A blue circle with white text

Description automatically generated

**COMSATS UNIVERSITY ISLAMABAD, ABBOTTABAD**

**Software Design Document of Hostel management system**

***Submitted by:***

Laiba Binte tahir FA21-BSE-019

Arfah Ali FA21-BSE-080

***Submitted to:***

Mam Neeli

Contents

[1.0 INTRODUCTION 3](#_Toc168838483)

[1.1 Purpose 3](#_Toc168838484)

[1.2 Scope 3](#_Toc168838485)

[1.3 Overview 4](#_Toc168838486)

[1.4 Reference Material 4](#_Toc168838487)

[1.5 Definitions and Acronyms 4](#_Toc168838488)

[2.0 SYSTEM OVERVIEW 5](#_Toc168838489)

[3.0 SYSTEM ARCHITECTURE 6](#_Toc168838490)

[3.1 Architectural Design 6](#_Toc168838491)

[3.1 Architectural Design 7](#_Toc168838492)

[3.2 Decomposition Description 10](#_Toc168838493)

[3.3 Design Rationale 10](#_Toc168838494)

[4.0 DATA DESIGN 11](#_Toc168838495)

[4.1 Data Description 11](#_Toc168838496)

[4.2 Data Dictionary 11](#_Toc168838497)

[5.0 COMPONENT DESIGN 13](#_Toc168838498)

[5.1 User Management Subsystem 13](#_Toc168838499)

[6.0 HUMAN INTERFACE DESIGN 15](#_Toc168838500)

[6.1 Overview of User Interface 15](#_Toc168838501)

[6.2 Screen Images 16](#_Toc168838502)

[6.3 Screen Objects and Actions 18](#_Toc168838503)

[7.0 REQUIREMENTS MATRIX 18](#_Toc168838504)

[8.0 APPENDICES 19](#_Toc168838505)

[Activity diagram 20](#_Toc168838506)

Software Design Document (SWDD) - Hostel Management System

# 1.0 INTRODUCTION

## 1.1 Purpose

This software design document (SWDD) describes the architecture and system design of the Hostel Management System. It serves as a comprehensive guide for the development team and stakeholders, providing detailed insights into the structural components and functionalities of the system. The primary goal of this document is to ensure that all parties involved in the project have a clear understanding of how the system is intended to be built, its various components, and their interactions.

The intended audience for this document includes:

* **Developers:** To provide detailed design specifications and guidance for coding.
* **Project Managers:** To understand the design and ensure that the project stays on track and meets the defined requirements.
* **Stakeholders:** To have a clear view of the system's architecture and design, ensuring it meets their needs and expectations.
* **Quality Assurance Team:** To understand the system's design for effective testing and validation.

## 1.2 Scope

The Hostel Management System aims to streamline the management of hostel facilities, addressing several core areas:

* **Room Allocation:** Automating the process of assigning rooms to students, including adding new rooms, viewing room details, generating fee structures, and terminating room allocations.
* **Student Management:** Efficiently handling student records, including adding, viewing, modifying, and deleting student information.
* **Staff Management:** Managing staff details, including wardens, security personnel, and cooks, with functionalities to add, view, modify, and delete records.
* **Incident Reporting:** Providing a systematic way to report, view, and manage incidents occurring within the hostel.
* **Feedback and Menu Management:** Collecting feedback from students and managing the hostel's menu.

The project goals include:

* **Improving Operational Efficiency:** Reducing manual workload through automation.
* **Enhancing User Experience:** Providing an intuitive interface for all users.
* **Ensuring Data Accuracy and Security:** Protecting sensitive data and maintaining accurate records.

## 1.3 Overview

This document is organized into sections detailing the system overview, architecture, data design, component design, human interface design, requirements matrix, and appendices. Each section provides specific information related to the overall design and implementation strategy.

## 1.4 Reference Material

The following documents were used as sources of information for the SWDD:

* Software Requirements Specification (SRS) Document: Provides detailed functional and non-functional requirements of the system.
* System Analysis Document: Offers an analysis of the current system, user needs, and potential improvements.

## 1.5 Definitions and Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Definition** |
| **SWDD** | Software Design Document, this document itself. |
| **SRS** | Software Requirements Specification, a document detailing the required functionalities and constraints. |
| **DFD** | Data Flow Diagram, a graphical representation of data flow within the system. |
| **OO** | Object-Oriented, a programming paradigm based on the concept of objects, which contain data and methods. |

# 2.0 SYSTEM OVERVIEW

The Hostel Management System is designed to facilitate the efficient management of hostel operations. Key functionalities include:

* **Room Allocation:** Managing the lifecycle of room assignments from addition to termination.
* **Student Records:** Handling student data comprehensively, including personal details and room allocations.
* **Staff Management:** Administering staff details, roles, and responsibilities.
* **Incident Reporting:** Allowing for systematic logging and management of incidents within the hostel.
* **Feedback and Menu Management:** Collecting and managing feedback from students and updating the hostel's menu accordingly.

The system supports multiple user roles, each with specific access rights and functionalities:

* **Administrators:** Full access to manage all aspects of the system.
* **Wardens:** Manage student and room allocations.
* **Staff:** Perform specific roles such as security and cooking.
* **Students:** Access to personal records, feedback submission, and view menu.

A diagram of a hotel management system

Description automatically generated

Figure 1 Use Case Diagram

By providing these functionalities, the Hostel Management System aims to enhance operational efficiency, user satisfaction, and data integrity within the hostel environment.

# 3.0 SYSTEM ARCHITECTURE

## 3.1 Architectural Design

For the Hostel Management System described in the requirements, the best architectural design would be a **Microservices Architecture**. Here’s why Microservices Architecture is suitable based on the provided requirements:

**Microservices Architecture**

**Benefits:**

1. **Modularity:**
   * Each subsystem (User Management, Room Management, Student Management, etc.) can be developed, deployed, and scaled independently. This aligns well with the modular decomposition described in the document.
2. **Scalability:**
   * Individual services can be scaled as needed. For instance, if the Room Management subsystem experiences high demand during semester start times, it can be scaled independently without affecting other services.
3. **Flexibility:**
   * Different subsystems can use the most appropriate technologies. For example, the Feedback and Menu Management subsystem can use a NoSQL database for flexible data storage, while the User Management subsystem might use a relational database for transactional integrity.
4. **Ease of Development and Deployment:**
   * Teams can work on different services simultaneously, accelerating the development process. Continuous Integration/Continuous Deployment (CI/CD) pipelines can be set up for each service, enabling faster releases and updates.
5. **Fault Isolation:**
   * If one service fails (e.g., Incident Management), it does not cause the entire system to fail. This improves the overall resilience of the system.
6. **Technology Heterogeneity:**
   * Each microservice can be developed using the most suitable technology stack, allowing for greater innovation and efficiency.

### 3.1 Architectural Design

#### High-Level Architecture:

1. **Client Layer:**
   * **Web Client:** Browser-based application built using React, Angular, or Vue.js.
   * **Mobile Client:** Mobile applications for Android and iOS using React Native or Flutter.
2. **API Gateway:**
   * Acts as a single entry point for all client requests, routing them to appropriate microservices. It can also handle cross-cutting concerns like authentication, rate limiting, and load balancing.
3. **Microservices Layer:**
   * **User Management Service:** Handles user registration, login, and logout functionalities.
   * **Room Management Service:** Manages room allocation, viewing, and termination.
   * **Student Management Service:** Manages student records.
   * **Staff Management Service:** Manages staff details.
   * **Incident Management Service:** Manages incident reporting.
   * A diagram of a web server

     Description automatically generated**Feedback and Menu Management Service:** Handles feedback submissions and menu updates.

Figure 2 Deployment Diagram

1. **Database Layer:**
   * **Relational Databases:** For structured data, use PostgreSQL or MySQL for services like User Management, Room Management, and Student Management.
   * **NoSQL Databases:** For flexible data storage, use MongoDB for services like Feedback and Menu Management.
2. **Authentication and Authorization:**
   * **Authentication Service:** Centralized service for handling user authentication using OAuth2.0, JWT, etc.
   * **User Directory:** Stores user credentials and related data.
3. **Inter-service Communication:**
   * **Synchronous Communication:** Using RESTful APIs for request/response interactions.
   * **Asynchronous Communication:** Using message brokers like RabbitMQ or Apache Kafka for event-driven communication.
4. **Monitoring and Logging:**
   * **Monitoring Tools:** Prometheus and Grafana for system monitoring.
   * **Logging:** Centralized logging using ELK stack (Elasticsearch, Logstash, Kibana).

A diagram of a computer program

Description automatically generated with medium confidence

Figure 3 Package Diagram

These subsystems interact with each other to provide seamless functionality to the users. The following diagram illustrates the interconnections between major subsystems and data repositories:

## 3.2 Decomposition Description

**User Management Subsystem:**

* **Modules:** Registration, Login, Logout
* **Database:** Relational (PostgreSQL/MySQL)

**Room Management Subsystem:**

* **Modules:** Add Room, View Room, Generate Fee, Terminate Allocation, Assign Facilities, Booked Room
* **Database:** Relational (PostgreSQL/MySQL)

**Student Management Subsystem:**

* **Modules:** Add Student, View Student, Modify Student, Delete Student
* **Database:** Relational (PostgreSQL/MySQL)

**Staff Management Subsystem:**

* **Modules:** Add Warden, View Warden, Modify Warden, Delete Warden, Add Security, View Security, Delete Security, Add Cook, View Cook, Delete Cook
* **Database:** Relational (PostgreSQL/MySQL)

**Incident Management Subsystem:**

* **Modules:** Add Incident, View Incident, Delete Incident
* **Database:** Relational (PostgreSQL/MySQL)

**Feedback and Menu Management Subsystem:**

* **Modules:** Add Feedback, Add Menu, View Menu, Update Menu
* **Database:** NoSQL (MongoDB)

## 3.3 Design Rationale

* **Modularity:** Microservices architecture naturally supports modular design, with each service handling specific functionalities.
* **Scalability:** Services can be scaled independently based on their load, ensuring efficient use of resources.
* **Flexibility:** Different technologies can be used for different services, allowing for optimized performance and development.
* **Fault Isolation:** Failures in one service do not cascade to others, improving overall system reliability.
* **Ease of Development:** Teams can work on services independently, enabling parallel development and faster delivery.

By adopting Microservices Architecture, the Hostel Management System can achieve high scalability, flexibility, and maintainability, aligning well with the project's requirements and goals.

# 4.0 DATA DESIGN

## 4.1 Data Description

The system transforms the information domain into structured data entities stored in a relational database. Key entities include users, rooms, students, staff, incidents, feedback, and menus.

## 4.2 Data Dictionary

**Hostel**

* **hostel\_name** (PK) (string): Name of the hostel
* **warden\_id** (FK) (int): ID of the warden in charge
* **location** (string): Location of the hostel
* **contact** (string): Contact information for the hostel

**Warden**

* **warden\_id** (PK) (int): Unique identifier for each warden
* **warden\_contact** (string): Contact number for the warden
* **warden\_email** (string): Email address of the warden
* **warden\_name** (string): Name of the warden
* **warden\_gender** (enum: Male, Female, Other): Gender of the warden
* **warden\_address** (string): Address of the warden
* **warden\_password** (string): Password for the warden's account

**Pending\_registration**

* **pending\_ID** (PK) (int): Unique identifier for each pending registration
* **stud\_id** (FK) (int): ID of the student waiting for registration

**Student**

* **stud\_id** (PK) (int): Unique identifier for each student
* **room\_no** (FK) (int): Room number assigned to the student
* **stud\_name** (string): Name of the student
* **stud\_address** (string): Address of the student
* **stud\_password** (string): Password for the student's account
* **stud\_email** (string): Email address of the student
* **stud\_contact** (string): Contact number for the student

**Room**

* **room\_no** (PK) (int): Unique identifier for each room
* **room\_type** (string): Type of the room (e.g., Single, Double)
* **room\_capacity** (int): Capacity of the room
* **room\_fee** (decimal): Fee for the room
* **room\_assignTo** (string): Assigned to (e.g., students)
* **room\_floor** (int): Floor number of the room
* **room\_building** (string): Building in which the room is located
* **room\_availability** (boolean): Availability status of the room

**Feedback**

* **feedback\_id** (PK) (int): Unique identifier for each feedback
* **stud\_id** (FK) (int): ID of the student giving feedback
* **room\_no** (FK) (int): Room number related to the feedback
* **feedback\_date** (datetime): Date when the feedback was given
* **feedback\_subject** (string): Subject of the feedback
* **contact\_no** (string): Contact number for follow-up
* **feedback\_category** (string): Category of the feedback (e.g., Complaint, Suggestion)

**Incident**

* **inci\_id** (PK) (int): Unique identifier for each incident
* **date** (datetime): Date of the incident
* **day** (string): Day of the incident
* **location** (string): Location where the incident occurred
* **description** (string): Description of the incident
* **time** (time): Time when the incident occurred

A diagram of a computer

Description automatically generated

Figure 4 ER Diagram

# 5.0 COMPONENT DESIGN

## 5.1 User Management Subsystem

* **Register:**

pseudocode

Copy code

function register(username, password, role): validate(username, password) if valid: addUserToDatabase(username, password, role) return success else: return error

* **Login:**

pseudocode

Copy code

function login(username, password): user = getUserFromDatabase(username) if user.password == password: return generateSessionToken(user) else: return error

* **Logout:**

pseudocode

Copy code

function logout(sessionToken): invalidateSessionToken(sessionToken) return success

A diagram of a project

Description automatically generated

Figure 5 Sequence Diagram - Add Room

Add Warden:

A diagram of a project

Description automatically generated

Figure 6 Sequence Diagram - add Warden

# 6.0 HUMAN INTERFACE DESIGN

## 6.1 Overview of User Interface

The user interface provides intuitive navigation through different functionalities. Users can log in, manage rooms, students, staff, report incidents, and provide feedback through an easy-to-use interface.

## 6.2 Screen Images

A screenshot of a login screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## 6.3 Screen Objects and Actions

* **Login Screen:**
  + Objects: Username field, Password field, Login button
  + Actions: User enters credentials and clicks Login
* **Dashboard:**
  + Objects: Navigation menu, Quick access buttons (e.g., Add Room, Add Student)
  + Actions: User navigates to different functionalities using the menu

# 7.0 REQUIREMENTS MATRIX

|  |  |
| --- | --- |
| **Functional Requirement** | **System Component** |
| Allocate rooms | Room Management Subsystem |
| Register | User Management Subsystem |
| Login | User Management Subsystem |
| Logout | User Management Subsystem |
| Manage students | Student Management Subsystem |
| Manage warden | Staff Management Subsystem |
| Manage staff | Staff Management Subsystem |
| Add incidents | Incident Management Subsystem |
| View incidents | Incident Management Subsystem |
| Delete incidents | Incident Management Subsystem |
| Add feedback | Feedback and Menu Management Subsystem |
| Add menu | Feedback and Menu Management Subsystem |
| View menu | Feedback and Menu Management Subsystem |
| Update menu | Feedback and Menu Management Subsystem |

# 8.0 APPENDICES

Appendices include additional diagrams, detailed workflows, or supplementary information relevant to the understanding of the SWDD.

### Activity diagram

A diagram of a computer program

Description automatically generated

Figure 7 Activity diagram (for student)

A diagram of a software project

Description automatically generated

Figure 8 Activity diagram for admin