CS30500 - Introduction to Software Engineering
Spring 2025

KAIST

Software Requirements Specification

KAISTime

AI-Powered Smart Queue Tracker

Team META4 Members	Student ID
Janghyun Kim	20200137
Javad Ismayilzada	20220830
Julien Dupont	20256086
Luke Foster	20256104
Marco Miglioranza	20256121

Date: April 28, 2025

Contents

1. Introduction

1.1 High Level Introduction of the KAISTime Software System (KSS)

This Software Requirement Specification (SRS) document provides a detailed overview of the KAISTime software system (AI-Powered Smart Queue Tracker) in terms of Functional & Non Functional requirements. KAISTime is a system that tracks queue times and the occupancy of shared spaces on the Korea Advanced Institute of Science and Technology (KAIST) campus. It helps students and other people on campus optimize their time and schedules by providing a real-time view of campus crowd levels.

The system will be app-driven, allowing users to access queue times and occupancy levels, as well as get useful information about key locations on campus. Based on a user's favorite places, the system will send notifications to alert them when a location becomes crowded or less busy. On the server side, an AI model will process data in real time and predict waiting times using historical patterns. Based on this real-time analysis, the system will both notify users and update information within the mobile application.

1.2 Definitions, acronyms and abbreviations

Below table lists the various definitions, acronyms and abbreviations used throughout the SRS document.

Term	Definition
KAIST	Korea Advanced Institute of Science & Technology
User	Someone who interacts with the mobile phone application
Admin/Administrator	System administrator who is given specific permission for managing and controlling the system
Facility staff member	Someone who is part of a place on campus tracked by the system and wants his facility to be a part the application
Stakeholder	Any person who has interaction with the system who is not a developer
KSS	KAISTime Software System
AI	Artificial Intelligence
IoT	Internet of Things
API	Application Programming Interface

CSV	Comma Separated Value
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HTTPS	HTTP Secure
JSON	JavaScript Object Notation
REST	Representational State Transfer
UI/UX	User Interface/User Experience
FR	Functional Requirement
SRS	Software requirements specification
RAM	Random Access Memory
TLS	Transport Layer Security
IDE	Integrated Development Environment
JWT	JSON Web Token

1.3 Contextual overview: the need for Queue Tracking at KAIST

The KAIST campus is a dynamic and highly frequented academic environment where thousands of students, professors, researchers, and staff move between classes, cafeterias, libraries, laboratories, and common areas on a daily basis. This high density of people, especially during peak hours, frequently results in long queues, overcrowded spaces, and inefficient use of facilities. These bottlenecks are not only a source of inconvenience but also contribute to lost time, increased stress, and reduced productivity, particularly during critical periods such as exam weeks. Students may spend excessive time waiting in line for meals or searching for available study spaces, while staff may struggle to anticipate and manage fluctuating demand. There is currently no unified system to provide real-time information or predictive insights regarding queue lengths and space occupancy across campus. The absence of such a system limits the ability of people to plan their time effectively and of facility managers to allocate resources efficiently.

Moreover, KAIST is a large campus, with many facilities spread out over a wide area. This geographic dispersion makes it challenging for students, professors, and international exchange students to easily access accurate and up-to-date information about campus services. For newcomers or visitors unfamiliar with the campus map, navigating and locating relevant facilities can be confusing and time-consuming.

The KSS is designed to address these challenges, by using IoT sensors with AI models, to provide real-time queue estimations, space occupancy predictions, and practical information such as opening hours, daily menus, and prices. This allows campus users to make more informed decisions about where and when to go, improving daily routines, reducing congestion, facilitating navigation, and enhancing the overall campus experience.

1.4 Scope & Features of the System

The system is designed to perform the following tasks:

- **Provide queue and occupancy information:** users on KAIST campus will receive live updates on the crowd levels and estimated waiting times at various facilities such as cafeterias, libraries, and study spaces.
- **Deliver up-to-date facility information:** the system offers current details about key places on campus (daily cafeteria menu, opening hours, availability), helping users plan their day more efficiently
- **Display an interactive campus map:** users can access a map of KAIST campus within the mobile app, allowing them to easily locate facilities and know their current occupancy levels.
- **Provide an interface for staff to manage facility information:** facility staff members will have access to an admin interface to update relevant information (daily menus, special events, or opening hours). They will also be able to view historical crowd levels and AI-generated predictions about future occupancy.

1.5 Stakeholders & Users

The system will have interactions with the following types of Stakeholders and Users, who benefit from its features in different ways:

- **People on campus daily** *(e.g., students, professors, employees)*: These users will use the system frequently to manage their daily routines on campus. They rely on it to better organize their schedules, avoid crowded areas, and receive real-time information during peak hours. These users will primarily interact with the system by receiving real-time mobile push notifications about the availability of their favorite places or by checking directly on the app the current waiting times and crowd level predictions for each location. They can also interact with the system by checking the app for practical daily information, such as cafeteria menus, to help them plan their meals. For example, during exam weeks, these users may heavily rely on the system to find available and quiet study spaces. Finally, they can use the map interface of the software to discover new or lesser-known locations on campus.

- **Facility staff members** *(e.g., restaurant or library staff)*: These users are responsible for managing specific campus spaces tracked by the system. They access the platform through a dedicated staff interface or dashboard, where they can update relevant information such as daily menus, opening hours, prices, and real-time status (e.g., availability, capacity). This allows them to keep people on campus informed and increase their visibility, potentially attracting new visitors to their facility. They may also consult occupancy predictions to optimize their services and improve space management. Additionally, they can receive advance notifications (e.g., the day before) about the predicted crowd levels of their facilities to better anticipate demand. Updates made by facility staff (e.g., availability or occupancy adjustments) can trigger new notifications to users if the updated status satisfies user-defined thresholds.

- **People from outside the campus** *(e.g., visiting family members, high school students)*: These users interact with the campus occasionally, so they are not the primary target of our software. However, they can still benefit from it. Since they are unfamiliar with the campus, their usage will differ slightly, they will primarily rely on the map interface to navigate the campus more easily and to locate dining or study spots. They may also use filters to find locations that accommodate specific dietary constraints or preferences.

- **Admin:** These users will monitor the system's functionality and data accuracy.

1.6 Overall Goal

Overall goal of the system is to:

- Captures data from multiple IoT sensors, to get various queue characteristics such as video, noise levels, and distance, to detect current queue conditions

- Process data using an AI model to accurately predict queue times and occupancy levels at various campus facilities, in a low latency

- Inform users with timely notifications and updates, using a clear and intuitive mobile interface, providing quick and easy access to information such as campus map, waiting times, and facility details

1.7 References

- IEEE Software Engineering Standards Committee, “IEEE Std 830-1998, IEEE Recommended, Practice for Software Requirements Specifications”, October 20, 1998
- Prof. In-Young Ko,”CS30500: Introduction to Software Engineering Lecture #11”
- Roger S. Pressman, and Bruce R. Maxim, “Software Engineering – A Practitioner’s Approach”, 9th Ed., Mc Graw Hill, 2020

- Ian Sommerville, “Software Engineering” (9th edition), Addison-Wesley, 2011
- KAIST Campus Map and Facilities Information, Internal Source, 2025
- Project Proposal KAIST team meta4, Internal Source, 2025
- KAIST team meta4 Mid term presentation, Internal Source, 2025.

Referred Tool Used

- OpenAI. ChatGPT, https://openai.com/chatgpt.
- UML Diagrams Generations. https://plantuml.com.

2. Overall Description

2.1 Product Perspective

To support a shared understanding of the KAISTime system's structure and functionality, the following diagrams have been developed: a Context Diagram, a Class Diagram and a Architecture Diagram. These visual representations clarify how the system interacts with external entities and how internal components are structured.

2.1.1 Context Diagram

The context diagram illustrates the KAISTime system as a single high-level process and identifies all external entities that interact with it. These include users (such as students, visitors, KAIST staff, and administrators) and external systems (such as Firebase, IoT Devices, and AI Libraries). It depicts the system's five primary functions: estimating queue lengths, sending timely queue notifications, visualizing crowd density through heatmaps, analyzing historical usage data, and managing user and system data.

By highlighting data flows between the system and its environment, the context diagram provides a clear overview of the system's scope and integration points, helping stakeholders understand the boundaries and responsibilities of the system.

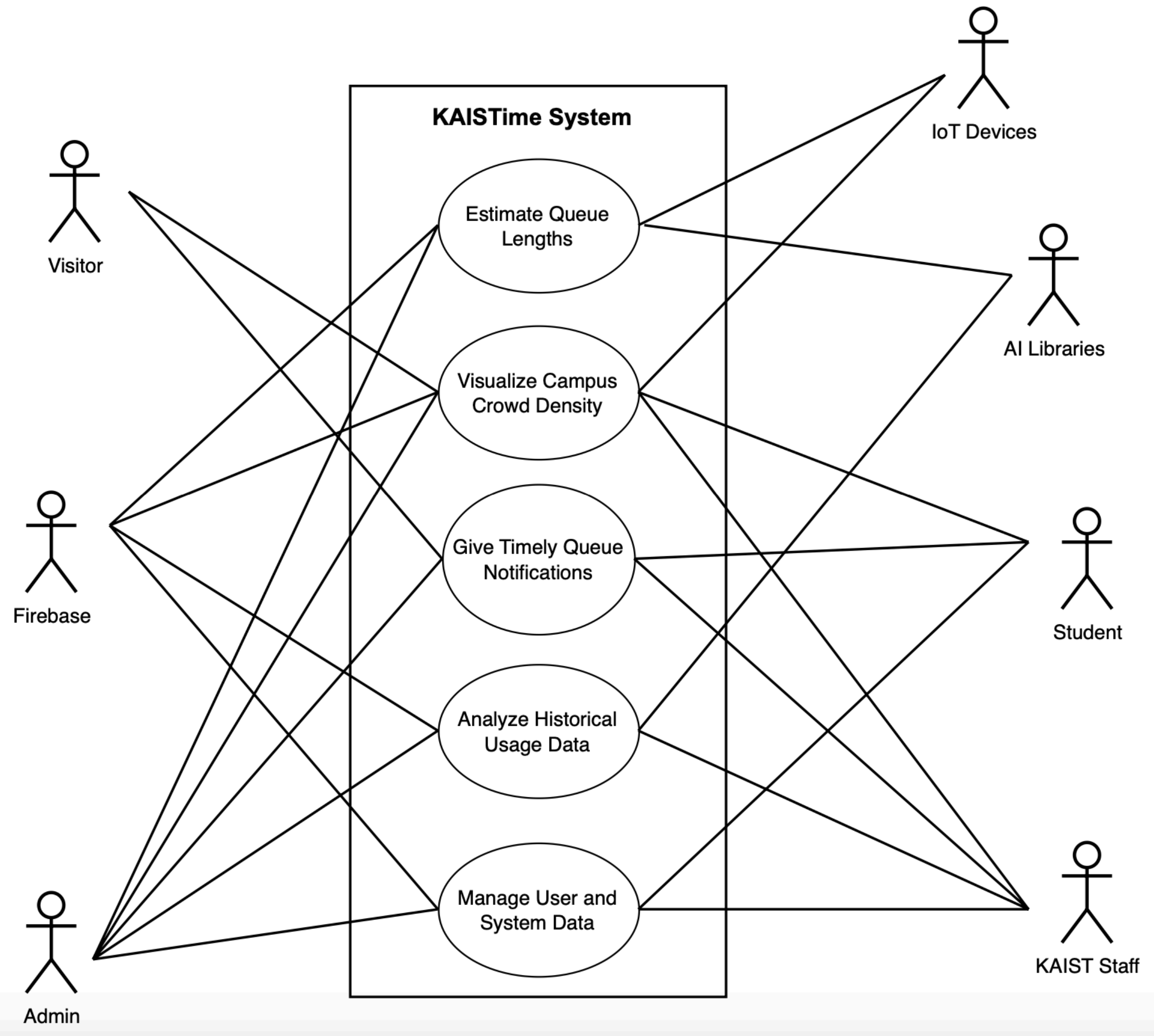

KAISTime System
Estimate Queue Lengths
Visualize Campus Crowd Density
Give Timely Queue Notifications
Analyze Historical Usage Data
Manage User and System Data
Visitor
Firebase
Admin
IoT Devices
AI Libraries
Student
KAIST Staff

Figure1: Use Case Diagram of KAISTime System

2.1.2 Block Diagram

The block diagram shown on the following page represents the internal structure of the KAISTime System in terms of its key software entities and their relationships. Each block corresponds to a core class within the system and includes relevant attributes and methods that define its role and responsibilities. The diagram also illustrates inheritance relationships, associations, and compositions between different blocks.

At the center of the system is a generalized User class, which is inherited by four specific user types: Student, Visitor, Staff, and Admin. These subclasses reflect the varying roles and permissions within the system. For example, students can manage preferences and receive personalized recommendations, while admins can manage users and system data.

Supporting these users are several key system components:

- **QueueData**, which stores live queue information
- **Location**, which represents physical areas on campus
- **HeatMap**, used for crowd density visualization
- **Preferences**, which capture user-specific data such as dietary needs or favourite places
- **Notification**, used to communicate alerts or suggestions
- **UsageReport**, which supports analytics for decision-making

Each block contains both attributes (data) and operations (behaviors), and relationships such as composition and aggregation are used to model how classes depend on or contain one another. For example, a User has a Preferences object, and QueueData is associated with a Location.

This block diagram provides a high-level structural view of the system's data model and object-oriented design, serving as a blueprint for future implementation

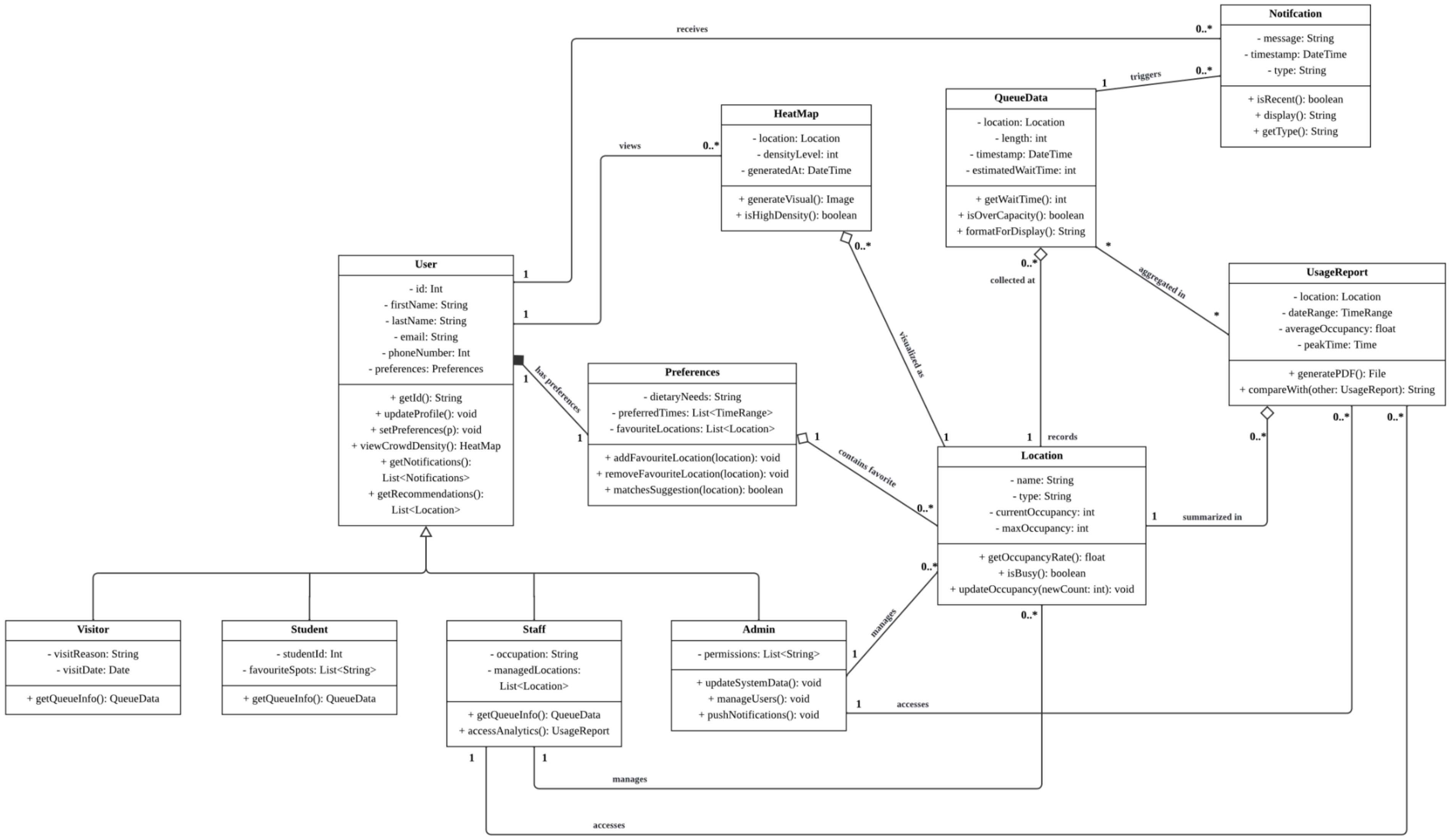


Figure2: Block Diagram of the System

2.1.3 Architecture Diagram

The architecture diagram below provides an overview of the design and core components of our system. We chose to adopt a microservices architecture following our agile methodology. Our system is structured in distinct layers, each responsible for specific functions:

The hardware layer is designed to collect diverse information about the facility environment, such as the number of people in the queue and seat occupancy. This is a crucial part of the system, as we will use this information to process data, make predictions, and inform users. Various hardware components and sensors will be installed throughout the different facilities to capture the necessary data. The collected data will then be preprocessed, formatted, and transmitted to our server via Wi-Fi.

The backend server acts as the central orchestrator, exposing an API to provide data for the mobile application. It receives incoming data from the hardware layer, and coordinates requests from the mobile application, and interacts with the database server to store data, AI server to manage predictions, and the cloud notification server. The backend will be responsible for handling authentication of the user, authorization, and all the business logic of the application.

The AI server is responsible for data analytics and machine learning tasks. Incoming video and sensor data is processed and formatted to perform model prediction. We are also storing the data in the database to train and improve our model and new data.

The database server stores all persistent data, including historical sensor readings for improvement of the model, user account and information, facility metadata, and AI model outputs. For real-time notifications and updates, the system will use a cloud storage and notification service (such as Firebase). Integration with an external API will enable the map feature, allowing users to visualize facility locations on campus

Lastly, the mobile application will serve as the frontend of our software and will be a crucial part of the development process. It will interact with the backend via APIs to retrieve data, with the external API to display the campus map, with the AI prediction service to obtain accurate estimates, and with the cloud server to handle real-time notifications.

Architecture Diagram

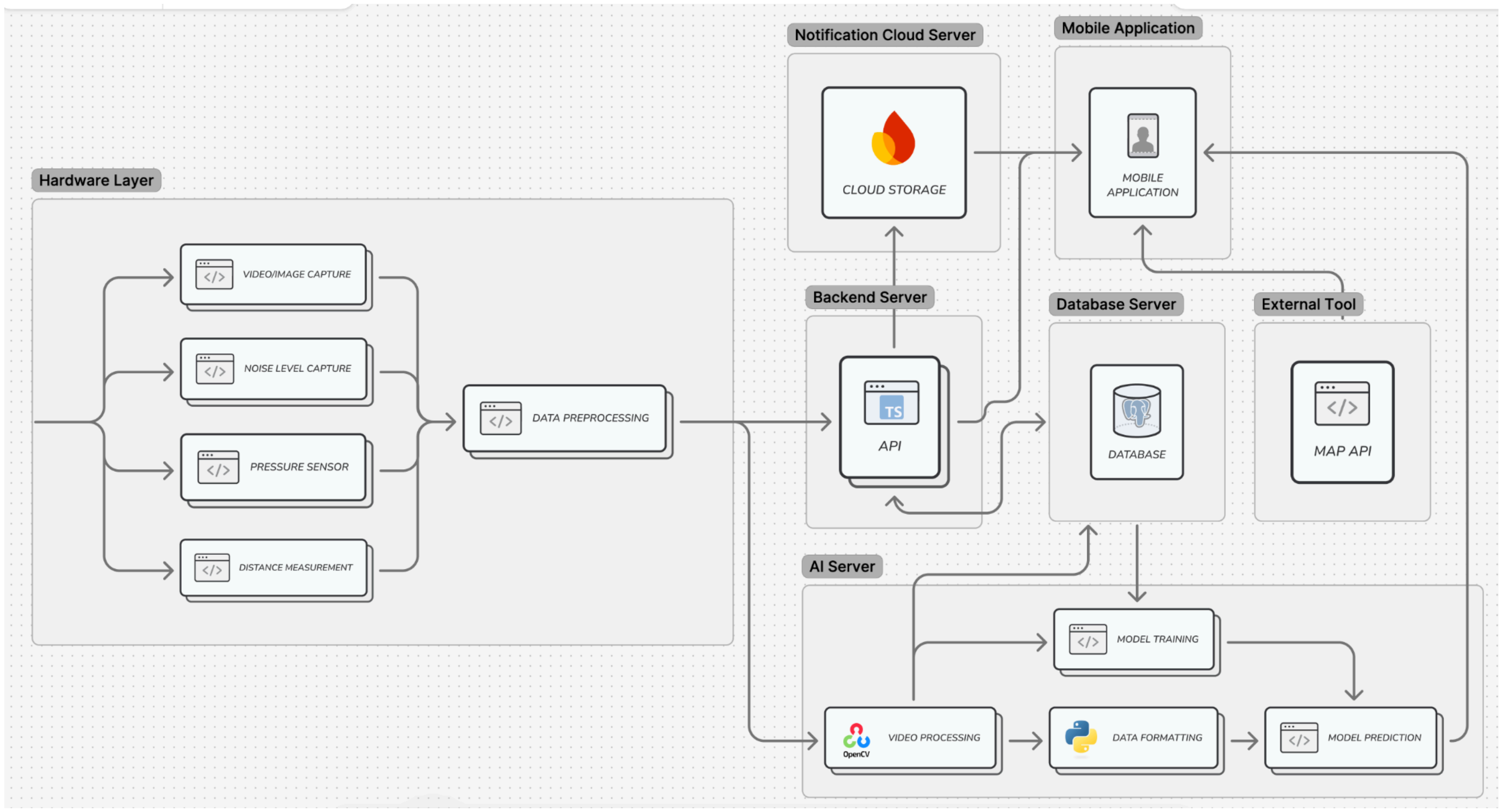


Figure3: Architecture Diagram of the system

2.2 Product Functions

The use case diagram below provides a functional overview of the KAISTime System by identifying the primary actors and the system's main functionalities. It illustrates the interactions between users and the system, outlining how different user roles engage with key system processes to achieve specific goals.

The system supports a range of user types:

- Students and Visitors use the platform to view real-time queue information, receive adaptive location recommendations, and visualize campus crowd density.
- KAIST Cafeteria Workers are responsible for monitoring live occupancy data, validating queue information, and accessing usage reports.
- Admins manage system data, update user information, and send system-wide notifications.
- Executive Staff—such as managers, professors, or campus leadership—use the system primarily for data-driven insights and long-term planning.

The diagram organizes functionality into use cases such as:

- Receive Recommended Locations
- View Queue Information and Crowd Density
- Manage Preferences and User Data
- Send and Receive Notifications
- Generate and View Usage Reports

These are supported by system-level processes including:

- Estimate Queue Lengths,
- Analyze Historical Data,
- Predict Wait Times, and
- Sync with Firebase.

<<include>> and <<extend>> relationships are used to clarify dependencies and optional flows between use cases. For example, the system always includes queue prediction when generating recommendations (<<include>>), while selecting a location from the recommendation list is an optional user action (<<extend>>).

This use case diagram captures the full scope of user interactions and system behaviors, and serves as a foundation for functional requirements, user stories, and future interface design.

Use Case Diagram

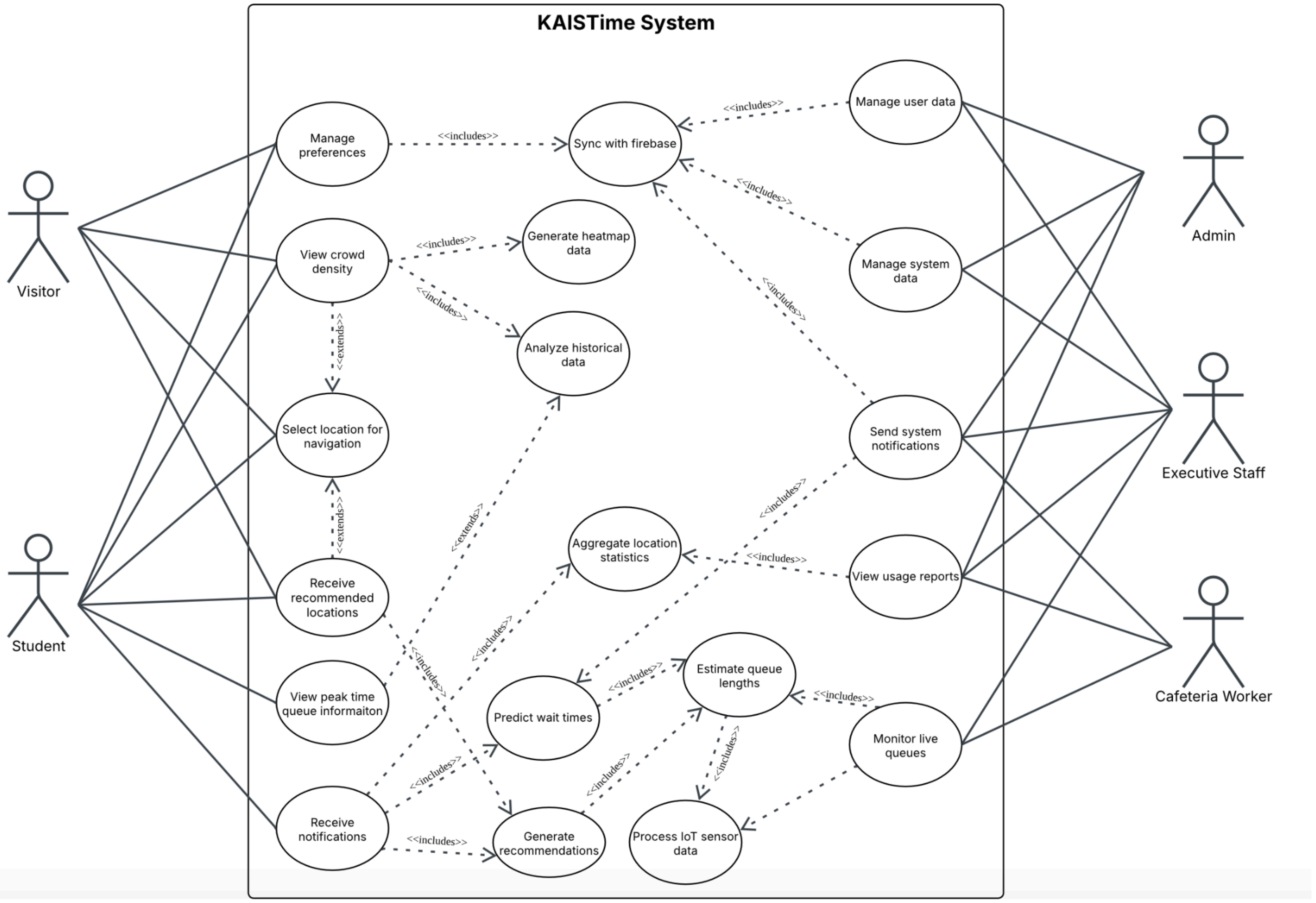


Figure4: KAISTime System Use Case Diagram

2.3 High-level Overview of the Interfaces

This section provides a high-level description of the necessary interfaces that enable interaction between the KAISTime System, users, hardware devices, and external services.

1. User Interfaces

The KAISTime System provides a user-friendly interface accessible via mobile devices and web browsers. Through this interface, students, visitors, staff, and admins can access live queue information, receive location recommendations, manage preferences, and view usage reports. The interface will prioritize accessibility, responsiveness, and clarity to support a seamless user experience.

2. Hardware Interfaces

The system integrates with IoT hardware devices, including ultrasonic sensors, time-of-flight sensors, and camera modules. These devices supply real-time occupancy and movement data to the system. Communication between the hardware and backend services occurs through standard wireless protocols.

3. Software Interfaces

The system connects with external software components, including a real-time cloud database (Firebase) for user authentication, data storage, and synchronization, and AI libraries (such as OpenCV and YOLO) for queue estimation and crowd analysis. These integrations ensure real-time data processing and intelligent recommendations.

4. Communication Interfaces

Secure communication protocols, such as HTTPS, are used to transfer data between users, hardware devices, and cloud services. The system also supports real-time push notifications to users' devices to deliver updates about queue statuses, recommendations, and important alerts.

2.4 User Characteristics

The KAISTime System is designed to support a diverse range of users with varying levels of familiarity with campus facilities, technical proficiency, and system needs. This section outlines the primary user groups and their expected interaction patterns with the system.

- **Students**
    - Primary users of the system.
    - Familiar with KAIST campus facilities.
    - Expected to have basic mobile and web application usage skills.
    - Capable of updating preferences and receiving personalized recommendations.

- **Visitors**
    - Occasional or one-time users.
    - Generally unfamiliar with the campus layout.
    - Require intuitive navigation assistance and real-time queue/crowd information.
    - Limited or no use of preference-setting features.

- **Staff**
    - University employees involved in campus operations (e.g., live queue monitoring, space management).
    - Expected to have moderate technical proficiency.
    - Interact with dashboards, occupancy data, and heatmaps.

- **Admins**
    - Technically skilled users with administrative access.
    - Responsible for managing user data, updating system content, and sending system notifications.
    - Expected to interact with backend systems and administrative dashboards.

- **Executive Staff**
    - Senior university leadership such as managers, directors, or professors.
    - Primarily view high-level usage reports and trend analyses.
    - Minimal interaction with operational features; focus on strategic insights.

- **General Requirements**
    - The system must be accessible and responsive across mobile and web platforms.
    - User interfaces must be intuitive and require no specialized technical training beyond standard mobile app usage.
    - Accessibility considerations must be incorporated to accommodate a diverse user base.

2.5 Constraints

The system's design and implementation are subject to several constraints that affect the development process, architectural choices, and deployment viability.

2.5.1 Hardware Constraints

The prototype relies on microcontroller-based hardware that impose several limitation described a follow:

a. The limited onboard RAM and processing power of the ESP32-CAM make local machine learning infeasible. The microcontroller is used only for basic data processing and communication. All inference and data analysis are handled server-side via platforms such as Google Colab and Firebase.

b. To perform all operations, sensors can produce current spikes that exceed **0.5 A**, which is the typical limit of standard **Micro USB 5V TP4056 charging modules** used with 18650 batteries. This can result in system instability, boot failures, or brownout resets.
c. Some sensors (e.g., ToF or ultrasonic) require **higher voltage levels (≥3.3V–5V)** and a **stable current supply**, which cannot be guaranteed by a single low-cost Li-ion power module under load.
d. Breadboard connection and jumper cable are suitable only for testing and prototyping purpose; they are unsuitable for permanent setups
e. Sensors are sensitive to environmental noise (e.g., lighting conditions, reflective surfaces, ambient sound, light physical pressure). Proper calibration and signal filtering are required to maintain data quality.

2.5.2 Software and platform Constraints

a. The backend should run on a cloud environment (e.g., Firebase) for persistent data storage, centralized computation, and execution of machine learning models
b. APIs should be **RESTful** and **JSON**-enabled for effective sensor data ingestion as well as frontend integration. Such APIs should be fault-tolerant against periodic disruptions of the underlying network and should include retry mechanisms or buffering at the local level.
c. The frontend must be responsive and accessible via both mobile and desktop browsers.
d. Machine learning-based predictions must be executed on the backend due to the hardware limitations of edge devices.
e. The system aims to deliver **low-latency updates (typically <2 seconds)**, which is acceptable for the intended use. However, it does **not meet strict real-time guarantees (<100 ms)**.
f. The system must support English interfaces
g. The UI should meet standard accessibility guidelines (e.g., WCAG 2.1) and provide accessibility for various user populations.

2.5.3 Integration Constraints

a. Data from multiple sensors must be **fused** to improve reliability, as no single sensor provides consistently accurate data under all conditions.
b. Facility staff must be able to **manually override** sensor-generated data in case of hardware failures or anomalies.
c. **Time synchronization and timestamping** are essential to correlate low-latency sensor inputs with historical data logs. While the system does not operate in real-time, it is designed to provide near-instant updates (typically below 2 seconds) under standard conditions.

2.5.4 Security and privacy Constraints

a. All communication between clients, sensors, and the backend must occur over **secure channels** (e.g., HTTPS with TLS 1.3)
b. The system must implement **role-based access control** to differentiate between general users, facility staff, and administrators.
c. The system must **not collect any personally identifiable information (PII)**. All data must be **anonymized and aggregated** before processing or presentation.
d. The system must comply with the **Korean Personal Information Protection Act (PIPA)**, including:
   i. Collecting only data that is strictly necessary
   ii. Ensuring explicit user consent where applicable
   iii. Providing users with access and deletion rights
   iv. Securing stored and transmitted data according to regulatory standards
e. The system in this case will not process or store any personal data (PII). Individuals are identified solely as **anonymous presence indicators** (e.g., silhouettes, blobs, movement zones) by the sensor system. No facial recognition, user account linkage, or device tracking is performed.

2.5.5 Development and Tooling Constraints

a. The firmware for embedded divided will be developed using the Arduino IDE
b. The backend data processing and ML components will use **Python** and **TensorFlow** or **Scikit-learn**, with deployment and testing handled in **Google Colab** or equivalent platforms
c. The frontend will be developed using **React Native** or a similar web app framework
d. The system's architecture and codebase shall be designed to support future extension and reuse, including modular service separation and component decoupling
e. All source code must follow common software engineering conventions, including **modular design**, **descriptive naming**, and appropriate **documentation** to ensure maintainability, reusability, and collaboration across team members.

- We rely on the facility staff members to update their facility information daily themselves

3. External Interface Requirements

3.1 User Interfaces

The KAISTime System provides a web-based and mobile-compatible user interface designed for a diverse user group, including students, visitors, staff, and administrative users.

Key interface characteristics include:

- A responsive dashboard that displays live queue lengths, crowd density heatmaps, and location recommendations.
- Standard navigation elements, including:
  - A consistent header with navigation links (Home, Queue Info, Recommendations, Profile).
  - A persistent footer with a "Help" button available on every screen.
  - Profile management accessible via a top-right dropdown menu.
- Common buttons and functions:
  - Home, Refresh, Settings, Help, and Logout.
  - Recommendations displayed in card layouts, each featuring a "View Details" and "Navigate" button.
- Standard error message formatting:
  - All error messages will be displayed in a red-colored banner at the top of the screen.
  - Messages will follow a consistent phrasing: "[Action] could not be completed. Please try again."
- Notification pop-ups:
  - Users will receive real-time push notifications for congestion warnings or recommendations.
- Accessibility standards:
  - All user interface elements will comply with WCAG 2.1 Level AA accessibility standards.
  - Support for adjustable font sizes and high-contrast modes.

User interfaces are required for:

- Students and Visitors (queue viewing, recommendations)
- Staff (peak time monitoring, report access)
- Admins (user and system data management)
- Executive Staff (usage report viewing)

3.1.2 Hardware Interfaces

The KAISTime System interfaces with IoT hardware devices for real-time occupancy detection.

Key hardware characteristics include:

- Supported device types:
  - Ultrasonic distance sensors
  - Time-of-flight (ToF) sensors
  - Camera modules with lightweight AI inference capabilities

- Nature of data exchange:
  - Devices will transmit occupancy readings (numerical counts or images) to the backend system at regular intervals (e.g., every 10–30 seconds).

- Communication protocols:
  - Devices will use Wi-Fi or Bluetooth Low Energy (BLE) for short-range communications to a local hub, which forwards data via HTTPS to the KAISTime servers.

- Device management:
  - The system must support heartbeat monitoring to ensure sensors are online and functional.

3.1.3 Software Interfaces

The KAISTime System integrates with the following external software components:

Software Component	Purpose
Firebase	Storage of queue data, user preferences, and notification logs.
Firebase Authentication	User authentication and session management.
OpenCV	Image processing for people detection and queue analysis.
YOLOv5	Queue length estimation based on visual input from cameras.

Data interactions:

- Incoming data:
  - Occupancy counts from IoT devices
  - User profile updates
  - System configuration changes from Admins

- Outgoing data:
  - Real-time queue lengths and heatmaps to users
  - Notifications pushed to user devices

- ○ Usage reports accessible by staff and executive staff

3.1.4 Communication Interfaces

The KAISTime System requires several communication functions:

- Web Browsers:
  - Interfaces must be accessible via Chrome, Safari, and Firefox.
  - Responsive design standards must be maintained for both desktop and mobile views.

- Network Communications:
  - All communication between devices, users, and servers must be conducted over HTTPS to ensure secure data transmission.
  - WebSockets will be used for real-time data updates (queue lengths, occupancy alerts).

- Push Notifications:
  - Users will receive real-time alerts on mobile and web devices via Firebase Cloud Messaging (FCM).

- Internal Communications:
  - API-based RESTful services will manage data flow between system components, including IoT hubs, cloud storage, and user dashboards.

No email, SMS, or telephony communications are required in the base version of the system.

3.2 Functional Requirements (FR)

The five most important FR are listed below, with an extensive list of requirements shown in Appendix 1.

FR No.	Requirement Title	Requirement Description	Related Components	Priority
FR-1	Customize Eating and Location Preferences	The system shall allow users to manage their eating preferences, favorite locations, and preferred visiting times to enable personalized suggestions based on real-time queue data.	Mobile application, Firebase, Authentication system	Medium
FR-2	Timely notification	The system sends real-time notifications about queue changes and occupancy updates to users.	Notification Manager, Mobile application, Queue monitoring	High
FR-3	Receive Recommended Eating Locations	The system shall provide users with personalized eating location recommendations based on their saved preferences, real-time queue data, and	Mobile application, Firebase, Queue monitoring,	High

		predicted wait times to optimize decision-making and reduce congestion.		
FR-4	Modify Facility Information	The system shall allow authenticated staff members and administrators to modify and update facility information, ensuring that all users have access to accurate and current facility details. All updates are reflected in the mobile application and map interface in real time.	Mobile application, Authentication system, Map interface	High
FR-5	Interactive Map/UI and Spatial Data Visualization	The system shall allow users to visualize, explore, and interact with geographic or spatial data in KAIST dynamically including real-time updates, and contextual actions depending on user inputs.	Mobile application, Firebase, Map interface	High

FR1: Customize eating and location preferences

1.1 Purpose: The purpose of this feature is to allow users to customize their eating preferences, favorite facilities, and preferred visiting times. Based on these preferences and most recent queue information, the system will provide personalized suggestions to optimize user decisions and reduce waiting time at busy facilities.

1.2 Stimulus/Response:

1.2.1 Stimulus: User updates personal preferences (eating style, favorite cafeterias, preferred time slots).
1.2.1 Response: System saves the updated preferences to the user profile database.

1.2.2 Stimulus: User opens the application or requests cafeteria recommendations.
1.2.2 Response: System fetches most recent queue data, matches it against saved user preferences, and displays a ranked list of recommended locations.

1.2.3 Stimulus: Most recent queue data updates significantly (queue length changes, new data received).
1.2.4 Response: System dynamically refreshes recommendations and may send notifications if preferred locations meet optimal conditions.

1.3 Associated functional requirements:
FR-15, FR-16, FR-17, FR-18

1.4 Use Case Scenarios:

Name	Receive Personalized Eating Suggestions
Summary	Provides users with cafeteria recommendations based on most recent queue data, user preferences, and predictive analytics.
Actors	Student, Staff, Visitor, Mobile Application
Precondition	· The user has installed and authenticated into the KAISTime mobile application. · User profile preferences have been set. · The system is actively receiving updated queue and occupancy data from IoT sensors.
Description	1. User updates or verifies his/her preferences in the app. 2. System stores and processes preferences. 3. User requests recommendations. 4. System matches the most recent queue data with user preferences. 5. System displays a sorted list of suggested cafeterias. 6. User selects a cafeteria to view more details or navigate there.
Alternatives	3a. If no preferences are set, system ranks cafeterias purely by shortest wait time. 4a. If queue data is unavailable, system shows cached recommendations or informs the user.
Postcondition	User receives a personalized, sorted list of cafeteria recommendations tailored to preferences and up-to-date conditions.

1.5 Use Case Diagram:

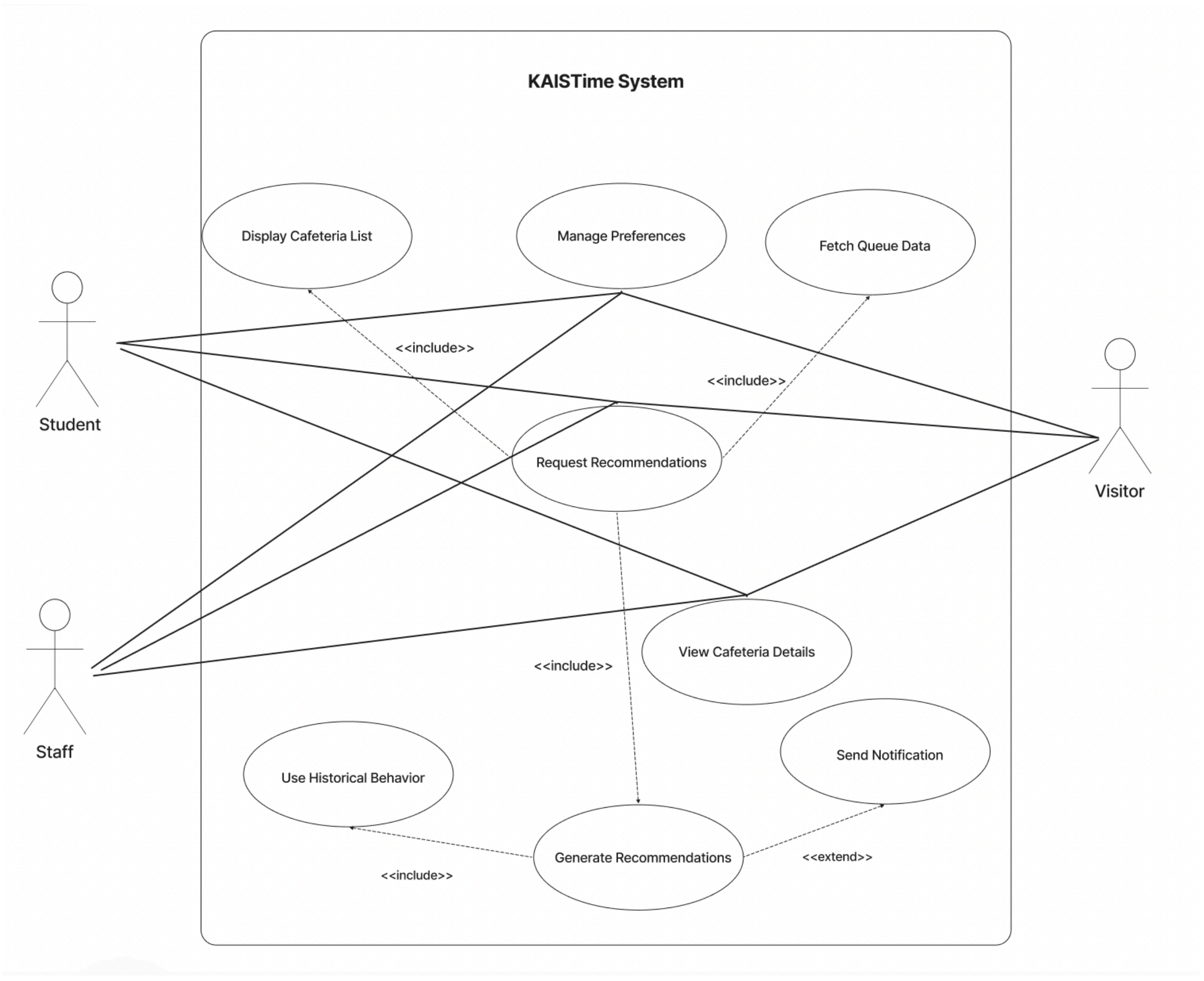


Figure5: Use Case Diagram Customization Eating

FR2: Timely Notification

2.1 Purpose: The purpose of the Timely Notifications feature is to provide users with updates about queue status and occupancy levels at facilities they are interested in. This allows users to make informed decisions and optimize their time on campus without relying on instantaneous real-time updates.

2.2 Stimulus/Response

- Stimulus: A monitored facility's occupancy level or queue time fluctuates and reaches preset notification thresholds according to user or from manual updates submitted by facility staff..
- Response: The system sends a push notification to the device of the user, generally within seconds of the occurrence of the condition, while accepting minor communication or processing delays

2.3 Associated functional requirements:
FR-6, FR-7, FR-8, FR-9, FR-10, FR-11, FR-12, FR-13, FR-14

2.4 Use Case Scenarios:

Name	Receive Timely Notification
Summary	Users receive a push notification when the queue waiting time or occupancy level at their favorite facilities meets their configured thresholds.
Actors	Users (Student, Visitor, Professors), Facility Member, Mobile Application
Precondition	· The user has installed and authenticated into the KAISTime mobile application. · Notifications are enabled for the selected facilities. · The system is actively receiving updated queue and occupancy data from IoT sensors and/or manual updates submitted by facility staff.
Description	1. Users configure notification preferences in the KAISTime app (e.g., maximum acceptable queue time: 5 minutes). 2. System monitors the queue and occupancy data every 30 seconds 3. A facility monitored by the user has a queue waiting time drop below the configured threshold. 4. The system triggers a notification generation event. 5. The mobile app receives the notification and displays it on the user's device. 6. User taps the notification and is redirected to the app's facility detail page showing the updated queue status
Alternatives	1a. If the user does not set any configuration of notification preferences, the system will use the system

	configuration (e.g. 10 minutes of queue and 70% of occupancy) 2a Facility Staff member update status of facility 3a. If the used device is offline the notification is queued and delivered upon reconnection if the data is still valid (within a 5-minute window). 3b. If the user has disabled notifications for that facility, the system suppresses the notification event and no push message is sent 4a. If the notification is generated during a user-defined quiet hours window, it is held and only delivered after the quiet hours end unless marked as critical. 6a. If the users do not open the notification in the window slot, the notification will be archived
Postcondition	User receive a push notification of a selected facility

2.5 Sequence Diagram and Use Case Scenario

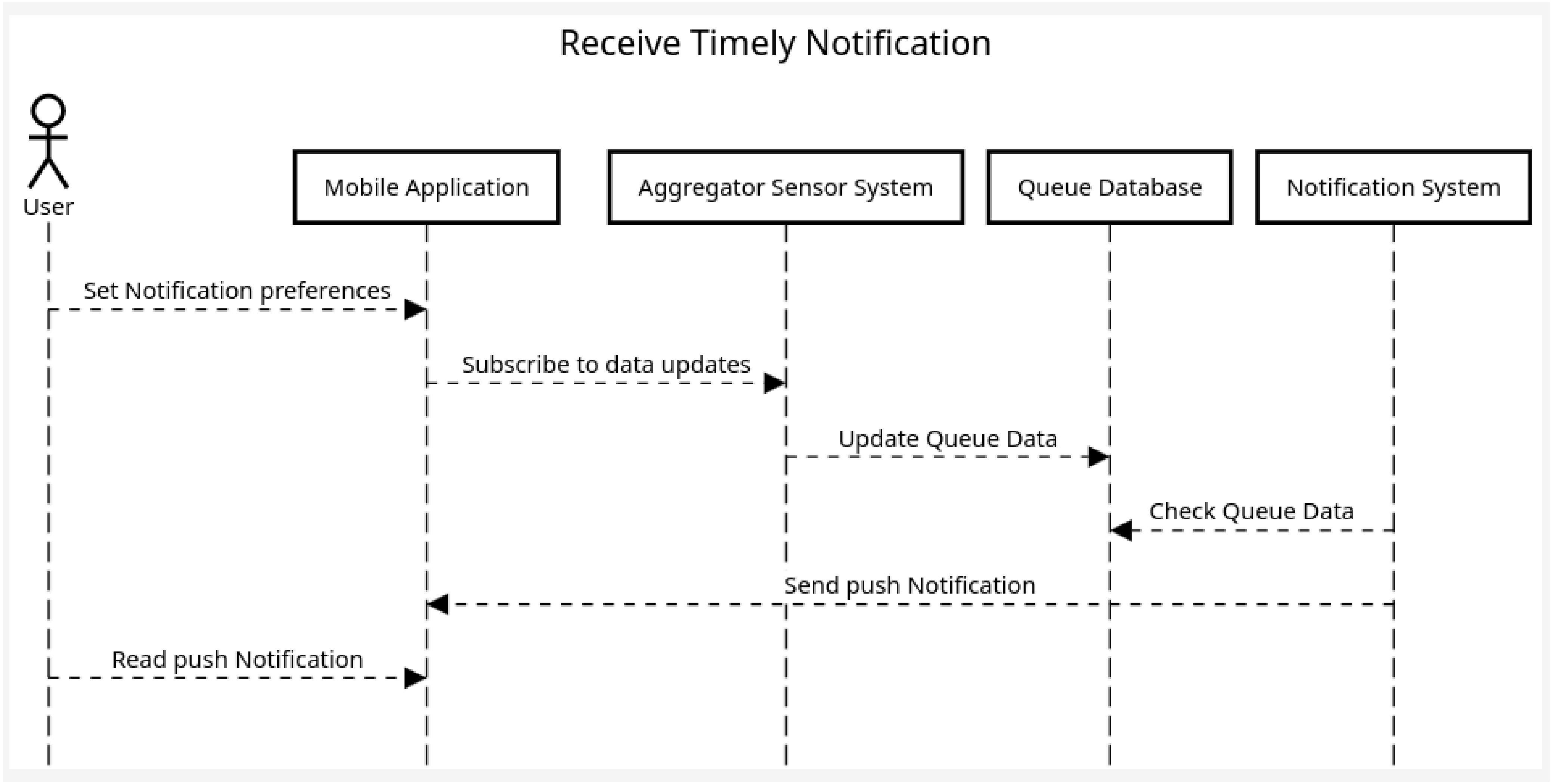


Figure6: Receive Timely Notification Sequence Diagram

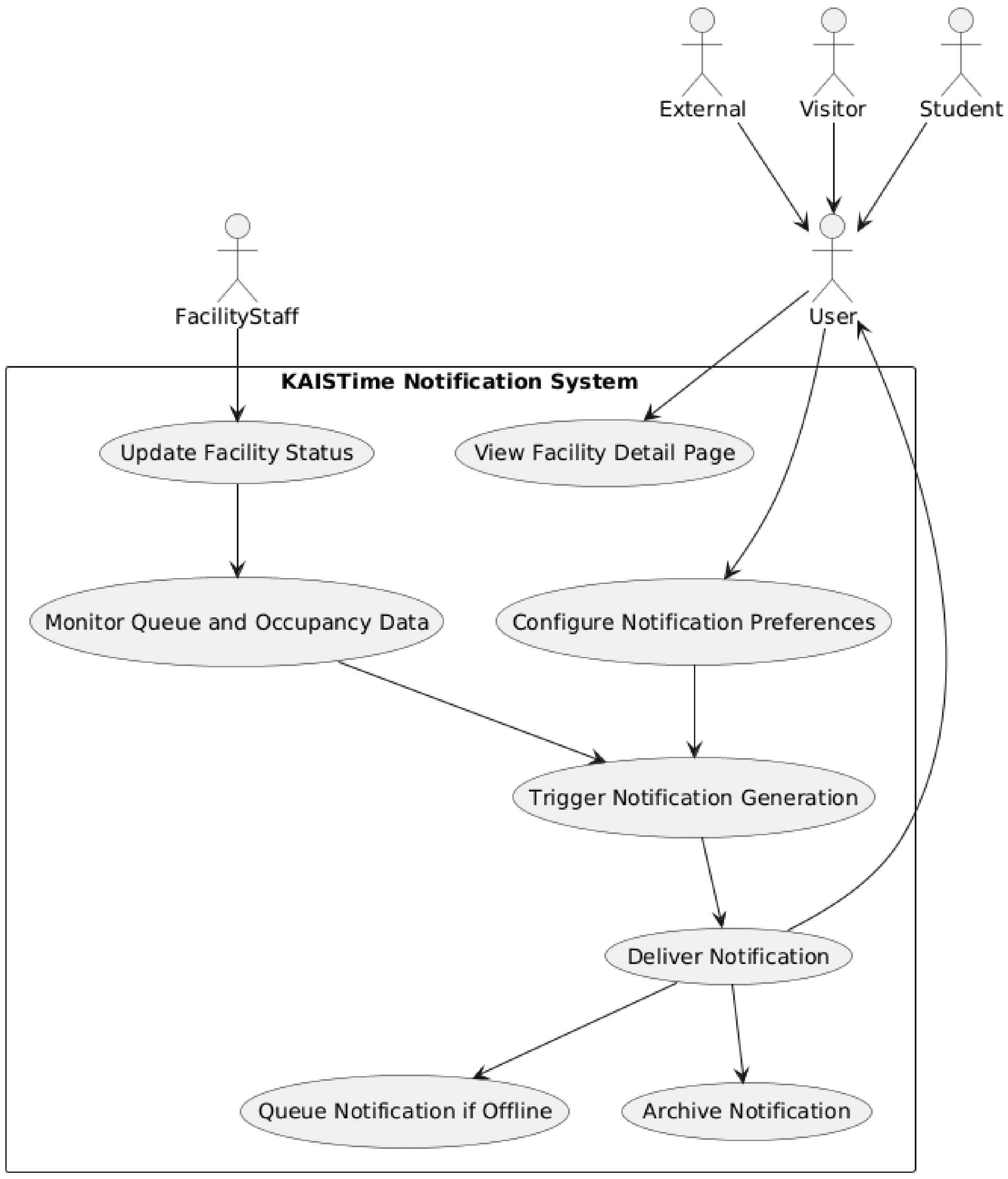

External
Visitor
Student
FacilityStaff
User
KAISTime Notification System
Update Facility Status
View Facility Detail Page
Monitor Queue and Occupancy Data
Configure Notification Preferences
Trigger Notification Generation
Deliver Notification
Queue Notification if Offline
Archive Notification

Figure7: Timely Notification Use case diagram

FR3: Receive Recommended Eating Locations
The system shall provide users with a list of recommended eating locations on the KAIST campus based on peak time queue data, crowd density, and the user's saved preferences (such as dietary needs, preferred locations, and preferred times). If the user has no preferences saved, the system will generate recommendations based on general factors such as proximity and current occupancy levels.

When a user requests eating location recommendations, the system will:

- Retrieve live queue and occupancy data from IoT devices and databases.
- Analyze user preferences (if available) to filter and prioritize locations.
- Apply predictive wait time estimates to rank options.
- Display the recommendations in an ordered list or card layout, including estimated queue lengths and expected wait times.
- Allow users to view additional details about each location or select a location for navigation assistance.

If real-time queue data is unavailable, the system will fall back to cached suggestions or display a notification advising users accordingly.

Dependencies

FR-24, FR-25, FR-36, FR-1, FR-15, FR-18

Use Case Description

Name	Receive Recommended Eating Locations
Summary	Provide users with a list of the best places to eat at a given time based on current queue lengths, preferences, and predicted wait times.
Actors	· Student · Staff · Visitor
Precondition	· User is authenticated · System has real-time queue data from IoT sources · System has access to user preferences
Description	1. User opens the app and selects “Find best place to eat” 2. System fetches current queue data from all cafeterias/dining areas 3. System filters or ranks locations based on: · Shortest wait time · User’s preferred or nearby locations 4. System returns a ranked list of recommended eating spots 5. User selects one to view more info or navigate
Alternatives	3a. If no preference exists, show most available spots 4a. If queue data is unavailable, show cached suggestions or notify the user that queue data is currently unavailable

Postcondition	User receives a sorted list of eating locations

Use Case Diagram and Sequence Diagram

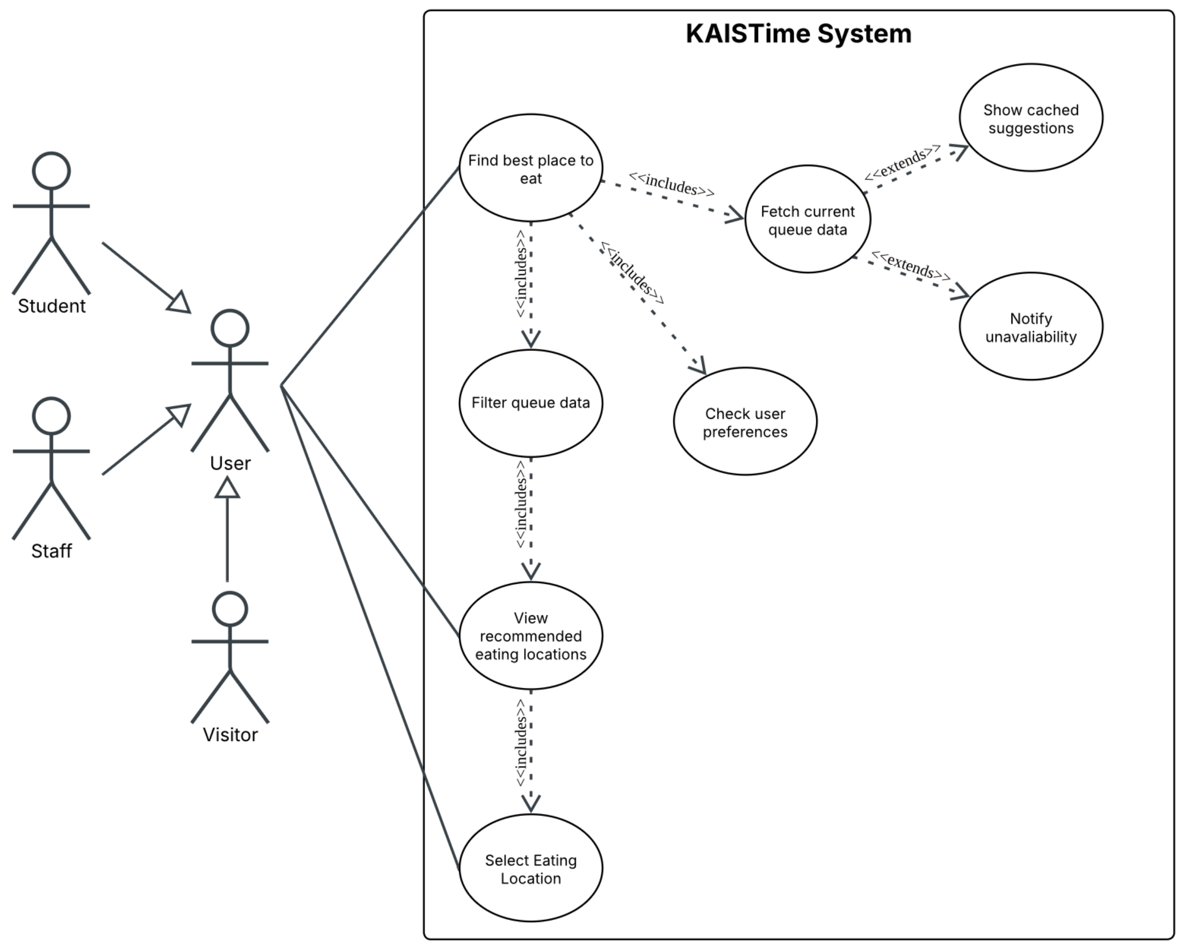


Figure8: Receive Recommended Eating Locations Use Case Diagram

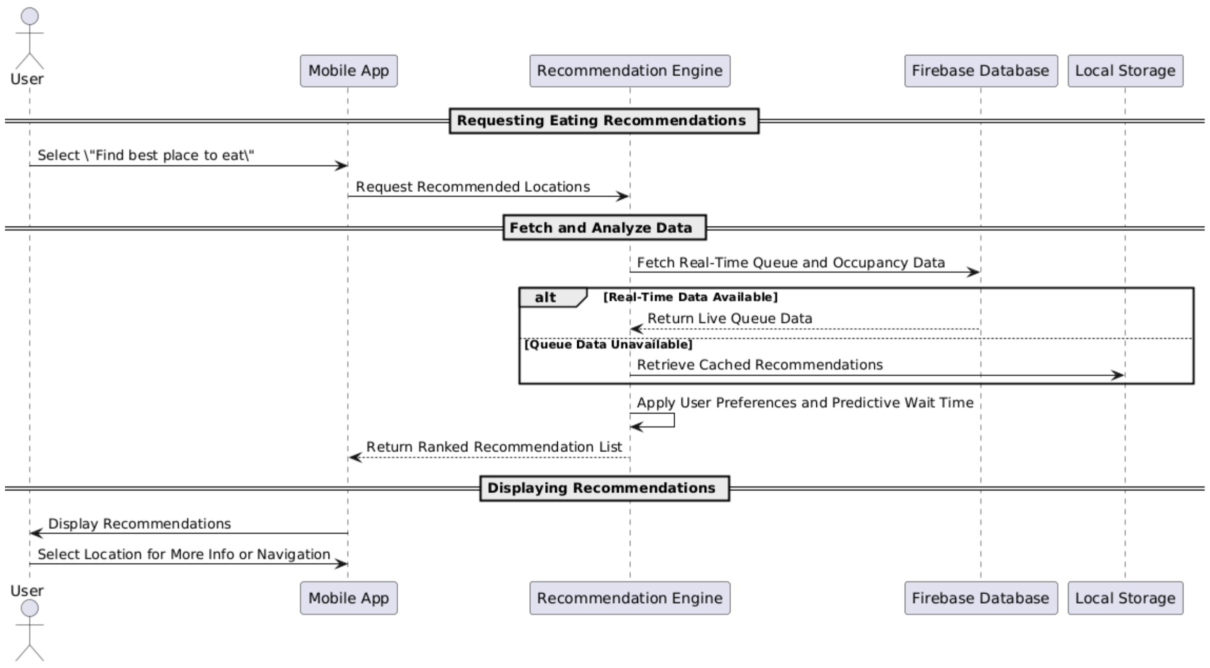

User
Mobile App
Recommendation Engine
Firebase Database
Local Storage
Requesting Eating Recommendations
Select \"Find best place to eat\"
Request Recommended Locations
Fetch and Analyze Data
Fetch Real-Time Queue and Occupancy Data
alt
[Real-Time Data Available]
Return Live Queue Data
[Queue Data Unavailable]
Retrieve Cached Recommendations
Apply User Preferences and Predictive Wait Time
Return Ranked Recommendation List
Displaying Recommendations
Display Recommendations
Select Location for More Info or Navigation

Figure9: Sequence Diagram Receive Recommended Eating Locations

FR4: Staff and administrators can modify and update facility information

4.1 Purpose: The purpose of this feature is to enable authorized staff members and administrators to modify, update, and manage facility information within the system. This ensures that users always have access to accurate and up-to-date details about facilities, including operational status, occupancy, queue information, pricing, daily menus, special announcements, and any other information staff wish to share with users.

4.2 Stimulus/Response:

- **4.2.1** Stimulus**:** A staff member that is logged in on his account can consult the admin dashboard to manage his facility. On this page he can use the edit mode to modify or update facility content (changes opening hours, updates status, updates the meny, updates the prices, marks facility as temporarily closed, updates temporary promotion, sends a notification about special notice to user, ….)
- **4.2.1** Response: The backend authenticates the user with their credentials and verifies their role while processing the request. If the user has the correct role, it applies the changes to the database, synchronizes the updated information, and updates the mobile application in real time.
- **4.2.2** Stimulus: A user creates an account using the facility staff member role. The mobile application will display a series of screens prompting the user to enter their account details, including information for the facility information page. This process will initiate a request to the API to create the facility.

- **4.2.2** Response: The backend will sequentially receive information provided by the user during the facility creation process. First, it will create a new facility entry in the database with the minimal required information. As the user proceeds through the next screens, the backend will update the facility record with additional details. Once the user completes the account creation phase, they will be redirected to the facility profile page. From this point, the facility will be visible in the application and on the map for users to view and consult.
- **4.2.3** Stimulus: An administrator who is logged into their account can access the admin dashboard to supervise all registered facilities. The dashboard provides a list of all facilities, allowing the administrator to select any facility to view its profile page and update its information if needed. The admin has the role of a super staff member that has the ability to manage all facilities to prevent mistakes, and maintain control of the platform. The admin can also create facilities via the dashboard for places that do not have a staff member managing them through the app.
- **4.2.3** Response: The backend authenticates the user and verifies the admin role when receiving a request. If the user has the correct role, it applies the changes to the database, synchronizes the updated information, and updates the mobile application in real time.

4.3 Associated functional requirements:

FR-45, FR-46, FR-47

4.4 Use Case Description

Name	Modify Facility Information
Summary	Staff and admin users can update facility details within the application to keep information up to date for all users and to promote their facility
Actors	Facility Member, Administrator, Mobile Application, Backend Server, Database Server
Precondition	· User is authenticated and authorized on the mobile application with a staff/admin account · To update a facility, it must already exist in the database and have a staff member account affiliated with it

	· The staff member should have the required information about their facility in order to create it
Description	1. Staff/admin user logs into the KAISTime application 2. User navigates to the facility management section 3. User selects and edits details (description, opening hours, menu, price, status, announcements, …) 4. User submits the changes 5. Backend server validates the request and updates the database 6. Updated information is synchronized, and the mobile application is updated in real time 7. If the update triggers a notification (facility closure, special announcement), relevant users receive a timely alert
Alternatives	1a. If the login fail the user cannot update facility details 4a. If the user lacks sufficient privileges, the system denies access 5a. If the backend validation fails (due to invalid data format or missing information), the system returns an error message to the mobile app, which then displays the message to indicate to the user which input should be changed 6a. If the database update fails, changes are rolled back and the user is notified via an alert message on the mobile application
Postcondition	Facility information is updated across all user interfaces, and users see the latest details in the mobile app. If applicable, a notification is sent out.

4.5. Use-case Diagram & Sequence Diagram

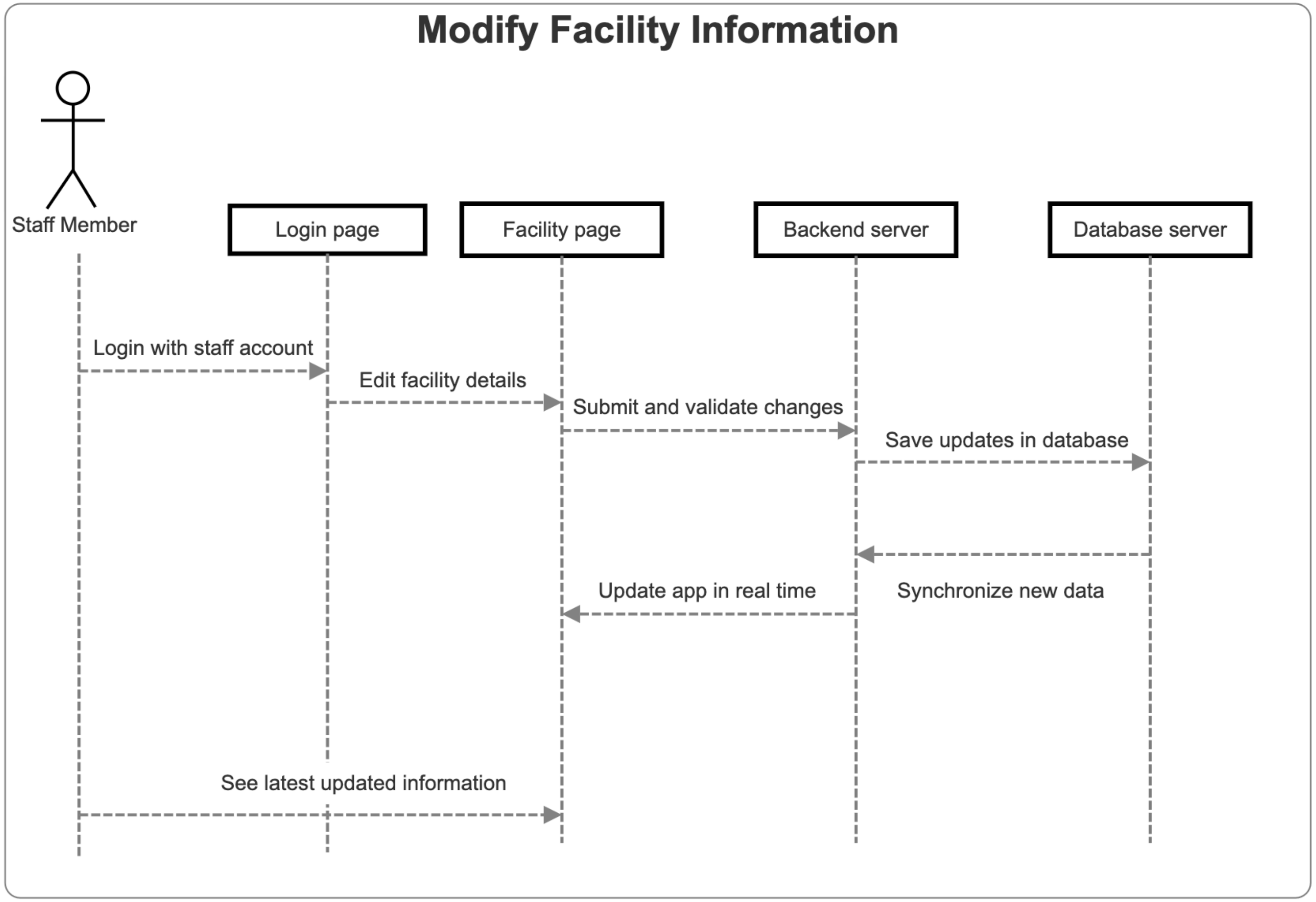


Figure10: Modify Facility Information Sequence Diagram

In this case sequence scenario shows only the general flow for staff member as example

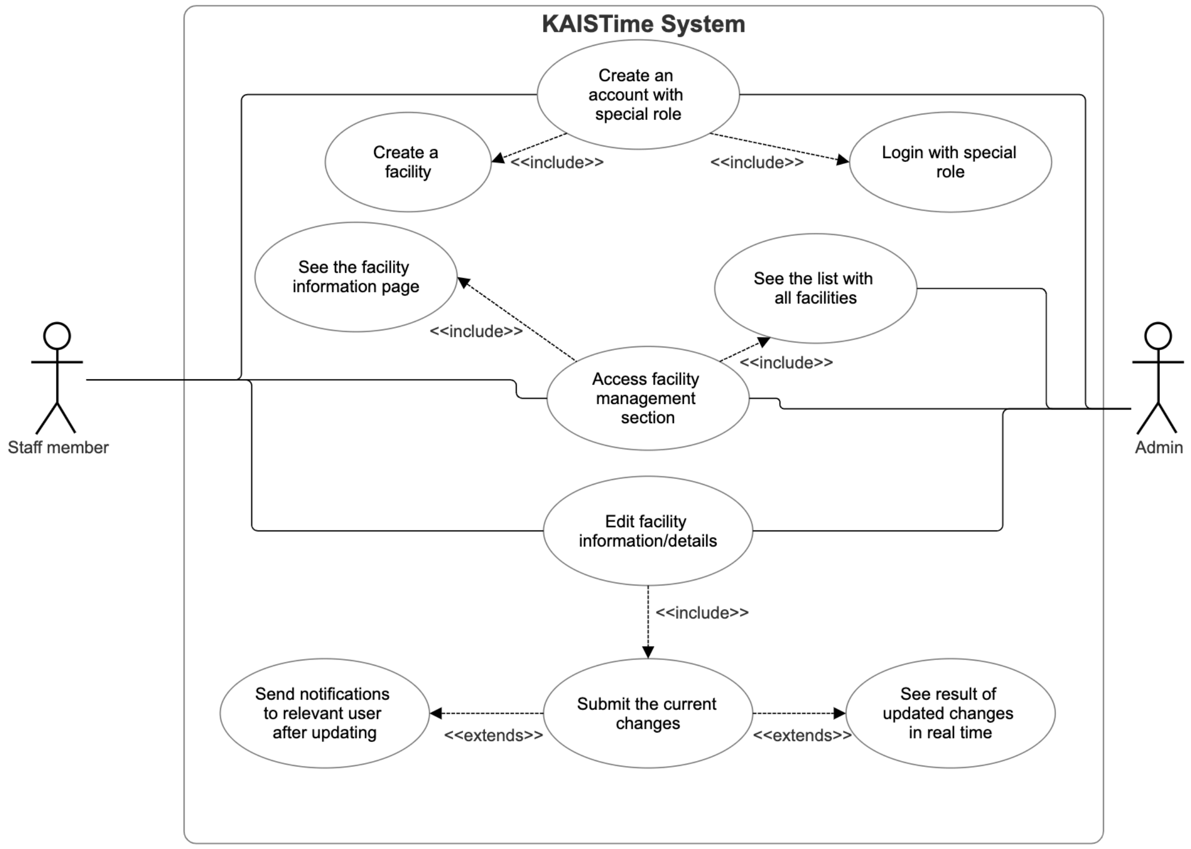

KAISTime System
Create an account with special role
Create a facility
Login with special role
<<include>>
<<include>>
See the facility information page
See the list with all facilities
<<include>>
<<include>>
Access facility management section
Staff member
Admin
Edit facility information/details
<<include>>
Send notifications to relevant user after updating
Submit the current changes
See result of updated changes in real time
<<extends>>
<<extends>>

Figure 11: Modify Facility Information Use Case Diagram

FR5: Interactive Map/UI and Spatial Data Visualization

5.1. Purpose: The purpose of the Interactive Map and Spatial Data Visualization component is to provide users with an intuitive, real-time geographic interface that allows them to easily understand the congestion status and spatial distribution of people across various KAIST campus facilities. By integrating live queue data and occupancy analysis into a dynamic, interactive map, the system aims to help students, faculty, and visitors make informed decisions about their movements, thereby reducing waiting times, improving the utilization of campus spaces, and enhancing overall campus life efficiency.

5.2. Stimulus/Response

5.2.1.

Stimulus: A user opens the application and selects the "Map" feature.
Response: The system renders an interactive map centered on KAIST, with real-time queue data and occupancy visualization. Users can click markers to get detailed information and receive real-time updates.

5.2.2.

Stimulus: A user clicks or taps on a specific building marker on the map.

Response: The system opens a detail popup showing the building's current queue length, peak hour prediction, and occupancy rate, along with options to set notifications.
5.2.3.
Stimulus: A user applies a filter (e.g., only cafeterias or only study areas) from the map’s filter panel.
Response: The system dynamically updates the visible markers, showing only those matching the selected category while hiding others.
5.2.4.
Stimulus: A user zooms or pans the map to explore different areas of the KAIST campus.
Response: The system dynamically loads and displays queue and occupancy data relevant to the new visible area, ensuring smooth navigation without lag.

5.3. Associated functional requirements

FR-6, FR-7, FR-18, FR-20, FR-26, FR-27, FR-40, FR-42

5.4. Use Case Description

Name	Interactive Map/UI and Spatial Data Visualization
Summary	A user — such as a student, staff member, visitor, or administrator — accesses the mobile application's Interactive Map feature to view real-time queue lengths and occupancy statuses of various KAIST campus facilities, interact with facility markers, apply filters, optionally set personalized notification alerts based on congestion thresholds, and for administrators, manage facility data and broadcast critical notifications when necessary.
Actors	● Student ● Visitor ● Administrator
Precondition	● The user has successfully accessed the mobile application (authentication is required for administrative features). ● The device has internet connectivity, or local cache data is available. ● The system has up-to-date queue and occupancy information from Firebase. ● The user has appropriate privileges if attempting to perform administrative actions (e.g., facility data updates, critical notifications).
Description	1. The user launches the mobile application and selects

	the "Interactive Map" feature from the main menu. 2. The system initializes and displays a dynamic map centered on the KAIST campus, loading real-time queue and occupancy data. 3. The map shows facility markers (e.g., cafeterias, study areas, libraries) with color-coded indicators reflecting congestion levels. 4. The user zooms or pans the map to explore different areas; the system dynamically refreshes markers based on the visible region. 5. The user taps on a facility marker. 6. The system displays a detailed information popup showing: ● Facility name ● Current queue waiting time (in minutes) ● Occupancy percentage ● Peak hour predictions (if available) 7. The user applies filters to view specific facility types (e.g., only cafeterias or only study rooms). 8. The system updates the markers according to the selected filter. 9. The user optionally sets a notification to be alerted when a selected facility's congestion falls below a preferred threshold (e.g., waiting time under 5 minutes). 10. The user optionally saves a facility as a favorite for easier access and personalized updates. 11. If the user has administrative privileges, the user may ● Edit facility information such as name, maximum occupancy, or coordinates. ● Mark facilities as temporarily closed due to maintenance or other issues. ● Send critical notifications to all users regarding facility closures, maintenance alerts, or crowd control advisories. 12. The system verifies the user's permissions before committing any administrative changes. 13. Administrative actions are logged for audit purposes.
Alternatives	● If real-time data is unavailable, the system displays the last known data with a visible warning. ● If the map fails to load, the system provides a fallback list-based interface showing facilities and their statuses. ● If no facilities match the applied filters, the system notifies the user with a "No results found" message. ● If an administrator attempts to modify facility data but loses connection, the changes are not applied and the administrator is notified of failure.

	● If an administrator attempts to send a critical notification and Firebase Cloud Messaging fails, the system logs the failure and retries within a limited window.
Postcondition	● The user successfully visualizes current queue and occupancy information. ● Any configured notification preferences are saved. ● Any selected favorite locations are saved. ● The user can make better-informed decisions about which facilities to visit based on congestion levels. ● Any successful facility updates are saved and reflected in real-time on the map.

5.5. Use-case Diagram & Sequence Diagrams

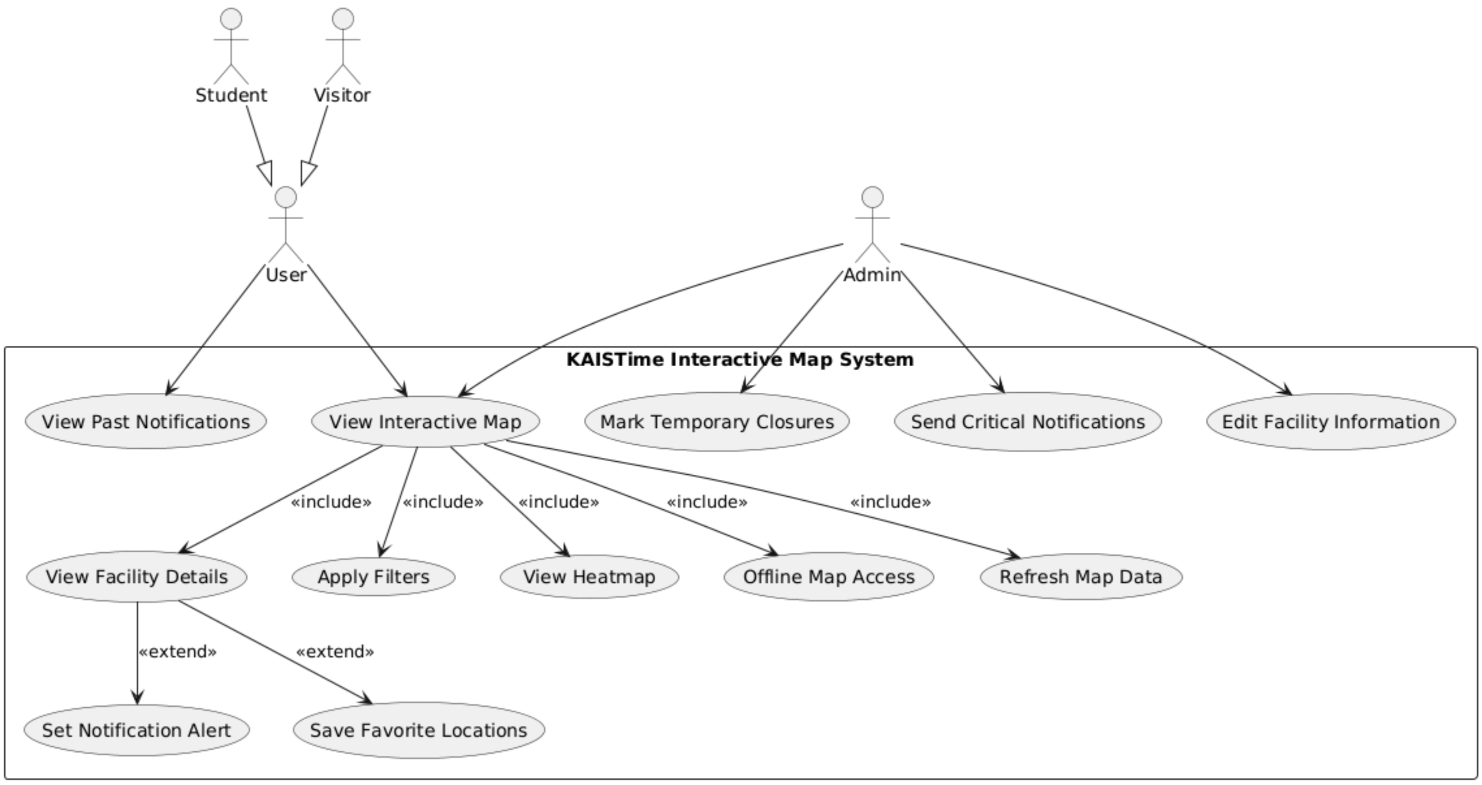


Figure12: Interactive Map/UI and Spatial Data Visualization Use Case Diagram

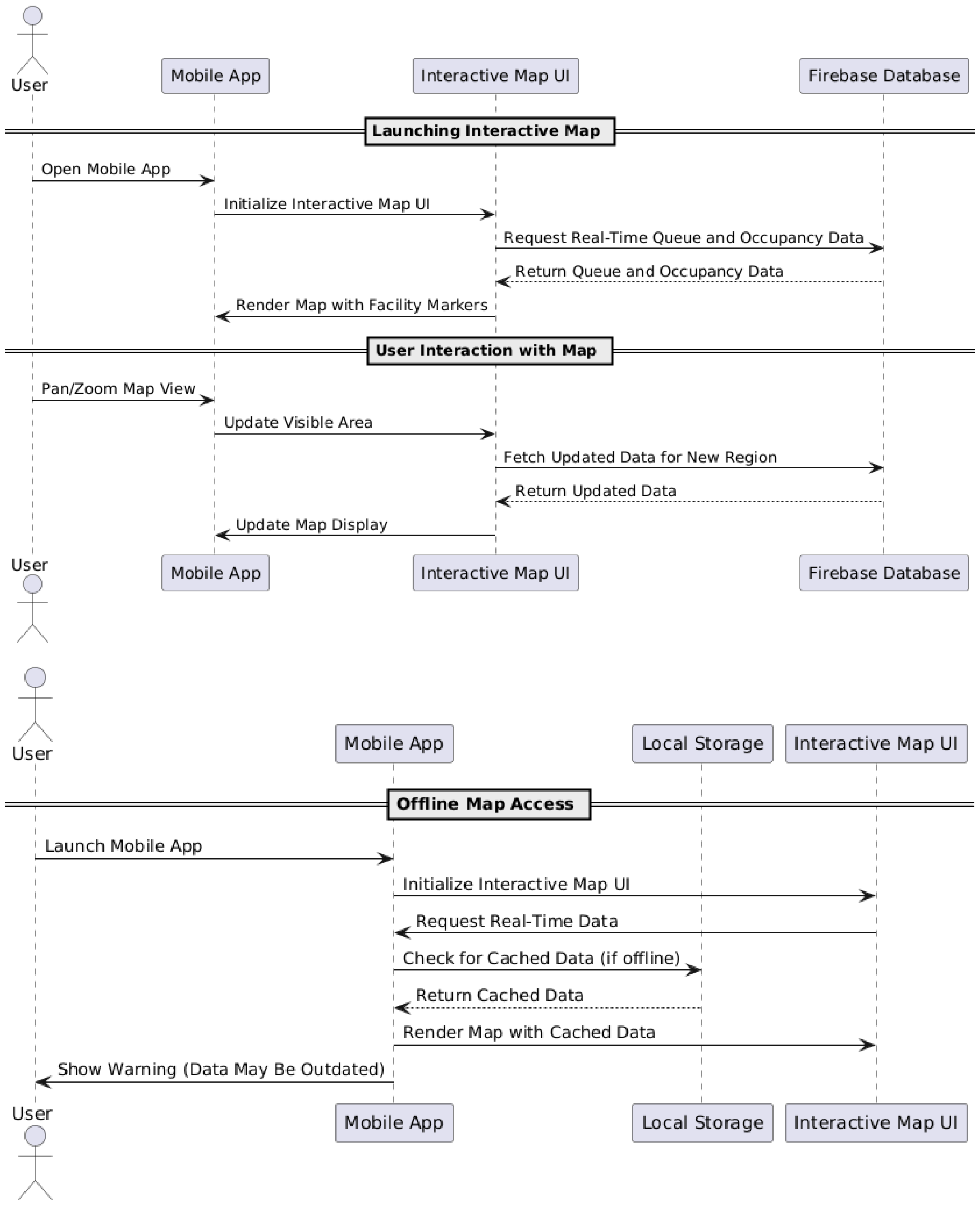


Figure 11: Interactive Map/UI and Spatial Data Visualization Sequences Diagram

4. Non Functional Requirements

4.1. Performance Requirements

The system must meet the following performance requirements to ensure both smooth operations, acceptable responsiveness, and accurate data processing under various usage scenarios:

4.1.1 Latency and Timing:

a. The system shall deliver sensor-based occupancy and queue updates to the user interface with an end-to-end latency not exceeding **2 seconds** in **95% of requests**, under typical conditions.
b. An active internet connection is required:
    i. On normal condition: receiving updates from the backend every <= **30 seconds** and uploading sensor data to the backend with a maximum delay of <= **3 seconds** between acquisition and transmission
    ii. Network failure: Sensor devices shall store sensor data in ram, wit a maximum retention window of **30 seconds**, All buffered data shall be sent to the backend immediately upon reconnection

4.1.2 Accuracy of predictions:

The system shall provide queue length and waiting time estimations with a minimum accuracy of **95%,** through periodic comparisons with real-world observed data

4.1.3 Resource utilization limit:

c. The mobile application shall not exceed **100 MB** of memory usage during normal operation
d. The total storage space used by the application (including cache and log) shall not exceed **200 MB** on client devices
e. Internet connectivity is required for all core functionality; in the absence of connection, the system shall gracefully degrade to offline mode (limited information, no live data)

4.2 Reliability Requirements

The system shall meet the following reliability objectives to ensure consistent and trustworthy delivery under both normal and adverse operating conditions:

1. **Data Transmission Reliability**
    a. Sensor shall reliably transmit at least **99%** of data collected to the backend in one week of continuous operation

   b. In case of temporary loss of network, sensor data shall be buffered in RAM and retried until successful delivery is made, up to a maximum of 3 consecutive retries with **30 seconds** between retries.

2. **Fault recovery**
   a. During loss of power or system reboot, sensor devices ought to re-initiate and resume work within **10 seconds**, assuming power and network availability.
   b. During network restoration, all the offline buffered data shall be transmitted automatically, and the system shall be in its normal operating condition without manual reset.

3. **Data Consistency and Coherence**
   a. The backend shall ensure data consistency across all clients (mobile, web, admin panel) by synchronizing updates at intervals of **≤ 30 seconds.**
   b. No conflicting updates shall be visible to users during concurrent access; all write operations (e.g., admin updates) must be atomic and resolved with eventual consistency logic.

4. **Uptime Stability (Sensor Devices)**
   a. Each sensor unit shall maintain at least **95%** uptime over a 7-day period, excluding planned maintenance windows.

4.3 Availability Requirements

The system must remain operational and accessible to end users and facility staff with minimal downtime. The following availability goals apply to the full service, including sensor devices, backend infrastructure, and frontend interfaces.

1. **Service Availability Requirements:**
   The system shall maintain a minimum uptime of 99%, measured during operational hours:
   a. Monday to Sunday, from 8:00 AM to 7:00 PM
   b. During the Spring, Summer, Fall and Winter Semesters
   c. The system shall be fully operational especially during critical periods, including:
      Lunch breaks (11.00 AM – 1.00 PM), Dinner times (5.00 PM – 7.00 PM) and Exam weeks. During these periods, system availability shall remain **≥ 99%**, with **no complete service interruptions** lasting more than **30 seconds**

2. **Downtime Requirements:**

   a. Downtime schedules must be announced at least 24 hours in advance and limited to no more than 30 minutes per week

3. **Status monitoring requirements:**
   a. Sensors shall be monitored periodically to ensure they are online; if a device is unresponsive for more than **60 seconds**, an alert must be triggered.
   b. The frontend must automatically **reconnect to the backend within 5 seconds** of a transient network error, and attempt retries every 10 seconds for at least 2 minutes.

4.4 Safety Requirements

Even if the system is not safety-critical, a number of safeguards must be considered to prevent damage to hardware, data integrity loss, and potential user confusion due to incorrect or unavailable information.

1. **Electrical Safety**
   a. All sensor devices must be powered using components rated for **3.3V logic and safe current limits (<500 mA)**.
   b. Overcurrent protection (e.g., fuses or current-limiting modules) should be used when appropriate to avoid component damage or overheating.
   c. The use of **unregulated power supplies** or **5V input to 3.3V-only devices** must be prevented through hardware design and documentation

2. **System Fail-Safe Behavior**
   a. In case of system fail or unexpected shutdown the system must:
      i. Fail silently, meaning it must not provide any misleading of false information (e.g. a false “empty” status when no data is available could lead into an overcrowded situation)
      ii. Frontend must clearly indicate when the data is unavailable or outdated by using warnings or status indicator
      iii. All sensors must **automatically reinitialize safely** after a reboot or power interruption
      iv. All values presented on the frontend must be verified and **marked invalid** if the backend detects missing, corrupted, or outdated input and none partially collected or inconsistent sensor data shall be shown to users.

3. **Installation Safety:**
   a. Sensors must be installed in locations that prevent any accidental damage, with no exposed wiring or sharp edges shall be present, especially in public spaces.

b. The system must be signaled or labeled to inform users his presence and not to interfere with hardware components

4.5 Security Requirements

To ensure the CIA principles (Confidentiality, Integration and Authentication) of data the system must implement the following mechanism:

1. **User Authentication**
    a. All users and admin must authenticate by using a password and email, with input validation and password strength enforcement (minimum 8 characters, including alphanumerical characters and at least one special symbol).
    b. Passwords must **not be encrypted** for storage purposes. Instead, they shall be **securely hashed** using a strong, one-way cryptographic algorithm such as **Bcrypt**, **PBKDF2**, or **Argon2**.
    c. Encryption is used for temporary transmission or processing (e.g., secure communication with external identity providers), it must be performed using the **AES-256 (Advanced Encryption Standard)** algorithm. Encryption keys must be stored in a **secure and isolated environment**, such as environment variables or a secrets manager.
    d. Authentication tokens shall be issued via **OAuth2 or JWT**, and must expire within **60 minutes** of inactivity. Tokens must be securely stored and invalidated on logout or session expiry.

2. **Authorization and access control**
    a. The system shall enforce **role-based access control (RBAC)** to distinguish:
        i. Users: read-only access to facility information
        ii. Facility staff: ability to update schedules, menus, or capacity status
        iii. Administrators: full access to all system settings and data
    b. Any unauthorized access attempt shall be **logged and notified** to administrators.

3. **Data Transmission Security**
    a. All data exchanged between clients (web, mobile, sensors) and the backend shall use **HTTPS over TLS 1.3** or higher
    b. No plain-text transmission of credentials or personal data is permitted at any stage.

4. **Data Privacy and Protection**
    a. The system shall not store or transmit personally identifiable information (PII). This includes biometric data (e.g. facial features).

   b. The occupancy and queue data shall be captured as anonymous presence indicator such as:
      i. Body silhouettes
      ii. Object blobs
      iii. Movement zones
      iv. Force/contact detection (FSR)
      v. Distance-to-object (ToF, ultrasonic)
   c. Anonymization must be performed on the device (edge level) prior to any transmission to the backend specifically:
      i. No sensible information such as camera frame or raw microphone data shall be stored
      ii. Only simple metrics may be sent (e.g. average distance, count of detected entities)
   d. If image-based sensor are used the system must:
      i. Apply on-device filtering
      ii. Avoid any face detection, recognition or identifiable pattern retention
      iii. Discard all raw frames after the entities are being estimated
   e. The system must comply with the principles of **data minimization**, **purpose limitation**, and **privacy by design**, as required by the **Korean Personal Information Protection Act (PIPA)** and relevant international standards (e.g., GDPR Article 5).

5. **Incident Handling**

   The system must provide logging for
   a. Authentication failures
   b. Unauthorized access attempts
   c. System configuration changes

   These logs must be stored securely and made available for audition purposes.

4.6 Maintainability Requirements

The system must be designed to support long-term maintances, modular updates and retraining of AI models requiring architectural changes or full redeployment

1. **AI Model Maintanence**
   a. Queue length estimation and occupancy prediction machine learning models must be retrainable using newly collected labeled data, without codebase modification
   b. Retraining pipelines must be documented and executed in **Google Colab or equivalent ML environments**
   c. Trained models shall be versioned, and new models shall be **hot-swappable** into the backend (e.g., via REST endpoint configuration or container update)

2. **Code Modularity and Documentation**
    a. All software components (frontend, backend, sensor firmware) will follow a **modular design**, so that each modules to be updated independently
    b. Each module must have a **clear** separation of concerns, In-code documentation and a High-level README file and setup script for reproducibility

3. **Testability and debugging**
    a. All core modules must be covered by unit test with at least 70% test coverage, and critical logic must be covered by integration tests.
    b. The system shall support diagnostic logging, including:
        i. Model inference error
        ii. Data flow success/failure
        iii. Sensor connectivity status
    c. Test environments must simulates:
        i. Sensor failure
        ii. Missing data packets
        iii. High-load periods

4. **Hotfix and Patch Capability**
    a. The system must allow to incremental updates for each single module
    b. Sensor firmware should be updatable over USB or via OTA (Over-The-Air) where hardware permits

4.7 Software Quality Attributes

The system shall exhibit the following quality attributes to ensure long-term effectiveness user satisfaction and ease of development and deployment:

1. **Usability**
    a. The user interface (mobile and web) must be intuitive and minimalistic, allowing users to retrieve occupancy and queue information within 5 **clicks or taps**
    b. Tooltips and contextual guidance shall be provided for all interactive elements

2. **Reusability**
    a. Core modules (e.g., API handling, sensor input, queue estimation) shall be reusable across different facility types and deployments (e.g., cafeterias, libraries, labs).
    b. Sensors, drivers and microservices must be written as independent modules, allowing for future updates and integration into a larger scale system such as the KAIST system.

3. **Scalability**
    a. The backend must support **horizontal scaling** to handle increases in the number of facilities and users without significant changes to the architecture
    b. Sensor data ingestion shall support **at least 100 concurrent devices** sending updates at 1 Hz.

4. **Portability**
    a. The frontend must run on all modern browsers (Chrome, Firefox, Safari) and Android/iOS devices
    b. Firmware code for sensor units shall be portable across different ESP32 variants with minimal modifications.

5. **Adaptability**
    a. System configuration (e.g., facility names, capacity thresholds, update frequency) must be editable via admin interface or configuration files, without recompiling the code.
    b. ML/AI models must be replaceable without altering backend code logic
    c. AI models must be replaceable trainable without altering backend code logic
    d. Model metadata (e.g., version, training date, accuracy) must be **logged and traceable** to ensure reproducibility and auditing of system behavior.

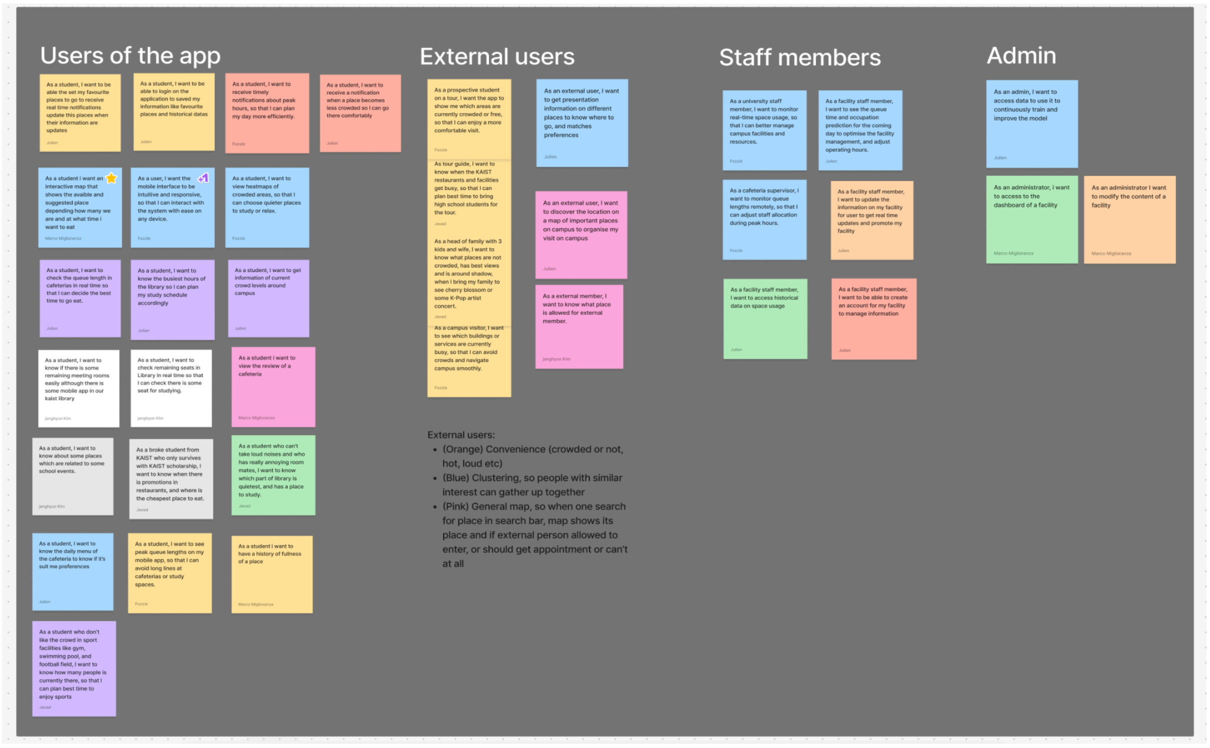
5. Prioritization and Release Plan

As we follow an Agile methodology using Scrum to develop this software, we have to effectively divide the requirements into different releases and determine which features should be included in each sprint. This approach allows us to prioritize features based on user value and deliver them incrementally through short, time-boxed development cycles. In this section, we explain the prioritization strategy used and provide a proposed release plan that aligns with our goal of rapid feature development and frequent delivery.

5.1 Choice of prioritization method

When prioritizing the requirements the five most important ones were picked out first. This was done during our elicitation sessions, we discussed together how to rank user stories by importance level. During these meetings we used the perceived ideas of what was important to the different stakeholders and feedback from the user we interviewed for prioritization.

Our prioritization strategy was to perform clustering of user stories by color to group them according to similarities. We started by collecting all user story ideas on a whiteboard using post-it notes, with no restrictions, to encourage creativity and open exploration from every stakeholder. Next, we grouped the user stories according to stakeholder roles and customer categories. After reviewing all the user stories, we identified distinct categories and reorganized the stories to ensure similar ones were grouped together, as shown in the diagram below:



Users of the app

As a student, I want to be able the set my favourite places to go to receive real time notifications update this places when their information are updates
Julien

As a student, I want to be able to login on the application to saved my information like favourite places and historical datas
Julien

As a student, I want to receive timely notifications about peak hours, so that I can plan my day more efficiently.
Fozzie

As a student, I want to receive a notification when a place becomes less crowded so I can go there comfortably
Julien

As a student i want an interactive map that shows the avaible and suggested place depending how many we are and at what time i want to eat
Marco Miglioranza

As a user, I want the mobile interface to be intuitive and responsive, so that I can interact with the system with ease on any device.
Fozzie

As a student, I want to view heatmaps of crowded areas, so that I can choose quieter places to study or relax.
Fozzie

As a student, I want to check the queue length in cafeterias in real time so that I can decide the best time to go eat.
Julien

As a student, I want to know the busiest hours of the library so I can plan my study schedule accordingly
Julien

As a student, I want to get information of current crowd levels around campus
Julien

As a student, I want to know if there is some remaining meeting rooms easily although there is some mobile app in our kaist library
Janghyun Kim

As a student, I want to check remaining seats in Library in real time so that I can check there is some seat for studying.
Janghyun Kim

As a student i want to view the review of a cafeteria
Marco Miglioranza

As a student, I want to know about some places which are related to some school events.
Janghyun Kim

As a broke student from KAIST who only survives with KAIST scholarship, I want to know when there is promotions in restaurants, and where is the cheapest place to eat.
Javad

As a student who can't take loud noises and who has really annoying room mates, I want to know which part of library is quietest, and has a place to study.
Javad

As a student, I want to know the daily menu of the cafeteria to know if it's suit me preferences
Julien

As a student, I want to see peak queue lengths on my mobile app, so that I can avoid long lines at cafeterias or study spaces.
Fozzie

As a student i want to have a history of fullness of a place
Marco Miglioranza

As a student who don't like the crowd in sport facilities like gym, swimming pool, and football field, I want to know how many people is currently there, so that I can plan best time to enjoy sports
Javad

External users

As a prospective student on a tour, I want the app to show me which areas are currently crowded or free, so that I can enjoy a more comfortable visit.
Fozzie

As tour guide, I want to know when the KAIST restaurants and facilities get busy, so that I can plan best time to bring high school students for the tour.
Javad

As a head of family with 3 kids and wife, I want to know what places are not crowded, has best views and is around shadow, when I bring my family to see cherry blossom or some K-Pop artist concert.
Javad

As a campus visitor, I want to see which buildings or services are currently busy, so that I can avoid crowds and navigate campus smoothly.
Fozzie

As an external user, I want to get presentation information on different places to know where to go, and matches preferences
Julien

As an external user, I want to discover the location on a map of important places on campus to organise my visit on campus
Julien

As a external member, I want to know what place is allowed for external member.
Janghyun Kim

External users:

- (Orange) Convenience (crowded or not, hot, loud etc)
- (Blue) Clustering, so people with similar interest can gather up together
- (Pink) General map, so when one search for place in search bar, map shows its place and if external person allowed to enter, or should get appointment or can't at all

Staff members

As a university staff member, I want to monitor real-time space usage, so that I can better manage campus facilities and resources.
Fozzie

As a facility staff member, I want to see the queue time and occupation prediction for the coming day to optimise the facility management, and adjust operating hours.
Julien

As a cafeteria supervisor, I want to monitor queue lengths remotely, so that I can adjust staff allocation during peak hours.
Fozzie

As a facility staff member, I want to update the information on my facility for user to get real time updates and promote my facility
Julien

As a facility staff member, I want to access historical data on space usage
Julien

As a facility staff member, I want to be able to create an account for my facility to manage information
Julien

Admin

As an admin, I want to access data to use it to continuously train and improve the model
Julien

As an administrator, i want to access to the dashboard of a facility
Marco Miglioranza

As an administrator I want to modify the content of a facility
Marco Miglioranza

Figure12: Brainstorming session

With these categories established, we prioritized the most important user stories by voting for our top five choices. The stories with the most votes were selected. We then developed the release plan based on this prioritization.

5.2 Release Plan

The requirements were divided releases based on the prioritization and their dependencies. The three different releases were assembled so that each would work as a fully functional application.

In the first release, we will implement the foundational features of the application, focusing on the highest-priority requirements and their dependencies. This includes user registration, support for different types of accounts, account management, and a campus map displaying locations with estimated waiting times. This initial release will serve as the Minimum Viable Product (MVP), allowing us to showcase the map feature and waiting time estimation to users, and to gather feedback that will be incorporated as improvements in future releases.

The second release will include additional important requirements that are not essential for the application's core functionality, but serve as improvements to make the product more attractive to users. These features include the ability for users to set food preferences, mark favorite places, retrieve passwords, and receive notifications.

The third release will include requirements that can be deprioritized or omitted if the project experiences delays or budget overruns. These features include a search function that allows users to find new places based on criteria such as price, user preferences, distance.

Appendix

Appendix 1 - List of KAISTime Functional Requirements

FR No.	Requirement Title	Requirement Description	Related Components	Priority
FR-1	Customize Eating and Location Preferences	The system shall allow users to manage their eating preferences, favorite locations, and preferred visiting times to enable personalized suggestions based on real-time queue data.	Mobile application, Firebase, Authentication system	Medium
FR-2	Timely notification	The system sends real-time notifications about queue changes and occupancy updates to users.	Notification Manager, Mobile application, Queue monitoring	High
FR-3	Receive Recommended Eating Locations	The system shall provide users with personalized eating location recommendations based on their saved preferences, real-time queue data, and predicted wait times to optimize decision-making and reduce congestion.	Mobile application, Firebase, Queue monitoring,	High
FR-4	Modify Facility Information	The system shall allow authenticated staff members and administrators to modify and update facility information, ensuring that all users have access to accurate and current facility details. All updates are reflected in the mobile application and map interface in real time.	Mobile application, Authentication system, Map interface	High
FR-5	Interactive Map/UI and Spatial Data Visualization	The system shall allow users to visualize, explore, and interact with geographic or spatial data in KAIST dynamically including real-time updates, and contextual actions depending on user inputs.	Mobile application, Firebase, Map interface	High
FR-6	Update Queue and Occupancy Data	The system shall monitor IoT sensor inputs and update queue and occupancy data at intervals of 30 seconds.	IoT Devices, Firebase, Queue Monitoring System	High
FR-7	Threshold-Based Notification Trigger	The system shall trigger a notification when the queue waiting time falls below or rises above user-defined time thresholds (e.g., waiting < 5 min) or when occupancy percentage matches specific user-defined occupancy criteria (e.g., occupancy < 70%).	Notification Manager, Mobile Application, Firebase	High
FR-8	Fast Notification Delivery	The system shall send notifications to users with a typical end-to-end latency of no more than 2 seconds under standard operating conditions.	Notification Manager, Mobile Application, Firebase Cloud Messaging (FCM)	High
FR-9	Notification Content Structure	Notifications shall include facility name, queue waiting time (in minutes), occupancy percentage (if measured), and a clear action recommendation (e.g., "Facility X now has low waiting time, recommended to visit now.").	Notification Manager, Mobile Application	High
FR-10	Notification Rate Limiting	The system shall limit notifications to at most one notification per facility every 10 minutes to avoid user overload.	Notification Manager, Notification Scheduler	Medium

FR-11	Notification Queuing for Offline Devices	If a user's device is offline, the notification shall be queued and delivered upon reconnection if it remains valid within a 5-minute window.	Firebase Cloud Messaging, Notification Manager	Medium
FR-12	Configure Notification Preferences	Users shall configure detailed notification preferences, including notification frequency (minimum interval between alerts), specific queue time thresholds, occupancy thresholds, and selectively enabling/disabling notifications per facility.	Mobile Application, Preferences Management, Firebase	Medium
FR-13	Expire Invalid Notifications	Notifications that remain undelivered due to prolonged offline status (more than 5 min) shall be automatically discarded to ensure only valid updates are presented.	Notification Manager, Firebase	Medium
FR-14	Define Quiet Hours	Users shall define quiet hours during which notifications will be silenced or delayed, except for critical occupancy alerts.	Mobile Application, Notification Manager	Medium
FR-15	Filter Suggestions by User Preferences	The system shall filter cafeteria suggestions based on the saved user preferences.	Recommendation Engine, Preferences Management, Mobile Application	High
FR-16	Update Suggestions Using Peak Time Data	The system shall update suggestions using peak time queue data from IoT sensors.	IoT Devices, Historical Data Analysis, Recommendation Engine	High
FR-17	Improve Suggestions Using Historical Behavior	The system shall use anonymized historical user behavior to improve future suggestions.	Recommendation Engine, Historical Data Analysis, Firebase	Medium
FR-18	Notify When Favorite Location is Ideal	The system shall notify users when a favorite location matches preferred conditions (e.g., low queue time).	Notification Manager, Preferences Management, Mobile Application	High
FR-19	Update Peak Time Occupancy	The system shall allow cafeteria workers to manually update the current occupancy of their assigned facilities if IoT device data is unavailable or inaccurate.	Staff Web Portal, Occupancy Management System	Medium
FR-20	View Peak Time Heatmaps	Staff members and users shall be able to view campus-wide crowd density heatmaps through the mobile and web applications.	Heatmap Generator, Mobile Application, Web Application	High
FR-21	Manually Trigger Notifications	Admins shall be able to manually send notification alerts to users about facility closures, maintenance, or urgent crowding conditions.	Notification Manager, Admin Portal	High
FR-22	Edit Facility Information	Admins shall be able to add, update, or remove facility information such as names, maximum occupancy, or location coordinates through the web application.	Facility Management Module, Admin Portal	High

FR-23	View Analytics Dashboard	Executive staff shall have access to an analytics dashboard that summarizes occupancy trends, queue statistics, and peak usage times.	Analytics Dashboard, Web Application	High
FR-24	Filter Recommendations by Travel Time	The system shall allow users to filter eating location recommendations based on estimated travel time to the facility.	Recommendation Engine, GPS Module, Mobile Application	Medium
FR-25	Save Frequently Visited Locations	Students shall be able to save frequently visited eating locations to improve the relevance of future recommendations.	Preferences Management, Mobile Application	Medium
FR-26	Provide Offline Access to Last Known Data	The mobile application shall allow users to view the last known queue and occupancy information even if the device is temporarily offline.	Mobile Application, Local Storage Cache	High
FR-27	Multi-Language Support	The mobile and web applications shall support at least English and Korean languages for all user interfaces and notifications.	Localization Module, Mobile Application, Web Application	Medium
FR-28	Staff Login and Role-Based Access	Staff users shall authenticate via a staff login portal and be granted role-specific permissions based on their assigned position (e.g., cafeteria worker, facility manager).	Authentication Module, Staff Web Portal	High
FR-29	Admin Login and Secure Management Portal	Admins shall log into a secure web portal to manage system-wide settings and monitor overall system health.	Authentication Module, Admin Portal, Security System	High
FR-30	Configure Facility-Specific Notification Rules	Admins shall configure custom notification rules for each facility, including thresholds for wait times and occupancy percentages.	Notification Manager, Admin Portal	Medium
FR-31	Manage Quiet Hour Policies	Admins shall configure and enforce default quiet hour policies across all users, while allowing personal overrides in the mobile app settings.	Quiet Hour Management System, Admin Portal, Mobile Application	Low
FR-32	View Historical Queue Trends	Staff and executive staff shall view historical queue and occupancy data for specific facilities through the web application.	Historical Data Manager, Web Application	High
FR-33	Personalized Recommendation Settings	The mobile application shall allow users to customize how strongly their preferences (dietary needs, favorite locations) influence recommendation generation.	Recommendation Engine, Mobile Application, Preferences Management	Medium
FR-34	Export Usage Reports	Admins and executive staff shall be able to export occupancy and usage reports in CSV or PDF format through the web application.	Reporting Module, Web Application	Medium
FR-35	Push Critical System Alerts	The system shall push critical system alerts (e.g., IoT device malfunction, database connection errors) to Admin users through the web portal and email.	Alert System, Admin Portal, Email Notification System	High
FR-36	Suggest Alternative	If the queue at a user's preferred facility exceeds	Recommendation	High

	Facilities	user-defined thresholds, the system shall automatically suggest alternative nearby facilities.	Engine, Queue Monitoring System, Mobile Application	
FR-37	Customize Notification Sounds	Users shall be able to select custom notification sounds for different types of alerts in the mobile app.	Notification Manager, Mobile Application	Low
FR-38	Dark Mode Support	The mobile and web applications shall support a dark mode theme for better accessibility during nighttime use.	UI/UX Module, Mobile Application, Web Application	Low
FR-39	Bookmark Facilities for Later	Users shall be able to bookmark facilities they intend to visit later for quick access.	Preferences Management, Mobile Application	Low
FR-40	Customize Heatmap Color Themes	Users shall be able to select alternative color schemes for the crowd density heatmaps.	Heatmap Generator, Mobile Application, Web Application	Low
FR-41	Facility Ratings and Feedback	Users shall be able to rate facilities and provide feedback after visiting.	Feedback Manager, Mobile Application	Low
FR-42	View Past Notifications	Users shall be able to view a history of past queue and system notifications within the mobile application.	Notification Manager, Mobile Application	Low
FR-43	In-App Tutorial for New Users	The mobile application shall include an interactive tutorial for new users on first login.	Onboarding Module, Mobile Application	Low
FR-44	Custom Quiet Hour Settings	Users shall define personal quiet hour periods during which non-critical notifications are muted.	Quiet Hour Management System, Mobile Application	Low
FR-45	Notify Relevant User on Facility update	After the staff member has updated the details of his facility with new announcements, the system should notify relevant users (users with this place as favorite).	Mobile Application, Notification Manager, Facility Management Module	Medium
FR-46	Possibility to Create Facility	Staff members or admins should be able to create a facility in the mobile application-on first login for members, and at any time for admins.	Mobile Application, Onboarding Module, Admin Dashboard	High
FR-47	Update Facility Details	Staff members or admins should be able to update facility details on the facility page.	Mobile Application, Facility Management Module	High