Room Booking System for Student Dormitories

Requirements and Technologies

Grace Anderson
Leo Curdi
Josh Evans
Calell Figuerres
Aaron Howe

February 09, 2025

Table of Contents

I. 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. 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).
   - **Justification:** these are basic requirements of all websites centered around user authentication
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.
   - **Justification:** snappier, more responsive websites provide better user experiences (and are, in general, easier to use compared to slower websites)
3. **Scalable** - the website should handle a reasonable number of current users, and be able to scale to handle high-traffic periods
   - **Justification:** supports high-traffic periods like term start
4. **Data Validation** - website must validate all input data before processing
   - **Justification:** maintains data integrity and prevents errors
5. **Error Resistant** - the website must provide clear error messages with suitable recovery options
   - **Justification:** improves user experience during problems
6. **FERPA Compliant** - the website must abide by standard FERPA regulations
   - **Justification:** required by all websites storing and handling data of students
7. **Reliable** - the system must have a high percentage uptime (99% or higher)
   - **Justification:** given the critical nature of the application, ensures that users can always access the platform
8. **Auditable** - the system should log all key actions (e.g., bookings, cancellations, admin overrides) to an audit log for security and compliance purposes.
   - **Justification:** helps in tracking issues, ensuring transparency, and maintaining accountability for changes made in the system.
9. **Configurable** - the system should allow administrators to configure certain parameters (e.g., booking limits, dorm policies, blackout dates) without requiring code changes.
   - **Justification:** helps in describing and tracking exact building and room conditions, both for administrators to keep track of and for students to be aware of when reserving
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
    - **Justification:** basic requirement for most online websites

III. Tools and Technologies

III. 1. IDE: Visual Studio Code

Visual Studio is a popular IDE that everyone in our team is familiar and comfortable with. It supports our functional and non-functional requirements by being the main method in which we write our code.

III. 2. Programming Language: TypeScript

TypeScript is a versatile language that is a superset of JavaScript. Though not everyone on our team has a great amount of experience in TypeScript, everyone does have experience with *JavaScript* and another statically-typed language (such as C# or Java), making learning curve for TypeScript fairly shallow.

TypeScript supports our functional and non-functional requirements by enforcing type safety throughout the backend, frontend, and with the addition of tRPC, the communication between them. This will significantly reduce the number type-based bugs we create, as well as ensuring the entire team is on the same page for what function inputs, outputs, and usage.

III. 3. Client Framework: React + React Router + Mantine

React is the most popular web framework for adding reactivity to a website. Combined with React Router, this will allow the team to create a full-fledged web application. Everyone on the team has some experience with React, and React Router is fairly easy to learn (additionally, our team lead, Calell, has extensive experience with using React Router.)

Mantine is then brought in to serve as our main component library; Mantine provides an extensive number of components and framework for building nice-looking web applications without needing to deal too much with CSS.

These libraries will support our functional and non-functional requirements by being the main way in which we build and style the application.

III. 4. Server Framework: Express.js + tRPC

Express.js is a well-known (if somewhat old but still updated) framework for creating APIs and server-side-rendered websites (akin to PHP but in JavaScript). For our purposes, we will only be using the API portion of Express.js, wherein tRPC runs “on top of” the Express.js server.

tRPC is a much newer framework for creating “procedures” (following the idea of remote procedure calls (RPCs) that clients can make; RPC itself predates the REST standard, but is still in use today). tRPC will effectively allow us to create “callable server functions” (with type safety) that our client can use whilst still allowing for separation of client and server code.

These support our functional and non-functional requirements by allowing us to write type-safe code, from client to server and back, which will hopefully reduce the bugs we introduce by ensuring client and server communication is consistent, especially when being worked on by different people.

III. 5. Version Control: <u>Git and GitHub</u>

Git is the most popular version control system in use today. Alongside it, GitHub is the gold standard for hosted remote Git repositories. The combination of these tools will make for easier and more "stable" collaboration (i.e., fewer people accidentally stepping over each other's code).

These support our functional and non-functional requirements by being our primary means of managing code between each team member.

III. 6. Deployment: <u>GitHub Actions</u>

GitHub Actions is a Continuous Integration and Continuous Delivery (CI/CD) pipeline offered by GitHub. The primary way this fits into our project is being a free (for small projects) and easy-to-use method of automatic deployment; alongside that, it remains a standard way of implementing CI/CD, and is fairly simple to learn.

GitHub Actions supports our requirements by being an easy "set and forget" method of automatic deployment, making for fewer mistakes during deployment and eliminating the need for any individual person to be "in charge" of deploying manually.

III. 7. Database: <u>PostgreSQL</u>

PostgreSQL (or simply Postgres) is a popular, open source, and industry-standard relational DBMS. Postgres will be the primary database management system we use for our project; we are using it instead of Oracle due mostly to its ubiquity amongst DBMSes, as well as its less-proprietary nature.

Postgres supports our requirements by being our primary database; it will be the primary space we store user data, rooms, etc.

III. 8. Hosting: <u>Amazon Web Services</u>

Amazon Web Services is the gold standard and leader in cloud computing products. AWS will fit into our project by being the primary medium of hosting our project (should hosting be desired and/or required); chiefly, Amazon EC2 (compute), Amazon RDS (database), and Amazon Cloud-Front (Content Delivery Network).

AWS supports our functional and non-functional requirements by eliminating the need for us to maintain physical servers to handle compute, database, and content delivery; additionally, the resources we use can quickly be spun up, spun down, scaled up, and scaled down, reducing the costs for hosting during times of little traffic or development.