabulary without the need for further adjustment.

3.6.3 Stage 2: LLM-based Entity Extraction and Intent Classification

In the second stage, structured scheduling data is extracted from unstructured text using Google's Gemini large language model. This approach addresses the ambiguities present in natural language scheduling questions by utilizing the model's semantic comprehension and contextual reasoning capabilities.

Two-Layer Semantic Processing

Layer 1: Intent Recognition and Entity Extraction

A carefully constructed instruction that specifies the extraction goal and output structure is sent to the LLM. The prompt engineering method includes:

```
Extract scheduling information from the user text
and return JSON only.

Expected fields:
- intent: [create, update, delete, query]
- title: event name or description
- date_phrase: temporal reference (e.g., "ngày mai", "thứ 6 tuần sau")
- time_range: start and end time if specified
- duration: event length if specified
- location: place information (optional)

If ambiguous, infer the most common meaning in Vietnamese culture.
Return only valid JSON without explanation.
```

This instruction is analyzed by the LLM using its transformer framework, which works as follows:

The input text is tokenized into subword units and mapped to high-dimensional embeddings that encode semantic relationships discovered through pretraining on large text collections during the tokenization and embedding process.

The model can understand long-range dependencies because the multi-head attention mechanism calculates associations between all token pairs. For example, the model uses attention weights to link "3 giờ chiều" and "thứ 6 tuần sau" in "Tạo lịch họp với khách hàng vào 3 giờ chiều thứ 6 tuần sau" (Create a meeting with client at 3 PM next Friday).

Gemini's framework resolves Vietnamese-specific temporal references by integrating linguistic and cultural knowledge from training data through contextual understanding. For instance, according to Vietnamese cultural customs, "chiều mai" (tomorrow evening) automatically corresponds to evening hours (usually 18:00-22:00).

Through constrained decoding, the model generates JSON-formatted output for structured output generation, ensuring syntactic validity and schema adherence.

Example Processing

Input: *"Nhắc tôi họp team vào 2 giờ chiều mai"*

LLM Output:

```
{
  "intent": "create",
  "title": "Họp team",
  "date_phrase": "mai",
  "time_range": "14:00",
  "duration": null,
  "location": null
}
```

3.6.4 Stage 3: Temporal Normalization with Duckling

In the third step, we have to take the time relative to the temporal expressions and turn them into absolute time stamps with the help of Duckling, a grammar parser built on Probabilistic Context-free Grammar.

Duckling's Parsing Mechanism

Duckling works based on a rule of thumb semantic parsing coupled with probability from the user's built time stamps Duckling. Parsing of time expressions by the system is first based on tokenization, and matching expressions with an inventory of verbatim temporal terms, such as: "mai" → tommarrow, "tuần sau" → next week, "tháng tới" → next month. Then, grammar parsing is done on the basis of duckling context-free and rules for composing temporal phrases into simple expressions. Specifications of such rules are, time of day can be expressed as "hôm nay", "mai", or "ngày kia"; a time with an hour "giờ" along with its minute in "phút"; and a time and date can be in any order. Then, on reference time the system resolves all expressions to an absolute time from relative expressions by doing arithmetic, and based on the phrase for example "mai" is the next date to reference, "tuần sau" is the date plus two weeks, and "2 tiếng"

is a few hour taken to mean a two hour period. Duckling resolves and parses time expressions based on probability and order based on context needed.

Example Normalization Process

Input: `{date_phrase: "mai", time_range: "14:00"}`

Duckling Processing:

- Reference time: `2025-11-29T10:30:00+07:00`
- "mai" → `2025-11-30`
- "14:00" → `14:00:00`
- Final output: `2025-11-30T14:00:00+07:00`

3.6.5 Stage 4: Schedule and Task Creation

The last step pours the structured data into the relational database as per the schema described in Section X. The system checks the extracted textttintent and the presence of temporal constraints to decide whether to create a textttSchedule (for time-boxed events) or a textttTask (for actions without time constraints).

Database Insertion Logic

```
if intent == "create":
    if time_range is not None:
        create_schedule(user_id, title, start_time, end_time, ...)
    else:
        create_task(user_id, title, description, ...)
elif intent == "update":
    update_schedule_or_task(entity_id, updated_fields)
elif intent == "delete":
    delete_schedule_or_task(entity_id)
elif intent == "query":
    return query_results(user_id, filters)
```

Furthermore, the system populates the `Message_to_Schedule_or_Task` mapping table to preserve traceability between the original NLP input and the produced database entry, facilitating explainability and debugging of the AI workflow.

3.6.6 Error Handling and Validation

The system has ASR validation checkpoints that trigger a bypass when confidence scores dip below particular levels, invoking user clarifications. Expected LLM results undergo JSON validation to determine schema compliance in complete, well-defined fields. Duckling outputs are validated using business logic to ensure functionality. New date flags, invalid date ranges of operations, and date ranges that exceed an operational window are all eliminated. Consistency in the database remains intact by employing atomicity around all data manipulations. The system was intentionally designed to allow for multiple forms of user input Yet having the validation system in place strengthens the system by making it more versatile and reliable.

3.7 Wireframe

Prior to the implementation phase, the user interface (UI) design was established to ensure a clear understanding of the system structure and intended functionality. Based on the system diagrams and the identified functional modules, the UI was designed to reflect the logical organization of the application.

The interface design follows key Gestalt principles to promote visual clarity, consistency, and intuitive interaction. UI components are grouped and arranged according to their functional relationships, enabling users to easily perceive structure and hierarchy across the application.

The overall UI flow is organized into three primary modules: Homepage, Schedule Management, and Task Management. This ordering reflects typical user interaction patterns and supports a smooth and coherent navigation experience. The following sections present a detailed description of each UI screen and its corresponding functionality.

HomePage

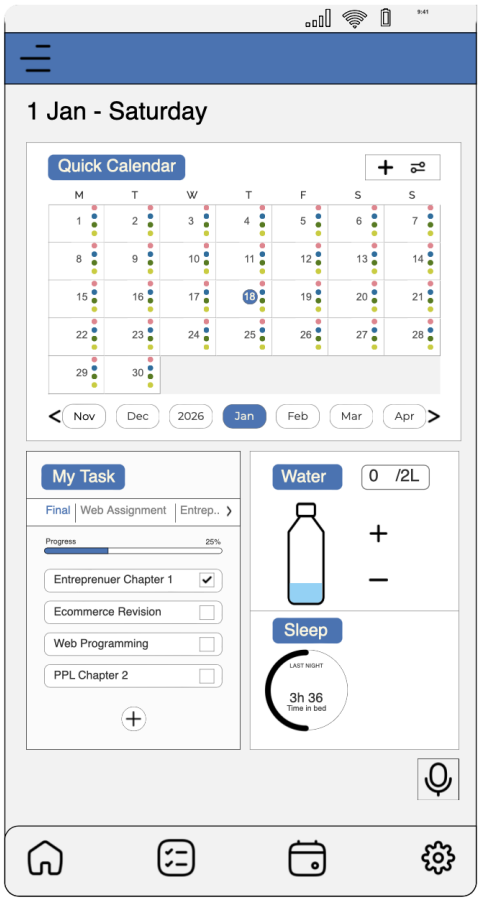

Figure 3.45: Home Page Wireframe

The Home Page in **Figure 3.45** functions as the central dashboard of the application and is the first screen presented to users upon launching the app. It provides a concise overview of daily schedules, tasks, and personal wellness metrics in a single view.

At the top of the screen, a quick-access monthly calendar displays color-coded entries from different categories, enabling users to visualize upcoming events at a glance. Below the calendar, the task panel presents current tasks along with progress indicators to support efficient task tracking. On the right side of the interface, health-related widgets display water intake and sleep information, helping users maintain awareness of their daily well-being.

Daily Schedule Display

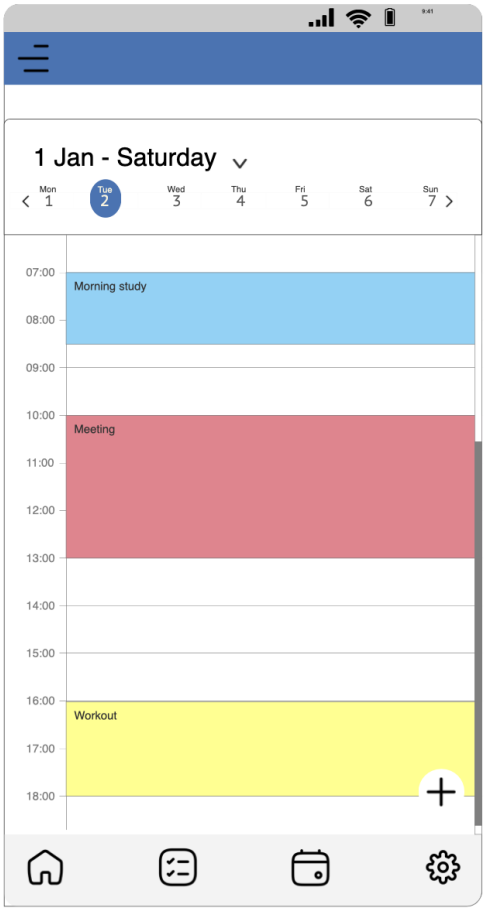

Figure 3.46: Daily Schedule Wireframe

The Schedule Page in **Figure 3.46** provides an hour-by-hour visualization of the user's daily activities, with the current day displayed as the default view. Users can filter and navigate schedules by day to review past or upcoming activities.

At the top of the screen, the selected date is prominently displayed along with a horizontal day selector, enabling quick navigation between days of the week. The main content area presents a vertical timeline in which scheduled events appear as color-coded blocks corresponding to different activity categories. This design allows users to easily understand the structure of their day, identify available time slots, and manage overlapping commitments.

A floating action button is provided to allow users to add new events instantly, supporting efficient and flexible schedule planning. Overall, the Schedule Page offers a clean and intuitive interface for effective time management and daily routine organization.

Weekly Schedule Display

Figure 3.47: Weekly Schedule Wireframe

The Weekly Schedule Display in **Figure 3.47** provides a comprehensive overview of the user's activities across an entire week. At the top of the screen, the selected date and weekday are displayed together with a horizontal week selector, allowing users to navigate between different weeks efficiently.

The main interface adopts a grid-based layout in which each column represents a day of the week and each row corresponds to a specific hour. Scheduled events are displayed as color-coded blocks positioned within their respective time slots, enabling users to quickly compare workloads, identify recurring patterns, and assess availability throughout the week.

This layout supports effective planning by offering a clear visualization of how tasks, study sessions, and personal activities are distributed over multiple days. A floating action button located at the bottom-right corner allows users to conveniently add new events from within the weekly view.

View Options Panel

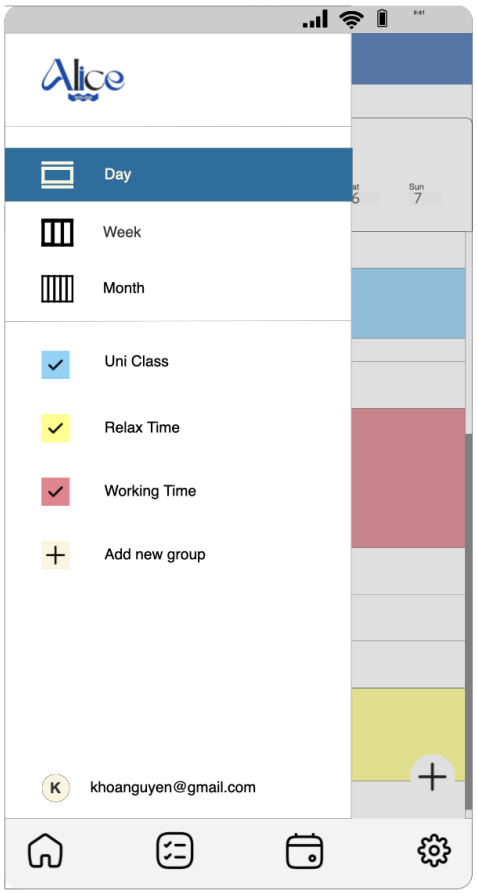

Figure 3.48: View Options Panel Wireframe

The View Options Panel in **Figure 3.48** provides users with quick access to calendar display modes and category-based filters. The upper section of the panel allows users to switch seamlessly between Day, Week, and Month views, enabling flexible navigation across different levels of scheduling detail.

Below the view selectors, a list of color-coded activity categories—such as Uni Class, Relax Time, and Working Time—allows users to control the visibility of specific categories within the schedule. This functionality enhances visual clarity by enabling users to focus on selected types of activities when planning their day or week. The panel also includes an option for creating new activity categories, supporting personalized organization.

At the bottom of the panel, the user's profile information, including the account name and email address, is displayed to reinforce user identity within the application. Overall, the View Options Panel provides intuitive and efficient control over schedule visualization and filtering.

Add Schedule Dialog

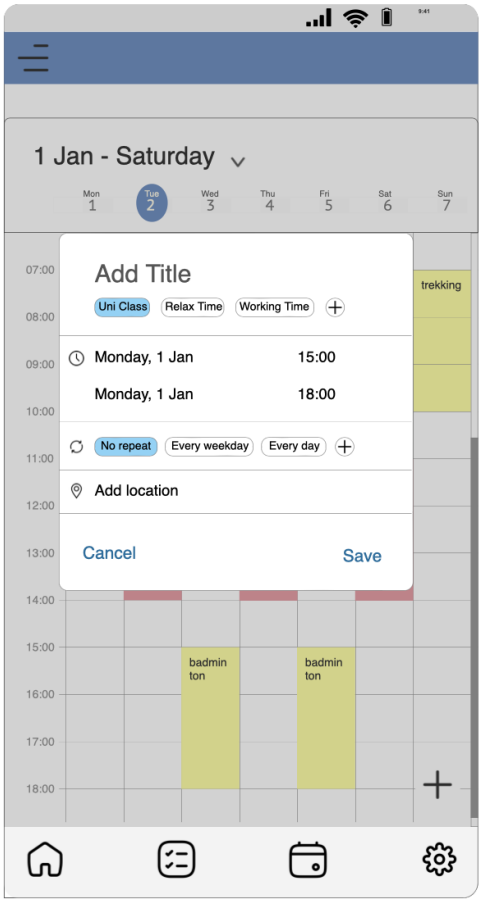

Figure 3.49: Add Schedule Dialog Wireframe

The Add Schedule dialog in **Figure 3.49** is presented as a centered pop-up overlay on top of the current schedule view, allowing users to create a new event without navigating away from the page. At the top of the dialog, a title input field enables users to specify the name of the activity. This is followed by selectable category tags, such as *Uni Class*, *Relax Time*, and *Working Time*, with an additional option to create a new category.

The central section of the dialog provides controls for configuring the start and end date and time of the event, ensuring accurate and precise scheduling. Below this section, repeat options (e.g., *No repeat*, *Every weekday*, and *Every day*) allow users to define recurring events. A location input field is also available for specifying where the activity takes place.

At the bottom of the dialog, *Cancel* and *Save* actions are provided, enabling users to either discard changes or confirm the creation of the new schedule entry. Overall, the dialog design streamlines the event creation process while maintaining visibility of the underlying schedule context.

Edit & Remove Schedule Dialog

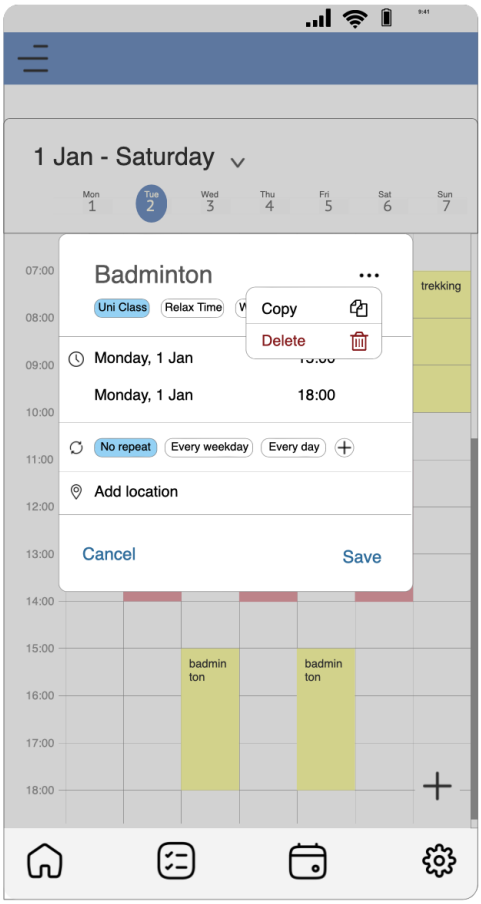

Figure 3.50: Edit and Remove Schedule Dialog Wireframe

The Edit and Remove Schedule dialog in **Figure 3.50** allows users to modify or delete an existing calendar event. Similar in structure to the Add Schedule dialog, it displays the event title at the top along with the assigned activity categories, such as *Uni Class*, *Relax Time*, and *Working Time*. A contextual options menu, accessed via a three-dot icon, provides additional actions including *Copy* for duplicating the event and *Delete* for removing it from the schedule.

The central section of the dialog contains start and end date–time fields that enable users to adjust the event duration. Repeat options are also provided, allowing users to modify recurrence patterns such as *No repeat*, *Every weekday*, or *Every day*. A location input field is included for adding or updating event details.

At the bottom of the dialog, *Cancel* and *Save* buttons offer clear options to discard changes or confirm modifications. Overall, this dialog supports efficient schedule editing and removal while maintaining consistency and usability across the scheduling interface.

Task Management

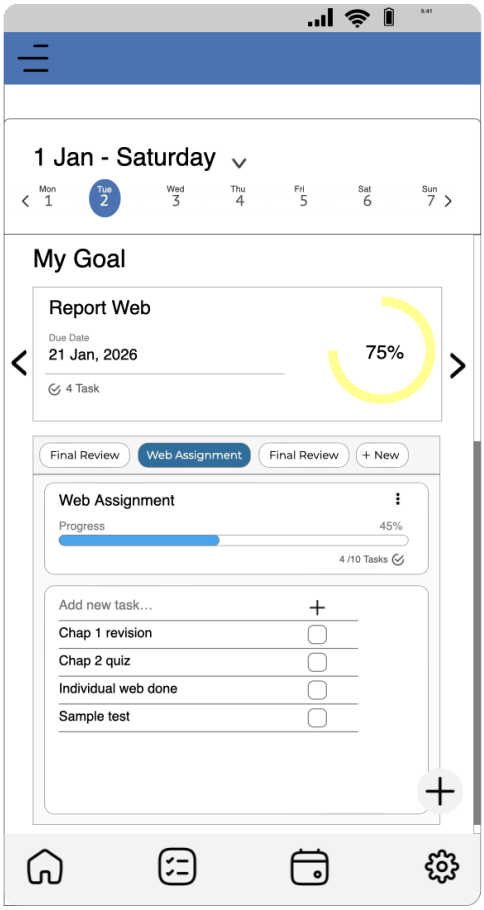

Figure 3.51: View Task Management Wireframe

The *Task Management* in **Figure 3.51** interface enables users to organize, monitor, and update progress toward their goals and associated tasks efficiently. Each goal is displayed with a clear title, due date, and a visual progress indicator, allowing users to quickly assess overall completion status. Related task items, such as assignments, quizzes, or revision activities, are grouped under each goal to support structured navigation and task organization.

Within a selected goal, users can switch between task categories or stages using a segmented control (e.g., *Final Review*, *Web Assignment*, *New*). Each category displays its associated task list with completion checkboxes and progress indicators that update dynamically. A contextual options menu, accessible via a three-dot icon, allows users to edit task details or manage task visibility.

New tasks can be added using the "Add new task..." input field and quick-add button at the bottom of the interface. Tasks are listed with checkboxes to enable fast and intuitive progress updates.

Overall, the Task Management module supports users in breaking down larger goals into manageable steps, tracking progress visually, and adjusting tasks as needed, while maintaining a clean, intuitive, and consistent user experience across the application.

Remove Task

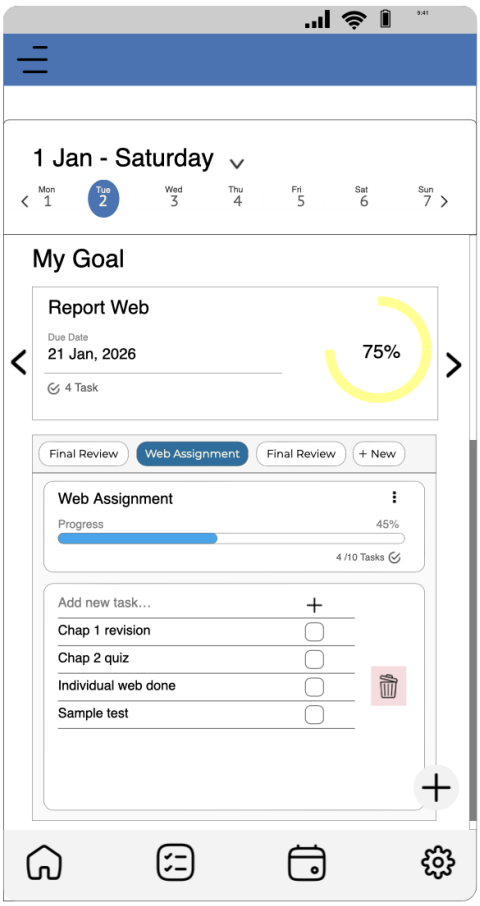

Figure 3.52: Remove Task Wireframe

The *Remove Task* in **Figure 3.52** function allows users to efficiently delete unnecessary tasks from their goal or assignment list. Tasks can be removed using a horizontal swipe gesture, which reveals a delete action represented by a bin icon.
Selecting the delete action removes the task from the list immediately. This gesture-based interaction minimizes navigation steps and supports a clean, responsive interface, enabling users to maintain an organized and up-to-date task list.

Chapter 4

Work Evaluation and Future Improvement

In Chapter 4, the authors conclude the specialized project. This chapter consists of two parts: an evaluation of the work completed in the first phase and proposed improvements for the next phase.

4.1 Work Evaluation

Throughout the project implementation process, our team organized the workflow effectively and clearly divided responsibilities among members. By applying the Agile Scrum methodology with iterative cycles of planning, implementation, reporting, and evaluation, and by using the Trello task management platform, the team was able to manage project requirements while balancing workloads from other subjects. Regular internal reviews and reporting sessions with the instructor enabled timely feedback and guided subsequent development iterations.

During the project, the team encountered several challenges, particularly in system design and the development of logical system views derived from requirements. In addition, team members faced constraints due to differing priorities across other major subjects. To address these issues, the team adapted its collaboration approach by increasing the frequency of meetings from one session per week to two or three sessions when necessary. These additional meetings provided opportunities for members to share difficulties, discuss technical problems, and collaboratively identify solutions, which contributed to improved project outcomes.

The project successfully delivered all required system documentation and analysis. Furthermore, the team selected an appropriate technology stack, designed application UI mockups, and implemented the voice-handling feature using an AI-based model. These deliverables demonstrate both the technical progress achieved and the effectiveness of team collaboration.

From an evaluation perspective, the implemented system satisfies the core functional requirements defined during the analysis phase, including task management, schedule management, and voice-based schedule creation. Functional validation through iterative development and instructor feedback confirmed that the primary user workflows operate as intended. While large-scale performance testing was beyond the scope of this academic project, the system exhibited

stable behavior in a controlled testing environment. A key limitation lies in the accuracy of voice interpretation for complex or ambiguous inputs, particularly those involving overlapping temporal expressions, which indicates directions for future refinement of the natural language processing component.

Overall, the project provided valuable experience in software engineering practices and system development. The outcomes achieved form a solid foundation for further enhancement and future development of the proposed work-management application.

4.2 Future Improvements

In the next development phase, the team plans to implement the application based on the completed system documentation and the selected technology stack. Initial development may progress at a moderate pace as efforts are concentrated on studying the chosen technologies, establishing a robust system architecture, and optimizing the overall code structure. Once these foundational components are stabilized, development efficiency is expected to improve significantly.

To enhance system reliability and software quality, a dedicated testing environment will be established. The team plans to adopt continuous integration and continuous deployment (CI/CD) practices to support a more reliable development workflow, reduce integration issues, and shorten development cycles. Knowledge related to DevOps practices, software testing, and mobile application development will be continuously reinforced throughout weekly development iterations. Following the Agile methodology, system features will be regularly reviewed and refined across future sprints, particularly those that were not fully specified during earlier phases.

In addition to technical enhancements, the team intends to further explore concepts related to work–life balance, productivity techniques, and effective personal management. These insights will support the design of features that are both practical and aligned with contemporary user needs.

Finally, while delivering a functional and reliable application remains the primary objective, the development process itself will continue to emphasize both technical and soft skill improvement. Regular weekly meetings will be used to reinforce individual learning, encourage knowledge sharing, and facilitate constructive feedback. Through this iterative and collaborative approach, the team expects to achieve continuous improvement in both the project outcomes and overall development competence.

References

[1] AWS (n.d.). What is Flutter?. Retrieve from: https://aws.amazon.com/what-is/flutter/

[2] Bonnie Eisenman (2015) What is React Native?. Retrieve from: https://www.oreilly.com/library/view/learning-react-native

[3] Ben Lutkevich (2022) Definition Kotlin?. Retrieve from: https://www.techtarget.com/whatis/definition/Kotlin/

[4] Wikipedia. Django (web framework). Retrieve from: https://en.wikipedia.org/wiki/Django_(web_framework)

[5] Django (n.d.). Official Website. Retrieve from: https://www.djangoproject.com/

[6] W3schools (n.d.). Django Introduction. Retrieve from https://www.w3schools.com/django/django_intro.php

[7] Wikipedia (n.d.). MongoDB. Retrieve from: https://en.wikipedia.org/wiki/MongoDB

[8] Mark Richards, Neal Ford (2020). Fundamentals of Software Architecture. O'REILLY

[9] Google Cloud (n.d.). What is Microservices Architecture? Retrieve from: https://cloud.google.com/learn/what-is-microservices-architecture

[10] BMI (n.d.). What is Java Spring Boot?. Retrieve from: https://www.ibm.com/think/topics/java-spring-boot

[11] Wikipedia (n.d.). Node.js. Retrieve from: https://en.wikipedia.org/wiki/Node.js

[12] Node.js (n.d.). About Node.js®. Retrieve from: https://nodejs.org/en/about

[13] FastAPI (n.d.). Official Website. Retrieve from: https://fastapi.tiangolo.com/

[14] Wikipedia (n.d.). FastAPI. Retrieve from: https://en.wikipedia.org/wiki/FastAPI

[15] Wikipedia (n.d.). Document-oriented database Retrieve from: https://en.wikipedia.org/wiki/Document-oriented_database

[16] Wikipedia (n.d.). PostgreSQL. Retrieve from: https://en.wikipedia.org/wiki/PostgreSQL

[17] PostgreSQL (n.d.). Official Website. Retrieve from: https://www.postgresql.org/

[18] Nearform (2023). Using Google Calendar with Natural Language via LangChain. Retrieve from: https://nearform.com/insights/using-google-calendar-with-natural-language-via-langchain/

[19] OpenAI (2022). Introducing Whisper. Retrieve from: https://openai.com/index/whisper/

[20] Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. Retrieve from: https://github.com/openai/whisper

[21] Sleep Cycle (n.d.). The Technology That Powers Sleep Cycle's App. Retrieve from: https://www.sleepcycle.com/features/the-technology-that-powers-sleep-cycle/

[22] Notion (2024). Introducing Notion Calendar - An Integrated Calendar for Work and Life. Retrieve from: https://www.notion.com/blog/introducing-notion-calendar

[23] Khadivi, M. et al. (2023). Deep Reinforcement Learning for Machine Scheduling: Methodology, the State-of-the-Art, and Future Directions. Retrieve from: https://arxiv.org/abs/2310.03195

[24] OpenAI (2022). Introducing Whisper. Retrieve from: https://openai.com/index/whisper/

[25] Radford, A., Kim, J.W., Xu, T., Brockman, G., McLeavey, C., Sutskever, I. (2022). Robust Speech Recognition via Large-Scale Weak Supervision. Retrieve from: https://github.com/openai/whisper

[26] Google (2023). Introducing Gemini: Google's most capable AI model yet. Retrieve from: https://blog.google/technology/ai/google-gemini-ai/

[27] Wikipedia (n.d.). Gemini (language model). Retrieve from: https://en.wikipedia.org/wiki/Gemini_(language_model)

[28] Facebook (n.d.). Duckling - Language, engine, and tooling for expressing, testing, and evaluating composable language rules on input strings. Retrieve from: https://github.com/facebook/duckling

[29] The Wit.ai Team (2017). Open Sourcing our new Duckling. Retrieve from: https://medium.com/wit-ai/open-sourcing-our-new-duckling-47f44b776809

Appendix A

Appendix: Task Assignment

Nguyen Anh Khoa

- Requirement analysis
- Functional Requirements
- Mobile Application Technology Research
- Use Case Design of Schedule Management Module
- Use Case Scenario of Schedule Management Module
- Activity Diagram Design
- Wireframe Design
- Mockup Design

Tran Chi Tai

- Project Scope Define
- Non-Functional Requirements
- NLP theory research
- Speech-to-text design, implementation and testing
- Related Work Research
- Use Case Design of Task Management Module
- Use Case Scenario of Task Management Module
- Architecture Diagram Design
- Class Diagram Design
- ERD Design

Dinh Ba Khanh

- Domain Context Analysis
- Mobile Application Technology Research
- Support Chatbot theory, design, implementation and testing
- Backend Technology Research
- Use Case Design of Authentication and Personal Information
- Use Case Scenario of Authentication and Personal Information
- Sequence Diagram Design
- Component Diagram Design

Appendix B

Appendix: Timeline

Sprint 1 25/08/2025 - 07/09/2025

- Domain Skate Hole
- Requirement Analysis

Sprint 2 08/09/2025 - 21/09/2025

- Tech Stack Discussion and Choose
- NLP Theory Reseach
- Use Case Diagram design
- Requirement Redefine

Sprint 3 22/09/2025 - 05/10/2025

- Use Case Scenario Implement
- Tech Stack Analysis

Sprint 4 06/10/2025 - 19/10/2025

- Activity Diagram Analysis
- Sequence Diagram Analysis

Sprint 5 20/10/2025 - 02/11/2025

- Component Diagram Analysis
- Entity-Relationship Diagram Analysis
- Wireframe Design

Sprint 6 03/11/2025 - 16/11/2025

- Entity-Relationship Diagram Redesign
- Class Diagram Analysis
- Wireframe Design

Sprint 7 17/11/2025 - 30/11/2025

- Speed-to-text Demo Implement
- Mockup Design

Sprint 8 01/12/2025 - 14/12/2025

- Speed-to-text Demo Implement
- Mockup Design