**Software Requirements Specification (SRS)**

**Smart Campus Services Portal**

**Document Version:** 1.0  
**Date:** April 16, 2025  
**Prepared by:** GB SFG (Group Assignment)

**Table of Contents**

1. [Introduction](#_1._Introduction)
   1. 1.1 [1.1 Purpose](#_1.1_Purpose)
   2. 1.2 [1.2 Scope](#_1.2_Scope)
   3. 1.3 [Definitions, Acronyms, and Abbreviations](#_1.3_Definitions,_Acronyms,)
   4. 1.4 [1.4 Overview](#_1.4_Overview)
2. [Overall Description](#_2._Overall_Description)
   1. 2.1 [Product Perspective](#_2.1_Product_Perspective)
   2. 2.2 [Product Functions](#_2.2_Product_Functions)
   3. 2.3 [User Classes and Characteristics](#_2.3_User_Classes)
   4. 2.4 [Operating Environment](#_2.4_Operating_Environment)
   5. 2.5 [Design and Implementation Constraints](#_2.5_Design_and)
   6. 2.6 [Assumptions and Dependencies](#_2.6_Assumptions_and)
3. [Specific Requirements](#_3._Specific_Requirements)
   1. 3.1 [Functional Requirements](#_3.1_Functional_Requirements)
   2. 3.2 [Non-functional Requirements](#_3.2_Non-functional_Requirements)
   3. 3.3 [Interface Requirements](#_3.3_Interface_Requirements)
4. [Other Requirements](#_4._Other_Requirements)
5. [Appendices](#_5._Appendices)

* [5.1 Glossary](#_5.1_Glossary)
* [5.3 Use Cases & UML Diagrams](#_5._2_Use)

1. [Use Case Specifications](#_6_Use_Case)

* [References](#_References)

# **1. Introduction**

## 1.1 Purpose

This document's objective is to outline the **Smart Campus Services Portal's** functional and non-functional needs. The design, development, testing, and validation phases are planned to use this Software Requirement Specification (SRS), which offers a thorough explanation of the functionality anticipated of the system.

## 1.2 Scope

A role-based web program called the Smart Campus functions Portal was created to provide a number of campus functions, like as:

* **User Authentication:** Access for Students, Lecturers, and Admins.
* **Service Booking:** Reserving study rooms, appointments, etc.
* **Timetable Viewing:** Providing students with their class schedules.
* **Maintenance Issue Reporting:** Allowing users to report issues.
* **Notifications:** Centralized notification centre for updates and alerts.
* **Admin Dashboard:** Analytics and management tools for campus administrators.

React will be used in the development of the user interface to guarantee a responsive and **consistent experience across various devices and browsers.**

## 1.3 Definitions, Acronyms, and Abbreviations

* **SRS:** Software Requirements Specification.
* **UI:** User Interface.
* **API:** Application Programming Interface.
* **PWA:** Progressive Web Application.
* **CRUD:** Create, Read, Update, Delete.
* **Roles:** Designated types of users (Students, Lecturers, Admins).

## 1.4 Overview

This document is divided into sections that describe the general system, detailed requirements, and supplemental materials. It is intended for usage by developers, testers, project managers, and other stakeholders.

# **2. Overall Description**

## 2.1 Product Perspective

The Smart Campus Services Portal is an independent system that interfaces with campus databases and third-party services (for notifications, geolocation, etc.). It is designed to integrate easily into existing campus IT infrastructures and extend functionalities with optional features like geolocation and PDF reporting.

## 2.2 Product Functions

* **User Account Management:** Registration, login, and profile management.
* **Service Booking:** Enabling users to book study rooms, appointments, or other campus resources.
* **Timetable Viewer:** Displaying class schedules for students in a calendar or list format.
* **Issue Reporting:** Allowing users to report and track maintenance issues.
* **Notification Centre:** Broadcasting campus updates and user-specific alerts.
* **Admin Analytics Dashboard:** Displaying usage statistics, service bookings, and maintenance reports.

## 2.3 User Classes and Characteristics

* **Students:** Seek easy access to schedules, room bookings, and notifications.
* **Lecturers:** Require timetable management and service booking for lecture facilities.
* **Administrators:** Need a comprehensive view of campus services, analytics, and system management tools.

## 2.4 Operating Environment

* **Client Side:** Modern web browsers (e.g., Chrome, Firefox, Edge) and mobile devices.
* **Server Side:** The back end will be hosted in a Java-based server environment, with the database managed using MySQL.
* **Platforms:** The client application will be developed using React for web interfaces, while the back-end services will be built using JAVA.

## 2.5 Design and Implementation Constraints

* **Technology Stack:**
  + Front-end: React (web).
  + Back-end: JAVA.
* **Standards & Guidelines:** Compliance with web accessibility standards and responsive design principles.
* **Security Constraints:** Implementation of secure communications (HTTPS), data encryption, and adherence to privacy standards.
* **Performance Constraints:** The system should provide a fast user experience with server response times under two seconds for critical operations.

## 2.6 Assumptions and Dependencies

* Reliable network connectivity is assumed for cloud services and third-party integrations.
* The campus IT infrastructure supports necessary API integrations.
* Future scalability and integration needs are considered for increasing user numbers and additional features.
* Dependencies on third-party services (e.g., mapping/geolocation APIs, notification services) are documented and managed.

# **3. Specific Requirements**

## 3.1 Functional Requirements

### 3.1.1 User Authentication and Authorization

* **1:** The system shall allow users to register with unique email addresses and validate these accounts.
* **2:** The system shall permit secure login and logout functionality.
* **3:** The system shall enforce role-based access control, providing differentiated experiences for Students, Lecturers, and Admins.
* **4:** Password recovery and account management functionalities shall be available.

### 3.1.2 Service Booking

* **5:** Users shall be able to view available rooms or appointment slots in real time.
* **6:** The system shall enable booking reservations with confirmation and calendar integration.
* **7:** Users shall receive email and in-app notifications confirming their bookings.

### 3.1.3 Timetable Viewer

* **8:** The system shall display a personalized timetable for students.
* **9:** Timetable data shall be synced with campus databases and updated dynamically.

### 3.1.4 Issue Reporting

* **10:** The system shall allow users to report maintenance issues by selecting a category and providing a description.
* **11:** Users shall be able to attach images or documents with their issue reports.
* **12:** A tracking system shall enable users and maintenance staff to monitor issue resolution progress.

### 3.1.5 Notification Centre

* **13:** The system shall provide a centralized area for campus-wide announcements and individual notifications.
* **14:** Notifications shall be delivered via both messaging and email based on user preferences.

### 3.1.6 Admin Dashboard with Analytics

* **15:** Admins shall have access to a dashboard featuring system usage statistics, service booking reports, and maintenance logs.
* **16:** The system shall support exporting data (e.g., PDF reports) for further analysis.
* **17 (Optional):** Geolocation-enabled features to show real-time room availability on an interactive map.

## 3.2 Non-functional Requirements

### 3.2.1 Performance

* **1:** The system shall respond to user actions within 2 seconds during peak loads.
* **2:** The system should support concurrent users with scalable architecture.

### 3.2.2 Security

* **3:** All user data must be encrypted at rest and in transit.
* **4:** The system shall implement robust authentication and session management protocols.
* **5:** Regular security audits and vulnerability assessments shall be conducted.

### 3.2.3 Usability and Accessibility

* **6:** The user interface shall be intuitive, with a responsive design for various devices.
* **7:** The system shall meet web accessibility standards (e.g., WCAG 2.1).

### 3.2.4 Reliability and Maintainability

* **8:** The system shall have an uptime of at least 99.5% over a calendar year.
* **9:** The codebase shall follow modular design principles to ease maintenance.
* **10:** Logging and error reporting mechanisms must be implemented to facilitate troubleshooting.

### 3.2.5 Scalability

* **11:** The architecture shall allow scaling horizontally to support increased user loads.
* **12:** The design should permit the introduction of new features without major refactoring.

## 3.3 Interface Requirements

### 3.3.1 User Interface (UI)

* **1:** The web UI should be built using React, providing dynamic, responsive interactions.
* **2:** The user interface will be developed using React to ensure a responsive and consistent experience across different browsers and devices.
* **3:** Consistent design elements (colors, typography, navigation) across all interfaces are mandatory.

### 3.3.2 Software Interfaces

* **1:** The application shall expose a set of RESTful APIs for communication between the front end and the JAVA back end.
* **2:** APIs shall support JSON as the primary data format.
* **3:** Integration with third-party services (e.g., email notifications, geolocation APIs) shall be documented and secured.

### 3.3.3 Hardware Interfaces

* **1:** While the application is web-based, it should be compatible with standard devices such as desktops, tablets, and smartphones.
* **2:** Any required peripheral integrations (e.g., barcode scanners for resource check-ins) should be considered in future expansion phases.

### 3.3.4 Communication Interfaces

* **1:** The system must use HTTPS for all data transmissions.
* **2:** Real-time features may utilize WebSocket’s or similar protocols for timely notifications.

# **4. Other Requirements**

* **Data Management:**The system will use a robust database solution (SQL or NoSQL, based on final architectural decisions) with scheduled backups and disaster recovery processes.
* **Development and Deployment:**The development process will follow an Agile methodology, with version control via GitHub. Continuous integration/continuous deployment (CI/CD) pipelines shall be established.
* **Documentation:**Detailed user guides, API documentation, and developer manuals must be produced and maintained.
* **Testing:**Comprehensive unit, integration, and user acceptance testing (UAT) plans should be developed. Automated testing frameworks will be integrated into the CI/CD pipeline.

# **5. Appendices**

## 5.1 Glossary

* **Smart Campus:** A digitally enabled campus environment integrating various services.
* **Role-Based Access Control (RBAC):** A method of regulating access to a system based on user roles.
* **PWA:** Progressive Web Application, which offers app-like features using modern web technologies.

## 5. 2 Use Cases & UML Diagrams

* Detailed UML diagrams (use case, class, and sequence diagrams) should be attached as separate documentation.
* Use case specifications corresponding to the functional requirements (e.g., “User Login”, “Book a Service”, “Report an Issue”) should be developed.

# **6. Use Case and Specifications**

## 6.1. Use case:

### 6.1.1. Student:

* + - 1. As a student, I want to update my account details so that my information is always accurate and up to date.
      2. As a student, I want to view the class timetable so that I can be aware of my class times.
      3. As a student, I want to view my assigned lecturer's schedule so that I know when and where I can find them.
      4. As a student, I want to book an appointment with my lecturer so that I can receive assistance.
      5. As a student, I want to report a maintenance issue so that it can be resolved promptly.
      6. As a student, I want to book a study room so that I have a place to focus and study.

### 6.1.2. Lecturer:

* + - 1. As a lecturer, I want to report a maintenance issue so that it can be resolved promptly.

2. As a lecturer, I want to update the class timetable so that all the time slots of each session are up to date.

3. As a lecturer, I want to update my personal schedule so that my daily availability is reflected correctly.

4. As a lecturer, I want to update my account details so that my user information remains current.

### 6.1.3. Admin:

1. As an Admin, I want to report a maintenance issue so that it can be addressed and resolved promptly.
2. As an Admin, I want to create user accounts so that users can be added to the system.
3. As an Admin, I want to view user accounts so that I can verify their existence in the system.
4. As an Admin, I want to delete user accounts so that they can be permanently removed from the system.
5. As an Admin, I want to generate reports so that I can analyse system and user activity.

### 6.1.4. System:

* 1. As for the system, I want to send notifications so that the users can be notified.
  2. As for the system, I want to send alerts so that the users can be reminded.

## Use Case:

# References

## *Software Engineering: A Practitioner’s Approach* (8th or 9th Edition) by Roger S. Pressman, 2019 – McGraw-Hill Education *(Note: Online purchase or access through university library is required)*

## *Object-Oriented Software Engineering: Using UML, Patterns, and Java* by Bernd Bruegge and Allen H. Dutoit *(Pearson Education, accessible through institutional login or purchase)*

## *Software Engineering (9th Edition)* by Ian Sommerville – Pearson, 2011 *(Commonly used academic reference, available in university libraries or eBook format)*