System Design Document (SDD)

for

Hostel Management System

Version 0.1

Prepared by

Group Number: 03

Kasireddy Koti Reddy	**B230373CS**	**CS01**
Katta Abhiram	**B230669CS**	**CS01**
Amit Kumar	**B230801CS**	**CS01**
Dumpuru Sridhar	**B230093CS**	**CS01**

Instructor:	**Dr. Pranesh Das**
Course:	**Database Management System**
Lab Section:	**--**
Teaching Assistant:	**--**
Date:	**07 / 03 / 2025**

Contents

1. Introduction:

1.1 Purpose:

The purpose of this System Design Document (SDD) is to outline the architecture, components, and implementation details of the **Hostel Management System (HMS)**. This document serves as a blueprint for developers and stakeholders to understand how the system is structured and will be implemented.

1.2 Scope:

The HMS is designed to automate and manage hostel-related activities such as room allocation, fee payments, visitor tracking, complaints, maintenance requests, leave registers, and occupancy reports. The system will be web-based and accessible to different users based on their roles.

1.3 Target Audience:

- Administrators
- Faculty & Hostel Authorities
- Students

1.4 System Overview:

The HMS will be accessible via a web interface where users can interact based on their roles. It will include user authentication, dashboards, and functionalities to manage hostel operations efficiently.

2. System Architecture:

2.1 Architectural Design:

The system follows a three-tier architecture:

1. **Presentation Layer**: User Interface (Web frontend using React Js, Flask framework)
2. **Application Layer**: Business logic implemented in Python (Flask framework)
3. **Data Layer**: MySQL database storing hostel management records

2.2 Technology Stack:

- **Frontend: React.js**
- **Backend**: Python (Flask framework)
- **Database**: MySQL
- **Hosting**: Apache Server (XAMPP)

2.3 Deployment Model

- Deployed on a cloud-based server or hosted locally on university infrastructure.

3. User Roles & Access Control:

3.1 Assumptions for Hostel Management System:

1. **User Roles:**
    a. **Admins** (Hostel admin) is responsible for adding, deleting, and updating student records, room details, and other system data.
    b. **Students** can only view specific data (room information, hostel fee payment status, etc.), submit maintenance requests, and apply for leave through the system.
2. **Data Storage:**
    a. All information related to students, rooms, hostel details, payments, maintenance, additional staff and leave is stored in a centralized database.
    b. The database will have sufficient capacity to handle the entire system's data, including historical data like maintenance requests and payment history.
3. **Access Control:**
    a. Only authorized personnel (admins) will have access to sensitive operations such as updating student or room data.
    b. Each user (admin, staff, student) will have unique login credentials, and the system will validate these credentials before granting access.
4. **System Availability:**
    a. The system is web-based and assumed to be available 24/7, allowing students and staff to access it from any location with internet access.
    b. A stable internet connection and basic knowledge of using browsers are assumed for all users.
5. **Room Allocation:**

a. Students will be assigned rooms based on fee payment status, availability, and the system will keep track of the room's status (vacant/occupied).

6. **Maintenance Requests and Leave Applications:**
    a. Maintenance requests from students are logged in the system, with their status tracked (pending, in progress, completed).
    b. Students can apply for leave through the system, and the leave dates will be recorded and monitored by hostel authorities.
7. **Hostel Fee Payments:**
    a. Students can view the status of their hostel fee payments (paid/pending) online.
    b. Payment records will be updated automatically once fees are paid.

3.2 Constraints:

1. **Security and Authentication:**
    a. Each user (student, staff, admin) must have a unique ID and password.
    b. Admin access is restricted and protected with exclusive credentials to prevent unauthorized access to sensitive operations such as modifying room availability, payment records, or student details.
    c. Passwords are securely stored and encrypted to maintain privacy and security.
2. **Scalability:**
    a. The system must handle multiple concurrent users (students, staff, admins) without performance degradation.
    b. The database should be optimized for fast access and efficient handling of large datasets, such as student and room records.
3. **Data Integrity:**
    a. Foreign key constraints ensure that student data refers to valid room and hostel data.
    b. All records (student, room, payment) must follow defined rules, such as not allowing room numbers to be assigned to more than one student at the same time.
4. **Transaction History:**
    a. The system will store a history of transactions, including room assignments, leave applications, fee payments, and maintenance requests. Users will be able to view this history for tracking purposes.
5. **Web-Based Constraints:**
    a. Since the system is web-based, it relies on a stable internet connection.

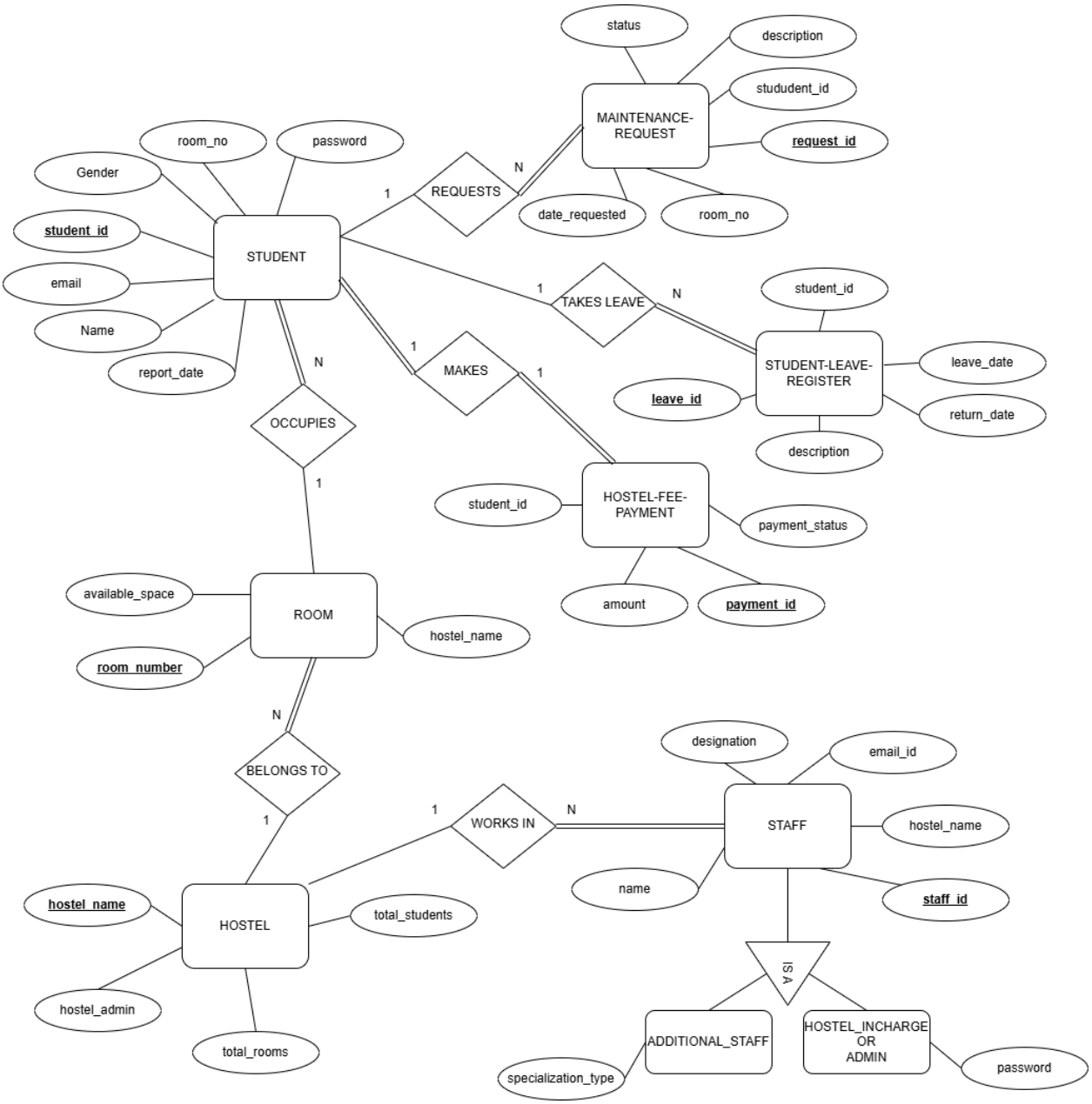
b. The user interface is designed for simplicity, assuming users have basic proficiency with web browsers and are familiar with operating computers (e.g., using the keyboard and mouse).

6. **Error Handling:**
    a. The system must handle errors (e.g., invalid data input, database errors) gracefully, providing meaningful feedback to users.
    b. There should be appropriate validation for user inputs, such as ensuring valid email formats, date ranges for leave applications, and non-empty fields for mandatory information.
7. **Backup and Data Recovery:**
    a. Regular backups of the database will be performed to ensure data is not lost in case of system failures or data corruption.
    b. There should be provisions for restoring data from backups in case of emergencies.

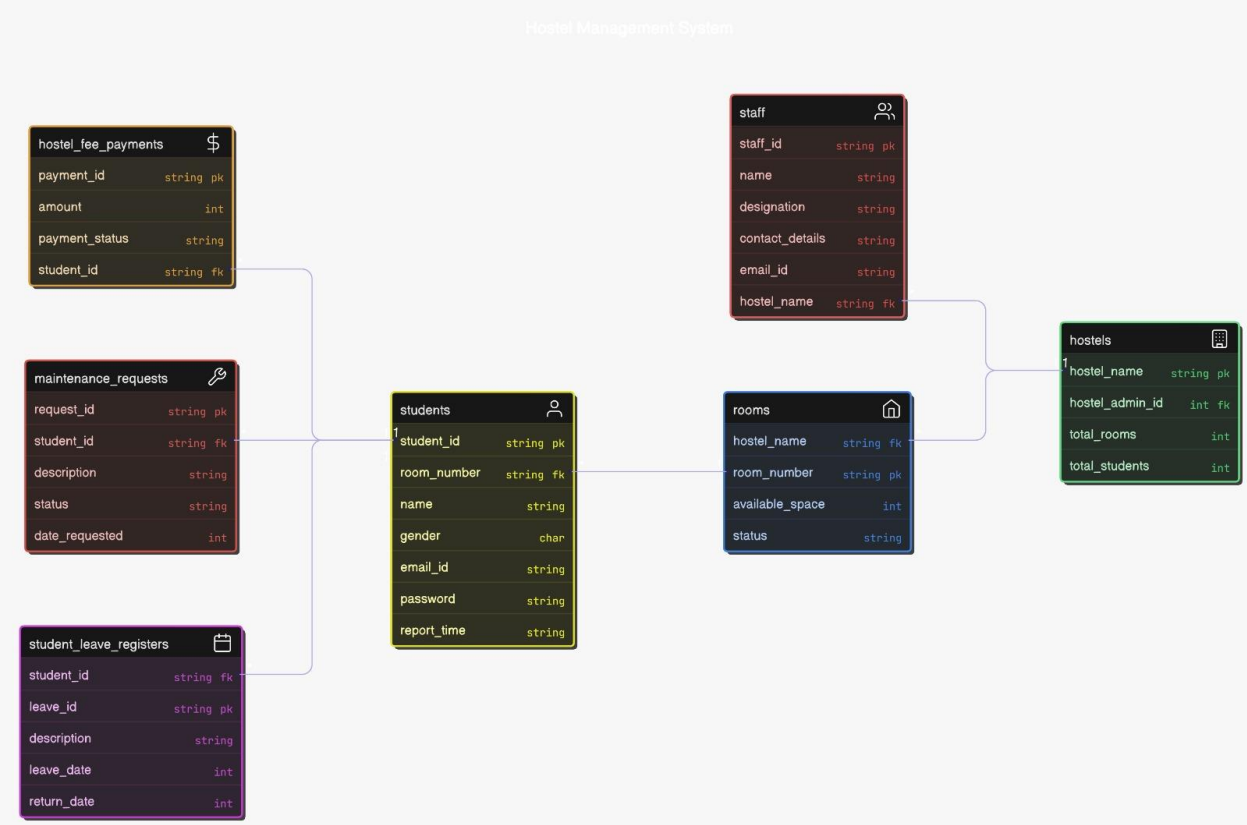
These assumptions and constraints help guide the development of a robust and reliable hostel management system, ensuring that all functional requirements and operational limitations are met.

4. Database Design:

4.1 Entity-Relationship (ER) Diagram:



4.2 Relational Schema Diagram:



4.3 Normalization:

1NF (First Normal Form)

The database schema satisfies the First Normal Form (1NF) as follows:

- All attributes contain **atomic values**. There are no composite or multivalued attributes. For example, in the STUDENT table, attributes such as Name, Email, Gender, and Room Number hold **single, indivisible values**.
- Each record is **unique**, ensuring no duplicate rows exist in any table. For instance, student_id serves as a unique identifier for students, payment_id uniquely identifies each payment transaction, and staff_id distinguishes each staff member in hostel.

2NF (Second Normal Form)

Since the database is already in 1NF, it must also satisfy the Second Normal Form (2NF). The schema follows 2NF because:

- **There are no partial dependencies** in any table. Each non-prime attribute is fully functionally dependent on the whole primary key and not just a part of it.

- In the STUDENT-LEAVE-REGISTER table, attributes like leave_date, return_date, and description depend **only on the composite primary key (leave_id, student_id)**. There is no partial dependency, as both fields together uniquely define each leave entry.

- Tables with **single-column primary keys**, such as ROOM and HOSTEL, inherently satisfy this condition since all non-key attributes are fully dependent on their respective primary keys.

Therefore, the database is in Second Normal Form (2NF).

3NF (Third Normal Form)

To ensure Third Normal Form (3NF), the schema must eliminate **transitive dependencies**. The database follows 3NF because:

- **There are no transitive dependencies** between attributes. Each non-key attribute depends **only on the primary key** of its table and not on any other non-prime attribute.
- For example, In the STUDENT table, attributes such as room_no and password are **directly dependent** on student_id, ensuring no transitive dependencies.

Therefore, the database meets 3NF normalization.

The normalization process ensures that the **Hostel Management System** database is **structured efficiently** by:

- Eliminating **redundant data** and preventing duplication.
- Ensuring **data integrity** by enforcing proper dependencies between attributes.

- Optimizing **storage and retrieval efficiency** for student records, room allocations, visitor tracking, fee payments, and complaints.

This structured database design improves the overall **reliability and maintainability** of the system while preventing anomalies in data storage.

5. System Components & Modules:

5.1 Functional Modules:

1. **User Authentication** (Login / Logout, Role-Based Access)
2. **Room Management** (Allocation, Availability, Status)
3. **Fee Payment Processing** (Due Payments, Transaction Records)
4. **Maintenance Requests** (Raise Requests, Assign Staff, Status)
5. **Student Leave Registers** (Student Leave)
6. **Report Generation** (Occupancy Reports, Payment Reports)

5.2 Non-Functional Requirements:

- **Security**: Role-based access control, encrypted passwords.
- **Scalability**: Support for multiple hostels.
- **Performance**: Optimized database queries.

6. Implementation Details:

6.1 Backend Implementation:

- Flask framework used to handle HTTP requests and database operations.
- RESTful APIs for interaction between frontend and backend.

6.2 Database Implementation:

- MySQL database with normalized tables and foreign key constraints.

6.3 Frontend Implementation:

- React.js for UI.
- JavaScript for interactive elements.

6.4. Future Enhancements:

- Mobile App Integration.
- Facial Recognition for Visitor Tracking.
- AI-based Room Allocation System.

7. Conclusion:

The **Hostel Management System** aims to simplify hostel administration and improve student experience through an automated, efficient, and secure system.

8. References:

- Hostel Management System Documentation
- Flask & MySQL Integration Guide

9. Data Dictionary:

ATTRIBUTE NAME	DATA TYPE	DESCRIPTION	CONSTRAINT	OPERATIONS
STUDENT:				
STUDENT_ID	String	Unique ID for each student	PRIMARY KEY	Create, Read, Update, Delete
Room Number	String	Assigned room	FOREIGN KEY	Assign, Update, Retrieve
Name	String	Students full name	NOT NULL	Retrive,Update
Gender	Char (1)	'M'/ 'F' / 'O'	CHECK (Gender IN ('M', 'F', 'O'))	Retrieve
Email_ID	String	College email	UNIQUE, NOT NULL	Retrieve,Update
PASSWORD	String	Used for login authentication	NOT NULL	Set,Update,Validate

REPORT_TIME	Timestamp	Last recorded report time	DEFAULT CURRENT_TIMESTAMP	Retrieve, Update
ROOM:				
Room_Number	String	Unique room identifier	PRIMARY KEY	Assign,Retrieve, Update
Hostel_Name	String	Associated hostel	FOREIGN KEY	Retrieve
Available_Space	Int	Number of vacant beds	CHECK (Available_Space >= 0)	Update,Retrieve
Status	String	'Vacant'/ 'Occupied'	CHECK (Status IN ('Vacant', 'Occupied'))	Update,Retrieve
HOSTEL:				
Hostel_Name	String	Unique hostel identifier	PRIMARY KEY	Retrieve
HOSTEL_ADMIN_ID	Int	Admin ID responsible	FOREIGN KEY	Assign,Retrieve
Total_Rooms	Int	Total number of rooms	CHECK (Total_Rooms > 0)	Retrieve
TOTAL_STUDENTS	Int	Max accommodation capacity	CHECK (TOTAL_STUDENTS >= 0)	Retrieve
HOSTEL FEE PAYMENT:				
PAYMENT_ID	String	Unique payment transaction ID	PRIMARY KEY	Retrieve, Validate
Amount	Int	Fee amount paid	CHECK (Amount >=0)	Retrieve
Payment Status	String	'Paid' / 'Pending'	CHECK (Payment Status IN ('Paid', 'Pending'))	Update,Retrieve
STUDENT_LEAVE_REGISTER:				
LEAVE_ID	String	Unique leave request ID	PRIMARY KEY	Retrieve

STUDENT_ID	String	Associated student	FOREIGN KEY → STUDENT(STUDENT_ID)	Retrieve
Description	String	Reason for leave	NOT NULL	Retrieve