

***UE21CS352B - Object Oriented Analysis & Design using Java***

**Mini Project Report**

**“Hotel Room Booking System”**

***Submitted by:***

|  |  |
| --- | --- |
| **K Virupakshi**  **LSS Praneeth Kumar**  **Mani Shankar M**  **Jyothiraditya D** | **PES1UG21CS264**  **PES1UG21CS305**  **PES1UG21CS307**  **PES1UG21CS921** |

*6th Semester E Section*

# Prof. Bhargavi Mokashi

Assistant Professor

**January - May 2024**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

FACULTY OF ENGINEERING

**PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

# Table of Contents

1. **Introduction**
2. **UML Diagrams**
3. **Design Principles**
4. **Design Patterns**
5. **Output Screenshots**
6. **Work Responsibility**
7. **Link to Github Repository**

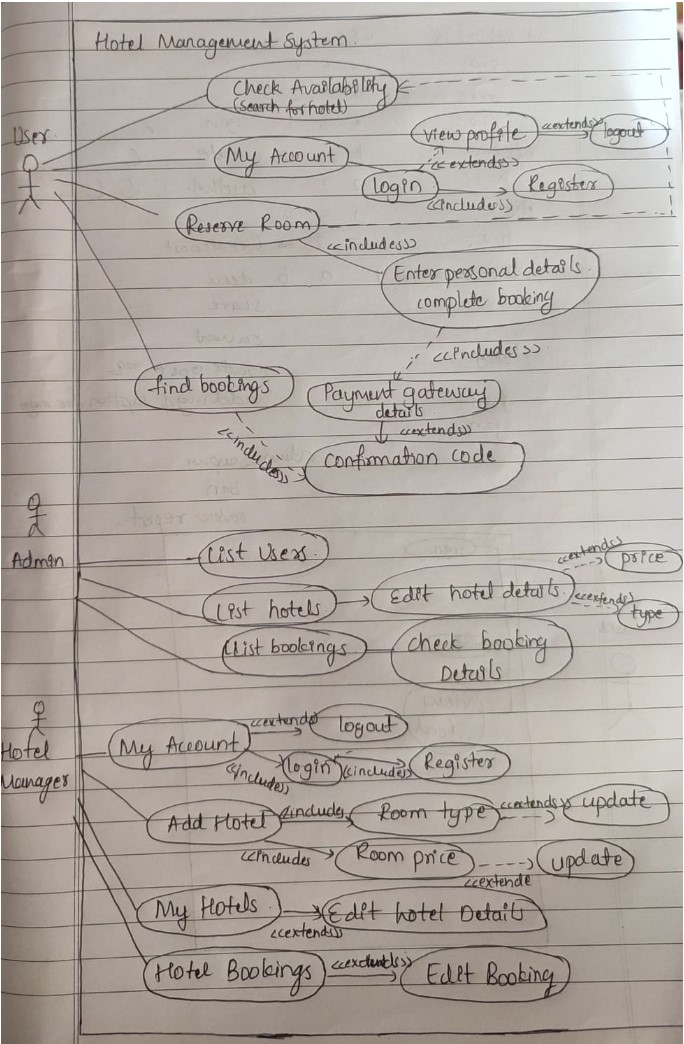
## INTRODUCTION

HotelBookingApp is a straightforward web application aimed at streamlining the hotel reservation process for customers and hotel managers alike. Developed using Java and Spring Boot in the backend, and Thymeleaf in the frontend, the system follows the Model-View-Controller (MVC) architectural pattern.

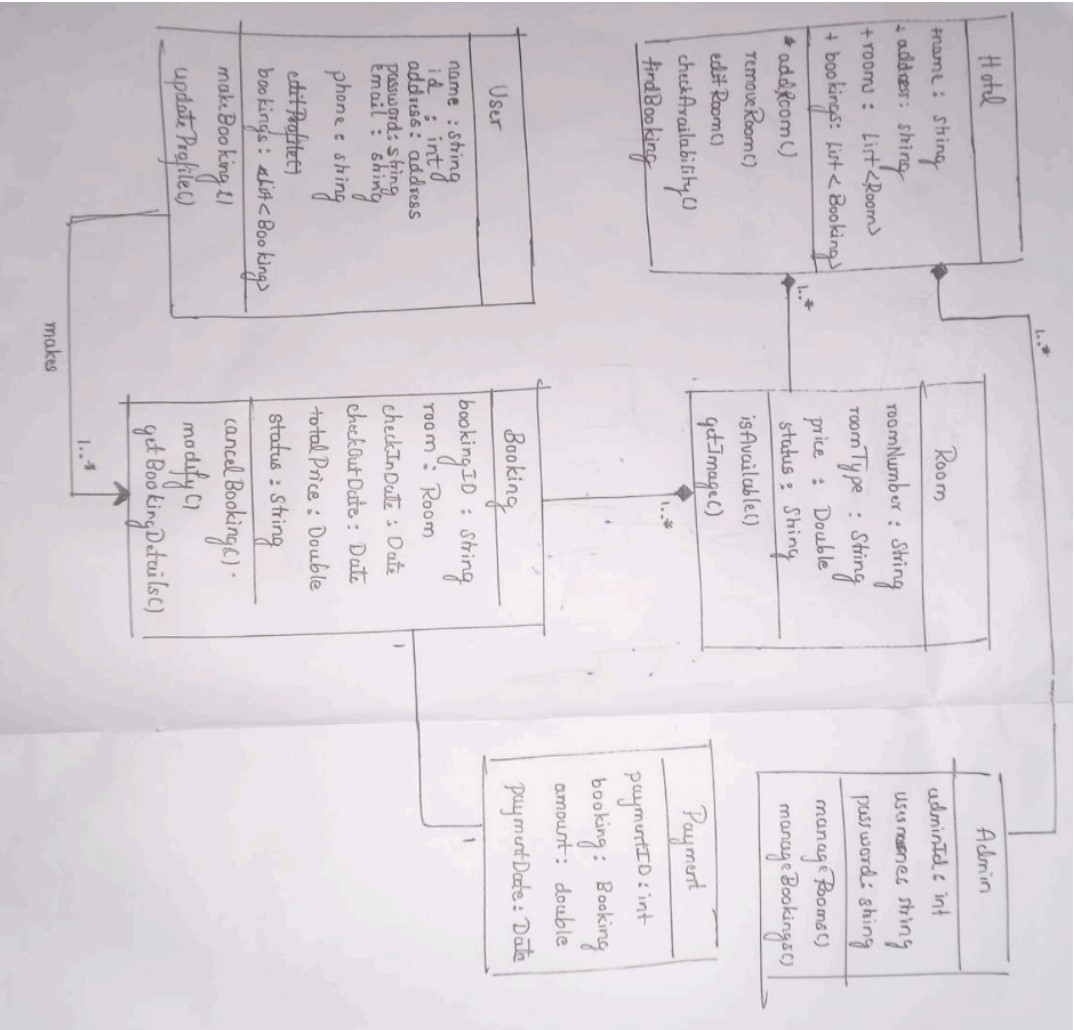
* User Registration & Management: Users can register, login, and manage their profiles. Various data validations (e.g. strong password requirement) have been implemented.
* Hotel Management: Hotel managers can add/edit hotels, specifying details (e.g. name, address, room counts, prices) in a single interface.
* Hotel Search: Enables customers to search for available hotels based on location and check-in/check-out dates.
* Hotel Listing: Displays a list of available hotels with relevant details such as name, available room counts, and prices.
* Hotel Details: Provides in-depth information on hotels, including name, address, room availability, pricing, and an interactive map leveraging the Nominatim geocoding API and Lea et library.
* Room Booking: Customers can select the desired number of rooms and get redirected to payment for nalizing the reservation.
* Payment Processing: Secure credit card payment with validations like Luhn checks and custom validators for expiry dates and CVV. (No third-party payment gateways are implemented.)
* Booking Management: Customers and hotel managers can view their bookings through the dashboard.
* Admin Panel: Allows administrators to manage users, hotels, rooms, and bookings.
* Responsive Design: The app is optimized for various devices including desktops, tablets, and smartphones.

## UML DIAGRAMS

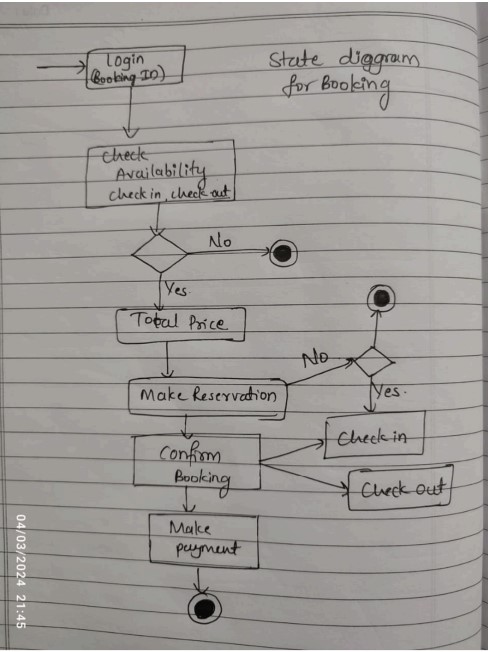
**Use case Diagram**

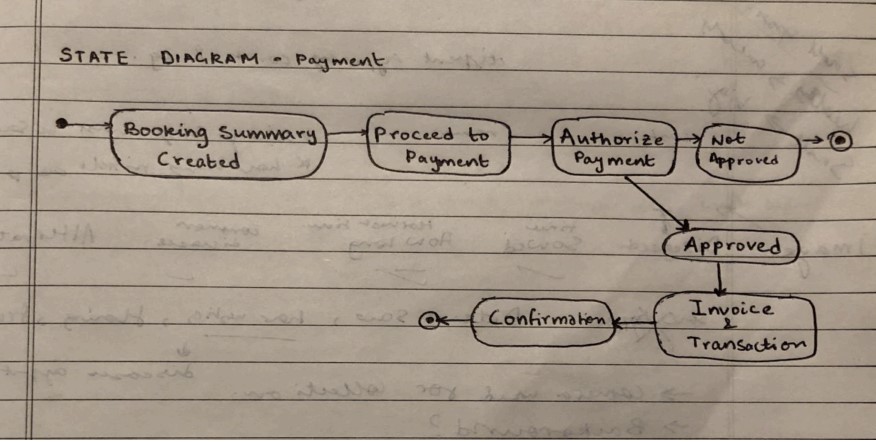


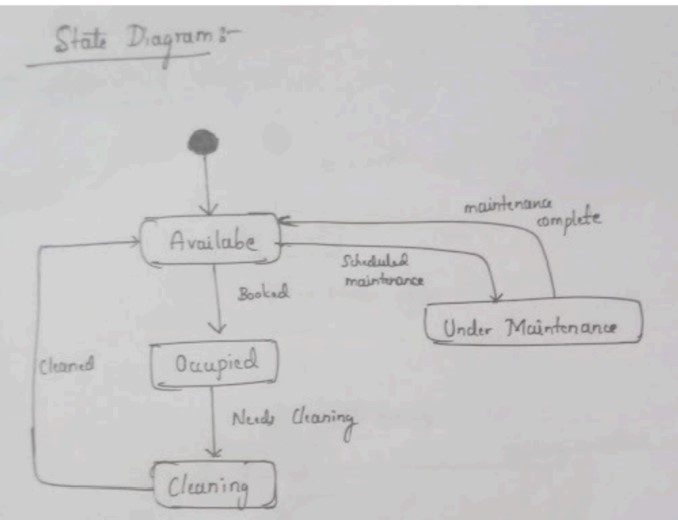
## Class Diagram

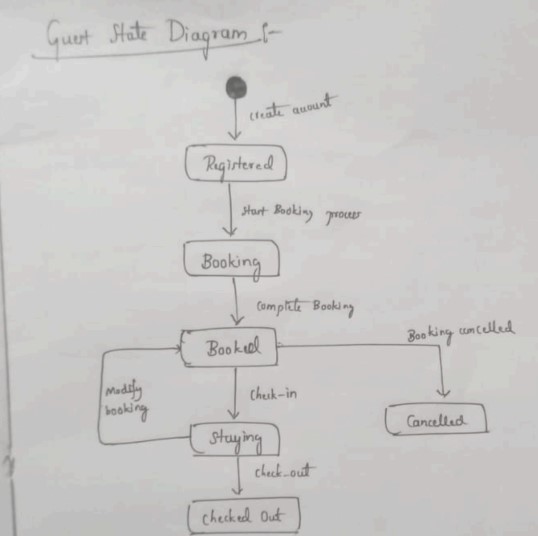


## State Diagram

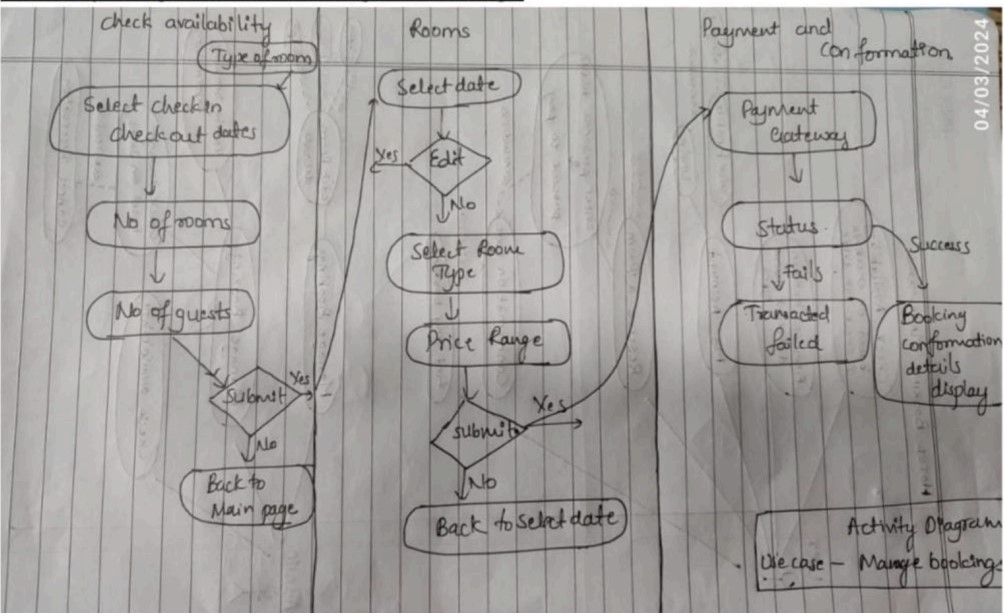


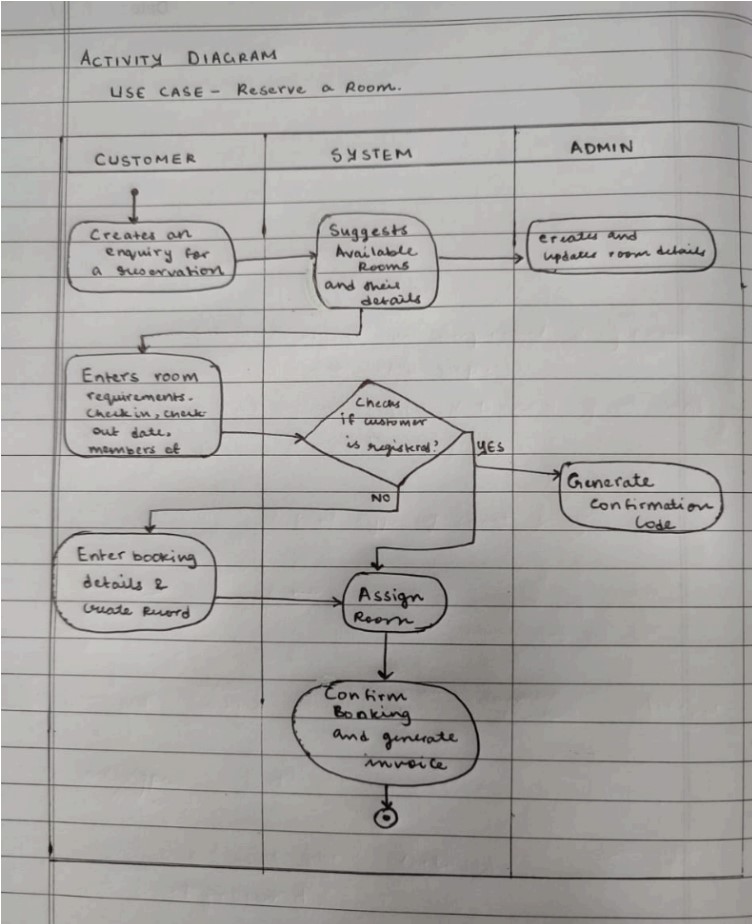


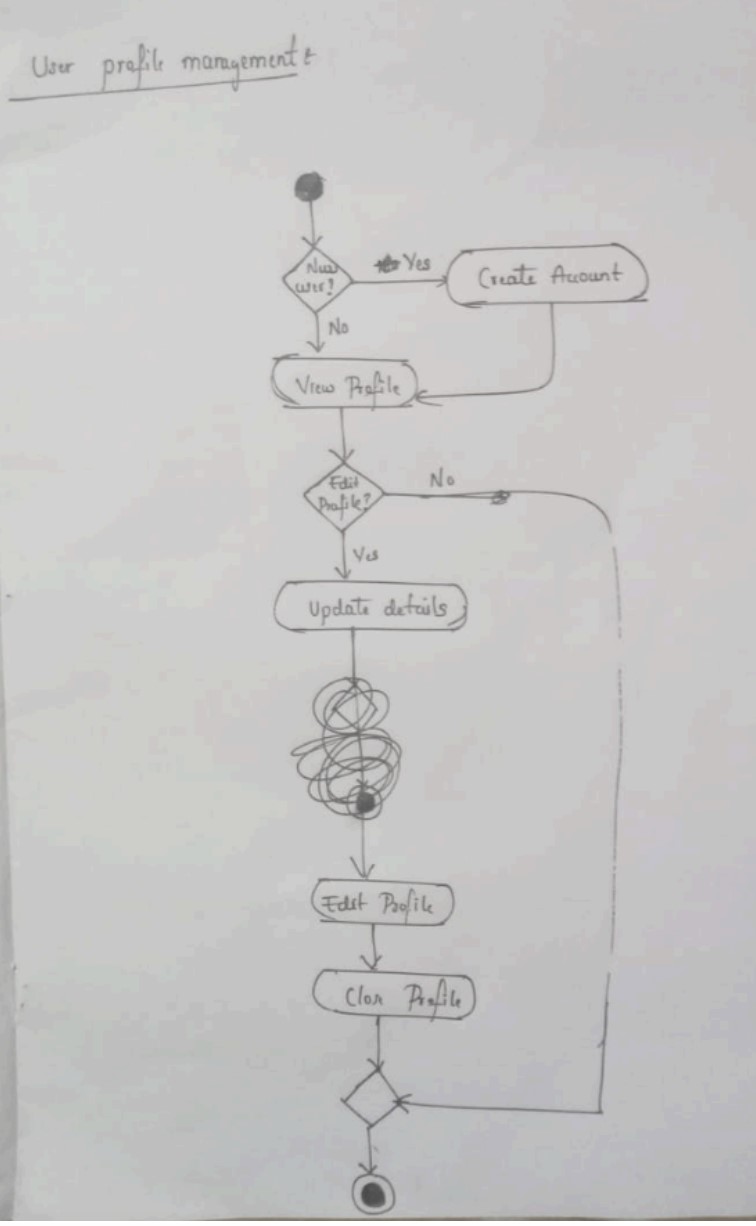


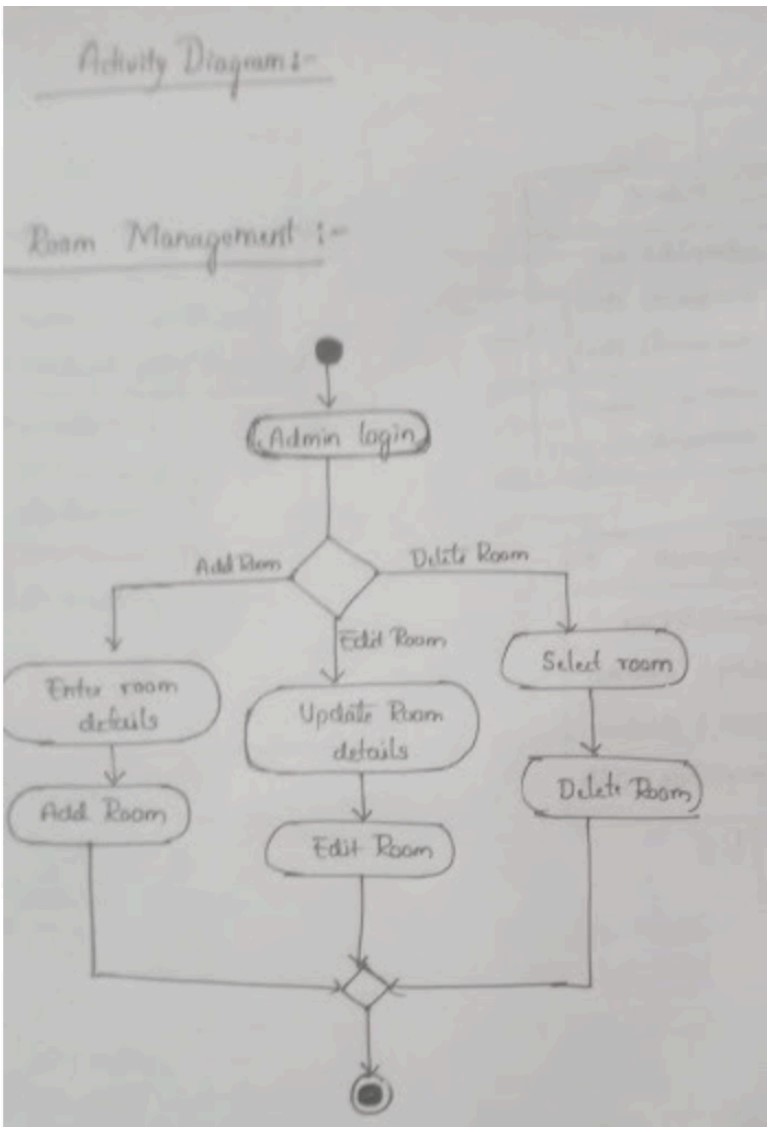


## Activity Diagram









**DESIGN PRINCIPLES**

## Single Responsibility Principle (SRP)

The Single Responsibility Principle (SRP) is applied to ensure that each class within the system has only one responsibility or reason to change. This principle is crucial for maintaining modularity, scalability, and testability in the codebase. By adhering to SRP, each class in the system is focused on a speci c aspect of user registration or management, making the codebase easier to understand, maintain, and extend.

SRP ensures that each class is focused on a single aspect of user registration or management, leading to a more modular, maintainable, and scalable codebase.

CardExpiryValidator class: This class is responsible for validating the format and validity of a credit card expiry date. It does this by implementing the ConstraintValidator interface and providing the logic for checking if the provided expiry date is in the correct format and is not in the past.

CardExpiry annotation: This annotation marks a eld as requiring validation for a credit card expiry date. It is responsible for associating the validation logic (de ned in CardExpiryValidator) with the annotated eld.

## Interface Segregation Principle

The decision to apply the Interface Segregation Principle to the

BookingRepository and CustomerRepository was made to ensure that these interfaces are tailored precisely to the speci c needs of their clients. By breaking down these interfaces into smaller, focused ones, unnecessary dependencies are avoided, and clients are only required to implement the methods relevant to their use cases. This promotes better code organization and reduces the risk of unintended side e ects when changes are made.

**DESIGN PATTERNS**

## Factory Method Pattern

The Factory Method Pattern is utilized to abstract the creation of di erent types of users (e.g., hotel managers, customers, administrators) based on prede ned roles or criteria. This pattern provides a exible and scalable approach to user creation by encapsulating the creation logic within factory classes. Each subclass of the factory is responsible for creating a speci c type of user, promoting code reuse and maintainability. By employing the Factory Method Pattern, the system can accommodate future changes or additions to user roles without requiring signi cant modi cations to the existing codebase.

The Factory Method Pattern facilitates exible and scalable user creation by encapsulating the creation logic within factory classes. This promotes code reuse and maintainability, allowing the system to accommodate changes or additions to user roles with minimal impact on existing code.

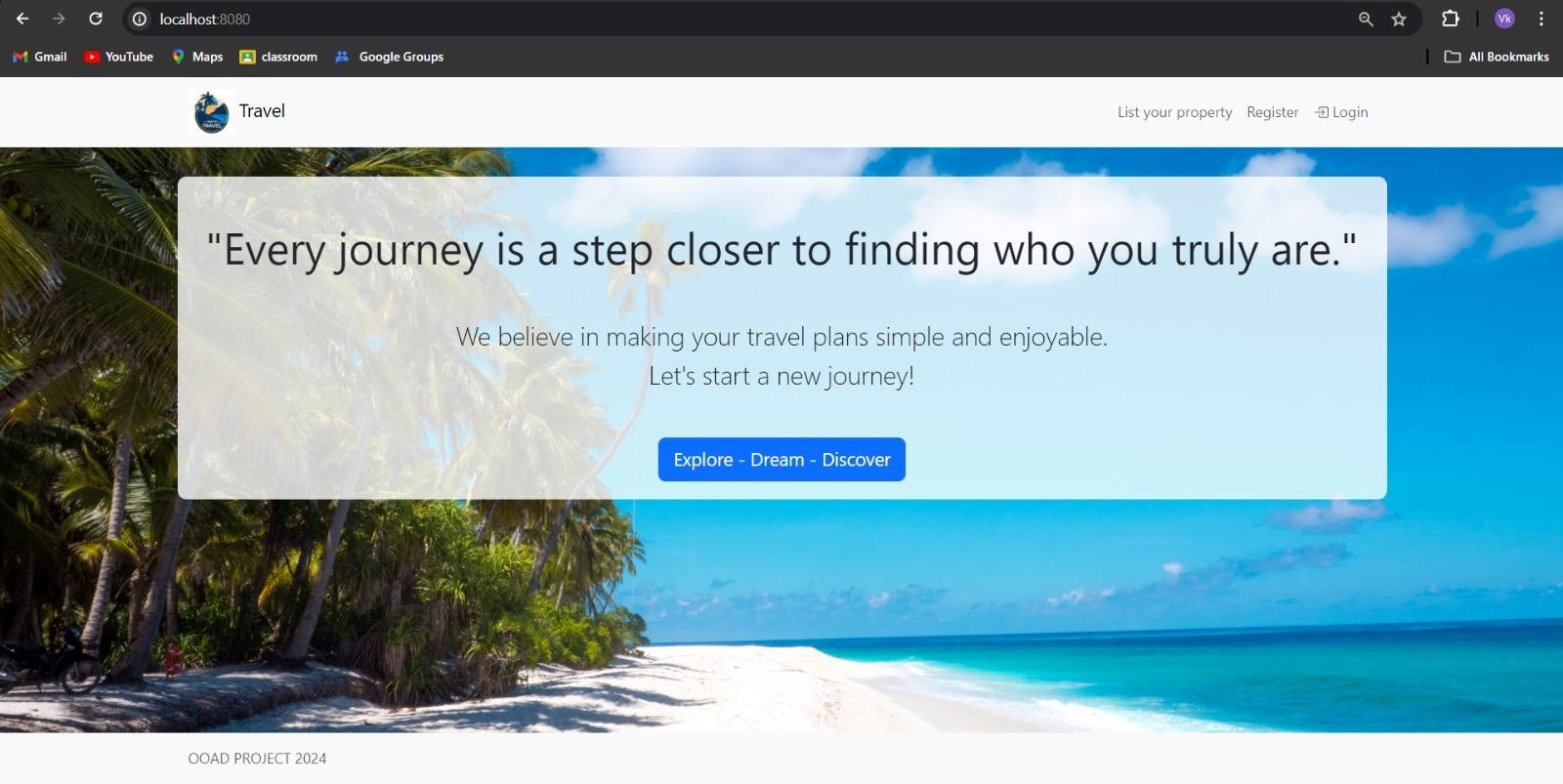
**Decorator Design Patterns:**

Validity: The isValid method within CardExpiryValidator is responsible for determining the validity of the provided credit card expiry date. It performs validation by checking if the date is in the correct format and if it is not in the past. This method encapsulates the validation logic.

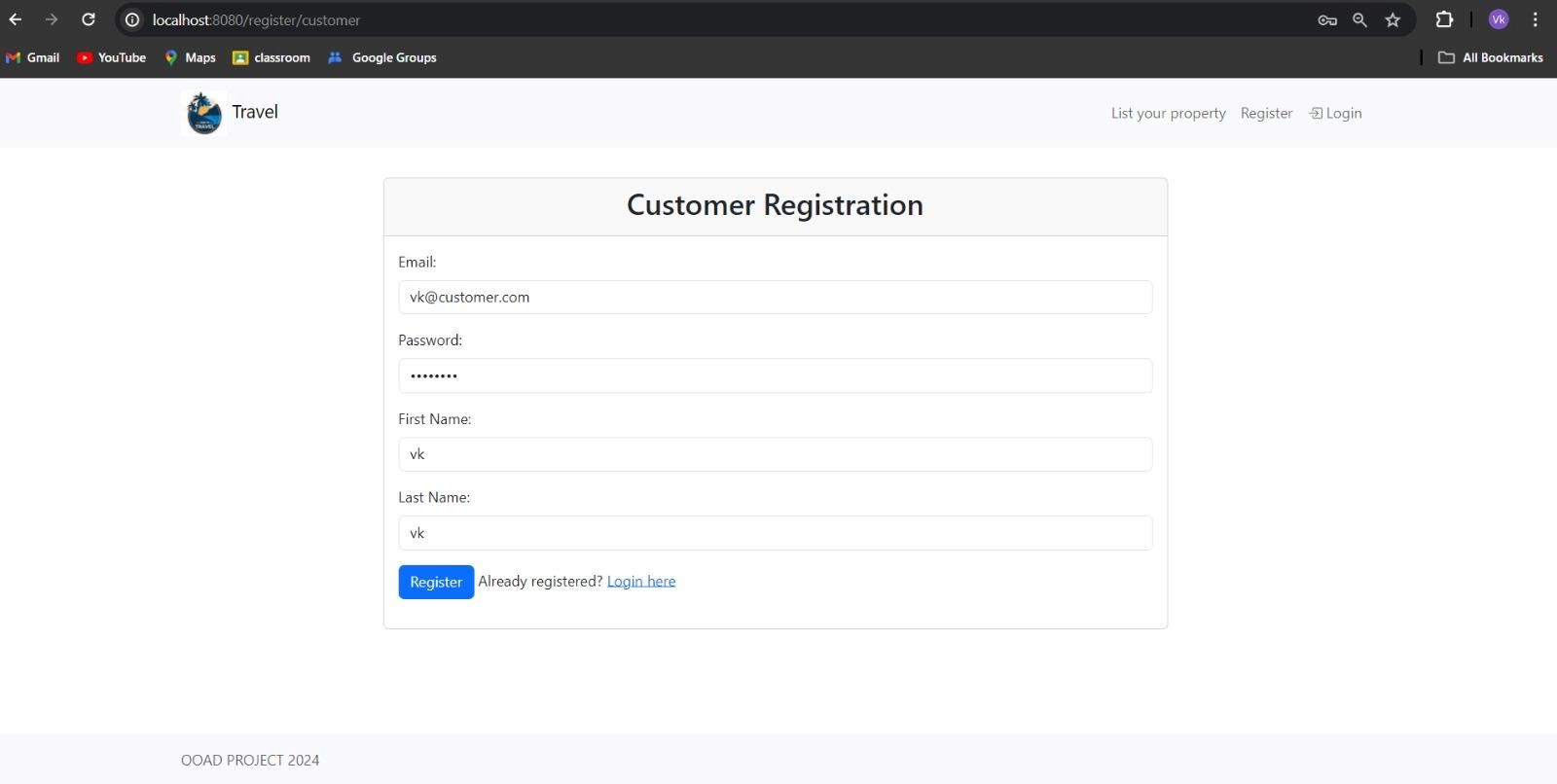
Decorator Design Pattern: The CardExpiryValidator class implements the

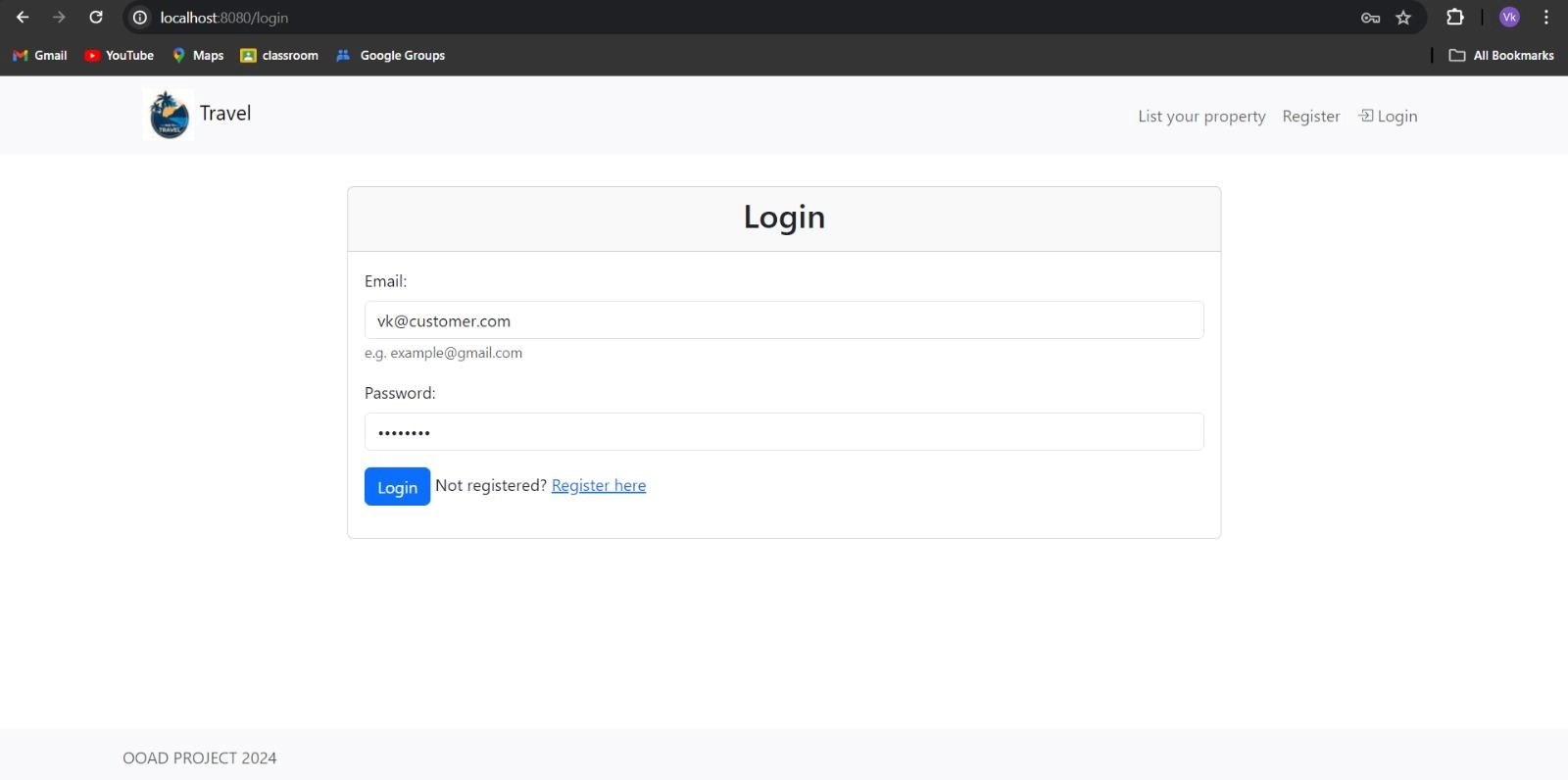
ConstraintValidator interface, which is a part of the decorator design pattern. This interface allows CardExpiryValidator to wrap the validation logic and apply it to elds annotated with CardExpiry. By implementing this interface, CardExpiryValidator can be seamlessly integrated into the validation framework of the project without directly modifying the core validation logic.

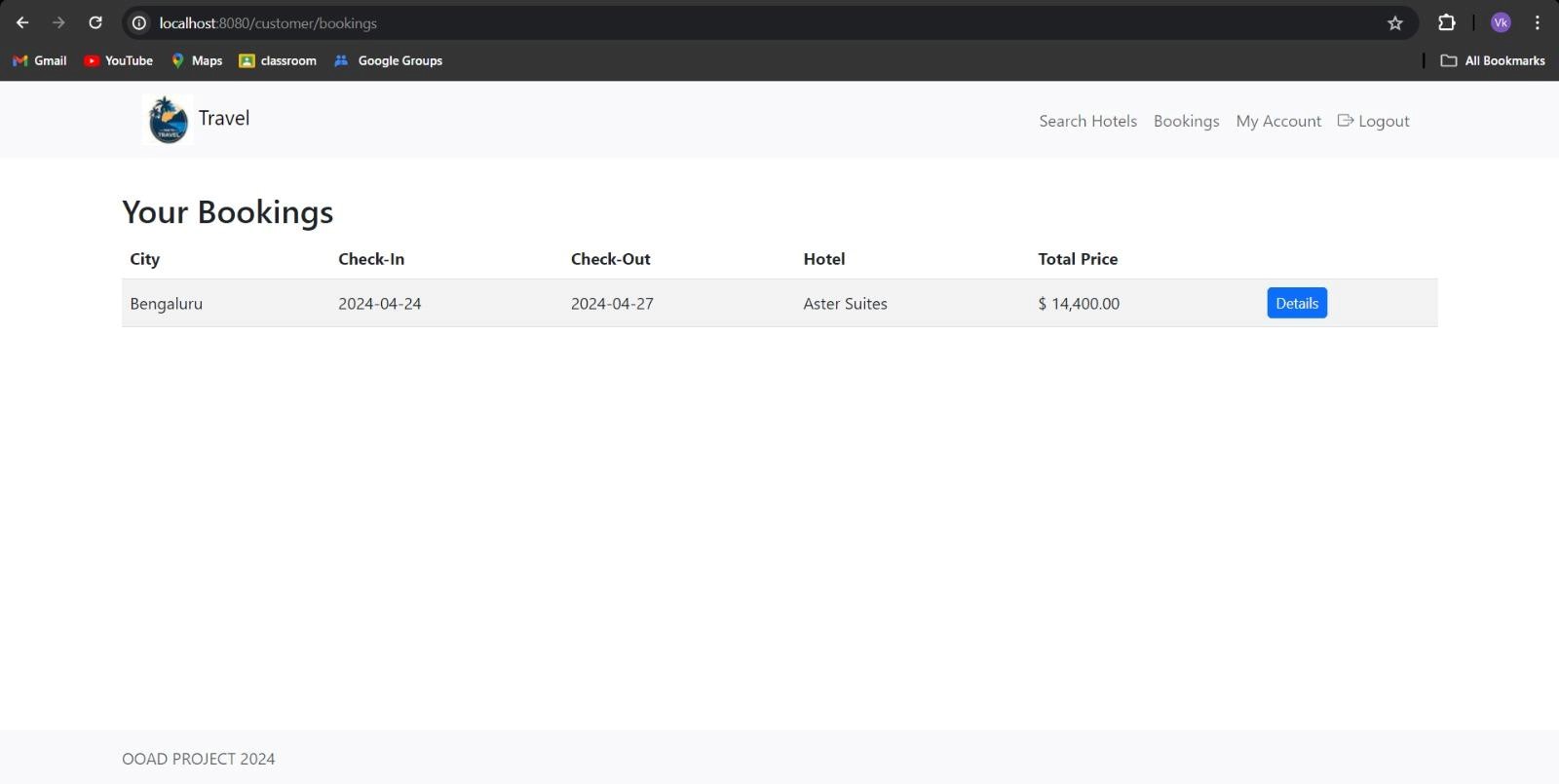
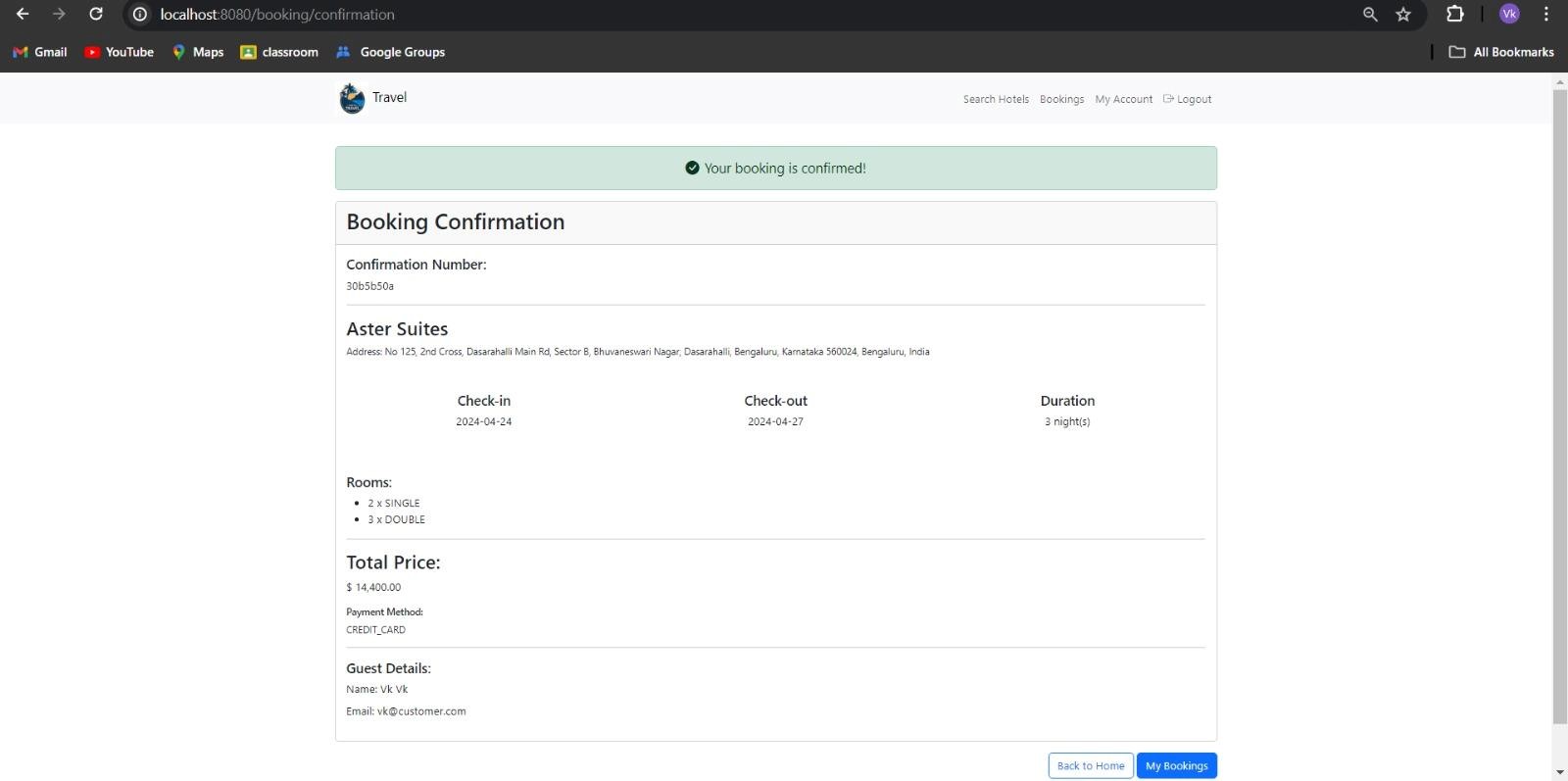
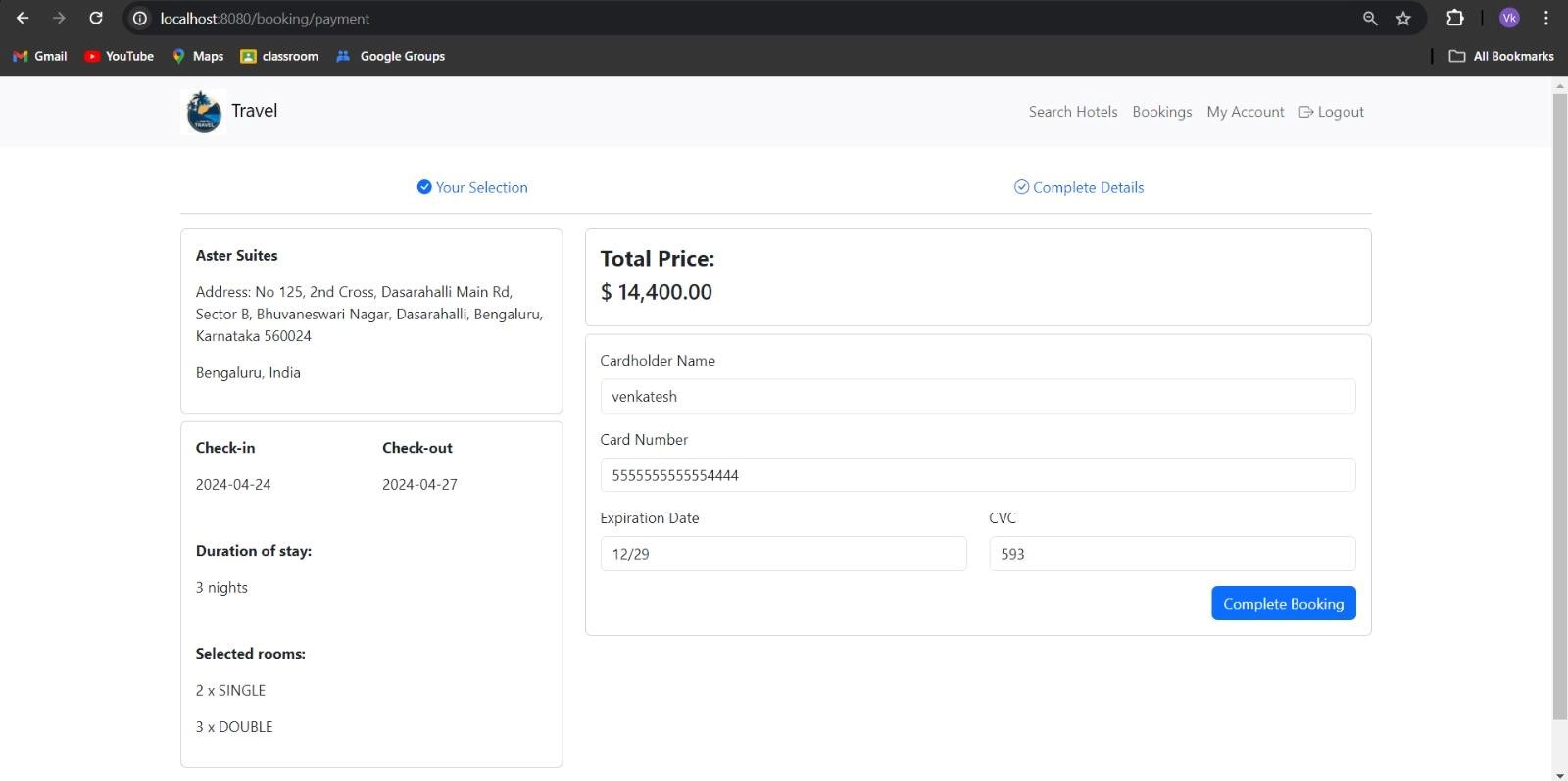
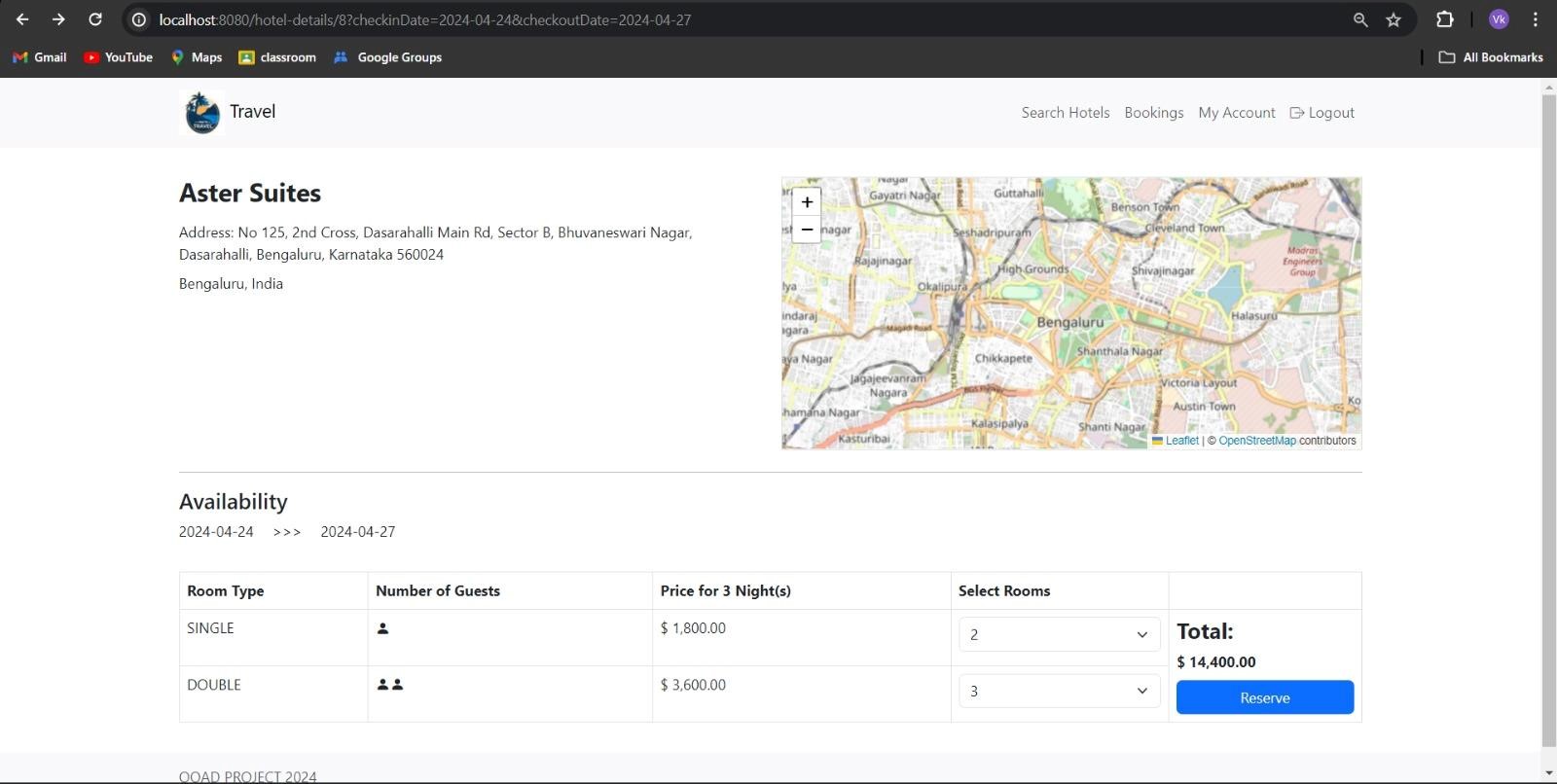
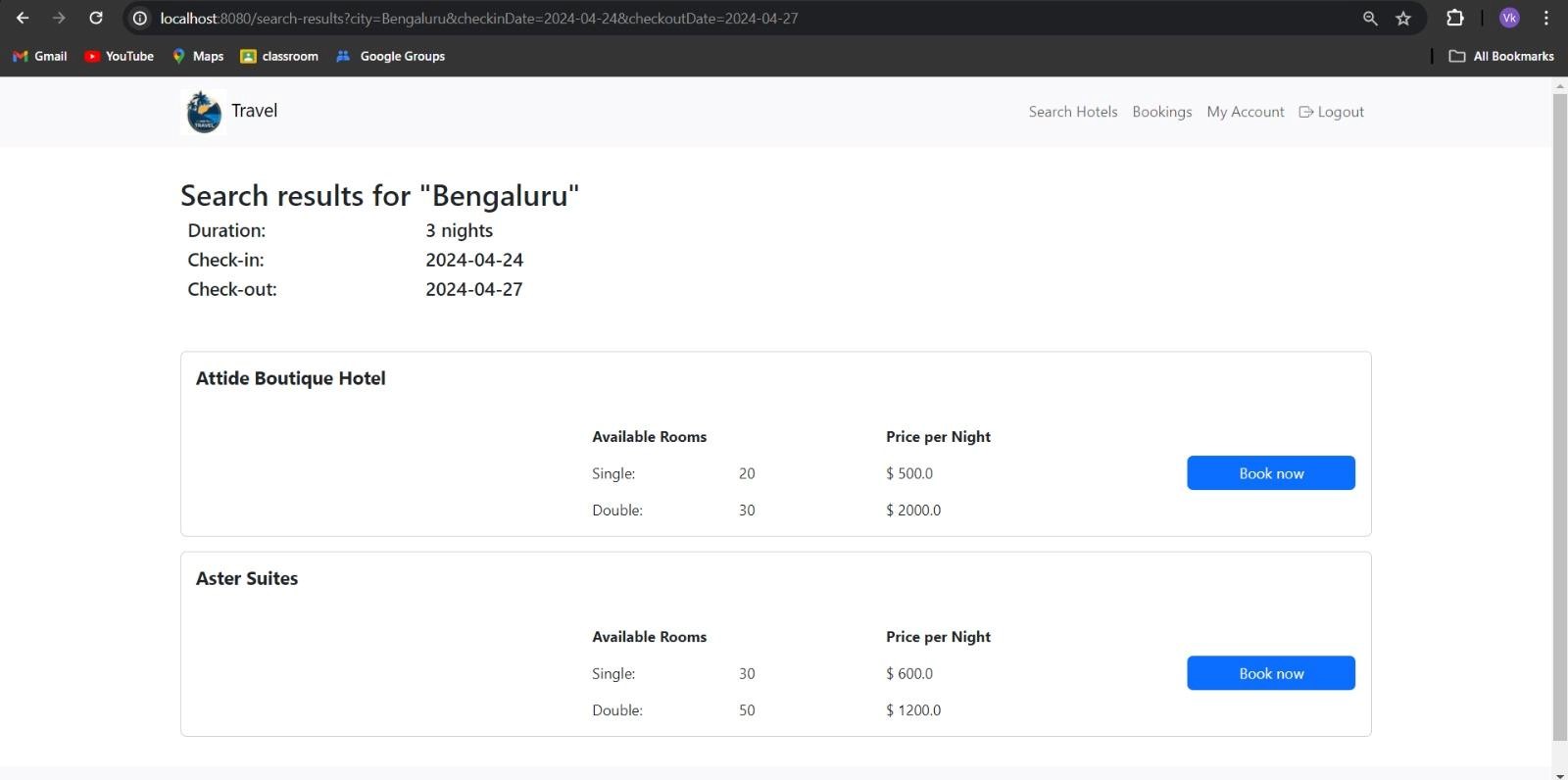
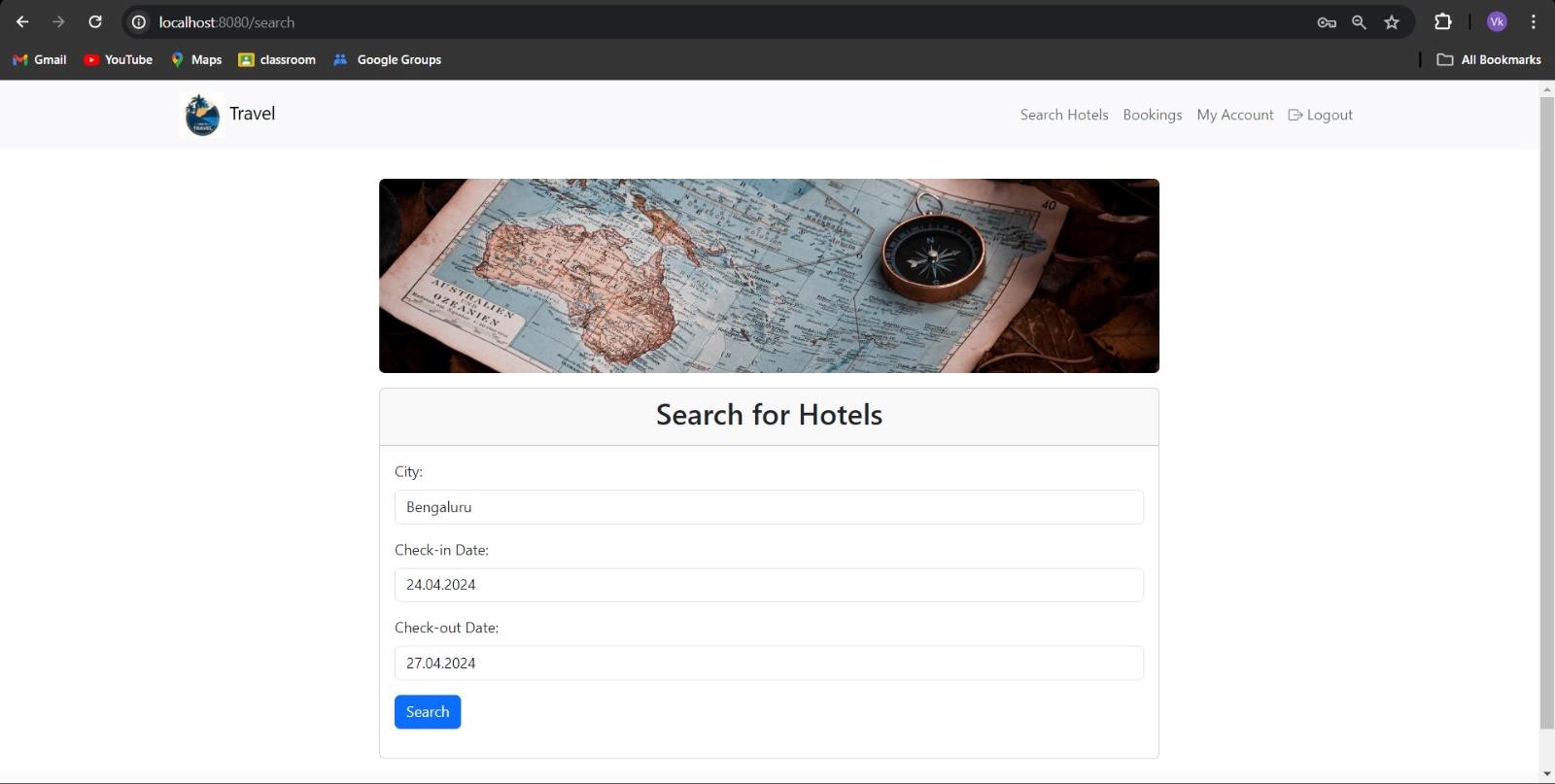
**OUTPUT SCREENSHOTS**



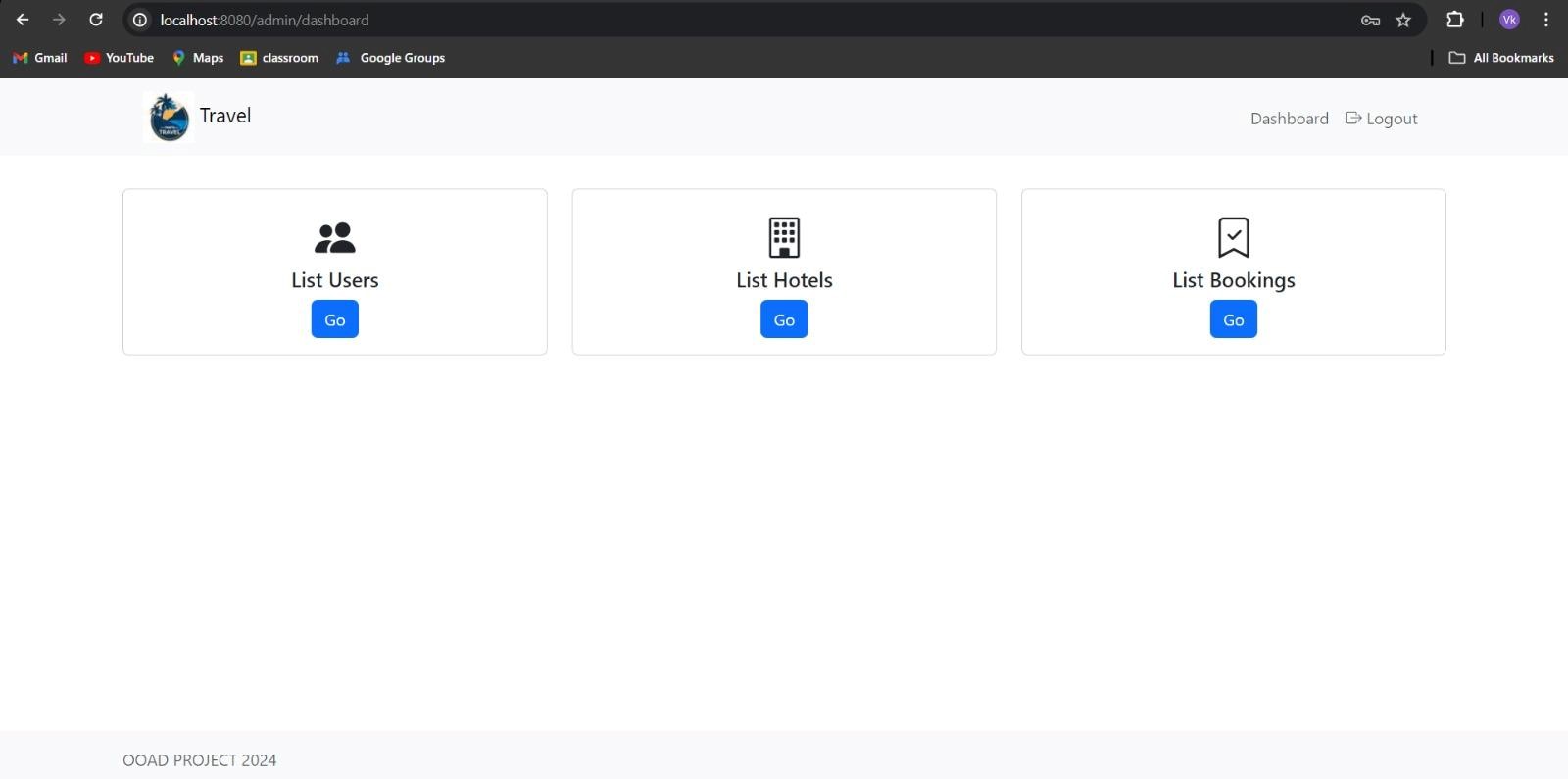
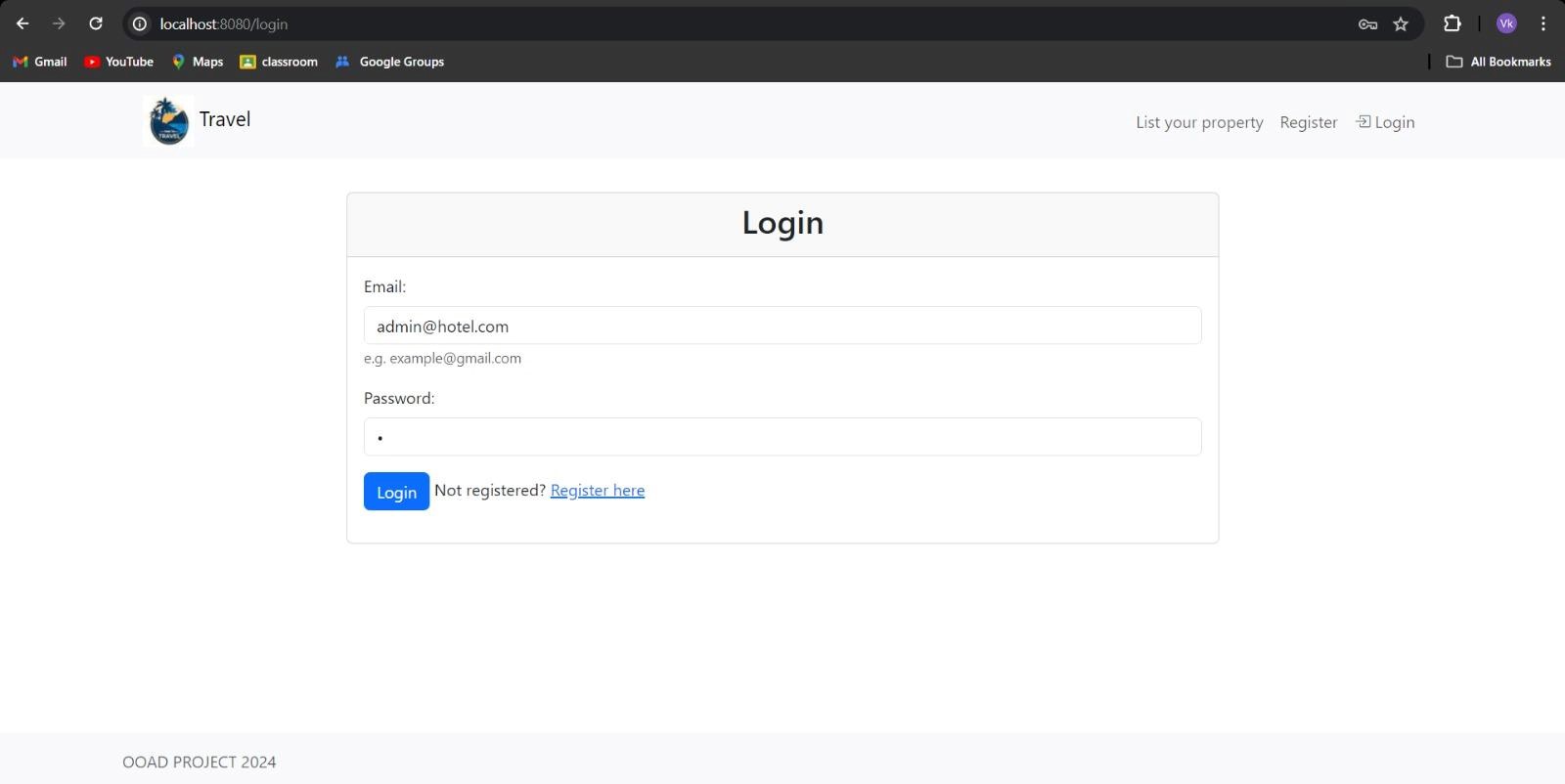
## Customer page

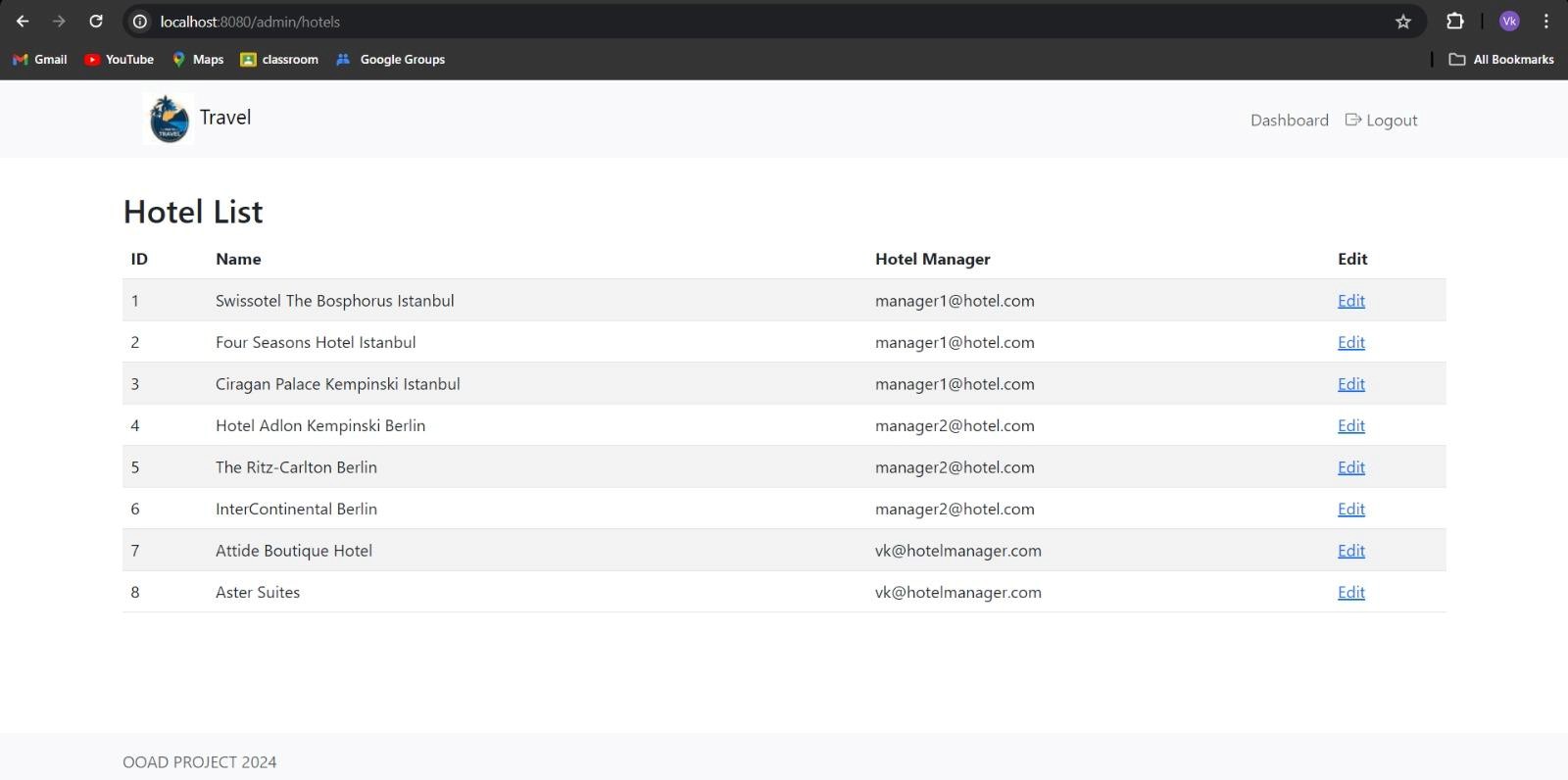




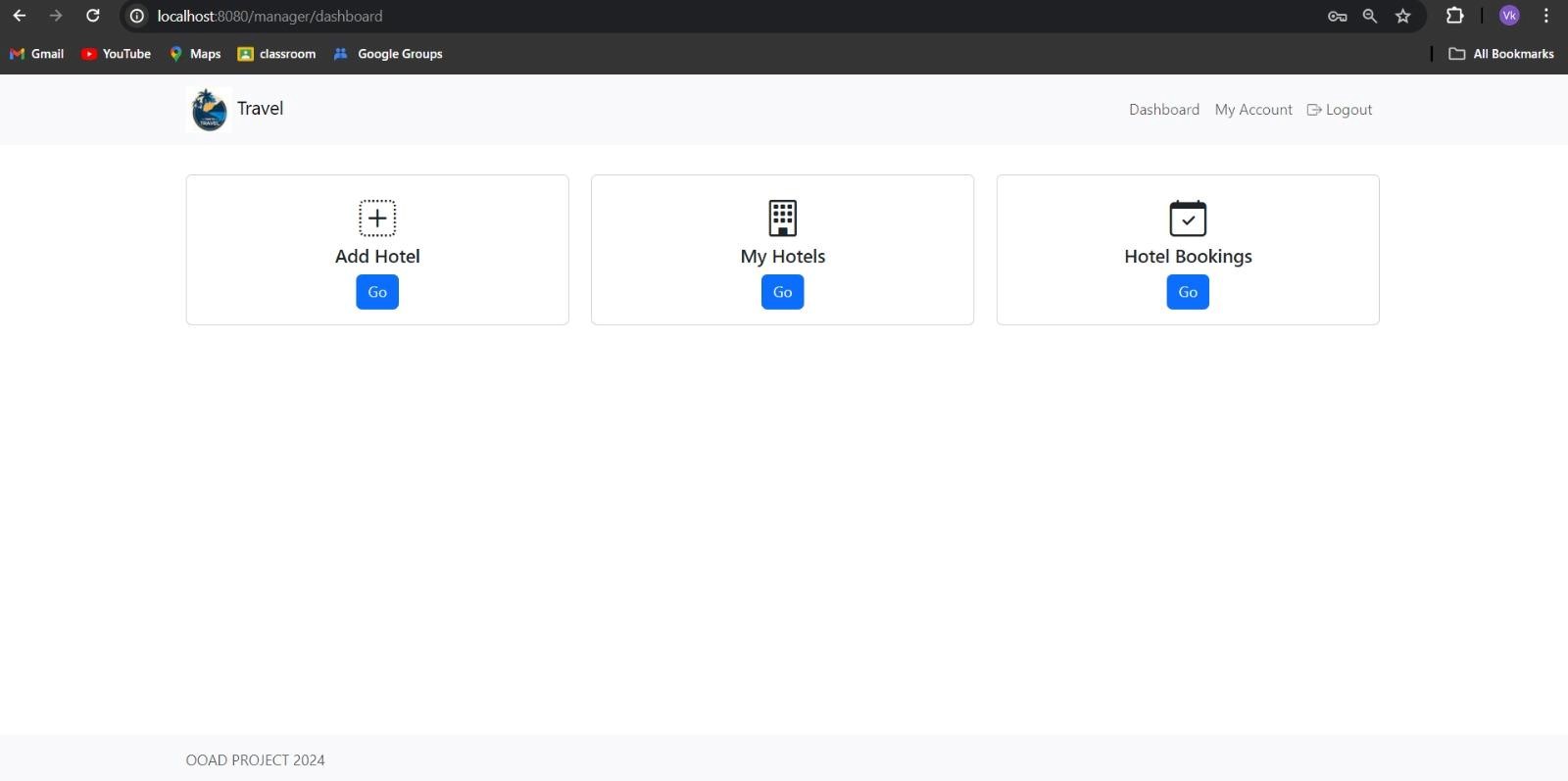


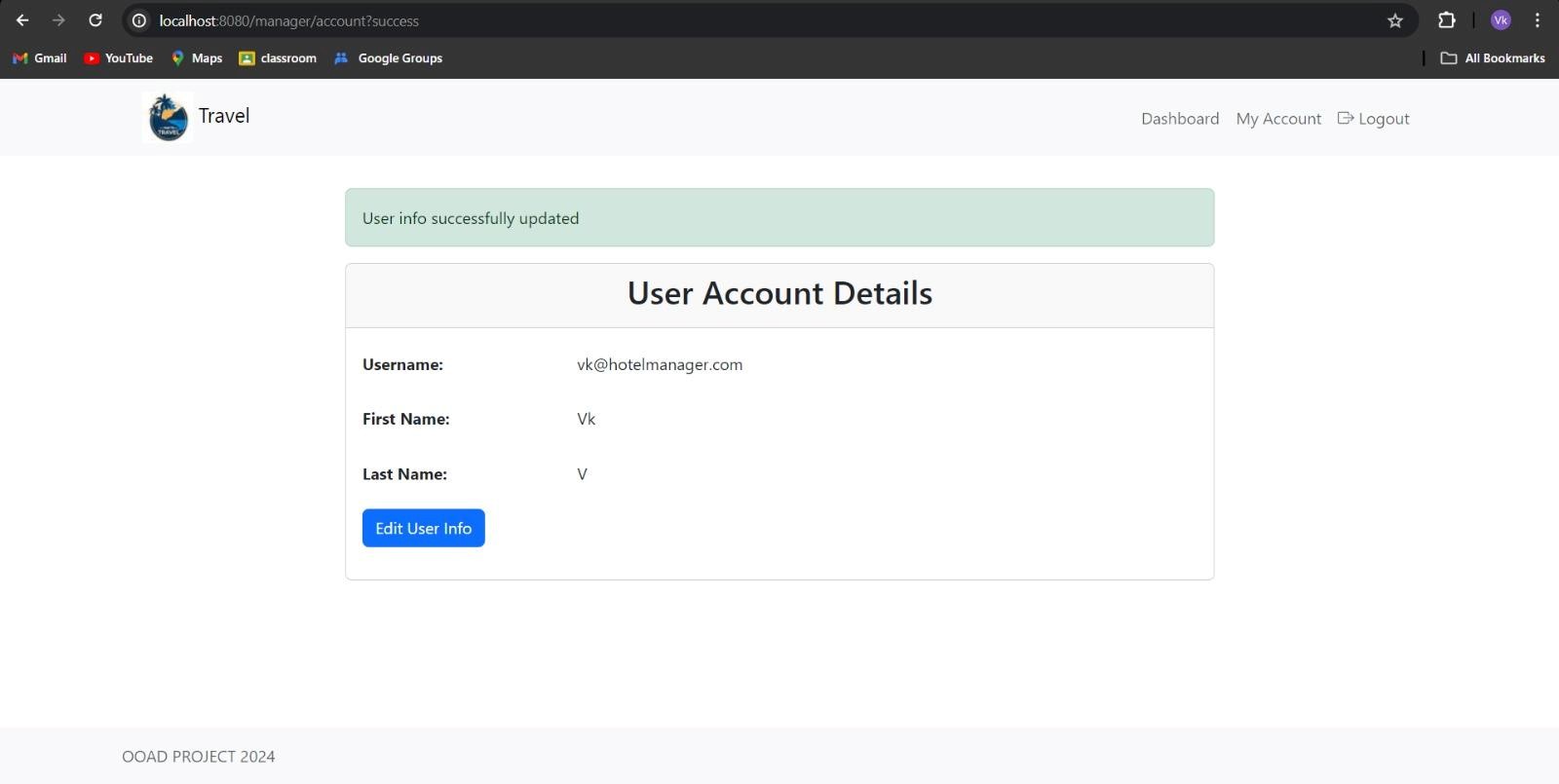
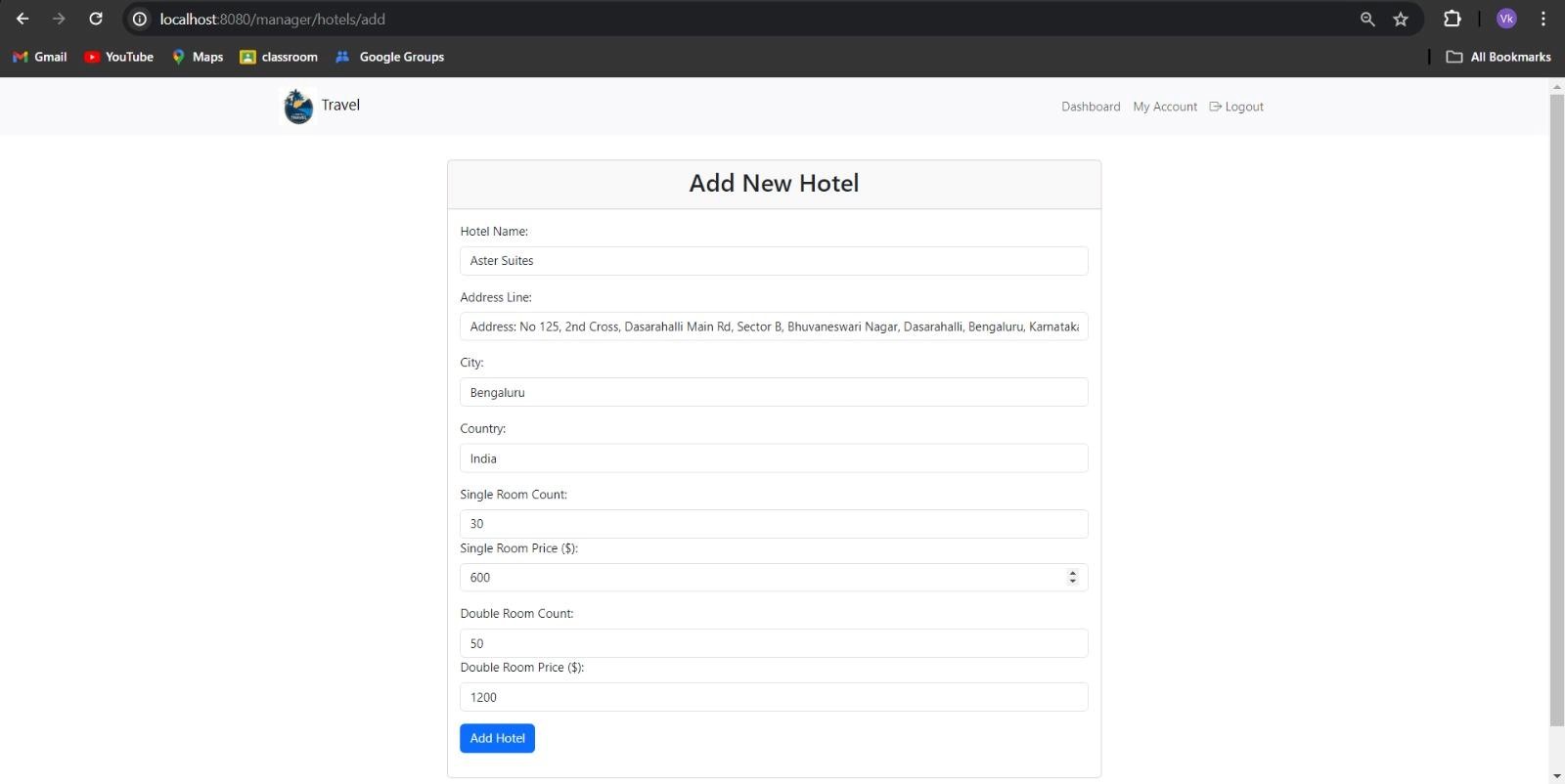
## Admin page

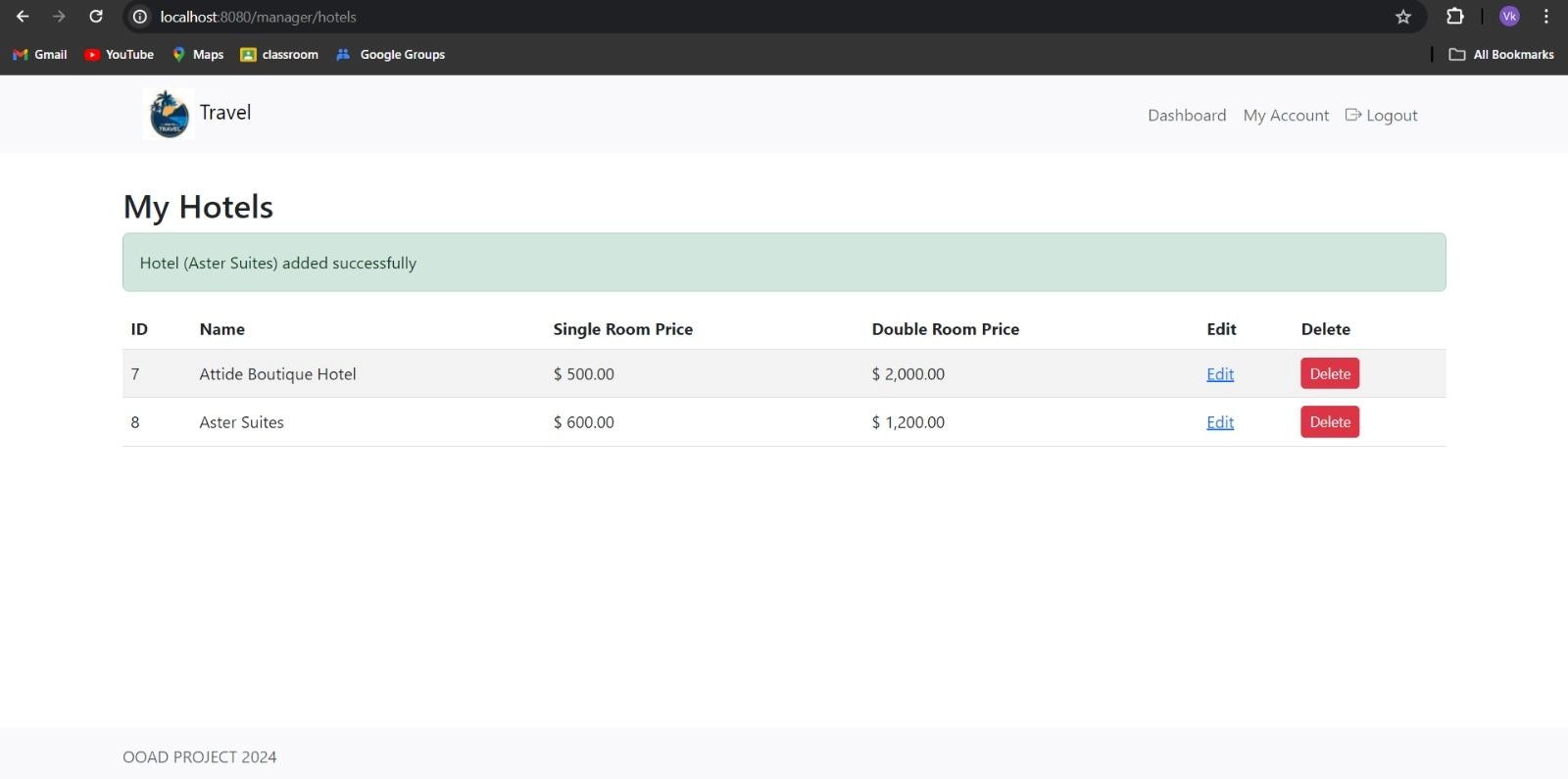


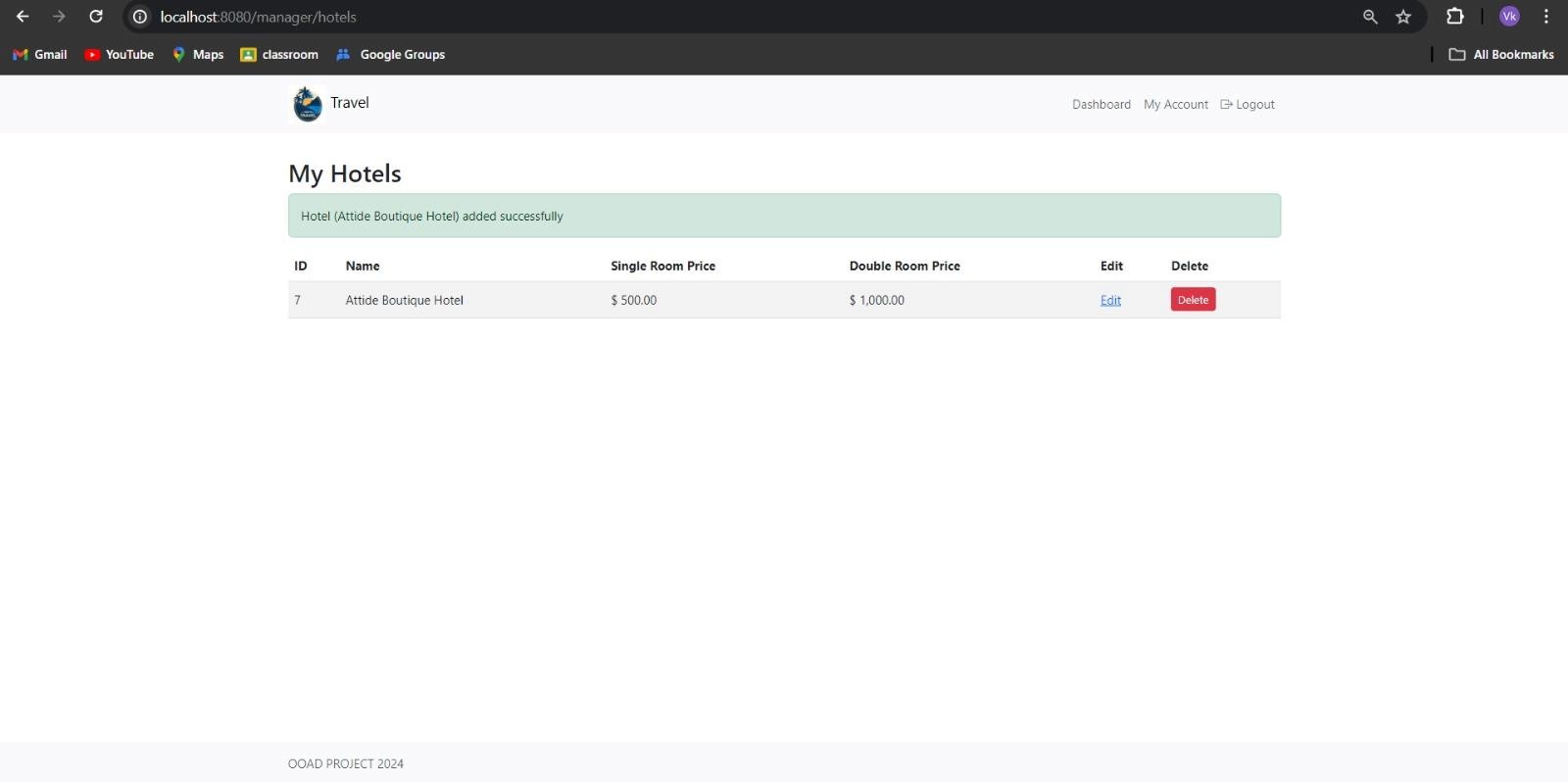
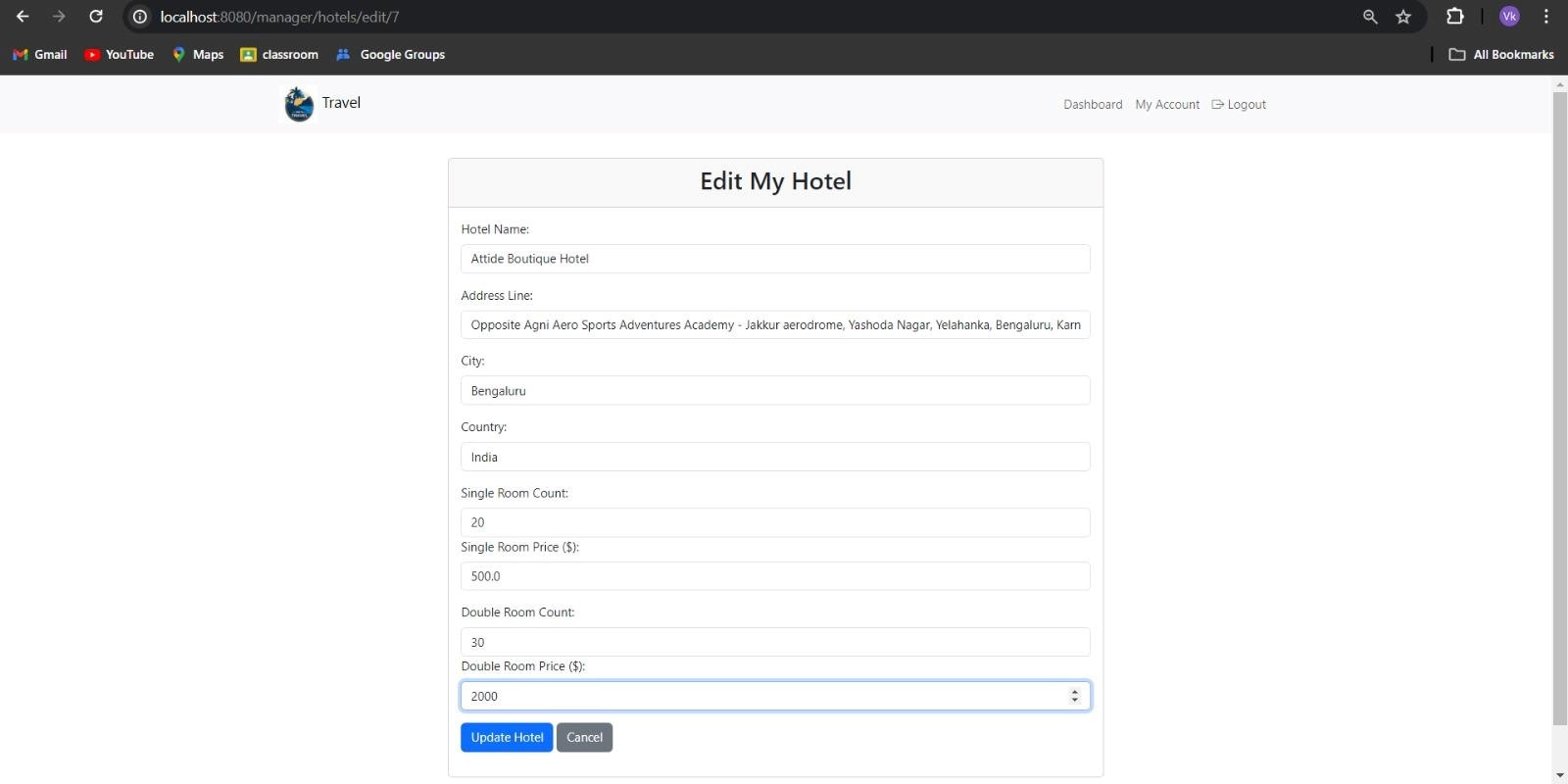


## HOTEL MANAGER PAGE









**WORK RESPONSIBILITY**

**K Virupakshi -** User Registration Registration Management

Payment

**LSS Praneeth Kumar-** Hotel Search & Listing Booking Management

**Mani Shankar M -** Hotel Management Admin Panel.

**Jyothiradithya D -** Room Booking

Payment Processing

Github Repository Link:

https://github.com/Virupakshi047/Hotel\_Management\_System.git