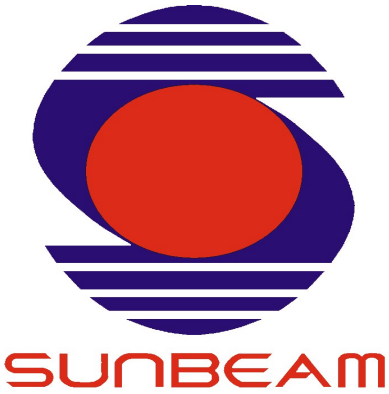
A PROJECT REPORT ON

Online Parking System

SUBMITTED IN
PARTIAL FULFILLMENT OF

DIPLOMA IN ADVANCED COMPUTING (PG-DAC)



BY

Manish Patil
Manish Gupta
Kartik Patil
Shreyash Gaikwad

UNDER THE GUIDENCE OF
Snehal Jadhav

AT
SUNBEAM INSTITUTE OF INFORMATION TECHNOLOGY, PUNE

SUNBEAM INSTITUTE OF INFORMATION TECHNOLOGY, PUNE.

CERTIFICATE

This is to certify that the project

Online Parking System

Has been submitted by

Manish Patil
Manish Gupta
Kartik Patil
Shreyash Gaikwad

In partial fulfillment of the requirement for the Course of **PG Diploma in Advanced Computing (PG-DAC AUG 2024)** as prescribed by The **CDAC** ACTS, PUNE.

Place: Pune Date: 11 Feb 2025

Snehal Jadhav
Project Guide

ACKNOWLEDGEMENT

We would like to take this opportunity to express our heartfelt gratitude to those who have guided and supported us throughout the journey of this project. A special thanks to Mr. Nitin Kudale (Center Coordinator, SIIT, Pune) and Mr. Yogesh Kolhe, Course Coordinator at SIIT Pune, for their invaluable guidance, unwavering support, and continuous encouragement. Their expertise, constructive feedback, and insightful suggestions at every stage of the project were instrumental in shaping it to its current form.

We are also deeply thankful to the entire faculty and staff members of Sunbeam Institute of Information Technology, Pune, for their support and cooperation.

This project would not have been possible without their assistance, and we are truly grateful for their contributions.

Manish Patil
Manish Gupta
Kartik Patil
Shreyash Gaikwad

0324 PG-DAC
SIIT Pune

ABSTRACT

The Online Parking System is a comprehensive web application designed to streamline the process of booking and managing parking spaces. The system caters to three primary user groups: customers, parking space providers i.e. manager, and administrators. Customers can easily browse available parking spaces, make bookings, and complete payments using QR code. Providers can manage their parking space availability, track bookings, and update space details. Administrators oversee the entire system, ensuring smooth operations and user account management. The system's design focuses on high performance, security, and user-friendly interfaces to ensure seamless interaction. Additionally, the database is structured to efficiently manage user data, parking space information, and booking records. The project adheres to industry-standard coding practices, ensuring maintainable and scalable code. The overall goal is to offer a reliable, efficient, and convenient solution to address parking management challenges in urban areas.

INDEX

LIST OF TABLES

LIST OF FIGURES

1. Introduction

With rapid urbanization and the growing number of vehicles, finding a suitable parking space has become a major challenge in metropolitan areas. The increasing demand for an efficient, accessible, and technology-driven parking solution has led to the development of the Online Parking System—a web-based platform that simplifies the process of locating, booking, and managing parking spaces.

This system provides a seamless experience for three primary user roles:

- Customers can search for nearby parking spots using location-based search powered by Google Maps API, make bookings, track their booking history, and manage payments.
- Parking providers can list and manage their available parking spaces, track reservations, and adjust availability in real time.
- Administrators oversee the platform's operations, enforce policies (such as banning users if necessary), and analyze business performance using built-in analytics tools.

To ensure security and efficiency, the system implements Role-Based Authentication and Authorization using Spring Security. User credentials are securely managed with BCRYPT hashing, and all sensitive data is transmitted over HTTPS. JWT-based stateless session management is used for authentication, ensuring a scalable and secure login mechanism. The backend is built with Spring Boot, leveraging Spring Data with Hibernate to persist data in a MySQL database. Additionally, email verification is implemented using a REST service built with Express.js and NodeMailer on Node.js.

The frontend, developed using React.js and Bootstrap, provides a user-friendly and responsive interface. Axios is used to integrate the frontend with the backend, ensuring smooth communication between components.

By streamlining parking space management, reducing congestion, and optimizing urban mobility, the Online Parking System contributes to smarter city planning and a hassle-free parking experience.

2. PRODUCT OVERVIEW AND SUMMARY

2.1 Purpose

The **Online Parking System** aims to address the growing challenge of finding and managing parking spaces in urban areas by providing a technology-driven solution for both drivers and parking providers. With the increasing number of vehicles and limited parking infrastructure, congestion and inefficient space

utilization have become common issues. This project seeks to streamline the parking process by offering a convenient, real-time booking system that enhances accessibility and optimizes space management.

For customers, the system provides a hassle-free way to search for nearby parking spaces using **Google Maps API**, make reservations, and track booking history. This eliminates the frustration of searching for parking manually and reduces traffic congestion caused by vehicles circling for vacant spots.

Parking providers benefit from a centralized platform that allows them to manage available spaces efficiently, track reservations, and maximize occupancy. Administrators have the authority to oversee system operations, monitor user activities, and enforce policies, such as banning fraudulent users and analyzing business performance using analytics tools.

Security and efficiency are key aspects of this system, ensured through **Spring Security-based authentication, JWT-based stateless sessions, and encrypted password storage using BCRYPT**. By integrating **React.js, Spring Boot, and MySQL**, this system provides a **scalable, secure, and user-friendly parking management solution**, ultimately contributing to smarter urban mobility.

2.2 Scope

The Online Parking System is designed to provide a seamless, efficient, and secure solution for parking space booking and management. It caters to three primary user roles: customers, parking providers, and administrators, each with distinct functionalities and access privileges.

Scope for Customers

1. Search for available parking spaces using Google Maps API based on real-time location.
2. Book parking spots in advance to avoid last-minute inconvenience.
3. View and manage booking history for reference and future planning.
4. Ensure secure transactions and user authentication through Spring Security and JWT-based authentication.

Scope for Administrators

1. Oversee the entire system, ensuring smooth operation and policy enforcement.
2. Manage user access, including the ability to ban users engaging in fraudulent activities.
3. Analyze system performance and usage trends using business analytics tools.

The system is built using React.js for the frontend, Spring Boot for the backend, MySQL for data storage, and Node.js for email verification. With role-based authentication, stateless session management, and encrypted credentials, it ensures a secure and scalable solution for urban parking challenges. The project can be expanded in the future by integrating automated parking sensors, mobile payment gateways, and AI-based parking predictions, making it a smart and adaptable solution for modern cities.

2.3 User Classes and Characteristics

The Online Parking System is designed to accommodate three distinct user roles, each with specific access levels, functionalities, and responsibilities. These user roles ensure an efficient and structured operation of the parking management system.

1. Customers (General Users)
   Customers are the primary users who book parking spaces through the system.
   Characteristics:

   i. Can search for parking spaces using Google Maps API.
   ii. Can book, modify, and cancel parking reservations.
   iii. Has access to booking history for reference.
   iv. Can register and sign in securely using Spring Security with hashed passwords (BCRYPT).
   v. Can make payments via secure transactions.
   vi. Receives notifications and booking confirmations.
   vii. Can submit reviews and ratings for parking lots.

2. Administrators
   Administrators oversee the entire system, manage user activities, and ensure smooth operations.
   Characteristics:

   i. Has the authority to ban users who violate policies.
   ii. Can monitor business analytics to track system performance.
   iii. Ensures system security and smooth functionality.

Each user class plays a critical role in the Online Parking System, ensuring a secure, efficient, and user-friendly experience for all stakeholders.

2.4 Design and Implementation Constraints

The Online Parking System is developed with specific design and implementation constraints to ensure security, scalability, and efficiency. These constraints define the limitations and considerations that guide the system's architecture and functionality.

1. Design Constraints

These constraints influence how the system is structured and built.

I. Technology Stack Limitation:
   a. The backend is implemented using Spring Boot with Spring Security for authentication.
   b. The frontend is designed with React.js and Bootstrap for a responsive UI.
   c. Data storage and management are handled using MySQL with Hibernate (JPA ORM framework).

II. Security Considerations:
   a. Role-Based Authentication and Authorization using Spring Security.
   b. JWT (JSON Web Token) for stateless session management.
   c. Password hashing with BCRYPT to prevent unauthorized access.
   d. Data transmission over HTTPS to enhance security.

III. Integration Constraints:
   a. The system must integrate with Google Maps API for location-based parking searches.
   b. Axios is used to handle API requests between the frontend and backend.
   c. NodeMailer with Express.js is implemented for email verification.

IV. User Roles and Access Control:
   a. Customers, Parking Providers, and Administrators must have clearly defined permissions and restricted actions.
   b. The Administrator has the authority to ban users and monitor analytics.

V. Scalability & Maintainability:
   a. The database schema is designed to support future expansions (e.g., AI-based parking predictions, IoT sensors).
   b. The RESTful API design ensures easy integration with third-party services.

2. Implementation Constraints

These constraints define technical limitations and restrictions during system development and deployment.

I. Database Constraints:
   a. The MySQL database must support complex queries for parking space management and analytics.
   b. ACID compliance is required to ensure transaction reliability.
   c. The system must efficiently handle large-scale data (bookings, payments, user data).

II. Performance Constraints:
    a. The system should be optimized for fast query execution to prevent booking delays.
    b. High traffic handling must be ensured with efficient indexing and caching strategies.
III. Deployment Constraints:
    a. The system should be cloud-deployable with containerization support (e.g., Docker, Kubernetes).
    b. A CI/CD pipeline is recommended for continuous updates and bug fixes.
IV. Device & Browser Compatibility:
    a. The web application must be responsive and work across various screen sizes.
    b. It should be compatible with modern browsers (Chrome, Firefox, Edge, Safari).

By adhering to these constraints, the Online Parking System ensures security, scalability, and efficiency while providing an optimal experience for all users.

3. REQUIREMENTS

3.1 Functional Requirements

Functional requirements define the specific operations and features the system must support to fulfill user needs. The system caters to three primary user roles: Administrators, Customers.

3.1.1 Use Case for the Administrator

The Administrator is responsible for managing users, overseeing operations, and maintaining the security and efficiency of the system.

I. Use Case: Manage Users
Actors: Administrator
Preconditions: The administrator must be logged in with appropriate privileges.
Flow of Events:

1. View the list of registered users (customers and parking providers).
2. Ban or restrict users violating system policies.
3. Grant or revoke access permissions.

Postconditions: Users are managed effectively, ensuring compliance and security.

II. Use Case: Monitor Business Analytics
Actors: Administrator
Preconditions: The system must have data on transactions and parking utilization.
Flow of Events:

1. Access reports on parking lot usage, user activity, and revenue.
2. View insights such as peak booking times and customer trends.
3. Generate analytical reports for business decision-making.

- Postconditions: The administrator can make informed decisions based on analytics.

III. Use Case: System Maintenance & Security Management
Actors: Administrator
Preconditions: The administrator must have security privileges.
Flow of Events:

1. Ensure system uptime and monitor performance.
2. Configure and enforce security policies (e.g., password rules, authentication methods).
3. Manage API integrations, such as Google Maps API for location services.

Postconditions: The system remains secure, functional, and up-to-date.

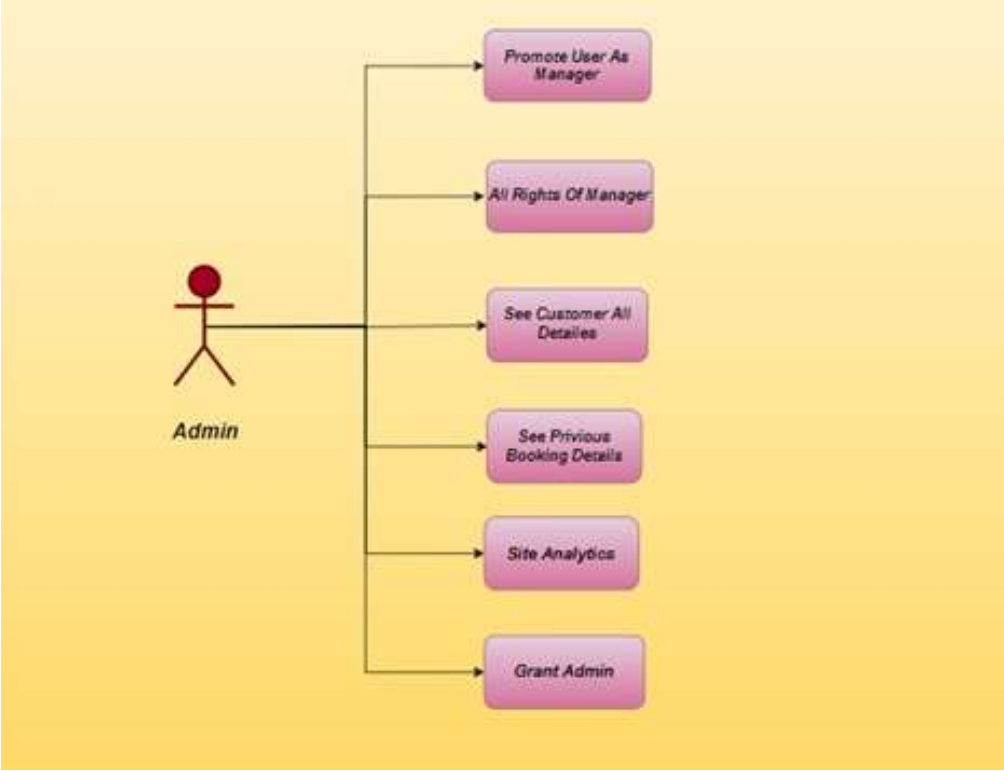


Figure 1Use Case For Admin

3.1.2 Use Case for the Customer

The Customer is the primary user who books parking spaces through the system. The system ensures a seamless experience by allowing users to search, book, and manage their reservations efficiently.

I. Use Case: Register and Authenticate
Actors: Customer
Preconditions: The user must provide valid credentials.
Flow of Events:

1. Sign up using email and password.
2. Verify email using a NodeMailer-based email verification system.
3. Log in securely via Spring Security with JWT-based authentication.

Postconditions: The user gains access to their account.

II. Use Case: Search and Book Parking Spaces
Actors: Customer
Preconditions: The customer must be logged in.
Flow of Events:

1. Enter location or use Google Maps API for nearby parking options.
2. View available parking spaces with real-time status updates.
3. Select a parking lot, choose a time slot, and confirm the booking.
4. Make a secure payment through the integrated payment system such as QR code.

Postconditions: The parking space is booked, and the user receives a confirmation notification.

III. Use Case: Review and Rate Parking Spaces
Actors: Customer
Preconditions: The user must have completed at least one booking.
Flow of Events:

1. Select a past booking and submit a review.
2. Rate the parking lot based on service quality.
3. View other customer reviews before booking.

Postconditions: User reviews help improve service quality.

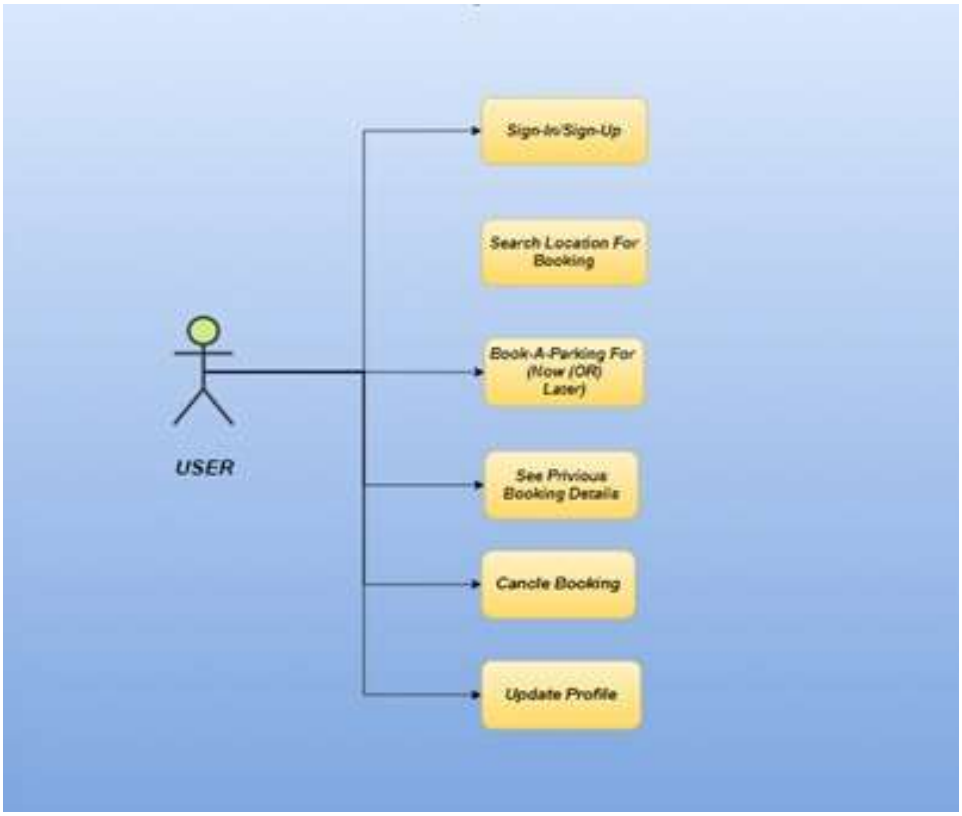


Figure 2 Use Case For Customer

3.2 Non-functional Requirements

Non-functional requirements define the quality attributes, system constraints, and overall operational characteristics of the **Online Parking System**. These ensure that the system performs efficiently, securely, and remains scalable for future enhancements.

3.2.1 Usability Requirement

The system must have an intuitive and user-friendly UI designed with React.js and Bootstrap.
It should support multiple devices (desktop, mobile, tablets) and work across modern web browsers.
Users should be able to complete key tasks (searching, booking, payment) within 3-5 clicks.

3.2.2 Performance Requirement

The system should handle multiple concurrent bookings without lag.
Response time for search queries and booking actions should be less than 2 seconds.
Efficient database queries and caching mechanisms should be used for optimal speed.

3.2.3 Reliability Requirement

The system should have 99.9% uptime to ensure availability.
All transactions (bookings, payments) should be ACID-compliant to prevent data inconsistencies.
Auto-recovery mechanisms should be in place in case of system failures.

3.2.4 Portability Requirement

The system should be deployable on cloud platforms (AWS, Azure, or on-premises).
It must support different operating systems (Windows, Linux, macOS) without major modifications.
The frontend should be responsive to different screen sizes and devices.

3.2.5 Security Techniques

Spring Security with JWT ensures secure authentication.
BCRYPT password hashing is used for user credentials.
HTTPS encryption protects data transmission.
Role-based access control (RBAC) restricts unauthorized actions.

Regular security audits help in identifying vulnerabilities.

4. PROJECT DESIGN

4.1 Data Model

4.1.1 Database Design

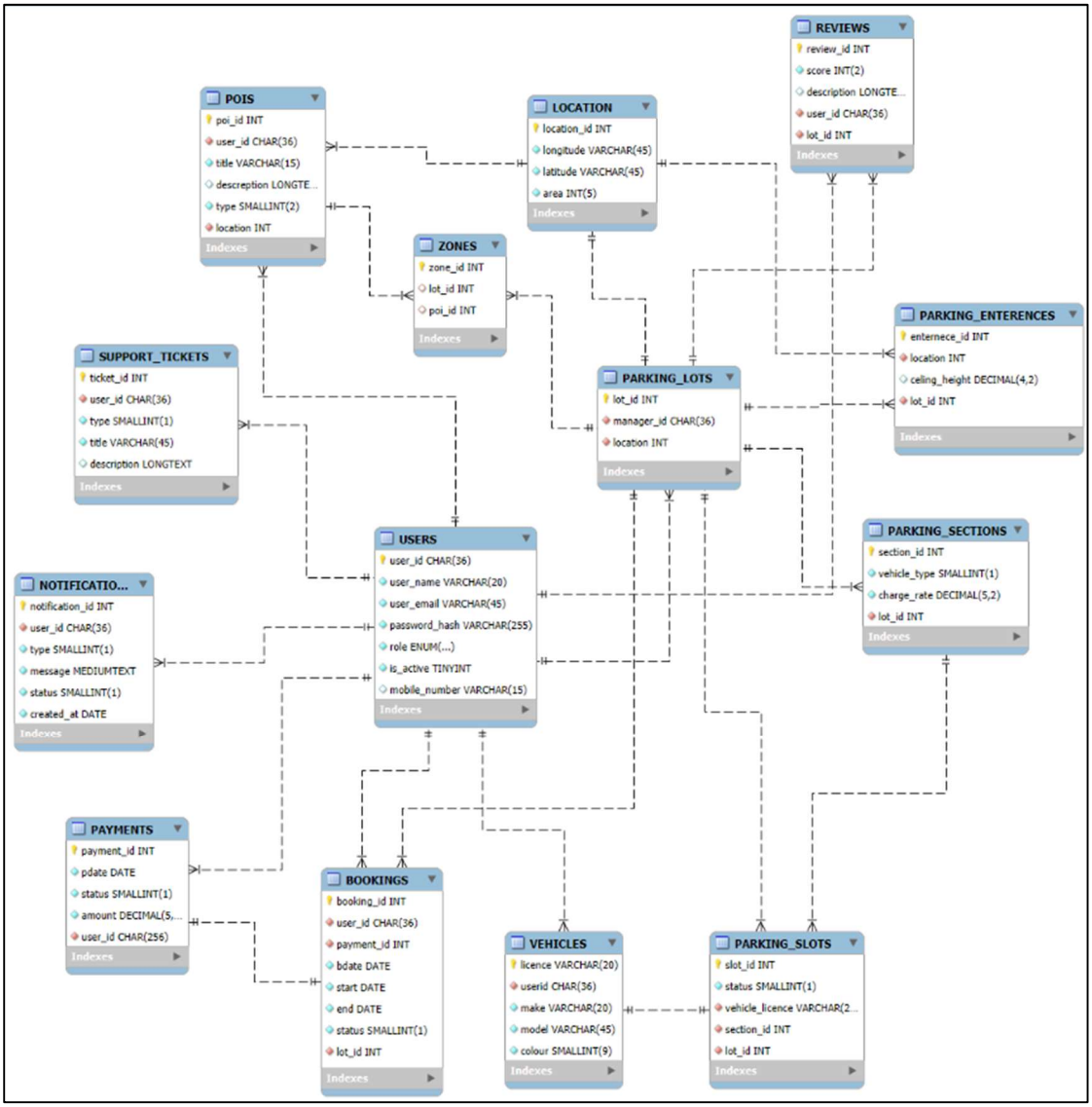


Figure 3 Database Design

4.2 Process Model

4.2.1 Functional Decomposition Diagram

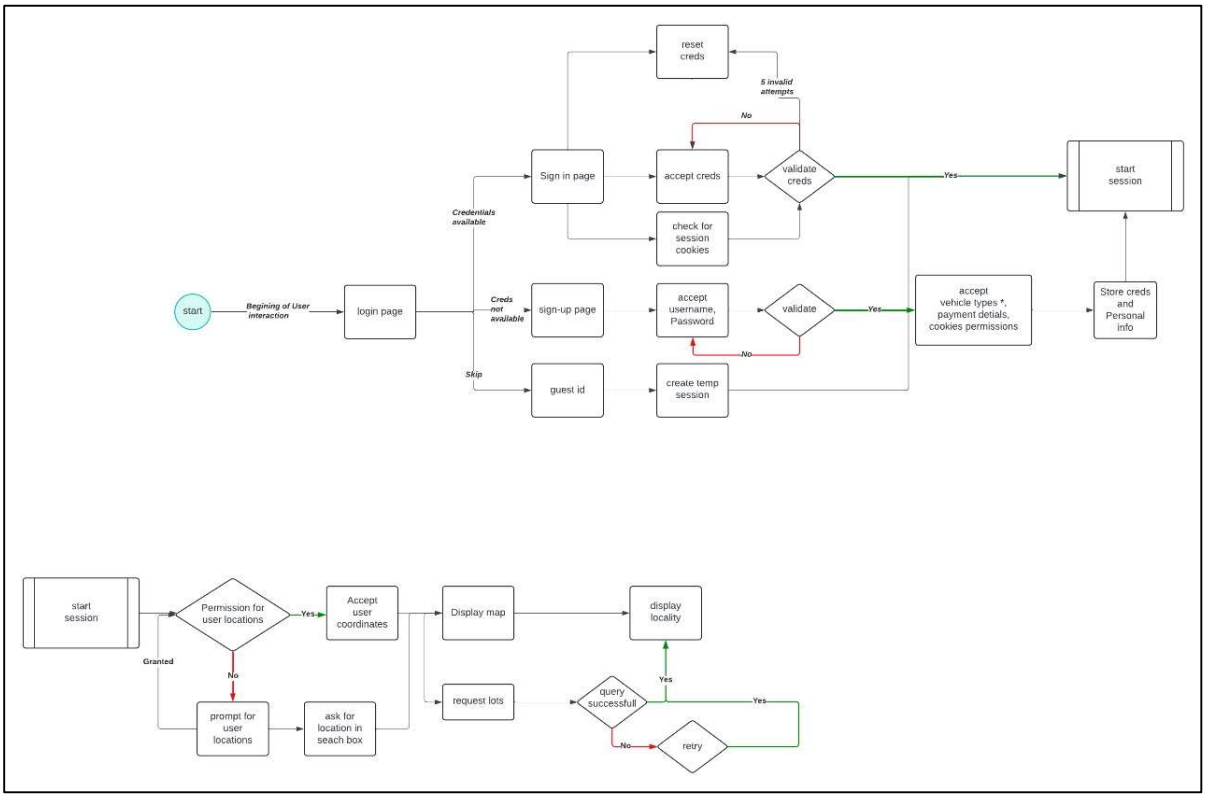


Figure 4 UI Flow Diagram

5. TEST REPORT

Table 1:Test Report

Test No	Tast Case Name	Objective	Status
1	User Registration and Authentication	Verify registration and login functionality.	**Pass**
2	Parking Space Search and Booking	Verify ability to search and book parking spaces.	**Pass**
3	Admin User Management	Verify admin can manage users (ban, view).	**Pass**
5	Usability and Interface Responsiveness	Verify UI responsiveness and ease of use.	**Pass**
6	Search Functionality	Verify search results accuracy.	Pass
7	Error Handling	Verify error handling for invalid actions.	Passs
8	Cross-Browser Compatibility	Verify functionality across different browsers.	Pass

6. OUTCOME

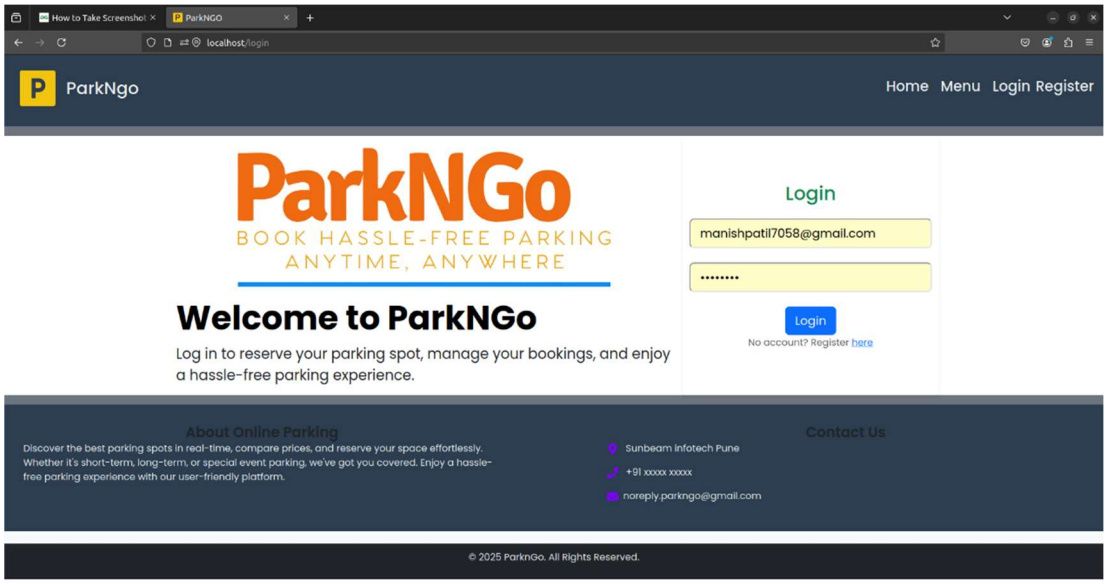


Figure 5 Login Page

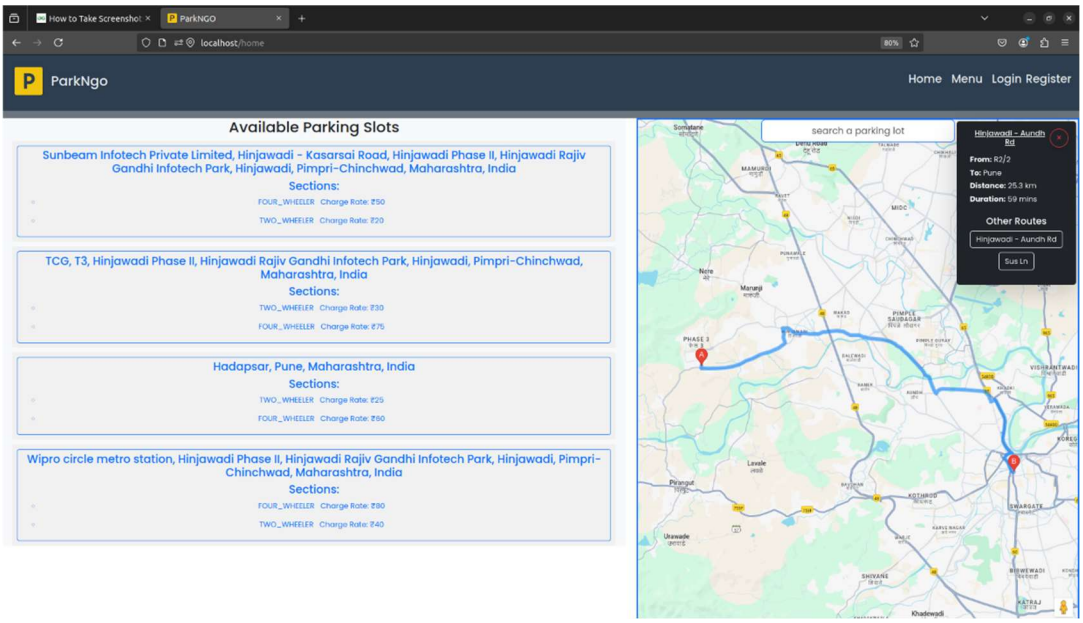


Figure 6 Parking Slots

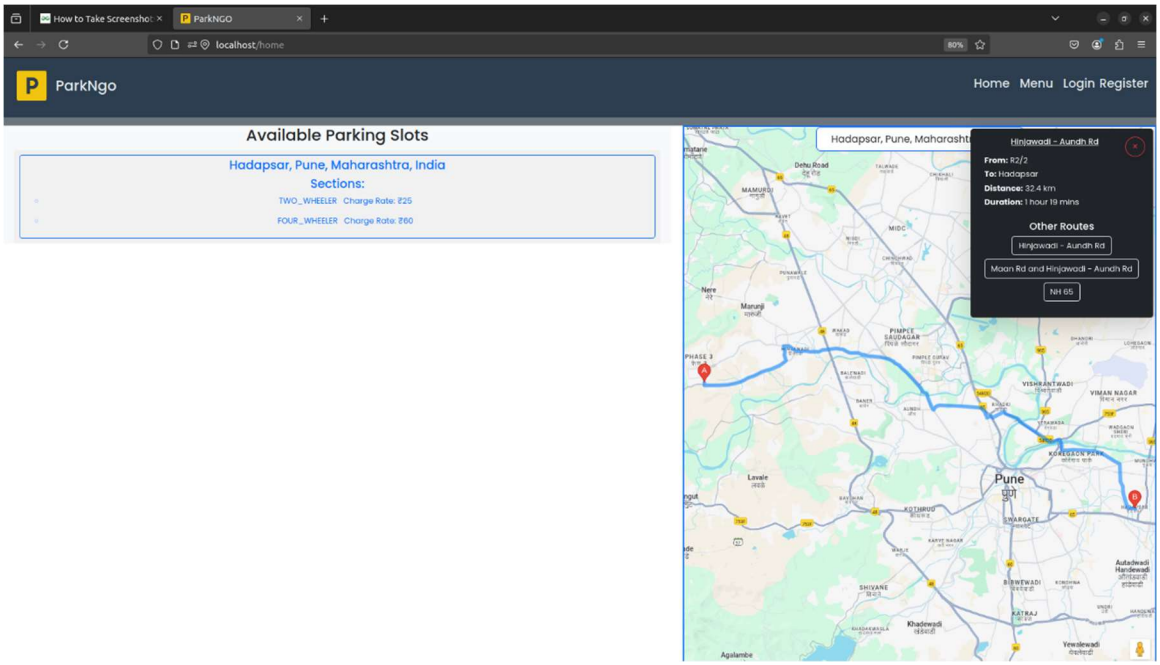


Figure 7 Search Location

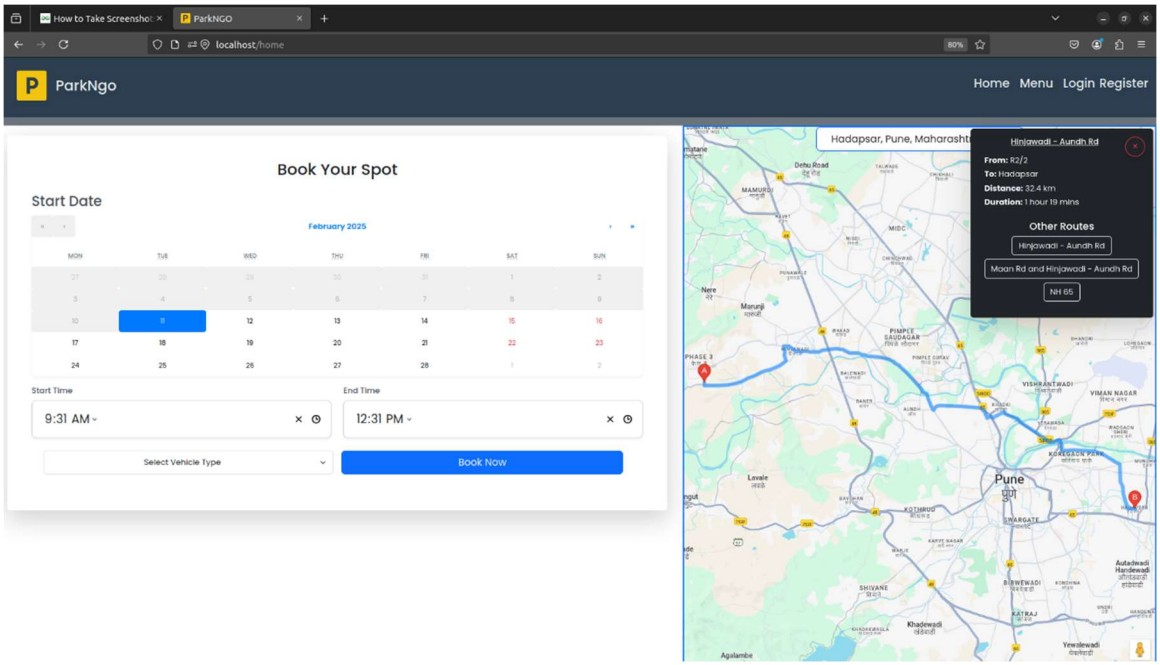


Figure 8 Book Slot

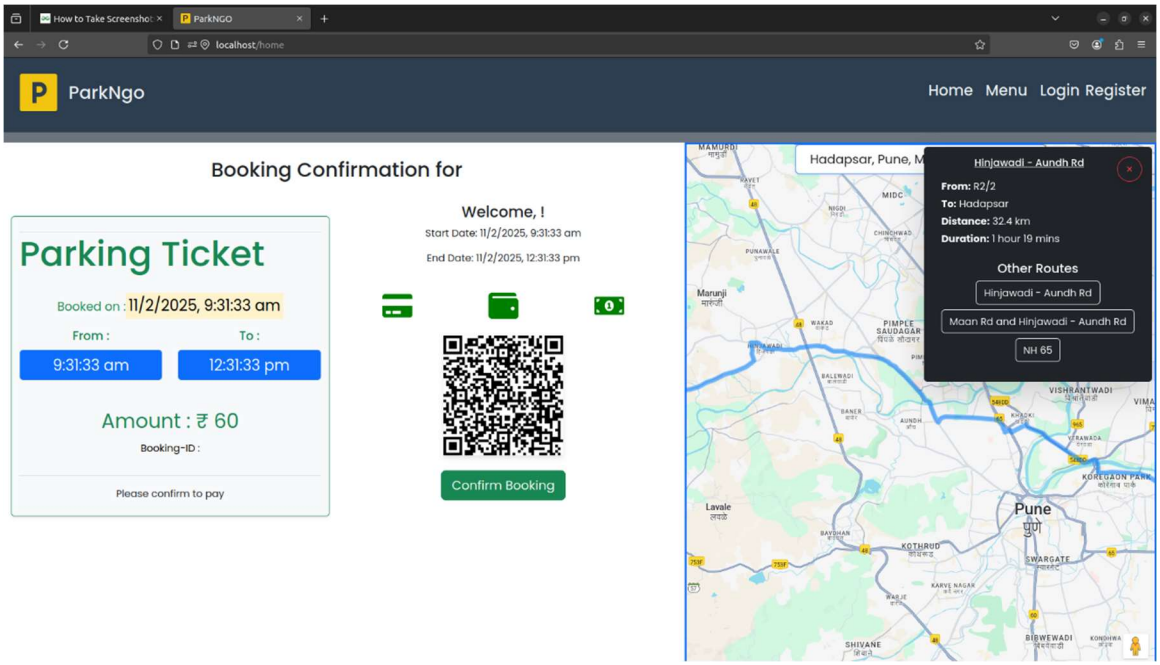


Figure 9 Booking Confirmation With QR Payment

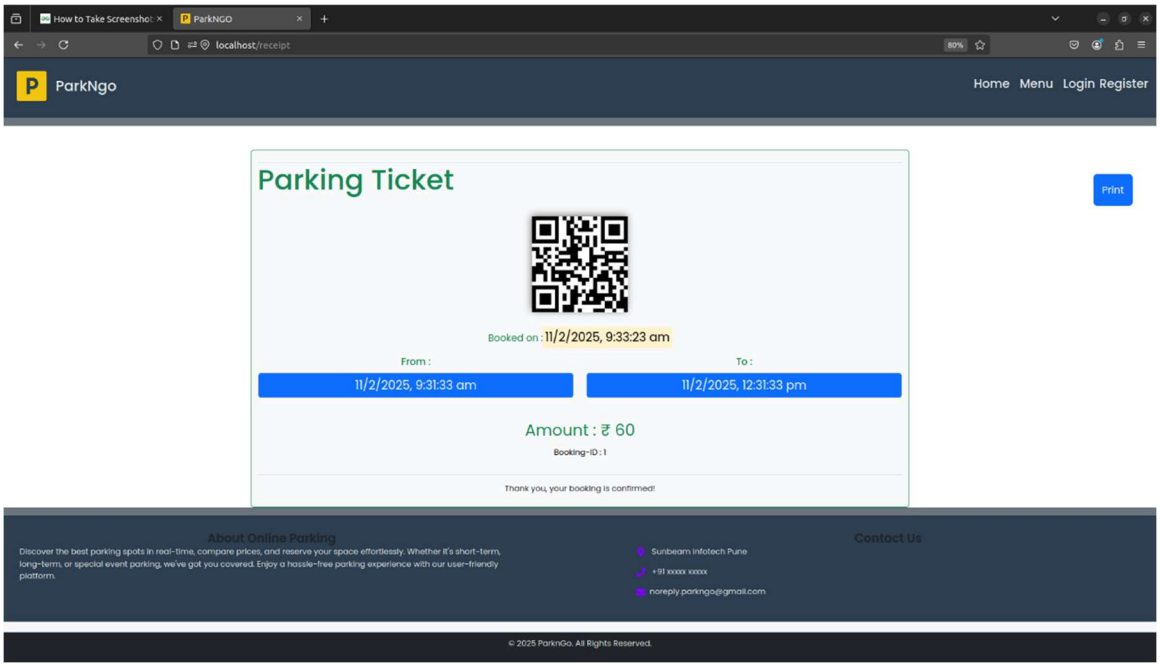


Figure 10 Booked Slot payment Receipts Page

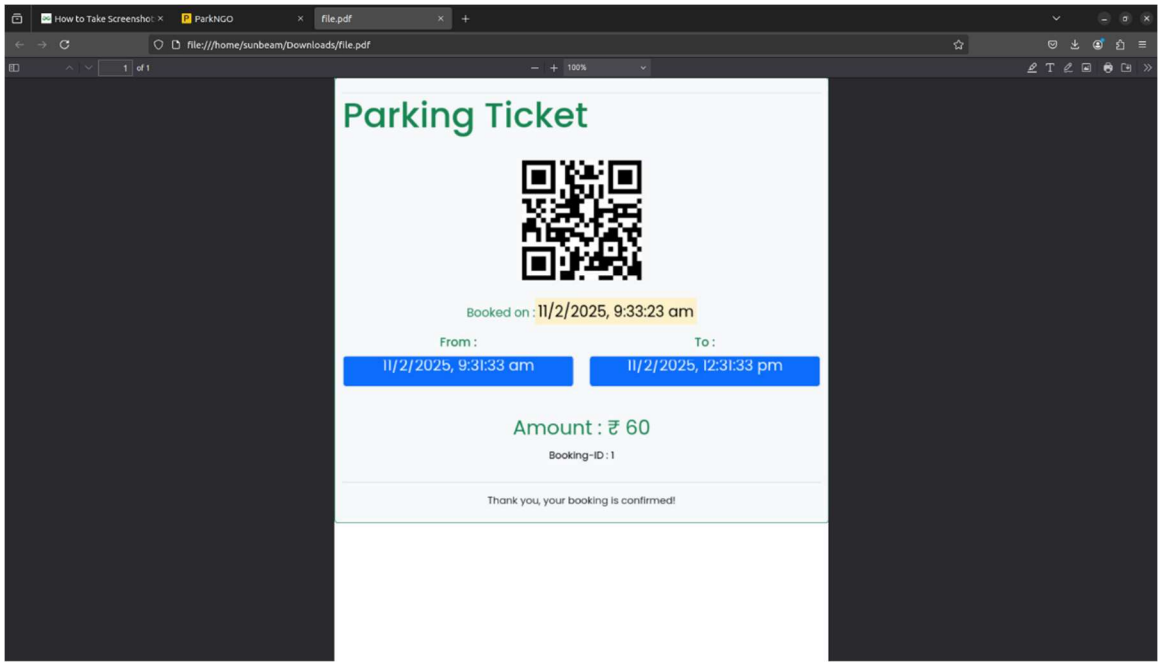
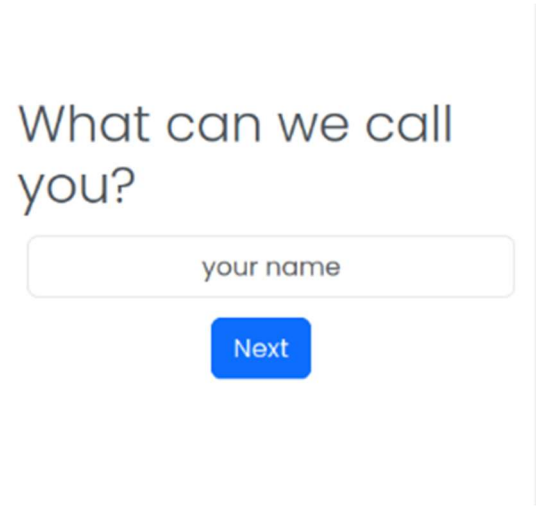


Figure 11 Printed PDF of Booked Slot



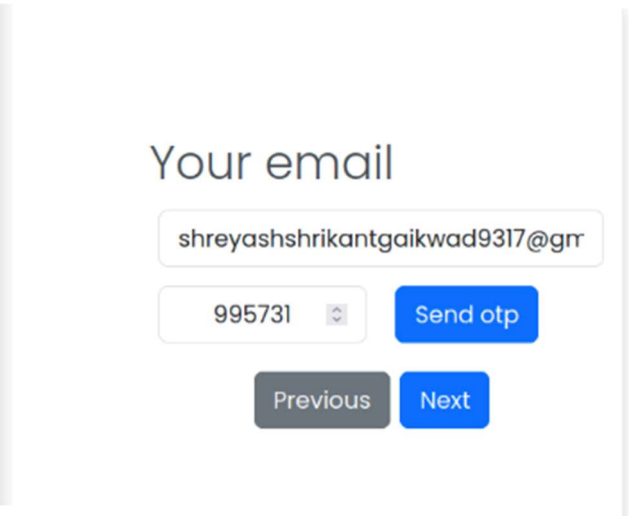


Figure 12 Registration page Enter Name Figure 13 Registration page Enter Email

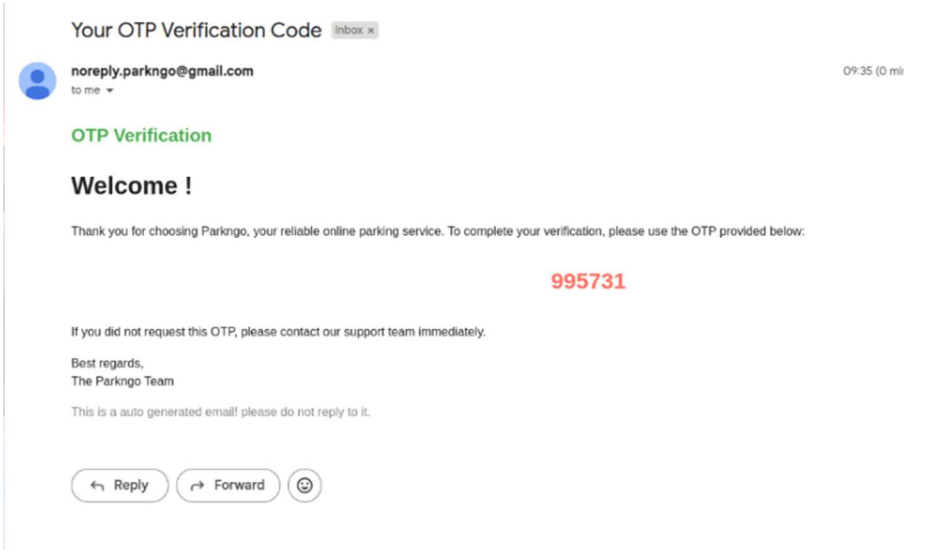


Figure14 Email OTP

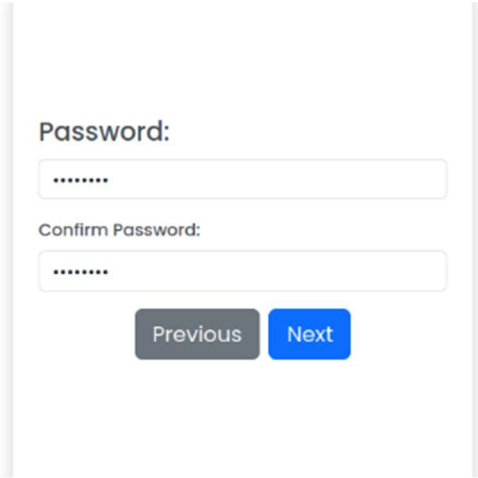


Figure 15 Password Page

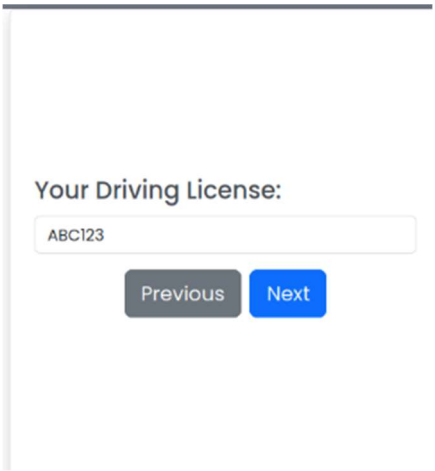


Figure 16 Driving License

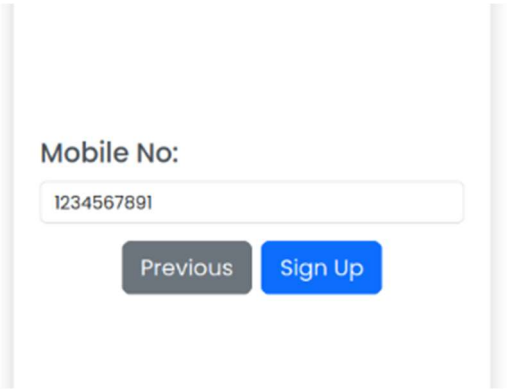


Figure 17 Mobile No

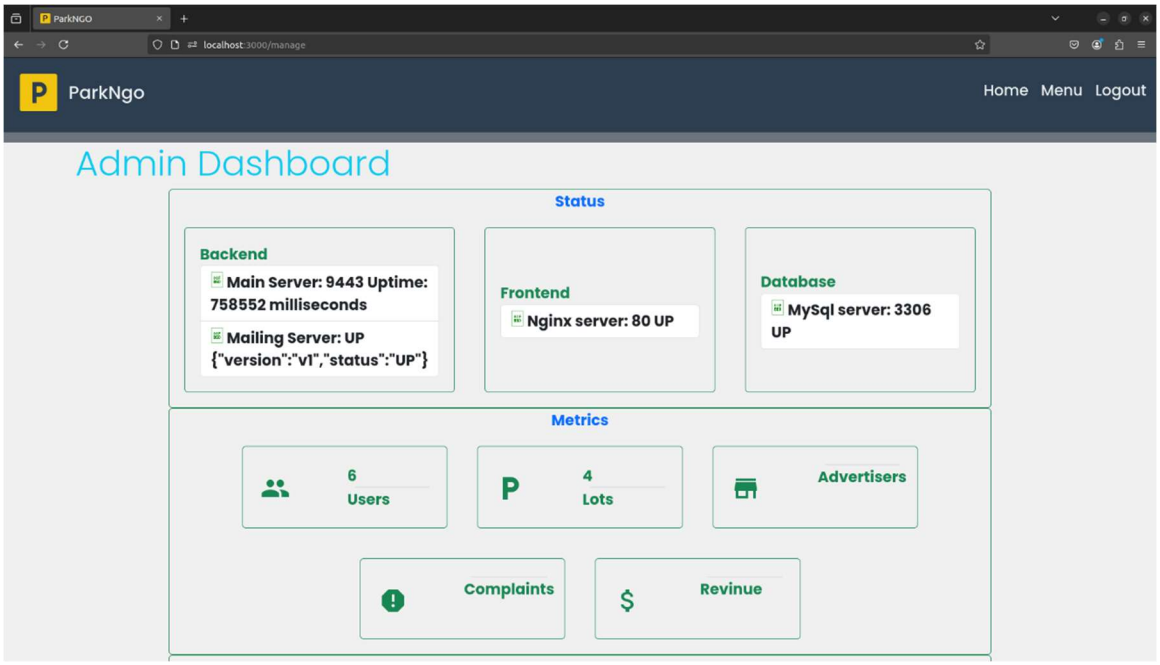


Figure 18 Admin Dashboard img.1

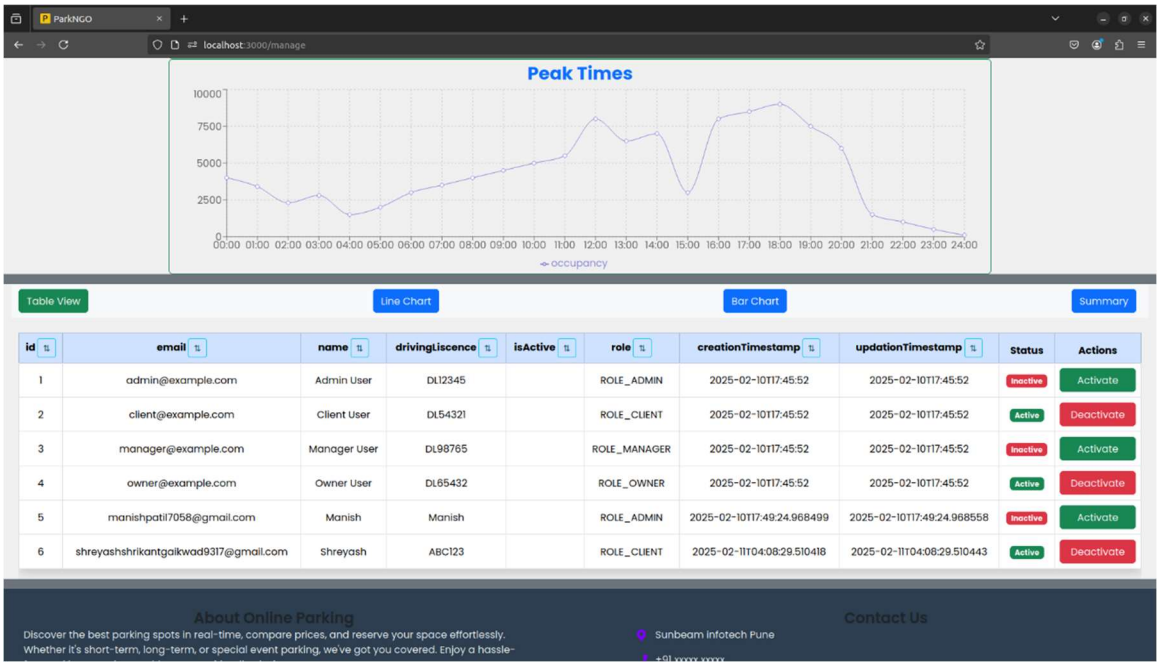


Figure 19 Admin Dashboard img.2

7. CONCLUSION

The Online Parking System is a comprehensive, technology-driven solution aimed at addressing the challenges of urban parking management. By integrating Google Maps API for location-based search, Spring Security for authentication, and React.js for an intuitive user interface, the system ensures a seamless and efficient parking experience.

This system caters to three primary user groups—customers, parking providers, and administrators—each with distinct functionalities that improve overall efficiency. Customers can search for available parking spaces, make real-time bookings, and track their history. Parking providers can manage slot availability and monitor demand, while administrators oversee user activities and system security.

With the implementation of JWT-based stateless session management, BCRYPT password hashing, and HTTPS-secured transactions, the system prioritizes security and data integrity. The integration of Spring Boot with MySQL and Hibernate ensures a reliable and scalable backend, while Express.js and NodeMailer facilitate smooth email verification services.

The Online Parking System contributes to improved traffic flow, reduced congestion, and enhanced urban mobility. By leveraging modern web technologies, it provides a user-friendly, secure, and scalable parking management solution, making it a vital tool for smart city initiatives

8. REFERENCES

1. Spring Boot Documentation
   URL: https://spring.io/projects/spring-boot
2. React.js Documentation
   URL: https://reactjs.org/docs/getting-started.html
3. Spring Security Documentation
   URL: https://docs.spring.io/spring-security/reference/index.html
4. JWT Authentication Guide
   URL: https://jwt.io/introduction
5. Google Maps API Documentation
   URL: https://developers.google.com/maps/documentation
6. MySQL Documentation
   URL: https://dev.mysql.com/doc/
7. Hibernate ORM Documentation
   URL: https://hibernate.org/orm/documentation/
8. Node.js and Express.js Documentation
   URL: https://expressjs.com/
9. Node-Mailer Documentation
   URL: https://nodemailer.com/about/
10. Axios Documentation for API Integration
    URL: https://axios-http.com/docs/intro