King Saud University
College of Computer and Information Sciences
Department of Software Engineering
SWE 321 – Software Design & Architecture
Second Semester | Spring 2025

HallGuide
Smart Campus Navigation

Phase 3

Group 2 | section 46475
Semester 2 | 2025

#	Student name	ID
1	Rand Khalid Alshalan	444200940
2	Danah Nasser Alraseed	444200630
3	Ghaida Abdulallah Alharbi	444201195
4	Rana Abdulallah Alanezi	444201047
5	Layan Yasser Aloufi	444200663
6	Hajar Mohammed Alonayq	444200784

Supervised by: Dr. Asma Alhussyen

Table of Contents:

Phase 3 Work distribution:

Student name	Work distribution
Rand Khalid Alshalan	- Component diagram - System Architecture
Danah Nasser Alrashid	- Deployment diagram - Updated class diagram
Ghaida Abdullah Alharbi	- System Architecture - Architecture diagram
Rana Abdullah Alanezi	- System Architecture - Updated class diagram
Hajar Mohammed Alonayq	- Component diagram - Architecture diagram
Layan Yasser Aloufi	- Deployment diagram - System Architecture

Table [1]: work distribution

Updates on phase one:

page	Section	Updates
8	*Functional requirements*	Modified requirements no. 17,18,23,35,37 added requirements no. 11, 13, 14, 24, 28, 33
11	*Challenges in development*	updated
11	*Non-functional requirements*	Updated section 4.9 scalability requirements
14	*Phase 2*	start of phase 2: system design document and Quality Assurance Plan

Table [2]: updates summery

Updates on phase two:

page	Section	Updates
11	*Non-functional requirements*	Updated section 4.6 reliability requirements
29	*Phase 3*	start of phase 3: component diagram, deployment diagram and system Architecture

Table [3]: updates summery

1. Problem Definition

1.1. Introduction

Universities are often large, consisting of several campus buildings and a mix of indoor and outdoor components. This complexity can make navigation challenging for students, instructors, and administrators, leading to confusion, wasted time, and missed opportunities. The absence of a streamlined navigation system creates obstacles in navigating between spaces, managing schedules, and finding available classes, ultimately impacting the overall campus experience.

To solve these challenges, we created **HallGuide**, a platform designed with multi-building navigation that seamlessly connects indoor and outdoor spaces. HallGuide enhances the campus experience by offering real-time updates, monitoring occupancy, and allowing users to easily book spaces. It also provides essential information such as space capacity and allows users to easily report maintenance or environmental issues. This ensures smooth transitions, better space utilization, and greater efficiency.

By offering a comprehensive, real-time view of campus resources, **HallGuide** transforms how universities manage space, communication, and navigation. With its advanced functionality, it fosters a more connected, efficient, and collaborative campus environment, ensuring that students, instructors, and administrators can move through their academic routines with ease.

1.2. Problem statement

In universities, the management of classroom reservations, scheduling, and communication between teachers and students can often be disorganized and inefficient. instructors struggle with conflicting schedules and room availability, leading to frustration securing proper teaching environments.
Moreover, New students or those unfamiliar with the campus layout often have difficulty finding their classrooms. There is a pressing need for a platform where both students and instructors can access classroom schedules, report issues, and for administrators to manage the educational activities effectively.

HallGuide is a solution that aims to address these challenges. It provides a streamlined approach for classroom reservations, notifications, GPS-guided directions to classrooms and issues reporting. By doing so, it significantly improves the overall communication and organization within the educational institution, offering a promising future for classroom management.

2. Proposed Solution

2.1. General Description

HallGuide is a unified platform designed to simplify navigation and resource management on large university campuses. It provides real-time updates on classroom status, an interactive campus map for seamless indoor and outdoor navigation, and integrated features for room booking and maintenance reporting. The solution aims to enhance efficiency and communication across campus, reducing confusion and saving time.
By combining scheduling, navigation, and reporting functionalities into a single system, HallGuide offers an innovative approach that surpasses traditional, fragmented solutions. Its real-time updates and comprehensive view of campus resource

2.2. Technologies Used

2.2.1 Artificial Intelligence (AI):

1. Objective and Appropriateness:

HallGuide will incorporate AI technologies to improve route navigation, particularly to determine the quickest route across campus, forecast maintenance requirements, and improve user interactions by making insightful suggestions. AI is a dependable solution for ongoing application innovation and enhancement due to its versatility and broad industry usage.

1. Essential Roles and Contributions:

- ***Pathfinding Algorithms****:* To determine the quickest and most effective routes throughout campus, AI-driven algorithms like Dijkstra's and A* (A-Star) will be employed. To give the optimum routes, these algorithms consider real-time data like congested locations or maintenance activities.
- ***Natural Language Processing (NLP):*** By enabling conversational language interaction, NLP makes the system accessible and easy to use for both staff and students.

2.2.2. Frameworks for Developing Mobile Applications (Flutter/React Native):

1. Objective and Appropriateness:

HallGuide will create a smooth mobile application for iOS and Android smartphones by utilizing cross-platform development frameworks such as React Native or Flutter. With a single codebase, these frameworks facilitate effective development, guaranteeing quicker updates, reliable performance, and less maintenance labor. They are perfect for developing reliable, responsive, and user-friendly mobile applications because of their scalability and vibrant developer communities.

2. Essential Roles and Contributions:

- **Cross-Platform Compatibility:** React Native and Flutter both make it possible to create applications that function flawlessly across iOS and Android, guaranteeing that HallGuide is accessible to all users on any device.
- ***configurable UI/UX:*** These frameworks include adaptable and configurable UI elements, making it possible to create aesthetically pleasing and user-friendly interfaces that are suited to the requirements of administrators, instructors, and students.

2.2.3. Indoor Global Positioning System (GPS):

1. Objective and Appropriateness:

Indoor GPS is an essential technology for enhancing wayfinding and navigation within large university campuses. Traditional GPS signals struggle to penetrate building walls and ceilings, making standard GPS unreliable for indoor use. It provides real-time location tracking and precise navigation inside multi-floor academic buildings. By integrating Indoor GPS, students and instructors can efficiently locate classrooms, offices, and other university facilities, ensuring timely arrivals and reducing confusion while navigating complex layouts.

2. Essential Roles and Contributions:

- ***Efficient Route Guidance & Time Management:*** Helps students and instructors find the shortest route to their destination, ensuring punctual arrivals to lectures and meetings, also it reduces time spent searching for rooms.
- ***Enhanced Campus Experience for New Users*****:** Simplifies wayfinding for first-time visitors and new students by providing accurate directions to key campus facilities such as libraries, cafeterias, and student service centers, reducing stress and improving overall engagement.

3. Functional Requirements:

Users (student, instructor, administrator):

1. Users shall be able to log in using their university credentials (ID, password).
2. Users shall be able to log out.
3. Users shall be able to reset their password by using their phone number.
4. Users shall be able to search for a specific classroom, instructor office or any facility using a search bar with criteria such as building, floor, room, or capacity.
5. Users shall be able to report issues related to classrooms by filling out a form (issue type, brief description, and hall identification).
6. Users shall be able to view an interactive map of the university, showing their current location and the best route to the classroom.
7. Users shall be able to add a stop point anywhere on campus before continuing to their destination.
8. Users shall be able to view estimated travel time based on their current location, selected stop points, and the best route.
9. Users shall be able to play voice-guided navigation to their destination.
10. Users shall be able to choose their preferred language (Arabic or English).
11. Users shall be able to add a new location.

Students:

12. Students shall be able to view an instructor's office hours, office location, and real-time availability status.
13. Students shall be able to view their booked appointments.
14. Students shall be able to view their class schedule.
15. Students shall be able to book an appointment with an instructor if they have available office hours.
16. Students shall be able to cancel their booked appointments with instructors.
17. Students shall be able to edit their booked appointments.

Instructor:

18. Instructors shall be able to set their availability status (Available, Busy).
19. Instructors shall be able to view their class schedule.
20. Instructors shall be able to view details of their assigned classrooms (location, capacity, facilities).
21. Instructors shall be able to update their office hours.
22. Instructors shall be able to view available classrooms for booking.
23. Instructors shall be able to request to book available classrooms.
24. Instructors shall be able to view their classrooms booking requests.
25. Instructors shall be able to edit their classroom booking request before administrator approval.
26. Instructors shall be able to cancel their classroom booking request before administrator approval.
27. Instructors shall be able to view a list of their upcoming appointments.
28. Instructors shall be able to cancel an upcoming appointment.

Administrators:

29. Administrators shall be able to add a new facility or classroom.
30. Administrators shall be able to edit classroom and facility information in the system.
31. Administrators shall be able to approve or reject instructor's requests for classroom bookings.
32. Administrators shall be able to view student’s and instructor's maintenance or environmental issues reports.
33. Administrators shall be able to update report status (new, in progress, resolved).

System:

34. The system shall send notifications related to schedule changes and reported issues to all users.
35. The system shall send confirmation notification to the instructor upon administrator approval or rejection of their booking request.
36. The system shall update the status of classrooms (Reserved, Available) based on the university schedule system or approved requests.
37. System shall send a confirmation notification to students 5 minutes before their appointment with an instructor.

4. Non-Functional requirements:

4.1. Usability:

- The instructor shall be able to reserve a classroom with no more than five clicks.
- users shall be able to learn how to use the system within 20 minutes.
- The user shall be able to report an issue in less than 3 minutes.
- No technical skills shall be required to use the system.

4.2. Portability:

- The mobile application shall be compatible with both iOS and Android platforms.
- The system shall be accessible as a web application and a mobile application, ensuring consistent functionality, UI components, and performance metrics.

4.3. Correctness:

- The navigation system shall calculate the shortest route with an accuracy deviation of no more than 5 meters.
- The estimated travel time shall have a deviation of no more than ±10% compared to actual walking time.

4.4. Interoperability:

- The system shall synchronize with the university's central database every 10 minutes to ensure schedule accuracy.
- The system shall support data export in standard formats (CSV, XML) for administrative reporting and analysis.

4.5. Security:

- Users shall only have access to features based on their roles, with a security breach probability of less than 0.001% annually.
- After five failed login attempts, the user account shall be temporarily locked for 10 minutes to prevent brute-force attacks.
- At least 99% of intrusions shall be detected within 10 seconds.
- All system's private data shall be encrypted via Secure Socket Layer (SSL).

4.6. Reliability:

- System shall be available 99.90% of the time
- Mean time between failures (MTBF) shall be at least 30 days.
- The system defect rate shall be less than 1 failure per 1000 hours of operation.
- The system shall perform daily full backups at midnight, stored in AWS cloud storage.

4.7. Maintainability:

- The system should be reviewed and refactored monthly to maintain a clean codebase over time.
- The software shall have clear well-commented code, using meaningful variable and function names that accurately describe their purpose.

4.8. Flexibility:

- The system shall be capable of integration and deploying new features with no more than 30 minutes of downtime.
- The system shall be configurable to support additional campuses or satellite buildings within 4 hours of setup time.

4.9. Scalability:

- The system should support up to 5000 concurrent users without degrading performance.

5. Challenges In Development:

- **Indoor Navigation Integration**: Implementing an accurate indoor navigation system with real-time campus updates is complex and we have limited experience with.
- **AI Path Algorithms**: Ensuring AI-based path algorithms function effectively with real-time data presents a challenge as we are still learning about real-time path optimization.
- **Real-Time Synchronization**: Managing live updates for schedules, bookings, and navigation while handling multiple users requires expertise in real-time databases.
- **Security & Access Control**: Implementing proper user access controls and protecting sensitive information is crucial yet complex.
- **User-Friendly Interface**: Designing an intuitive UI that caters to administrators, instructors, and students is essential and requires mastering UI/UX design.
- **Scalability & Reliability**: Ensuring the system handles up to 5,000 concurrent users efficiently require load balancing, and cloud deployment knowledge.

6. UI Design:

The HallGuide UI is designed to be intuitive, user-friendly, and accessible for students, instructors, and administrators. It includes an interactive map for navigation, a search bar for locating facilities, and easy booking management. The UI supports both Arabic and English, ensuring a seamless experience across desktop and mobile devices.

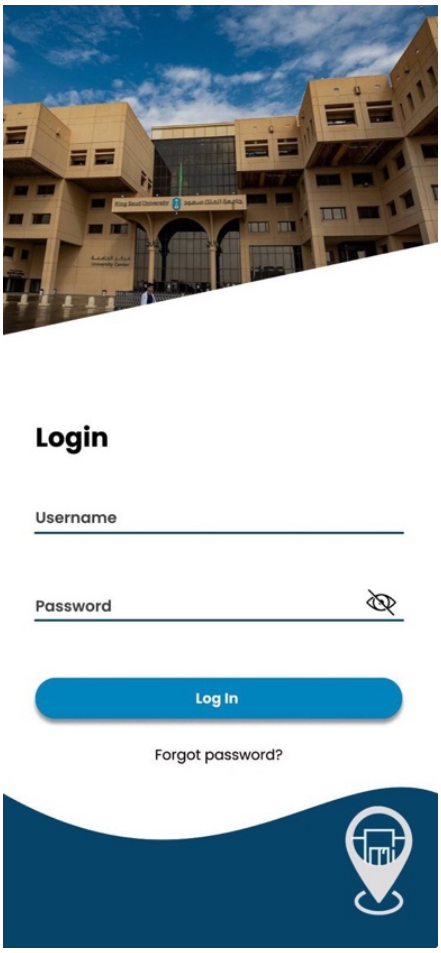

Figure 1:user login page

Figure 2: Student home screen

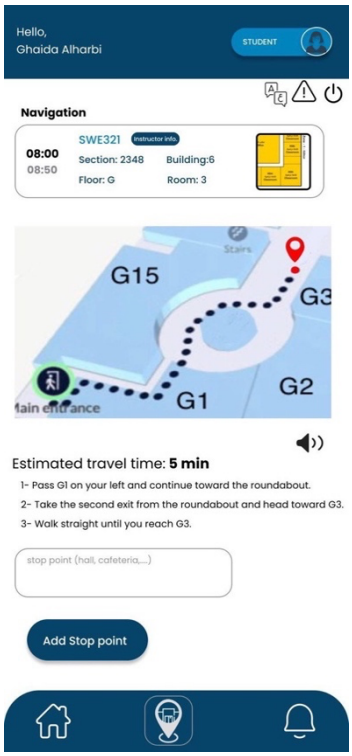

Figure 3: interactive map screen

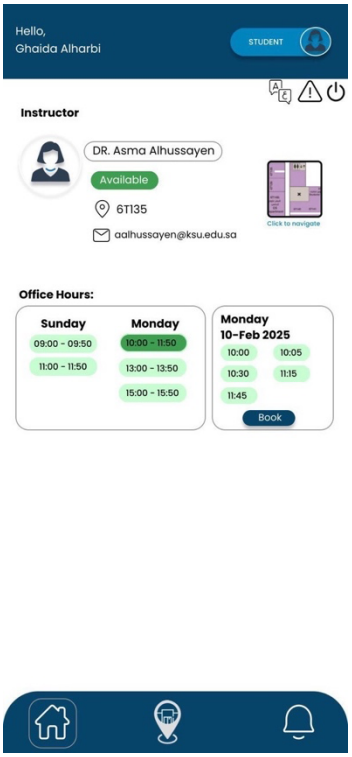

Figure 4: instructor profile screen

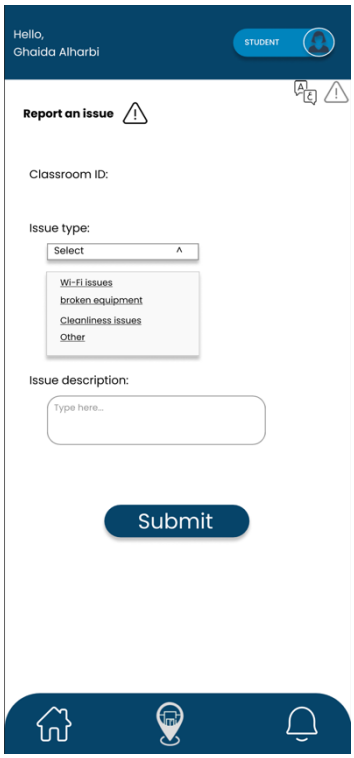

Figure 5: report issue screen

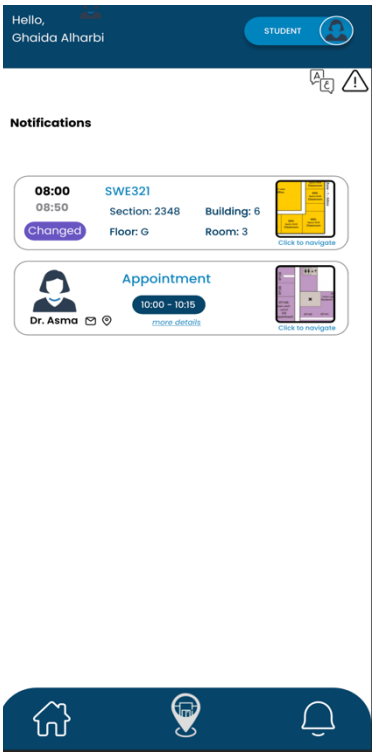

Figure 6: student notification screen

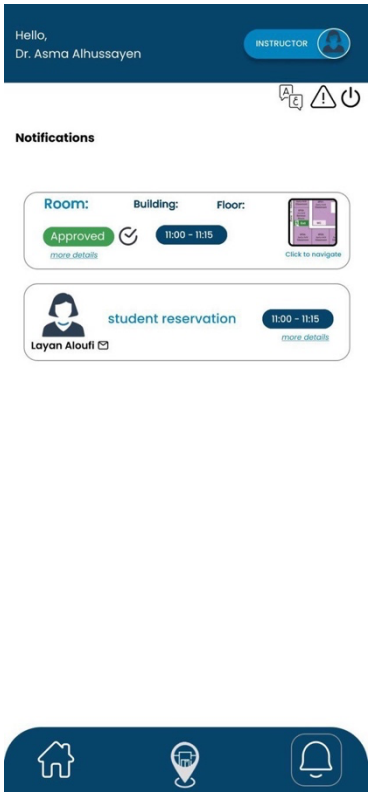

Figure 7: instructor notification screen

Figure 8: instructor home screen

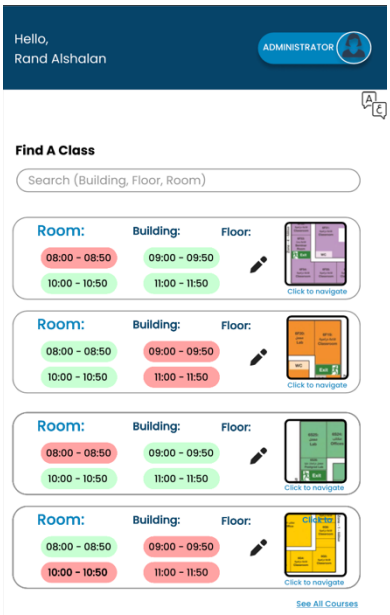

Figure 9: Administrator home screen

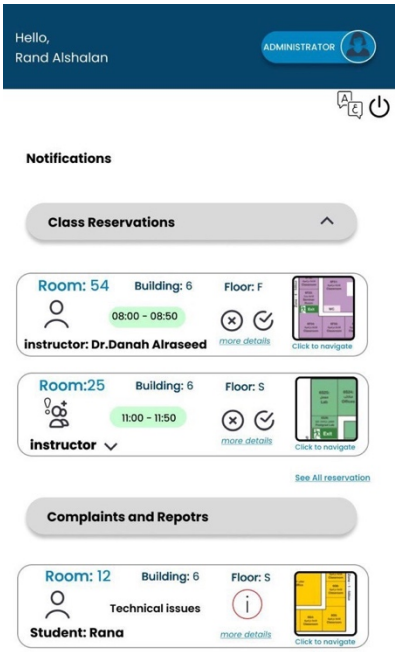

Figure 10: Administrator notification screen

7. System design document

7.1. Class Diagram

A class diagram is a UML diagram that visually represents a system's classes, attributes, methods, and relationships, providing a blueprint for object-oriented design.

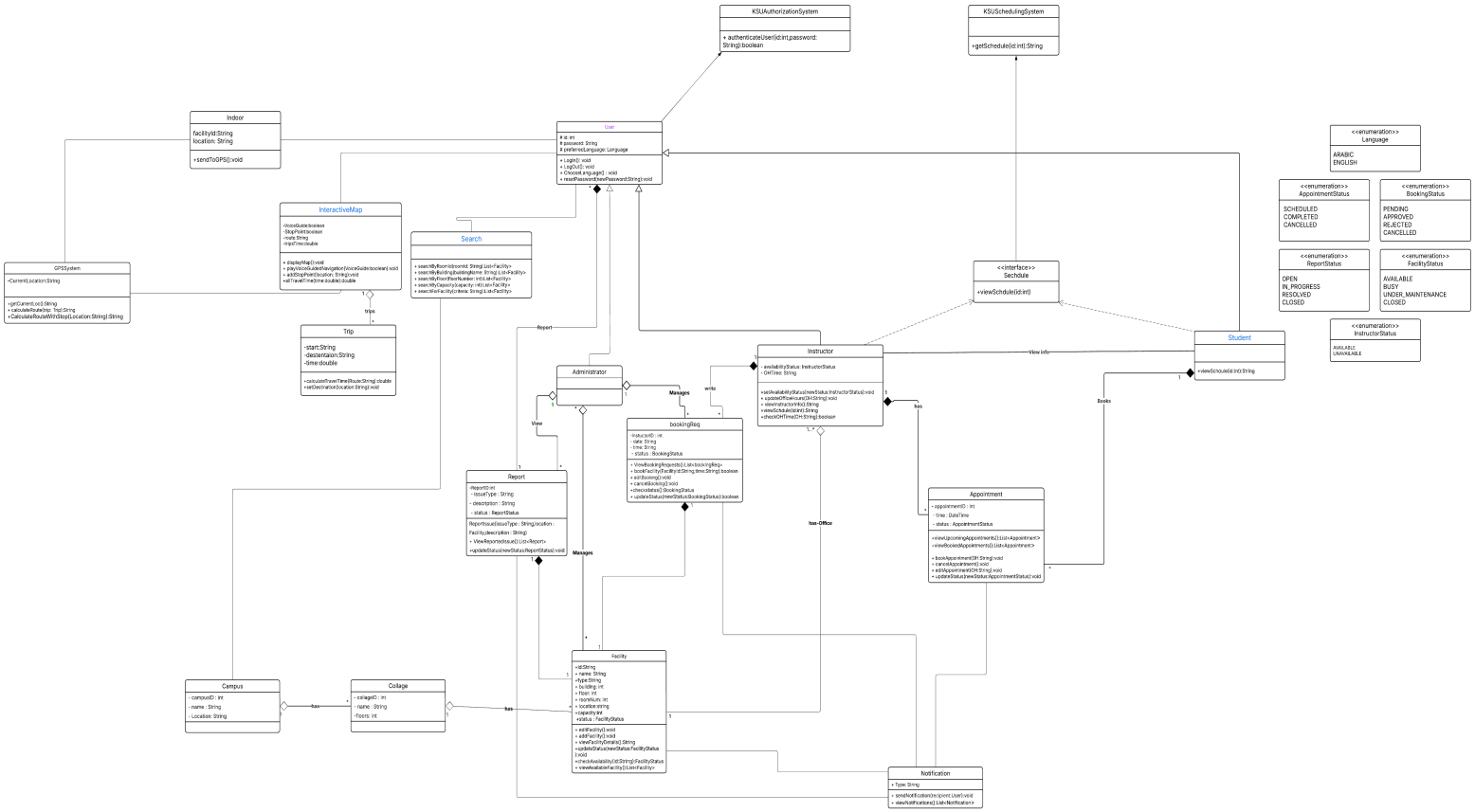

Figure 11: Class Diagram

7.2. Use Case Diagram

UML use case diagram is a visual representation that depicts the interactions between users (actors) and the system being developed. It provides an overview of the different use cases or functionalities of the system, and the actors involved in each interaction.

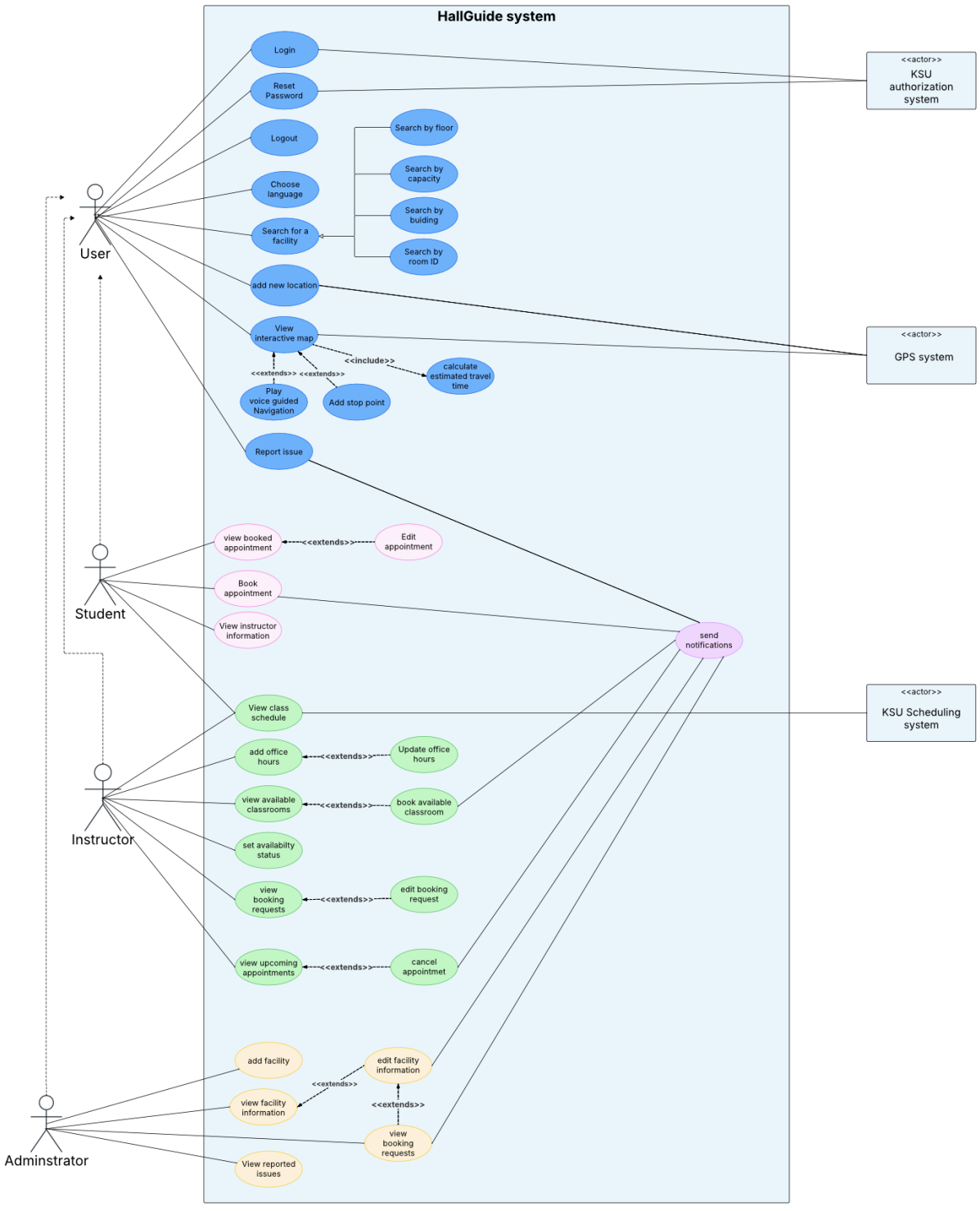

Figure 12: Use Case Diagram

7.3. Use Case Descriptions:

Use Case Description is a simple document that explains how a user interacts with a system to achieve a specific goal. It describes the steps or actions taken by the user and how the system responds. It also mentions any conditions or rules that need to be followed during the interaction.

7.3.1 Book available classroom:

Use Case Description	
System: HallGuide System	
Use Case Name: Book available classroom.	
Primary actor(s): Instructor	**Secondary actor(s):** Admin
Description: This case describes how an instructor can book an available classroom.	
Relationships: • **Includes:** none • **Extends:** none	
Pre-conditions: 1. The instructor must be logged in. 2. The facility must be registered	
Basic Flow:	
Primary actor (Instructor)	**System**
1. the instructor requests to book a facility by selecting a facility and time slot.	2. The system checks facility availability by facility id. 3. The system creates the booking request 4. notification is sent to the Administrator for approval.
Alternative and exceptional flows: **A2.** Facility is not available The request is denied immediately, and an error is displayed.	
Post-conditions: **Successful condition:** • Notification sent to admin. • Booking status (pending).	

Figure 13: Use Case Description [1]

7.3.2 Book an appointment:

Use Case Description	
System: HallGuide System	
Use Case Name: Booking an Appointment	
Primary actor(s): Student	**Secondary actor(s):** -
Description: A student wants to book an appointment with an instructor during their available office hours.	
Relationships: • **Includes:** none • **Extends:** none	
Pre-conditions: 1- The student is logged in. 2- The instructor has set their availability for appointments.	
Basic Flow:	
Primary Actor (Student)	**System**
1-The student asks for Instructor's information. 2- The student views the available office hours then selects an open time slot. 3-The student books the appointment by confirming the selected time slot and submitting the booking request.	4- The system checks instructor's office hour availability. 5- The system changes office hour status. 6- A **notification is sent** to the **Instructor and Student** confirming the appointment.
Alternative and exceptional flows: **A4.** instructor is not available The request is denied immediately, and an error is displayed.	
Post-conditions: **Successful condition:** 1-The instructor's availability status is updated and marks the booking slot as Busy. 2- The Student and Instructor receive notifications.	

Figure 14: Use Case description [2]

7.3.3 view interactive map:

Use Case Description	
System: HallGuide	
Use case name: view interactive map	
Primary actor(s): users	**Secondary actor(s):** GPS System
Description: This use case describes how users can use the interactive map to find the best route along with a voice navigation and get the estimated travel time.	
Relationships: • **Extends:** Play voice guided navigation, Add stop point. • **Includes:** Calculate estimated travel time.	
Pre-conditions: 1. User must be logged in 2. The facility must be registered 3. User must select a specific facility as a destination.	

Basic Flow:		
Primary actor (users)	**System**	**Secondary actor (GPS system)**
1-User selects "view interactive map" for a specific facility	2-The system sends a request to the GPS system to determine the user's current location. 4-The system initiates a trip with start and destination. 6- The system invokes the "Calculate Estimated Travel Time" use case to determine and display travel time along the route. 7- the system displays the interactive map showing the user's current location, the best route and the estimated travel time.	3- GPS system returns the current location coordinates to the system 5-using current location and the specific destination the GPS finds the best route.
Alternative and exceptional flows: **A.3, T**he GPS system fails to provide the current location, The system notifies the user and prompts for manual entry of the starting point. **A.4, T**he system is unable to compute the best route due to data or connectivity issues, The user is informed of the error and offered the option to retry. **A.6a,** user can add a stop point to include along the route. **At step 6b,** user can play voice guided navigation to receive auditory navigation cues during travel.		
Post conditions: **Successful condition:** the interactive map is displayed with the user's current location, calculated best route and the estimated travel time to the selected destination. Additional features like "add stop points" or "voice-guided navigation" are properly integrated and reflected in the map view.		

Figure 15: Use Case description [3]

7.4. Sequence Diagrams:

A sequence diagram models the interactions between objects in a single use case. It illustrates how different parts of a system interact with each other to carry out the function and the order in which the interactions occur when a particular use case is executed.

7.4.1. Book Available Classroom:

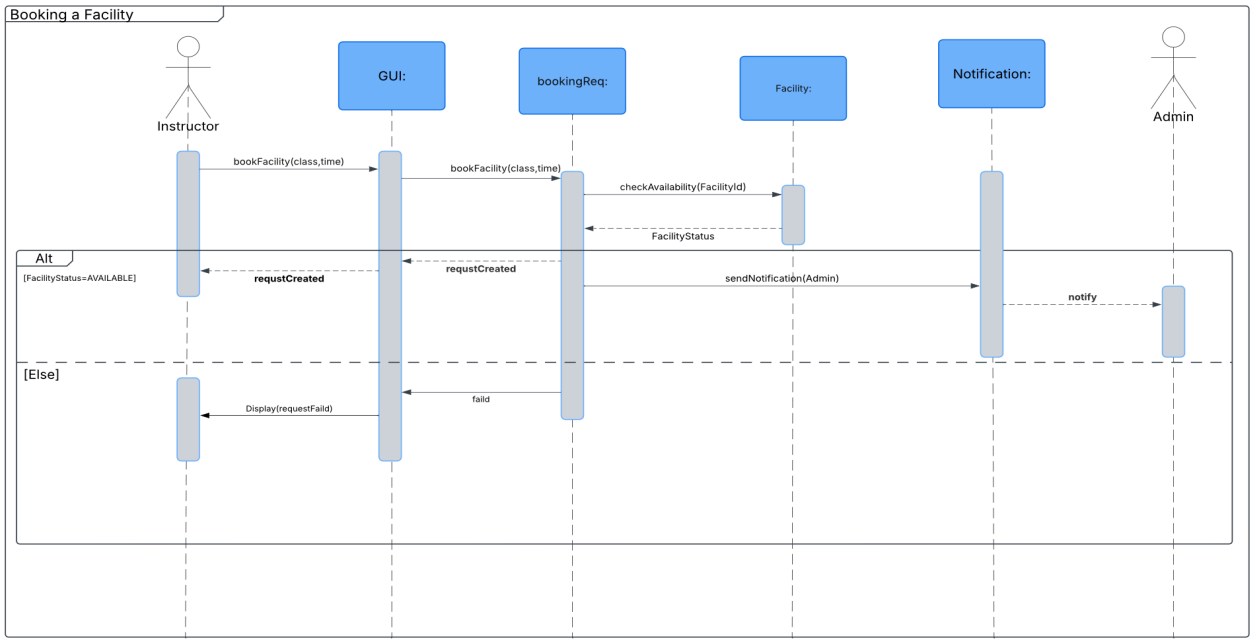

Figure 16: sequence diagram [1]

7.4.2. Book Appointment:

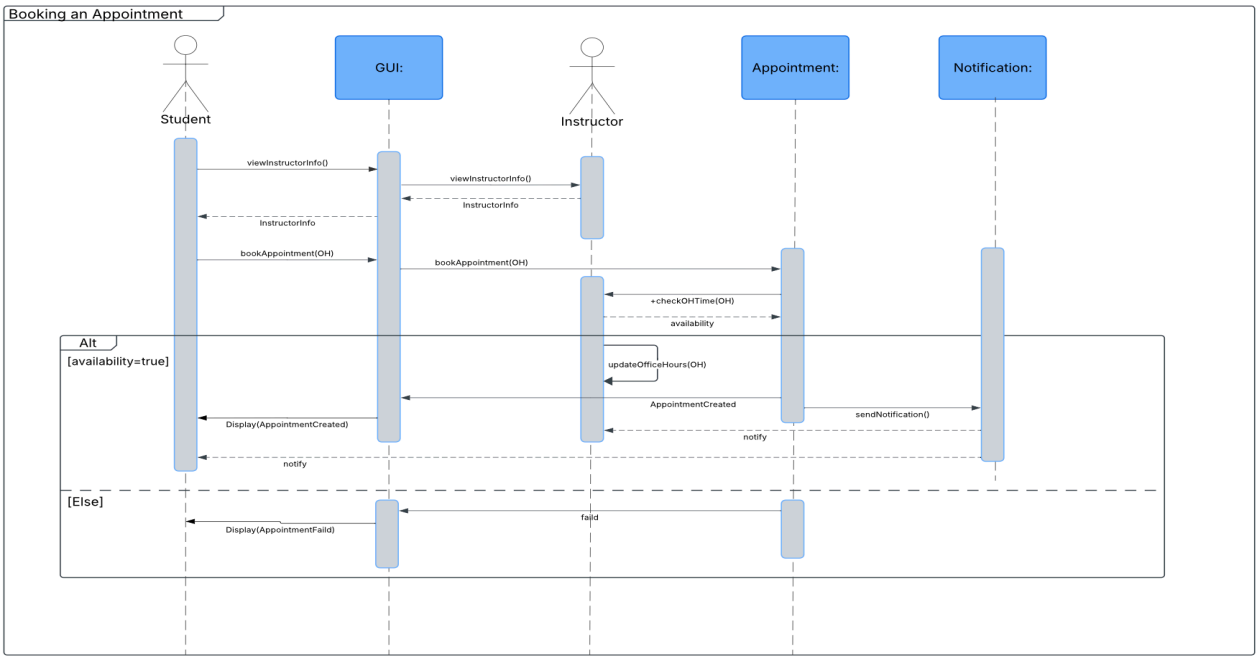

Figure 17: sequence diagram [2]

7.4.3. View Interactive Map:

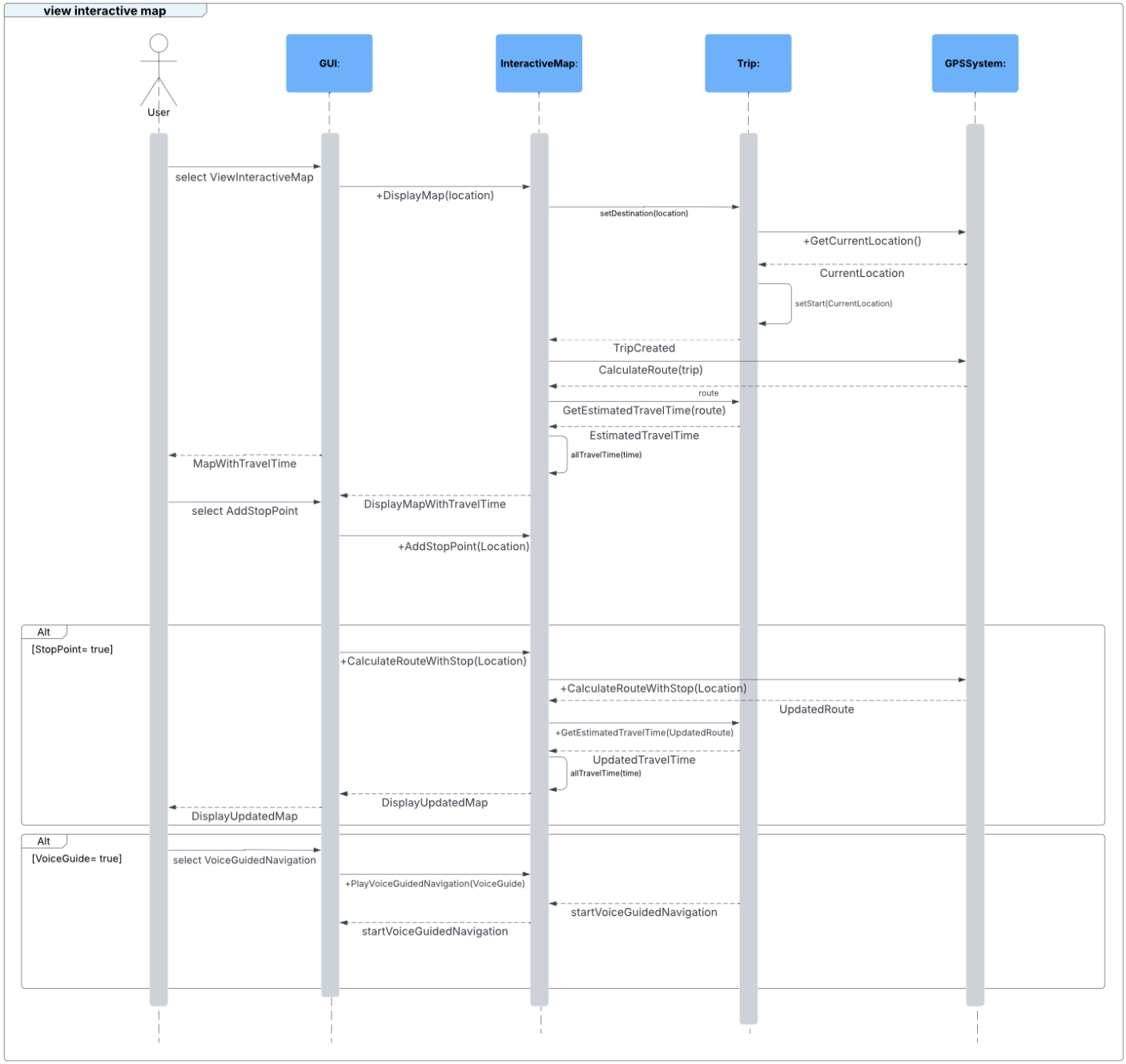

Figure 18: sequence diagram [3]

8. Quality Assurance Plan

8.1. Testing Strategy

the following types of testing will be conducted to ensure hallGuide system meets its functional and non-functional requirements:

8.1.1 Unit Testing: focuses on testing individual components of the system, such as:

- GPS module retrieving location
- Search functionality (finding classrooms/facilities)
- Interactive Map module (displaying routes)
- User authentication module

Conducted by developers using automated unit test frameworks.

8.1.2 Integration Testing: Ensures that modules work correctly when combined.
Test interactions between:

- HallGuide System and GPS System
- Interactive Map and Route Calculation modules
- User login and university authentication system

Helps identify data flow and communication errors.

8.1.3 System Testing: Evaluates the end-to-end functionality of HallGuide.
Ensures all functional requirements are met, including:

- Viewing an interactive map
- Adding stop points
- Playing voice-guided navigation
- Booking classroom requests
- Searching for facilities/classrooms

8.1.4. User Acceptance Testing (UAT): Confirms the system meets user requirements and is ready for real-world use.

- Conducted with real users (students, instructors, administrators) to validate usability.
- Ensures the system meets user needs and expectations.
- Feedback is collected to refine the system.

8.1.5 Compatibility Testing

- Tests the system on various devices, browsers, and screen sizes (e.g., mobile vs. desktop).
- Ensures smooth performance across platforms.

8.1.6 Performance Testing: Measures system response times and load capacity, Includes:

- Load Testing (Simulating multiple users accessing the system at once)
- Stress Testing (Testing under extreme conditions)
- Response Time Testing (Measuring map rendering speed)

8.1.7 Security Testing: Validates the system against security threats such as:

- Unauthorized access attempts
- Data breaches
- Encryption of sensitive data (e.g., login credentials)

8.2. Test Cases

8.2.1. Functional Test Cases:

TC-F1: User Login: Verify that users can log in with valid university credentials.

Scenario:

1. Open the app or website.
2. Enter valid university credentials (ID, password).
3. Click “Login”

Expected Result: Login is successful, and the user is redirected to the homepage.
Failure Result: If the credentials are incorrect, it will display an error message: *"Invalid username or password."*

TC-F2: Facility Search: Verify that users can search for facilities using valid search criteria.

Scenario:

1. Log in.
2. Click on the search bar.
3. Enter valid search criteria (e.g., "Building 06, Floor 1, Room 81").
4. Click the "Search".

Expected Result: The system displays the correct matching results.
Failure Result: If search input is invalid, it will display a message: *"No results found. Please enter a valid search input."*

TC-F3: Administrator Approving or Rejecting Booking Requests: Verify that administrators can approve or reject instructor booking requests.

Scenario:
1. Log in as an administrator.
2. Open Booking Requests.
3. Select a request.
4. Click Approve or Reject.

Expected Result: status changed to "Booked" if approved or remains "available" if rejected. Instructor receives an update notification about the status of their request.
Failure Result: status isn't updated, or notification isn't sent to the instructor.

TC-F4: Choosing Language Preference: Verify that users can switch between Arabic and English.

Scenario:
1. Log in.
2. Select the "Language" button.
3. Choose "Arabic" or "English."

Expected Result: The UI updates to the selected language immediately, the preference is saved for future logins.
Failure Result: If the language does not change the system fails to update the UI.

TC-F5: Viewing Instructor's Details: Verify that students can view an instructor's information, office hours, location.

Scenario:
1. Log in as a student.
2. Navigate to the Class Schedule section.
3. Select a course from the displayed schedule.
4. Click on the Instructor's Name associated with the selected course.

Expected Result: The profile displays instructor details, office hours, office location and real-time availability status.
Failure Result: If the instructor's details are missing, profile does not show office hours or location. Or Incorrect availability status is shown.

TC-F6: Instructor Booking a Classroom: Verify that instructors can request to book an available classroom.

Scenario:

1. Log in as an instructor.
2. Searches for a classroom by entering criteria (e.g., capacity, room, floor).
3. Select a classroom and time slot.
4. Confirm the booking.

Expected Result: Booking request is submitted, and the administrator is notified.
Failure Result: If the classroom is already booked, it will display an error message: *"This time slot is unavailable."*

8.2.2. Non-Functional Test Cases:

TC-NF1: Usability-System Learnability: Verify that the users able to learn the system within 20 minutes.
Scenario:

1. A new user accesses the system for the first time.
2. They navigate through the main features (login, navigation, search, booking, reporting).
3. They attempt common tasks based on provided instructions.

Expected Result: Users feel confident in using the system within 20 minutes.
Failure Scenarios: If the User needs more than 20 minutes to understand navigation.

TC-NF2: Portability-Mobile Compatibility: Verify the system works on both iOS and Android.

Scenario:

1. Install the mobile app on iOS and Android devices.
2. Perform basic operations (login, navigation, search, booking, reporting issues).

Expected Result: All functions work consistently across both platforms.
Failure Scenarios: If the Features work on one OS but not the other.

TC-NF3: Correctness-Travel Time Estimation: Verify the estimated travel time within ±10% of actual time.

Scenario:

1. Set a starting location and destination.
2. Start navigation and compare estimated time with actual time.

Expected Result: Travel time is accurate within ±10%.
Failure Scenarios: Travel time is off by more than ±10%.

TC-NF4: Security-Account Lockout after Failed Logins: Verify the accounts will be locked for 10 minutes after five failed login attempts.

Scenario:

1. Attempt to log in with an incorrect password five times.
2. Try logging in again.

Expected Result: System locks the account for 10 minutes.
Failure Scenarios: Users can keep attempting passwords indefinitely.

TC-NF5 :Reliability-System Availability : Verify that system is available 99.90% of the time .

Scenario:

1. Monitor system uptime over a month.
2. Record any downtime instances.

Expected Result: System uptime is 99.90% or higher.
Failure Scenarios: Uptime falls below 99.90%.

8.3. Metrics

Software quality metrics are quantifiable attributes used to gauge the performance, reliability, security, usability, and maintainability of a software system. The metrics provide objective information that helps development teams measure the extent to which the software meets user expectations, industry standards, and business needs.

By tracking software quality metrics, organizations can identify issues early, improve efficiency, and increase user experience. Software quality metrics are crucial in software engineering, product management, and IT operations for making the system stable, scalable, and secure and meeting functional and non-functional requirements

Category	Metric Name	Formula	Target Value
Usability	Time to Learn System	Total Training Time / Number of Users	≤ 20 minutes
Portability	Platform Compatibility	(Number of Supported Platforms / Total Targeted Platforms) × 100	100%
Correctness	Navigation Accuracy	Actual Distance - Calculated Distance	≤ 5 meters
Interoperability	Data Export Capability	Number of Supported Export Formats	At least CSV & XML
Security	Security Breach Detection Rate	(Detected Intrusions / Total Intrusions Attempted) × 100	≥99%
Reliability	System Uptime (%)	(1 - Total Downtime / Total Time) × 100	≥99.9%
Maintainability	Comment Density	(Number of Commented Lines / Total Lines of Code) × 100	≥ 55%
Flexibility	Deployment Downtime	Time Taken to Deploy a New Feature	≤ 30 minutes
Flexibility	Campus Expansion Time	Time Taken to Configure Additional Campuses	≤ 4 hours
Scalability	Concurrent User Handling	Max Number of Users Supported Without Degradation	5000 users

Table [3]: metrics

8.4. Tools

To ensure HallGuide meets its quality standards, various tools will be used for automation, bug detection, performance validation, and security testing. These tools help improve reliability, scalability, and user experience across different environments, ensuring the system functions smoothly under real-world conditions.

Testing type	Tools used
Unit Testing	JUnit, PyTest
Integration Testing	Postman (API testing)
System Testing	Selenium, Cypress
Performance Testing	JMeter, LoadRunner
Security Testing	OWASP ZAP, Burp Suite
Compatibility Testing	BrowserStack, LambdaTest

Table [4]: tools used for testing

8.5. Schedule

Testing

Name	Subitems	Status	Timeline - Start	Timeline - End	Dependent On	Duration	Completion Date
unit testing	module retrieving location,Search functionality (finding classrooms/facilities),Interactive Map module (displaying routes),User authentication m	Done	2025-02-15	2025-02-19		5	2025-03-20
Subitems	**Name**	**Timeline - Start**	**Timeline - End**	**Dependency**			
	GPS module retrieving location	2025-03-15	2025-03-17				
	Search functionality (finding classrooms/facilities)	2025-03-15	2025-03-18				
	Interactive Map module (displaying routes)	2025-03-18	2025-03-19	GPS module retrieving location			
	User authentication module	2025-03-17	2025-03-19				
Integration testing	HallGuide System and GPS System,Interactive Map and Route Calculation modules,User login and university authentication system	Done	2025-02-20	2025-02-23	unit testing	4	2025-03-20
Subitems	**Name**	**Timeline - Start**	**Timeline - End**	**Dependency**			
	HallGuide System and GPS System	2025-02-20	2025-02-21				
	Interactive Map and Route Calculation modules	2025-02-22	2025-02-23	HallGuide System and GPS System			
	User login and university authentication system	2025-02-20	2025-02-23				
System Testing	Viewing an interactive map,Adding stop points,Playing voice-guided navigation,Booking classroom requests,Searching for facilities/classrooms	Done	2025-02-24	2025-03-02	Integration testing	7	2025-03-20
Subitems	**Name**	**Timeline - Start**	**Timeline - End**	**Dependency**			
	Viewing an interactive map	2025-02-24	2025-02-27				
	Adding stop points	2025-02-28	2025-03-02	Viewing an interactive map			
	Playing voice-guided navigation	2025-02-28	2025-03-02	Viewing an interactive map			
	Booking classroom requests	2025-02-26	2025-03-02				
	Searching for facilities/classrooms	2025-02-27	2025-03-02				
User Acceptance test	users (students, instructors, administrators) to validate usability.,Ensures the system meets user needs and expectations.,Feedback is collected	Done	2025-03-03	2025-03-07	System Testing	5	2025-03-20
Subitems	**Name**	**Timeline - Start**	**Timeline - End**	**Dependency**			
	Conducted with real users (students, instructors, administrators) to validate usability.	2025-03-03	2025-03-05				
	Ensures the system meets user needs and expectations.	2025-03-03	2025-03-05				
	Feedback is collected to refine the system.	2025-03-06	2025-03-07	al users (students, instructors, administrators) to validate usability., Ensures the system meets user needs and expectations.			
Compatibility Testing	Tests the system on various devices, browsers, and screen sizes (e.g., mobile vs. desktop).,Ensures smooth performance across platforms.	Done	2025-03-08	2025-03-10	System Testing	3	2025-03-20
Subitems	**Name**	**Timeline - Start**	**Timeline - End**	**Dependency**			
	Tests the system on various devices, browsers, and screen sizes (e.g., mobile vs. desktop).	2025-03-08	2025-03-10				
	Ensures smooth performance across platforms.	2025-03-08	2025-03-10				
Performance Testing	ulating multiple users accessing the system at once),Stress Testing (Testing under extreme conditions),Response Time Testing (Measuring m	Done	2025-03-11	2025-03-14	tegration testing, System Testi	4	2025-03-20
Subitems	**Name**	**Timeline - Start**	**Timeline - End**	**Dependency**			
	Load Testing (Simulating multiple users accessing the system at once)	2025-03-11	2025-03-12				
	Stress Testing (Testing under extreme conditions)	2025-03-13	2025-03-14	Load Testing (Simulating multiple users accessing the system at once)			
	Response Time Testing (Measuring map rendering speed)	2025-03-11	2025-03-13				
Security Testing	Unauthorized access attempts,Data breaches,Encryption of sensitive data	Done	2025-03-15	2025-03-18	system Testing, Integration testi	4	2025-03-20
Subitems	**Name**	**Timeline - Start**	**Timeline - End**	**Dependency**			
	Unauthorized access attempts	2025-03-15	2025-03-16				
	Data breaches	2025-03-17	2025-03-18	Unauthorized access attempts			
	Encryption of sensitive data						
			2025-02-15	**2025-03-18**			**2025-03-20**

Figure 19: testing schedule

9. *Component diagram*

The HallGuide component diagram shows the GUI interacting with eight micro-services: Navigation, Search, Schedule Viewer, Booking Management, Appointment Management, Issue Reporting, Notification, and Authentication. Services access a shared Central Database, while Authentication, Schedule Viewer, and Navigation integrate with external campus systems. Notification also pushes updates back to the GUI. The diagram highlights a clean separation between interface, logic, data, and external services, following a structured three-tier design.

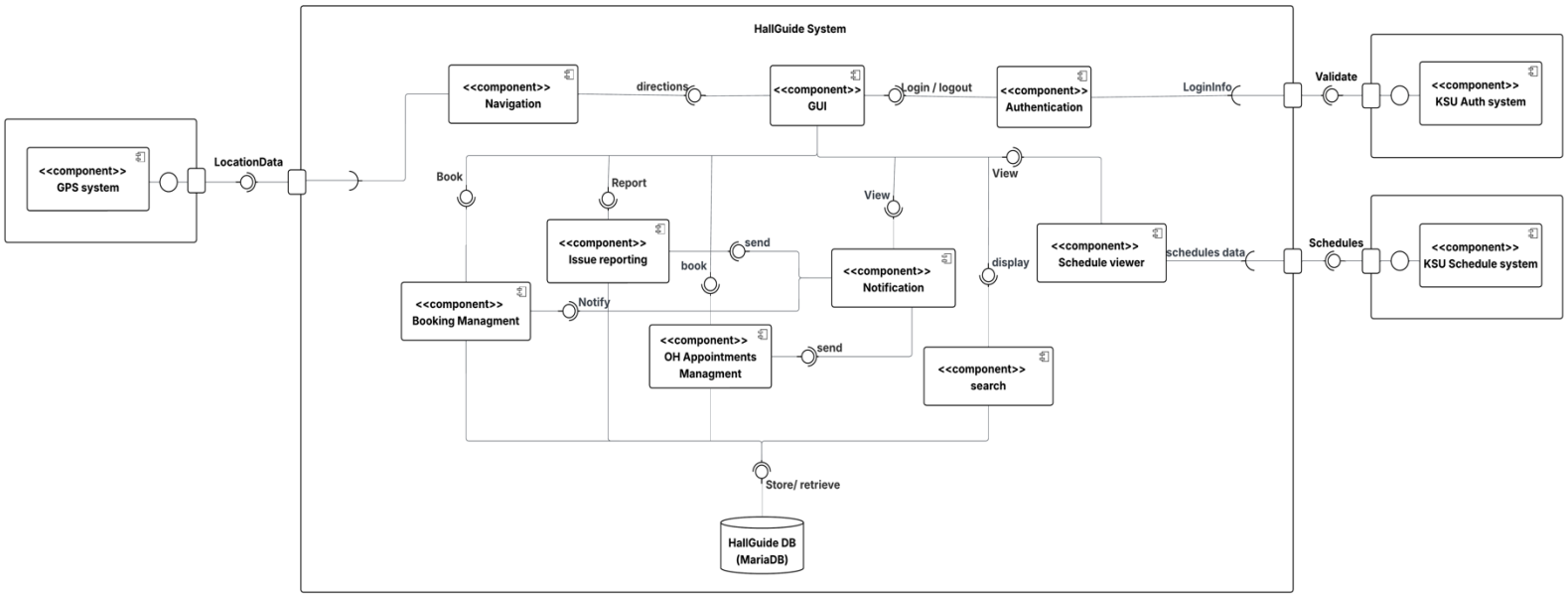

Figure 20: Component Diagram

For better quality: click here

10. *Deployment diagram*

The deployment diagram arranges HallGuide across three physical tiers so the path of every request is easy to follow.
Presentation tier: Two client nodes *clientMobile* and *clientPC* render the UI locally and send every action via HTTPS to a lightweight web server that serves static files and forwards API calls.
Application tier: an application server runs stateless micro-services Navigation, Search, Schedule Viewer, Booking Management, Appointment Management, Issue Reporting, Notification, and Authentication. These services can scale horizontally and are the sole point of contact with external systems: Authentication consults KSU Auth, Schedule Viewer pulls data from KSU Scheduling, and Navigation queries the GPS feed, all over secure HTTPS.
Data tier: a dedicated DB server hosts MariaDB, holding users, facilities, bookings, schedules, and issue reports. Services reach it through JDBC; no client connects directly. Nightly encrypted replicas are pushed to AWS for off-site backup.

This layout keeps UI, logic, and data isolated, allows independent scaling, and secures campus information end to end.

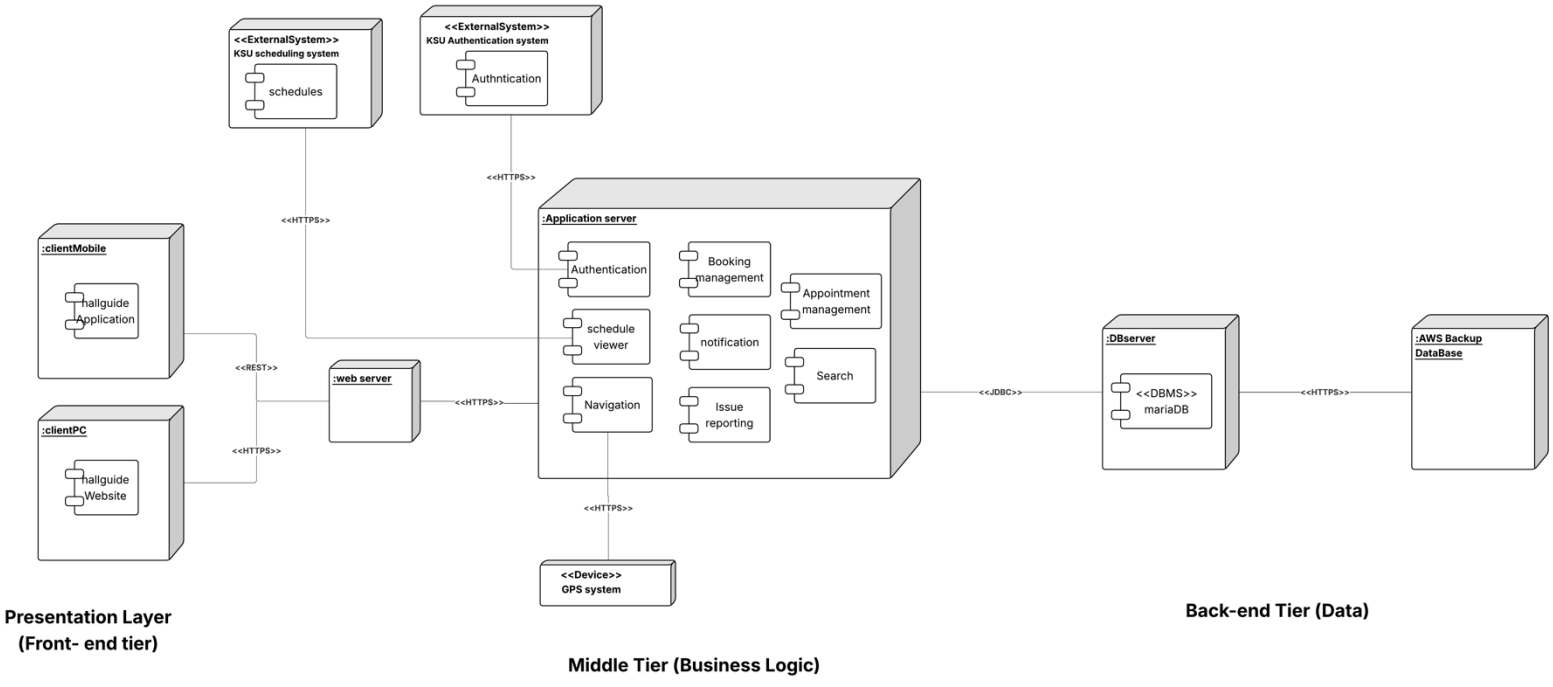

Figure 21: Deployment diagram

For better quality: Click here

11. *System Architecture*

Since **HallGuide** must deliver real-time navigation, bookings, and updates to thousands of campus users, scalability, maintainability, and secure data handling are essential.
To meet these needs, we adopt a structured, **distributed three-tier architecture**: a presentation tier, an application tier composed of discrete services, and a data tier.
By cleanly separating interface, logic, and data, HallGuide achieves independent scaling, straightforward maintenance, and strong protection of user information qualities that are critical for a campus-wide way-finding system.

11.1. Architecture description

HallGuide adopts the classic three-tier model, a pattern widely used for building scalable and maintainable software.
At the top, the presentation layer (client side) hosts a web client and a cross-platform mobile client that render maps, search panels, booking forms, and alerts; they capture user input and forward it to the next tier exclusively over HTTPS APIs.

The middle tier the **application server tier** implemented as a suite of microservices processes every incoming request and enforces HallGuide's business logic. Each independent service validates data, authenticates users, performs route calculations, resolves booking conflicts, triggers notifications, or integrates with external platforms such as the campus GPS feed and the KSU authentication and scheduling systems. Intensive tasks such as path-finding and schedule matching run inside these micro-services, which then communicate with the data tier via well-defined queries while shielding the database from direct client access.
Because the tier is decomposed into loosely coupled micro-services, HallGuide can scale hotspots, deploy updates, or recover failures in one module without affecting the others, all while delivering consistent rules, secure workflows, and smooth handling of simultaneous campus-wide requests.

The third tier the **data layer** operates a central MariaDB instance replicated nightly to encrypted AWS storage. It stores all persistent information (users, facilities, bookings, schedules, and issue reports) and exposes controlled data-access methods that the micro-services in the application tier use to read and update information while maintaining referential integrity and transaction safety. Isolated by strict access controls and automated backups, the data layer safeguards campus data and supports future growth.
This clean separation of interface, logic, and data allows HallGuide to scale each tier independently, simplifies maintenance, and safeguards user informationqualities essential for a campus-wide way-finding system.

11.2. Why this architecture?

We chose a **distributed three-tier architecture** because it aligns directly with HallGuide's non-functional goals scalability, reliability, security, and seamless campus integration.
By cleanly separating the presentation, application, and data tiers, each layer can scale independently: micro-services in the application tier can be replicated to handle spikes in booking or navigation traffic, and database replicas can be added without disrupting the other layers.
This separation also streamlines **maintenance**, allowing front-end teams to refine the interface while back-end developers deploy or roll back a single micro-service, thereby avoiding system-wide releases.
Security is strengthened by confining sensitive data to a private data tier; every request passes through the API gateway and business logic, where validation and authentication are enforced before any database access.
Performance improves as thin clients off-load computation to services optimized for pathfinding, search, and booking, while the data tier concentrates exclusively on transactional integrity and indexed queries.
Nightly encrypted backups to AWS and the **loose coupling** of services provide **fault isolation** if one service fails, the rest of the platform continues to operate. Taken together, these attributes make the distributed three-tier model the most practical and effective foundation for a system like HallGuide.

11.3. Alternative architecture style

We examined two alternative styles before adopting the distributed three-tier design. **MVC** was attractive for its clear separation of view, controller, and model, yet placing user-interface and business logic in a single deployable would have limited horizontal scalability and introduced tight coupling; for example: a traffic in routing requests during the first week of classes or a spike in booking traffic ahead of mid-term exams could saturate that single MVC back end and throttle the entire application..

We also assessed a **traditional layered architecture** that stacks presentation, business, and data-access layers, but each extra layer adds processing overhead and memory consumption. For HallGuide's real-time map and booking responses even fractions of a second matter those additional hops would degrade performance, and a change in one business module would still require redeploying the entire layer. Because performance, fine-grained scalability, and independent deployment are critical to a campus-wide way-finding service, we rejected both MVC and layered approaches in favor of the more flexible distributed three-tier architecture.

12. Architecture diagram

This Architecture diagram shows how hallGuide follows 3 tier architecture. **Clients** form the presentation tier, sending all user actions over HTTPS. **Micro-services** receive those requests, call the secure MariaDB data tier, or interact with external systems, then return a response. **Data** live exclusively in MariaDB, which services access through controlled queries.

Example flow: a logged-in student books an office-hour appointment. The mobile app sends a "reserve slot" request to the Appointment Management service. Appointment Management checks the instructor's open times, verifies availability in HallGuideDB, records the reservation, and then triggers the Notification service to push a confirmation back to the student.

The request touches all three layers: UI, logic, and data yet each remains isolated, preserving scalability, maintainability, and security

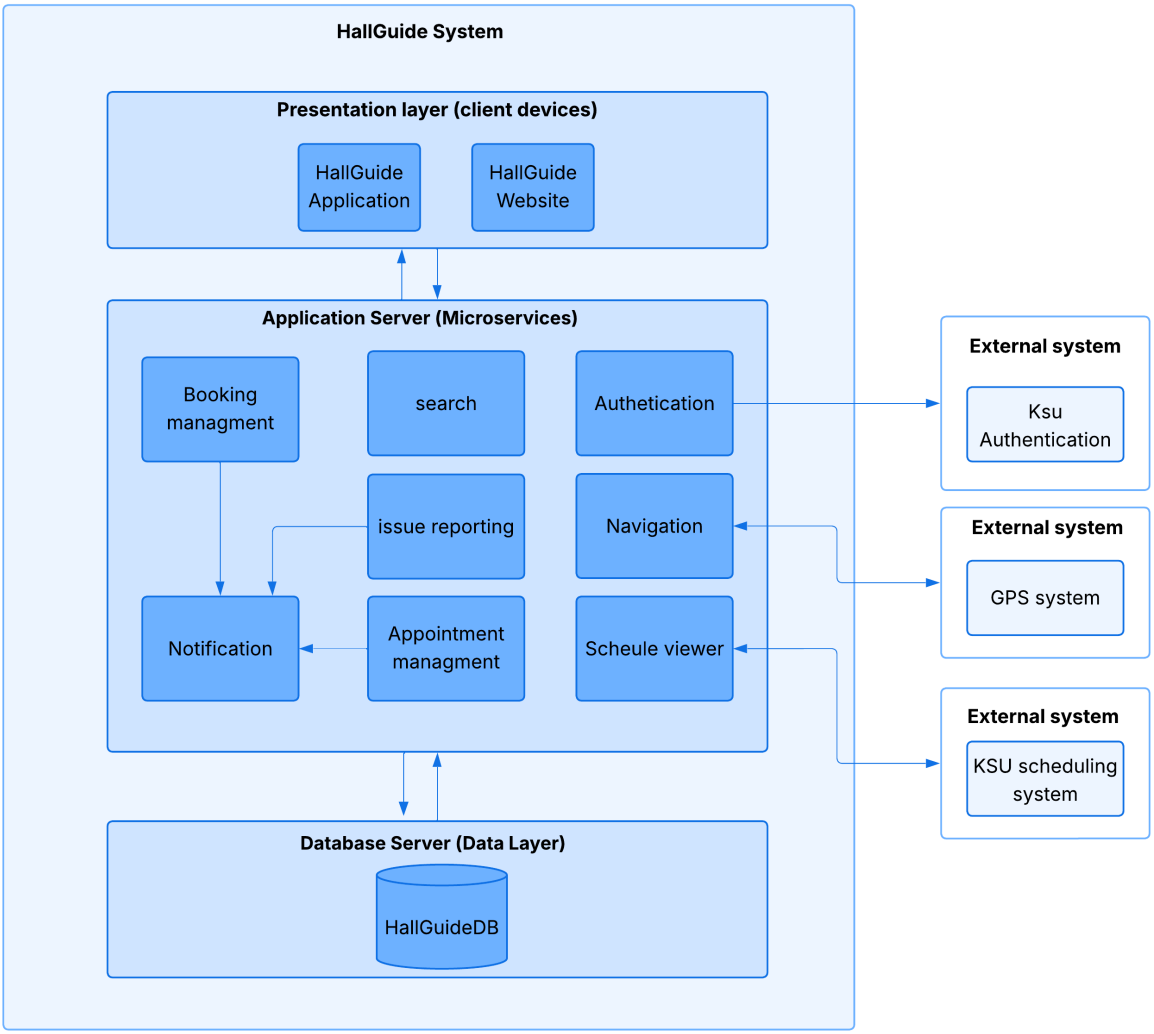

Figure 22: Architecture diagram

12.1. Updated Class Diagram

After choosing the HallGuide system architecture and finalizing its three-tier structure, updates were applied to the class diagram. These updates aligned the entities, services, and user views specifically with the HallGuide design, clearly organized the system layers, and accurately represented core relationships, such as the association between facilities and reports within the campus environment.

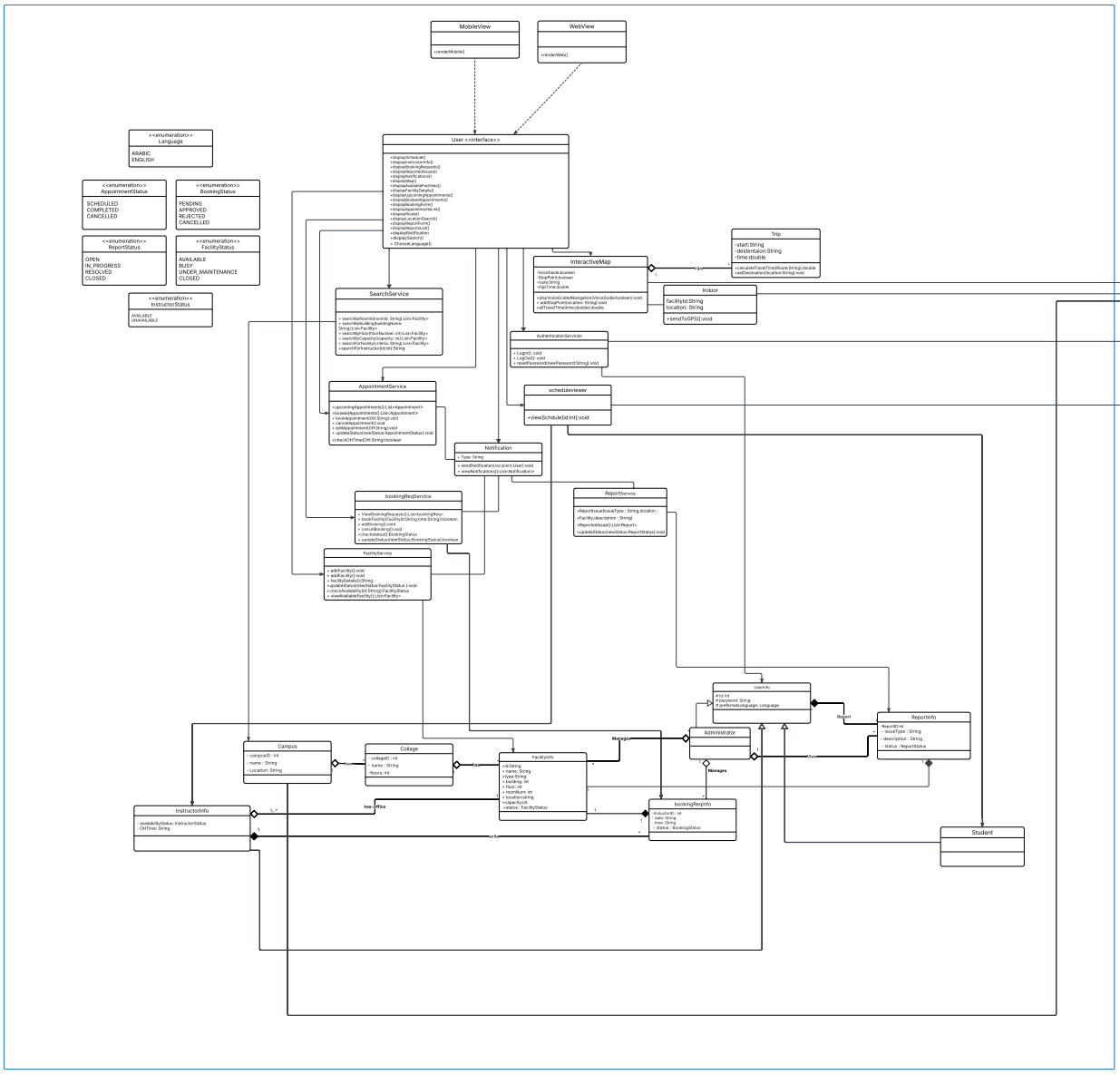

Figure 23: Updated Class Diagram

For better quality: Click here

13. references

[1] A. Dennis, B. Wixom and R. Roth, System Analysis & Design, the United States of America: John Wisley & Sons, Inc., 2012.

[2] "What Is Indoor GPS and How Does It Work?," Resonai.com, 2022. https://www.resonai.com/blog/indoor-gps

[3] Figma, "Figma: the Collaborative Interface Design tool.," Figma, 2024. https://www.figma.com/

[4] Dennis, Wixom, and Roth Systems Analysis and Design, 3rd Edition Copyright 2006 © John Wiley & Sons, Inc.

[5] GeeksforGeeks, "Software Testing Tools," GeeksforGeeks, Jan. 2020, doi: https://doi.org/10/2839/5625/10464/3713/7127.

[6]D. Galin, Software quality assurance. Harlow, England ; New York: Pearson Education Limited, 2004.

[7]"15 Quality Metrics Every Project Manager Needs to Know," by Shane Drumm, ProjectManagers.net, June 11, 2024. https://projectmanagers.net/15-quality-metrics-every-project-manager-needs-to-know/ (accessed Mar. 12, 2025).

[8]"Online Diagram Software & Visual Solution," Lucidchart. https://www.lucidchart.com/pages

[9]B. A. Doctor, "Use Case Description Basics," Business Analysis Doctor, Feb. 19, 2019. https://thebadoc.com/ba-techniques/f/use-case-description-basics

[10]K. Wiegers and J. Beatty, Software Requirements. Sydney: Pearson Education, 2013.

[11]M. Richards, Fundamentals of software architecture : an engineering approach. O'reilly Uuuu-Uuuu, 2020.