Table of Contents

[**1.** **Requirements** 2](#_Toc155527408)

[1.1. Target Audience 2](#_Toc155527409)

[1.2. Included Features and Justifications 2](#_Toc155527410)

[**2.** **Design** 2](#_Toc155527411)

[2.1. System Architecture 2](#_Toc155527412)

[2.2. UML diagrams 4](#_Toc155527413)

[**3.** **Testing** 4](#_Toc155527414)

[3.1. Automated Testing 4](#_Toc155527415)

[3.2. Usability Testing 5](#_Toc155527416)

[**4.** **DevOps Pipeline: Navigating the Development environment** 5](#_Toc155527417)

[4.1. Development Environment 5](#_Toc155527418)

[4.2. Continues Integration Pipeline 6](#_Toc155527419)

[CI/CD Tools: GitHub Actions 6](#_Toc155527420)

[Integration Process 6](#_Toc155527421)

[**5.** **Personal Reflection** 6](#_Toc155527422)

[5.1. What Worked Well 6](#_Toc155527423)

[5.2. What did not Work Well 6](#_Toc155527424)

[5.3. Lessons for Future Projects 6](#_Toc155527425)

# **Requirements**

## Target Audience

This application is carefully crafted to meet the evolving needs of a specific target audience: hotel customers who seek convenience of accessing and booking hotel rooms seamlessly, particularly while on the go, the primary focus is on a demographic that values the flexibility and the immediacy of a digital booking experiences.

The chosen target audience is distinctive in its preference for smaller boutique hotels recognizing a shift in consumer behavior, the application is attuned to the growing demand for digitized booking solutions in a sector that predominantly has been associated with high-end establishments. This demographic often finds existing technologies centered around larger hotels, leaving a gap in the market for a platform tailored to the unique characteristics of smaller accommodations

\Users in this category are inclined towards a more personalized hospitality experience, seeking the charm and individuality that small hotels offer. Their preference leans towards establishments that may not have expansive resources of large hotel chains but boast character, and authentic commitement to personalized service.

The decision to target users interested in digitizing bookings for smaller hotels aligns with the broader trend of technology democratization, making digital conveniences accessible to a wider range of business. By focusing on this niche, the application aims to bridge the technological gap and empower smaller hotel owners to streamline their booking processes and enhance customer satisfaction.

In conclusion, the target audience for this application comprises discerning hotel customers who appreciate the on-the-go convenience of digital booking, with a specific emphasis on those seeking unique experiences in smaller, boutique establishments.

## Included Features and Justifications

* + 1. **Easy Login & Signup**

**Description:**  The Ease of Login feature streamlines onboarding process, allowing the user to effortlessly create accounts or login.

**Justification:** Making the Login and signup process easy and simple enhances user experiences especially those on the go, who use the system once thereby attracting and retaining users. This creates a hassle-free start, reducing friction for the first-time users and encouraging repeated interactions.

* + 1. **Simple Dashboard with Room View**

**Description:** The simple dashboard offers an intuitive interface, displaying key information at glance, Users can easily navigate through the rooms available, view and details and access booking options

**Justification:** A clear and straight forward dashboard simplifies user interactions, making room selection and booking straightforward. Users can find quickly relevant information, promoting engagement and improving overall satisfaction

* + 1. **Simple Booking & Payment System**

**Description:** The Simple Booking System facilitates a smooth reservation process. Users can select rooms enter details and complete payments seamlessly without any hustle

**Justification:** Streamlining the booking and payment process ensures a user-friendly experience reducing transactional complexities, encouraging users to finalize booking with confidence, resulting to increased user satisfaction and higher conversion rates

# **Design**

## System Architecture

The Hotel room booking system is designed with a client server architecture, a widely adopted model that efficiently handles communication and data flow between the user interface(clients) and the back-end server.

In this architecture, clients representing users interacting with the application, sends requests to the central server. The server, responsible for processing these requests, executes the necessary actions, retrieves data from the database, and sends back appropriate responses. This separation of concerns optimizes performance, scalability, and maintenance.

* + 1. **Components Interaction**

The Interaction between components in this system is achieved through a series of well-defined processes. Clients, including web browsers or mobile devices, communicate with the server using HTTP requests. These requests are processed by the server, which in turn interacts with the database to retrieve the data and perform CRUD operations

The server acting as an intermediary, manages user authentication, authorization and other business logic. Upon receiving a request for room information, for instance, the server retrieves the relevant data from the database and sends it back to the client, where it is presented in a simple user-friendly format.

* + 1. **Data and Code Structure**
       - 1. **Data Structure**

Data in a system is organized in a relational database, chosen for its ability to efficiently manage structured information. The database stores user details and information, room information, booking and payment transactions. This uses a no-SQL database(MongoDB) for easy scalability.

* + - * 1. **Code Structure**

The code base follows the Model-View-controller pattern, a design principle that separate the Application into three interconnected components:

**1. Model:** Handles data management and storage interacting with the database to retrieve or update information. This system the model includes functions for user authentication, room management, booking transactions and payment processing.

**2.** **View:** Represents the user interface, ensuring a clear and intuitive presentation of information. The view component encompasses the dashboard, Room View and booking and Payment interface providing a seamless end user experience.

**3.** **Controller:** Manages the flow of data between the model and the view handling user input and triggering appropriate responses. The controller orchestrates the interaction between the client and the server, making sure that requests are processed and responses are delivered effectively

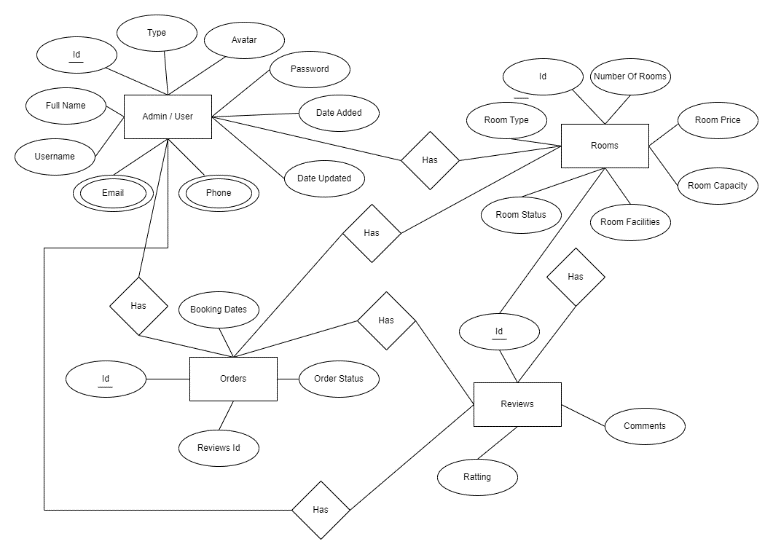
* + 1. **Appropriateness of Structure**

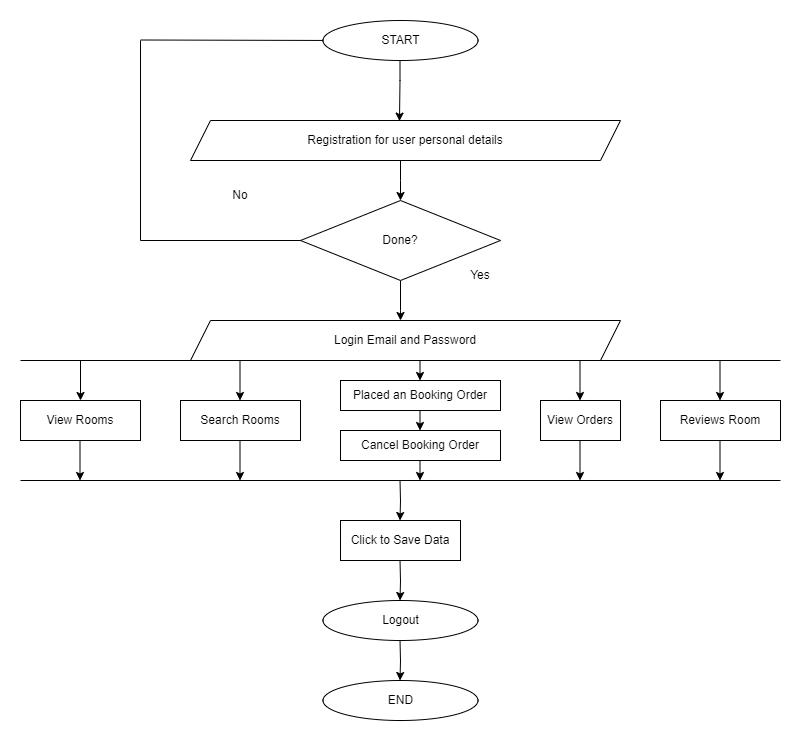
The chosen client-server architecture and the utilization of the MVC for code organization are highly appropriate for the hotel room booking system. The client-server model facilitates efficient communication and scalability, ensuring responsive and reliable application.

The use of non-relational database with the MVC enhances the scalability and maintainability as change to one component does not necessitate modification across the entire system.

In summary, the client-server architecture, coupled with MVC and a database creates a well organized and scalable foundation for the hotel room booking system. The architecture ensures optimal performance, streamlined data management and a user-friendly experience for all stakeholders.

## UML diagrams

****

****

# **Testing**

## Automated Testing

* + 1. **Test Coverage**

The automated testing strategy for the hotel room booking system focused on achieving comprehensive coverage across critical functionalities. This includes testing user authentication, room management, booking transactions and payment processing. Unit Tests were developed to ensure individual components such as controllers and models function correctly. Integration tests were employed to verify the seamless interaction between various system components

* + 1. **Tools Used**

The testing framework selected for automated testing was Jest, a widely used JavaScript testing library. Jest provided a robust set of tools for writing unit and integration testing, enabling effective testing of both front to back end components.

* + 1. **Results**

The automated testing phase yielded positive results, with the majority of test cases passing successfully. Unit tests verified correctness of individual functions, while integrating tests confirmed the seamless collaboration between the different components automated testing proved invaluable in detecting and rectifying potential issues early in the development process, ensuring the stability and reliability of the application

## Usability Testing

* + 1. **Test Scenarios**

Usability testing was conducted to evaluate the applications user interface and overall user experience. Key scenarios included user registration, room selection, booking a room, and completing a payment transaction.

* + 1. **Testing Approach**

Usability testing adopted a user-centric approach, involving real users through A/B testing. A/B testing was employed to compare variations of specific feature and determine the more user friendly option

* + 1. **Results and Improvements**

Usability testing provided valuable insights into the user interactions and preferences. Results showed high level of satisfaction on the straightforward registration process and the simplicity of the booking system. However, some of the users suggested visual enhancement of the room for better room information presentation. Based on these improvements were made

# **DevOps Pipeline: Navigating the Development environment**

## Development Environment

In crafting the project, a robust technology stack was chosen to ensure efficiency, scalability and seamless integration. The key technologies utilized encompass React for the front end, Node.js to power the server and MongoDB as the database. Reacts component based architecture provided a modular and maintainable front-end structure, while Node.js, known for its asynchronous capabilities, supported the backend, facilitating rapid development and handling concurrent user requests effectively. MongoDB, a NoSQL database, offered a flexibility in storing and retrieving the data, aligning with the projects dynamic requirements.

In the development toolkit, Visual Studio Code emerged as the IDE of choice. Its lightweight yet powerful features including built-in Git support and rich ecosystem of extensions, enhanced the coding experience. Git, for version control played a pivotal role in collaborative development, allowing seamless code collaboration and tracking changes across the team

## Continues Integration Pipeline

For the continuous Integration (CI) and Continuous Deployment (CD) processes, a tailored approach was adopted to ensure smooth workflow from development to production. GitHub Actions, a CI/CD tool seamlessly intergraded into the GitHub repository, became the backbone of the project’s automation strategy

## CI/CD Tools: GitHub Actions

GitHub Actions handled the CI/CD pipeline automating various stages of development. This included triggering builds upon code pushes, running automated tests and deploying successfully tested code to the production

## Integration Process

The CI/CD workflow was intricately woven into the development process, aligning with the principles of continuous integration delivery. Upon each code push to the GitHub repository, GitHub actions initiated the CI process. This Involved pulling the latest code, installing dependencies and running a suite of automated tests, including unit tests and integration tests any failures in the CI process triggered notifications, ensuring prompt attention to the potential issues

Following a successful CI, the CD process swiftly took over. GitHub Actions Orchestrated the deployment of the application to a staging environment. For the final validation. Once the staging was verified, the CI/CD pipeline automatically deployed the code for production

The streamlined CI/CD workflow not only accelerated development cycles but also minimized the risk of deploying faulty code to the production environment. By automating the key development processes, GitHub Actions contributed to a reliable consistent and agile development environment.

# **Personal Reflection**

## What Worked Well

Throughout the project development of a hotel room booking system, several elements contributed to its success. The adoption of a client-server architecture, powered by React, Node.js and MongoDB provided a scalable and efficient foundation. Leveraging GitHub Actions for continuous Integration and Continuous Deployment streamlined the development process, automating testing and deployment seamlessly. Visual studio Code emerged as a versatile IDE enhancing the coding experience with its light weight yet powerful feature.

Easy use of the selected programming tools gave a psyche in developing the project more while still keeping it simple and easy for users use, creating a curtain to blind the user from the complexities of the processes behind the simple outlook of the system.

## What did not Work Well

Challenges during the project included occasional issues with third-party dependencies and library compatibility especially the frequent updates in node. Integrating libraries sometimes led to unexpected conflicts, requiring additional troubleshooting and resolution time.

## Lessons for Future Projects

The project emphasized the importance of comprehensive automated testing, leading to a more robust code and increased in confidence in deployments. The adoption of a client server architecture showcased its scalability and also its efficiency, influencing future architectural decisions. Lessons learned from overcoming the challenges underscored the significance of effective decisions and proactive issue resolution within development

In future projects, a refined onboarding process will be implemented, including a clearer documentation.

# **Appendix**

