Room Booking System for Student Dormitories

Project Report

Grace Anderson	11759304
Leo Curdi	11704166
Josh Evans	11802879
Calell Figuerres	11776119
Aaron Howe	11635434

February 23, 2025

CptS 451 - Introduction to Database Systems
Parteek Kumar

Abstract

Dormitory reservation software causes adverse effects on student bodies in higher education when lacking an efficient database management system. Many universities are seeing either a rising student enrollment, or seeing severe drops in enrollment, and either situation can be difficult to manage without a well-implemented, automated database. Our database management system follows a design structure that allows students to intuitively complete and submit applications for room reservations and gives administrative entities a system that's simple enough to easily navigate, and complex enough to have authoritative control. Our fullstack utilizes TypeScript for type safety and sound communication between the frontend and backend frameworks, React on the client-side, Express.js API with tRPC, and PostreSQL for databasement management. Our goal is to cut reservation assignment and processing time by at least half of what is currently shown by real-world data, and reduce conflict-causing errors such as double-reservations.

Table of Contents

I. Introduction

Managing room bookings in student dorms can be a difficult, complex, and time consuming process. Our booking system for student dorms aims to streamline this process by giving an efficient, user friendly, and automated solution. The system allows students to do all the tasks they need to do that are related with finding, scheduling and booking dorms all on the same platform. The system also allows for admins to oversee all actions being made by students,

Our system is designed to enhance accessibility, reduce admin workload and improve the experience for students. With many features like real time updates, automatic notifications, tracking, and more we are able to ensure a seamless booking process while maintaining the necessary regulations. This report details the design, development and implementations of our room booking system. It will cover the functional and non-functional requirements, structure, and key feature that make is an effective solution.

II. Functional and Non-Functional Requirements

II. 1. Functional Requirements

FR-1: Registration

Description	Users should be able to register with their academic credentials (ID, name, etc.)
Justification	Key “root” in allowing rooms to be assigned to people in the first place (e.g., users are the ones completing actions on the website)

FR-2: Administrator Designation

Description	Users should be able to be designated as system administrators who can approve/disapprove room bookings, room assignments, and occupancy reports.
Justification	Key “root” in allowing rooms to be assigned to people in the first place (e.g., administrators are the ones to figure out room assignments at all)

FR-3: Student Designation

Description	Users should be able to designated as students who can request certain rooms, make roommate requests, reserve lounges, submit maintenance requests, etc.
Justification	Key “root” in allowing rooms to be assigned to people in the first place (e.g., students are the ones who want to be assigned to rooms)

FR-4: Maintenance Request System

Description	Students must be able to submit and track maintenance requests for the room they’ve been assigned.
Justification	Ensures proper room maintenance and student satisfaction

FR-5: Room Search and Filtering

Description	Students must be able to search and filter rooms based on type, facilities, and availability

Justification	Enables efficient room selection process and filtering based on student preferences

FR-6: Real-time Booking Management

Description	System must handle room bookings in real-time with immediate availability updates
Justification	Prevents double bookings, ensures data accuracy, and allows for a much snappier user experience

FR-7: Roommate Requests

Description	Description: Students must be able to make and respond to roommate requests
Justification	Facilitates student preferences and social arrangements

FR-8: Booking History

Description	Users must be able to view their booking history and current reservations
Justification	Users must be able to view their booking history and current reservations

FR-9: Compliance Tracking

Description	System stores data related to each and every dorm room (names, safety compliance, max limit on occupancy, maintenance, etc.)
Justification	Administrators must keep track of how each room is being used, whether rules are being violated (and, if so, how often they are)

FR-10: Automated Notification System

Description	Website must send automatic notifications for booking confirmations, move in/out dates/times, maintenance updates, and inspections

Justification	Ensures that students receive important communications in a reliable, effective, and timely manner, to prevent students from missing important dates and times

FR-11: Data Analytics and Reporting

Description	Website should aggregate data to generate reports on room occupancy, bookings, and maintenance records
Justification	Provides administrators with valuable insights necessary for critical decision making, planning, and optimization.

II. 2. User Stories

1. As a user I want to register with my credentials
2. As a user I want to be designated as an admin who can approve/disapprove room bookings, etc.
3. As a user I want to be able to be designated as a student who requests certain rooms, etc.
4. As a student I want to be able to submit and track maintenance requests
5. As a student I want to be able to search and filter rooms based on criteria.
6. The system mush handle room bookings in real-time with immediate updates
7. As a student I want to make and respond to roommate requests
8. As a user I want to be able to view my booking history and current reservation
9. As the system it should store data related to each and every dorm room
10. As the website it should send auto notifications for booking confirmations
11. As the website, it should aggregate data to generate reports on room occupancy, etc.

II. 3. Non-Functional Requirements

1. **Secure** - the website should be reasonably secure, such as preventing user information from being leaked; at the same time, users should as be prevented from doing actions outside their "jurisdiction" (e.g., students shouldn't be allowed to handled administrator tasks).
2. **Responsive** - the website should be reasonably responsive and "snappy"; e.g., it should take as little time as possible for user pages to load or be updated after submitting data changes.
3. **Scalable** - the website should handle a reasonable number of current users, and be able to scale to handle high-traffic periods
4. **Data Validation** - website must validate all input data before processing
5. **Error Resistant** - the website must provide clear error messages with suitable recovery options
6. **FERPA Compliant** - the website must abide by standard FERPA regulations
7. **Reliable** - the system must have a high percentage uptime (99% or higher)
8. **Auditable** - the system should log all key actions (e.g., bookings, cancellations, admin over-rides) to an audit log for security and compliance purposes.

9. **Configurable** - the system should allow administrators to configure certain parameters (e.g., booking limits, dorm policies, blackout dates) without requiring code changes.
10. **Accessible** - the website (particularly the frontend) should be usable by as many people as possible, from “regular” users who have no hearing or sight impediments, to blind or deaf users who would have a harder time using the website otherwise

III. Database Design

III. 1. ER Diagram

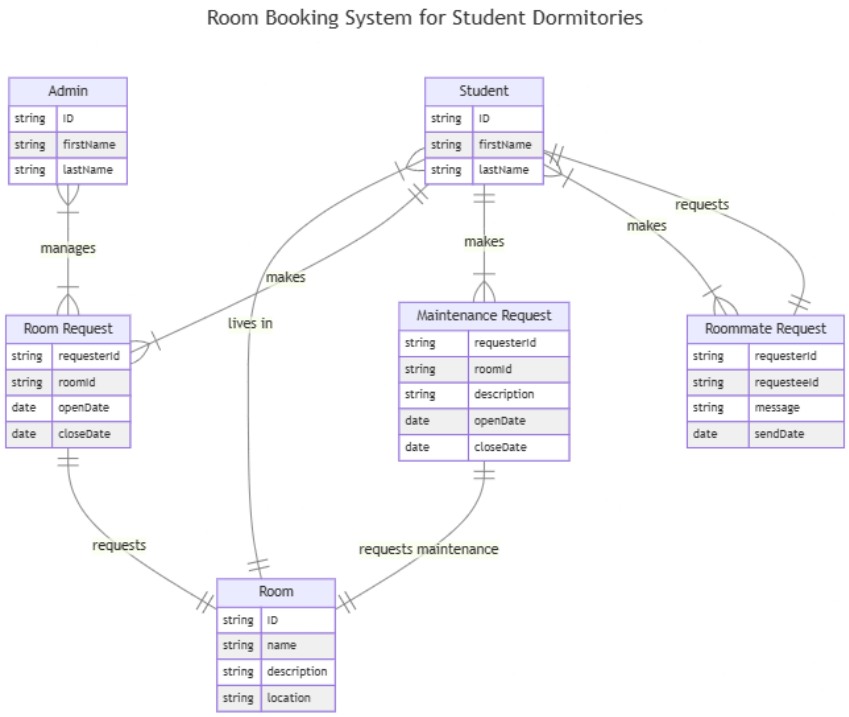

Figure 1: ER Diagram describing the main entities in our project and the relationship each entity has with the others it interacts with.

III. 2. Conversion Process

To convert our ER diagram into relational tables, we need to translate the entities and relationships: entities become tables, attributes become columns, primary keys and constraints are created to enforce the schema, and foreign keys are created to link relationships between entities.

The conversion process will vary based on different cardinalities for relationships:

- One-to-one: the foreign key can be placed in either of the tables.
- One-to-many / many-to-one: the foreign key should be placed in the table on the many side of the relationship, to avoid needing to create another table.
- Many-to-many: you need to create a new table to store the relationship, since both sides of the relationship could have many connections and thus need to reference an unbounded amount of foreign keys. Instead, the foreign key of the new table will point to the primary keys of the entities involved in the relationship.

III. 3. Tables (from ER Diagram)

III. 3. a. Admin

Attribute	Type	Constraints	Description
`ID`	string	Primary Key	The admin's unique ID
`firstName`	string	Not Null	the first name of the admin
`lastName`	string	Not Null	the last name of the admin

III. 3. b. Student

Attribute	Type	Constraints	Description
`ID`	string	Primary Key	The student's unique ID
`roomID`	string	Foreign Key	the room the student is staying in
`firstName`	string	Not Null	the first name of the student
`lastName`	string	Not Null	the last name of the student

III. 3. c. Room

Attribute	Type	Constraints	Description
`ID`	string	Primary Key	The room's unique ID
`name`	string	Not Null	The name of the room
`description`	string	Not Null	A description of the room
`location`	string	Not Null	The location of the room

III. 3. d. Room Request

Attribute	Type	Constraints	Description
`requestID`	int	Primary Key	A unique ID for the request
`studentID`	string	Foreign Key	The ID of the student making the request
`roomID`	string	Foreign Key	The ID of the room being requested
`openDate`	Date	Not Null	The date when the request was created
`closeDate`	Date		The date when the request was closed

III. 3. e. Maintenance Request

Attribute	Type	Constraints	Description
`requestID`	int	Primary Key	A unique ID for the request
`studentID`	string	Foreign Key	The ID of the student making the request

`roomID`	string	Foreign Key	The ID of the room being requested
`description`	string	Not Null	A description of the problem
`openDate`	Date	Not Null	The date when the request was created
`closeDate`	Date		The date when the request was closed

III. 3. f. Roommate Request

Attribute	**Type**	**Constraints**	**Description**
`requestID`	int	Primary Key	A unique ID for the request
`requesterID`	string	Foreign Key	The ID of the person making the request
`requesteeID`	string	Foreign Key	The ID of the person being requested as a roommate
`message`	string		An explanation of why they want to be roommates
`sendDate`	Date	Not Null	The date when the request was sent

III. 3. g. Admin Room Request Management

Attribute	**Type**	**Constraints**	**Description**
`managementID`	string	Primary Key	A unique ID for the management transaction
`adminID`	string	Foreign Key	The admin who's managing a room request
`requestID`	string	Foreign Key	The room request being managed

III. 4. Relationships

III. 4. a. Student lives in Room (M:1)

Many students can live in one room.

We model this by creating a foreign key `roomID` in the student table to store the room they are in.

III. 4. b. Student makes Room Requests (1:M)

One student can make many room requests, but a room request belongs to one student.

Put a foreign key `studentID` in the room request table to point toward the student who made the request.

III. 4. c. Room Requests request a room (M:1)

One room request points towards one room, but a room can have many room requests at any given time.

Best option is to put a `roomID` foreign key in the room requests table to point towards the room being requested.

III. 4. d. Student makes Maintenance Requests (M:M)

A student can make many maintenance requests, but each maintenance request belongs to one student.

To handle this, we'll add a `studentID` in the maintenance requests table for the student that made the request.

III. 4. e. Maintenance Requests requests maintenance for a Room (M:1)

A maintenance request points to one room, but a room can have many maintenance requests at a given time.

To model this, we put the foreign key `roomID` into maintenance request table, pointing at the room it is requesting.

III. 4. f. Admin manages Room Request (M:M)

One admin can manage many room requests, and one room request can be managed by many admin.

To handle this, we must create a separate table for the relationship (Admin Room Request Management). This table will store a management relationship instance, an `adminID` and a `requestID` as foreign keys to point to the admin doing the managing and the request being managed.

III. 4. g. Student makes/manages Roommate Requests (M:1)

A student can make many roommate requests, but a request is only made by one student.

We need to put a foreign key for the student requesting the roommate and the student getting requested as foreign keys in the roommate request table to reference the students table.

III. 5. Table Creation Commands

```
CREATE TABLE Admin (
    ID VARCHAR(255) PRIMARY KEY,
    firstName VARCHAR(255) NOT NULL,
    lastName VARCHAR(255) NOT NULL
);


CREATE TABLE Student (
    ID VARCHAR(255) PRIMARY KEY,
    roomID VARCHAR(255) REFERENCES Room(ID),
    firstName VARCHAR(255) NOT NULL,
    lastName VARCHAR(255) NOT NULL,
);


CREATE TABLE Room (
    ID VARCHAR(255) PRIMARY KEY,
    Name VARCHAR(255) NOT NULL,
    Description TEXT,
    Location VARCHAR(255) NOT NULL
);


CREATE TABLE RoomRequest (
    requestID INT PRIMARY KEY,
    studentID VARCHAR(255) REFERENCES Student(ID),
    roomID VARCHAR(255) REFERENCES Room(ID),
    openDate DATE NOT NULL,
    closeDate DATE NOT NULL
);


CREATE TABLE MaintenanceRequest (
    requestID INT PRIMARY KEY,
    studentID VARCHAR(255) REFERENCES Student(ID),
    roomId VARCHAR(255) REFERENCES Room(ID),
    description TEXT,
    openDate DATE NOT NULL,
    closeDate DATE NOT NULL
);


CREATE TABLE RoommateRequest (
    requestID INT PRIMARY KEY,
    requesterId VARCHAR(255) REFERENCES Student(ID),
    requesteeID VARCHAR(255) REFERENCES Student(ID),
```

```
    Message TEXT,
    sendDate DATE NOT NULL
);


CREATE TABLE AdminRoomRequestManagement (
    managementID VARCHAR(255) PRIMARY KEY,
    adminID VARCHAR(255) REFERENCES Admin(ID),
    requestID INT REFERENCES RoomRequest(requestID)
);
```

III. 6. Redundancy Elimination and Data Normalization

Each column contains only one value. We created separate tables to represent many to many relationships, so that we never have to store many values within one column. This is vital for accessing and modifying the data within the tables.

Each table only contains data relevant to the entity or relationship the table is based on (aka the primary key). We don't have information about other entities or relationships within the table, and instead we only reference that information through foreign keys. A byproduct of this is that no data field is repeated anywhere in the schema and thus we only have one single source of truth for all data, ensuring data consistency and helping with data integrity.