**System Requirements Document for IU Spaces**

**Cover Page**

**Project Name:** IU Spaces – Issue Tracker  
**Student Name:** Dorris Tazi  
**Course Name:** System Implementation and design  
**Instructor:** **Jafrina Jabin**  
**Date:** 12/21/24

**Table of Contents**

1. **Introduction** 1.1 Purpose  
   1.2 Scope  
   1.3 Definitions
2. **Customer Problem Statement and System Requirements** 2.1 Customer Problem  
   2.2 System Requirements
3. **Functional Requirement Specification** 3.1 User Registration and Authentication  
   3.2 Issue Reporting  
   3.3 Issue Viewing  
   3.4 Issue Status Management
4. **System Sequence Diagram**
5. **Activity Diagram**
6. **User Interface Specification** 6.1 Layout and Functionality  
   6.2 Visual Design
7. **Traceability Matrix**
8. **System Architecture and System Design**
9. **Algorithms and Data Structures** 9.1 Data Structures  
   9.2 Algorithms
10. **User Interface Design and Implementation** 10.1 Design Process  
    10.2 Frontend Implementation
11. **Test Design and Implementation** 11.1 Test Plan  
    11.2 Test Cases
12. **Project Plan**
13. **References**

**1. Introduction**

**1.1 Purpose**

The purpose of this document is to define the system requirements for the **IU Spaces** project. This application will help in tracking, managing, and reporting maintenance issues within campus buildings and rooms.

**1.2 Scope**

The system is a web-based solution that enables users to submit issues related to campus facilities. It provides a platform for users to report problems, which can then be tracked by the administrators.

**1.3 Definitions**

* **Issue:** A problem related to campus facilities that requires attention.
* **User:** A person submitting or viewing issues.
* **Administrator:** A person managing and resolving issues reported by users.

**2. Customer Problem Statement and System Requirements**

**2.1 Customer Problem**

Universities are faced with managing campus facilities and resolving maintenance issues in a timely manner. Currently, there is no centralized system for reporting and managing these issues, leading to inefficiency and delayed resolutions.

**2.2 System Requirements**

* **Functional Requirements:** The application must allow users to report issues and view their status.
* **Non-Functional Requirements:** The system must be scalable, secure, and provide real-time feedback to users.

**3. Functional Requirement Specification**

**3.1 User Registration and Authentication**

* Users must create an account to access the system.
* Authentication will require a secure login method.

**3.2 Issue Reporting**

* Users should be able to submit issues with necessary details.
* Issues must be categorized by building, room, description, and priority.

**3.3 Issue Viewing**

* Users can view reported issues and filter by building and priority.
* Administrators will have additional functionality to manage and resolve issues.

**3.4 Issue Status Management**

* Administrators can change the status of reported issues (resolved, in progress, etc.).

**4. System Sequence Diagram**

[Include your system sequence diagram here. This diagram should depict the interaction between users, the system, and the database during an issue report process.]

**6. User Interface Specification**

**6.1 Layout and Functionality**

* **Home Page:** Displays available actions like reporting an issue and viewing reported issues.
* **Report Issue Page:** Allows users to enter details for a new issue report.
* **View Issues Page:** Displays all reported issues with filter options.

**6.2 Visual Design**

The user interface is designed for ease of use, with a clean layout, responsive design for mobile and desktop views, and accessible form fields.

**7. Traceability Matrix**

[Include a table here linking each functional requirement to its corresponding system design component. For example:]

| **Requirement ID** | **Description** | **Design Component** |
| --- | --- | --- |
| FR1 | User Registration | User Authentication Module |
| FR2 | Issue Reporting | Issue Submission Form |
| FR3 | Issue Viewing | Issue Display Table |

**8. System Architecture and System Design**

The system will follow a **client-server architecture**:

* **Frontend (Client-side):** Built using HTML, CSS, JavaScript (React or similar).
* **Backend (Server-side):** PHP with MySQL for data storage.
* **Database:** MySQL for managing user data and issue reports.

**9. Algorithms and Data Structures**

**9.1 Data Structures**

* **Linked List:** Used for storing issues in the database.
* **Hash Tables:** For fast lookups of issues by priority or building.

**9.2 Algorithms**

* **Sorting Algorithm:** QuickSort will be used to sort issues based on priority or date reported.

**10. User Interface Design and Implementation**

**10.1 Design Process**

The user interface was designed keeping in mind accessibility, simplicity, and responsiveness. All pages have consistent navigation, and forms are clearly labeled.

**10.2 Frontend Implementation**

HTML, CSS, and JavaScript were used to develop the user interface. The front-end is responsive and can adjust according to screen size.

**11. Test Design and Implementation**

**11.1 Test Plan**

* **Unit Testing:** Each module (like form validation, issue reporting) is tested individually.
* **Integration Testing:** Testing interactions between modules (e.g., submitting an issue, viewing it).

**11.2 Test Cases**

* **Test Case 1:** Verify that users can submit a valid issue.
* **Test Case 2:** Verify that invalid input (e.g., non-numeric room number) triggers an error.

**12. Project Plan**

The project is divided into **phases**:

1. **Planning and Analysis (1 week)**
2. **System Design (2 weeks)**
3. **Implementation (4 weeks)**
4. **Testing (1 week)**
5. **Deployment and Maintenance (Ongoing)**