CERTIFICATE

This is to certify that the project entitled

“Pay and Park”

Has been satisfactorily completed by

1) Devam Shah

2) Sudhanshu Khanduri

3) Anurag Kharche

4) Prajkta Ghadge

5) Akshay Fegade

Under supervision and guidance for partial fulfilment of the

PG Diploma in Advanced Computing (DAC) (24 weeks Full Time) Course

of

Centre for Development of Advanced Computing (C-DAC), Pune.

at

Academy of Information Technology

(YCP) Nariman Point, Mumbai –400 021.

Faculty **Course Co-ordinator**

Batch: PG DAC March 2024 Batch
Date: August 2024.

PREFACE

In today's fast-paced urban environments, finding a parking space is often a time-consuming and stressful experience. As cities grow and the number of vehicles on the road increases, the demand for convenient and accessible parking solutions becomes more critical. The traditional methods of searching for parking spaces are inefficient, leading to wasted time, increased traffic congestion, and unnecessary fuel consumption. Recognizing the need for an innovative solution, this project, titled "Pay and Park," aims to revolutionize the way drivers reserve, book, and pay for parking spaces, offering a seamless and efficient experience for users.

"Pay and Park" is a web application designed to provide users with real-time access to available parking slots at their desired destinations. The application leverages modern web technologies to enable users to instantly reserve and pay for parking spaces while on the go, reducing the hassle of searching for parking and ensuring a guaranteed spot upon arrival. By integrating real-time data and user-friendly interfaces, the application caters to both drivers seeking convenience and parking lot owners looking to optimize the utilization of their spaces.

This project is the culmination of extensive research and development focused on understanding the challenges faced by urban drivers and parking facility managers. The application not only simplifies the parking process but also contributes to reducing traffic congestion and lowering carbon emissions by minimizing the time spent driving in search of parking. The system's architecture is built with scalability in mind, ensuring that it can accommodate the growing demand for parking solutions in cities worldwide.

The development of "Pay and Park" involved the collaboration of five PG-DAC students, to create a solution that is both technically robust and user-centric. This report documents the entire development process, from the initial concept and design phase to the implementation and testing stages. It also explores the potential impact of this application on urban mobility and the ways in which it can be expanded and improved in the future.

ACKNOWLEDGMENT

We extend our heartfelt gratitude to the countless individuals and teams who have contributed to the development and success of the “Pay and Park Project. Our dedicated team of developers, designers, and educators has poured countless hours of effort, creativity, and expertise into creating a platform that redefines the parking experience. Their unwavering commitment to excellence has been instrumental in bringing our vision to life.

We are also immensely grateful to our users for their invaluable feedback and contributions, which have played a crucial role in shaping and improving Pay and Park. Their engagement and enthusiasm have inspired us to continuously innovate and refine our platform to better meet their needs.

Additionally, we would like to acknowledge the support and collaboration of our partners and collaborators. Their guidance, resources, and expertise have been invaluable in the development and promotion of Pay and Park. Together, we have worked tirelessly to create a platform that not only simplifies parking but also fosters a sense of community and collaboration among users.

As we look to the future, we are excited about the possibilities that lie ahead for Pay and Park. We remain committed to our mission of revolutionizing the parking management system. Thank you to everyone who has been a part of this incredible journey. Your dedication and support have been instrumental in making Pay and Park areality, and we are deeply grateful for your contributions.

Last but not least, we would like to express our deepest gratitude to our families and friends for their unwavering support, patience, and understanding during the course of this project. Your love, encouragement, and belief in our abilities have been a constant source of strength and inspiration.

To everyone who has played a role, big or small, in making this project a reality – thank you from the bottom of our hearts.

Team Members;

1) Devam Shah

2) Sudhanshu Khanduri

3) Anurag Kharche

4) Prajkta Ghadge

5) Akshay Fegade

INDEX

INTRODUCTION

Urbanization has intensified the challenge of finding parking in crowded cities, often leading to frustration and increased traffic. The "Pay and Park" web application addresses this by offering a solution that allows users to reserve, book, and pay for parking slots in real-time, streamlining the parking experience. This application aims to reduce the time spent searching for parking, easing the burden on drivers and contributing to smoother urban traffic flow.

"Pay and Park" is designed to be user-friendly, providing real-time information on available parking slots near a user's destination. Users can make instant reservations and complete secure payments directly through the application, ensuring a convenient and stress-free parking experience. This system not only benefits drivers but also enhances parking facility management by optimizing the use of available spaces.

The technical development of "Pay and Park" leverages Java Spring Boot for a robust backend and MySQL for efficient database management. The frontend is crafted using modern web technologies to ensure a seamless experience across devices. Real-time updates and notifications are integral features, keeping users informed of their parking status and improving overall system responsiveness.

This report outlines the motivation behind "Pay and Park," the technical framework used, and the challenges faced during development. It also discusses the application's potential impact on urban mobility, particularly in reducing traffic congestion and improving the efficiency of urban transportation. Through this project, we aim to demonstrate how innovative parking solutions can contribute to more organized and stress-free urban environments.

Software Requirements :

• Eclipse IDE

• MySQL

• Browser

• Spring Boot Tool suite

Hardware Requirements :

• PC / Laptop

• i3 Processor minimum

• 4 GB Ram minimum

• PC / Laptop

Tools Requirements :

• Postman

• Git

• GitHub

ER Diagram

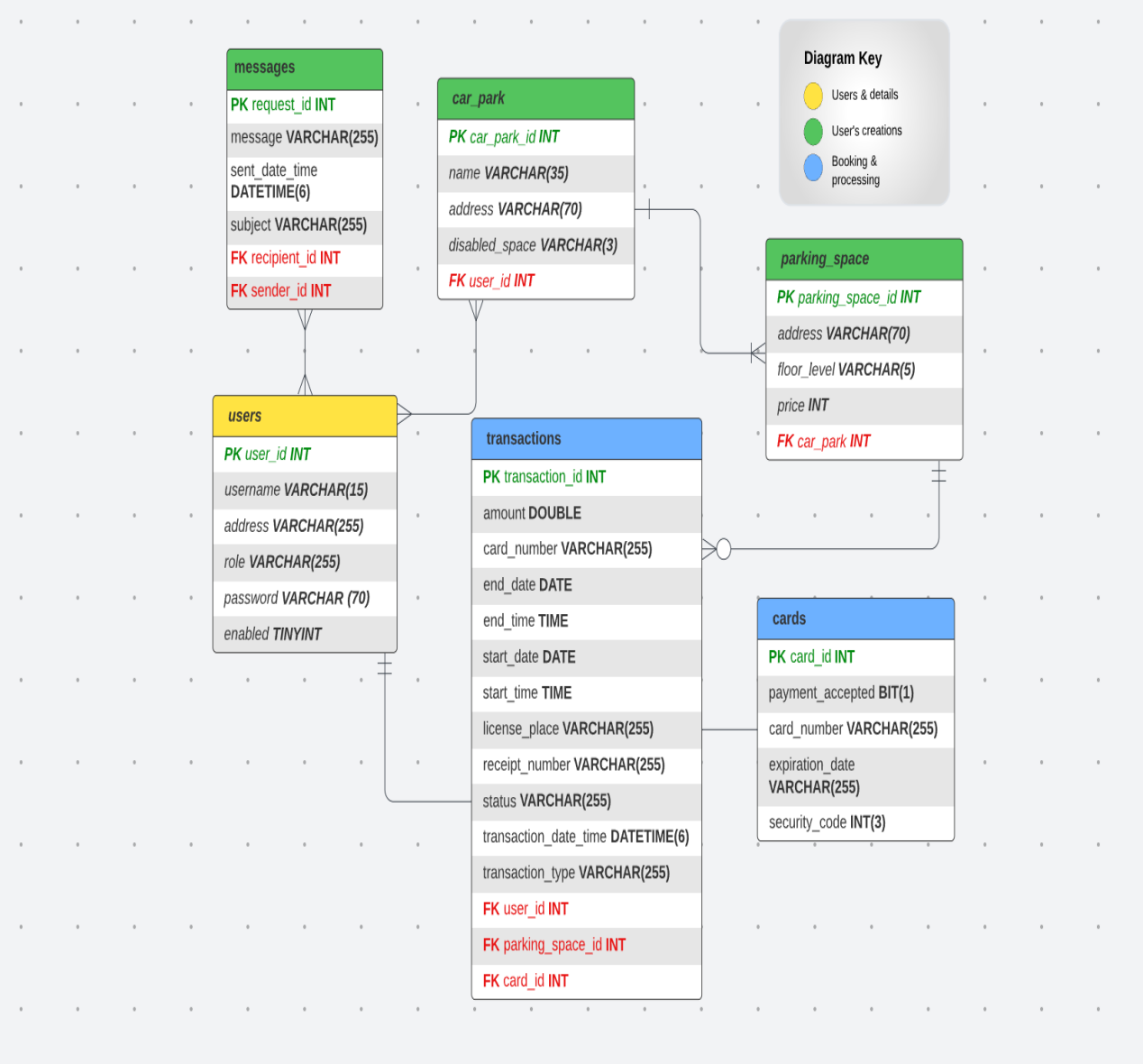


Diagram 1- ER Diagram

Data Flow Diagram

Diagram 2- Data Flow Diagram

Use case Diagram

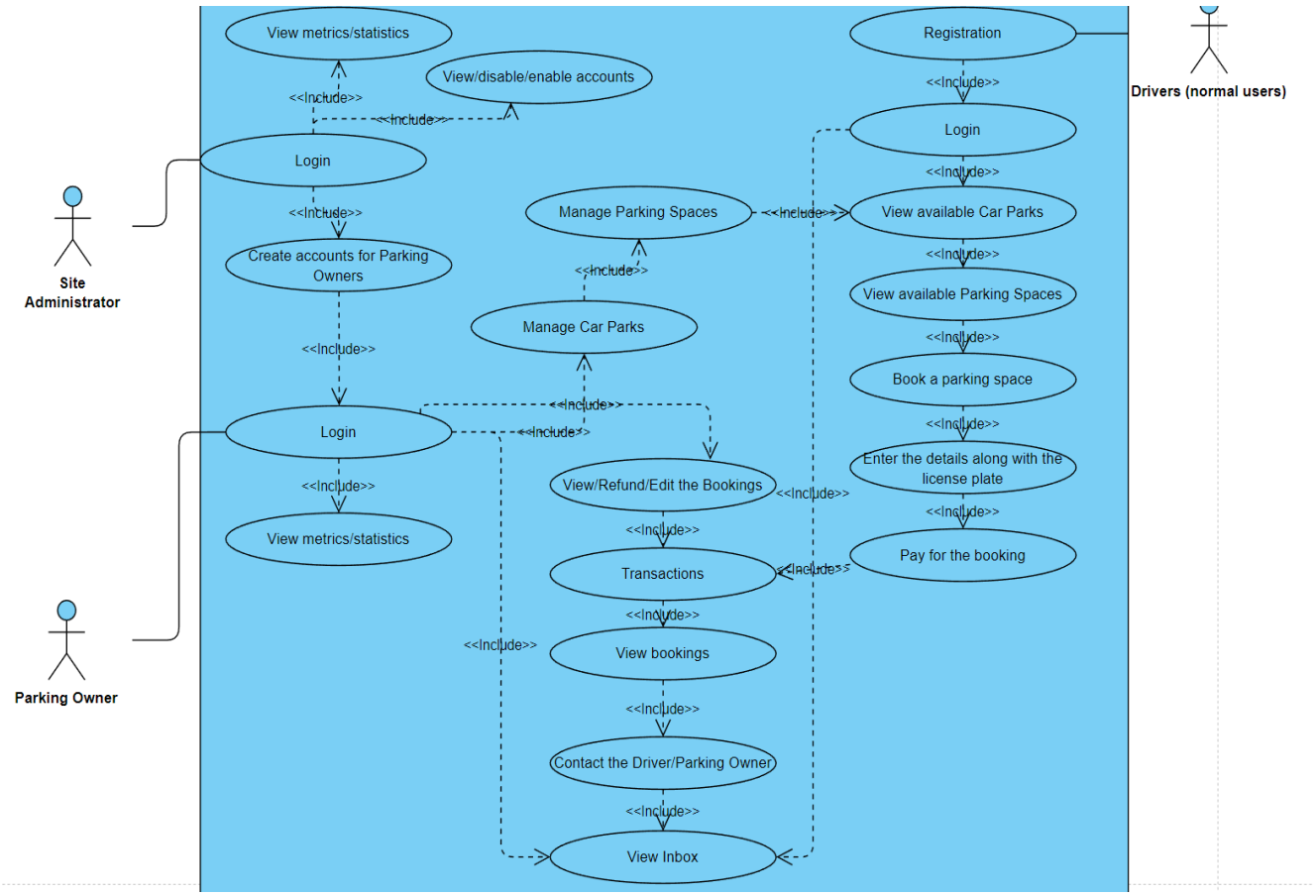


Diagram 3- Use Case Diagram

BACKEND DETAILS

What technology used?

• Spring Boot (Framework)

• MySQL (Database)

Dependencies:

• Spring web

• Spring-data-Jpa

• MySQL-connector

• Spring-security-core

End Points:

• Site Admin Login Functionality

• Driver Login Functionality

• Parking Owner Login Functionality

• Booking Functionality

• Transaction functionality

FRONTEND DETAILS

HTML (Hypertext Markup Language)

HTML is the standard markup language used to create and structure content on the web, such as text, images, and links. It provides the foundational structure of a webpage, defining elements like headings, paragraphs, lists, and multimedia.

CSS (Cascading Style Sheets)

CSS is used to control the presentation and layout of HTML elements, allowing developers to style and position content on a webpage. It enables customization of colors, fonts, spacing, and other visual aspects, creating a visually appealing and responsive design.

JavaScript

JavaScript is a powerful scripting language that enables dynamic content, interactivity, and complex functionality on websites. It allows developers to manipulate the DOM (Document Object Model), handle events, and perform tasks like form validation, animations, and asynchronous data loading.

Bootstrap

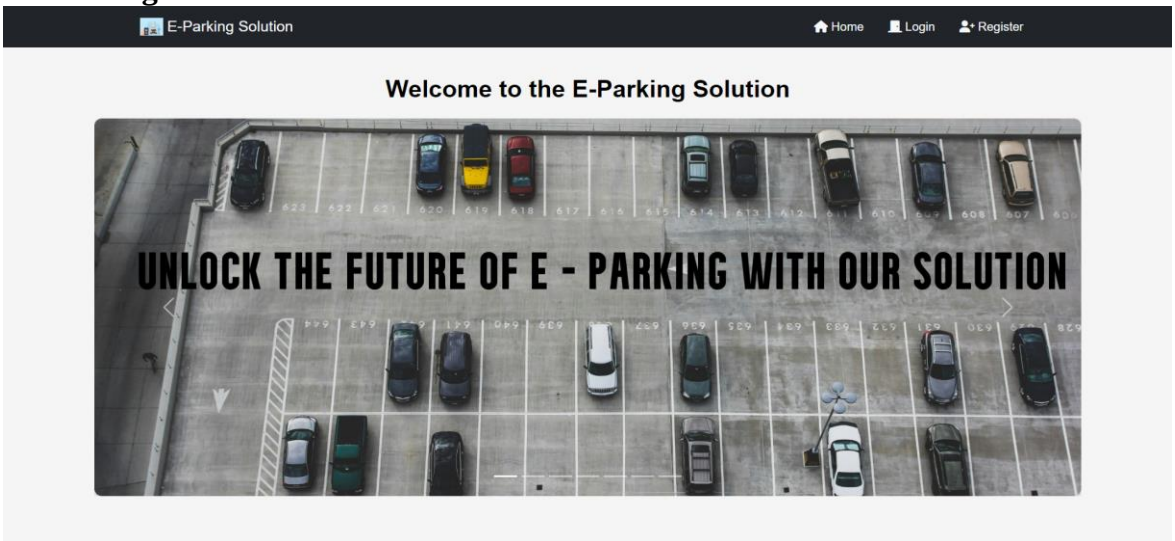
Bootstrap is a popular front-end framework that provides a collection of pre-designed components, grid systems, and utilities for building responsive and mobile-first websites. It simplifies the process of creating consistent and modern-looking web pages with minimal custom CSS and JavaScript.

jQuery

jQuery is a fast, lightweight JavaScript library that simplifies DOM manipulation, event handling, and AJAX interactions. It provides an easy-to-use API that works across different browsers, allowing developers to write less code while achieving more complex functionalities.

Interface :

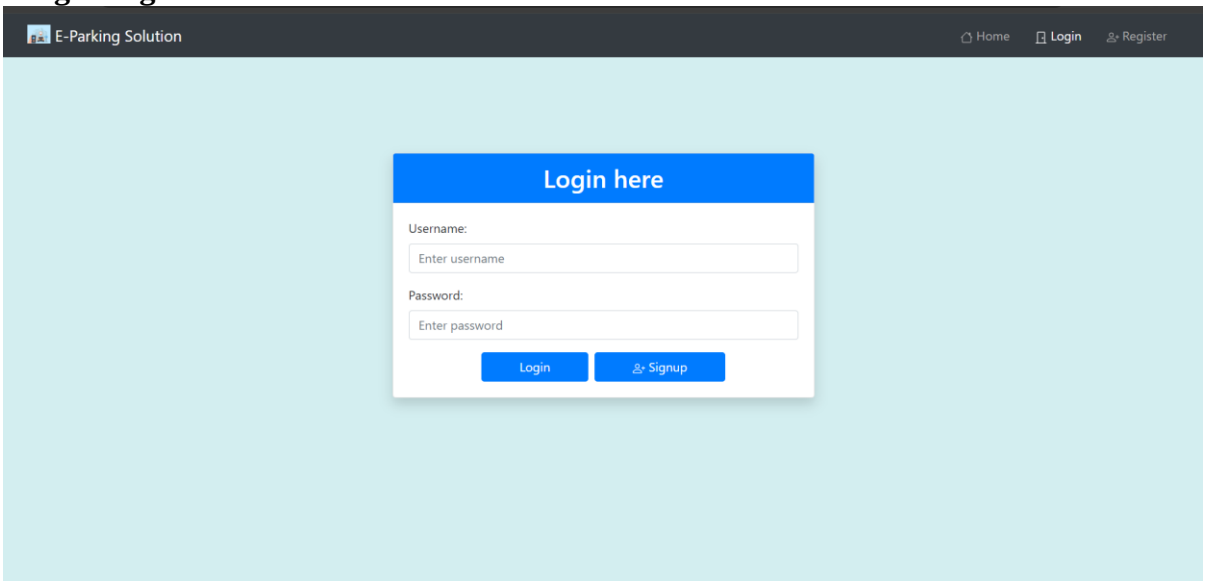
Home Page :



Screenshot-1

This is our Home Page you can see there is option for selection for Home, Login and Register. When you click on given option on navbar you can redirect to the Particular page.

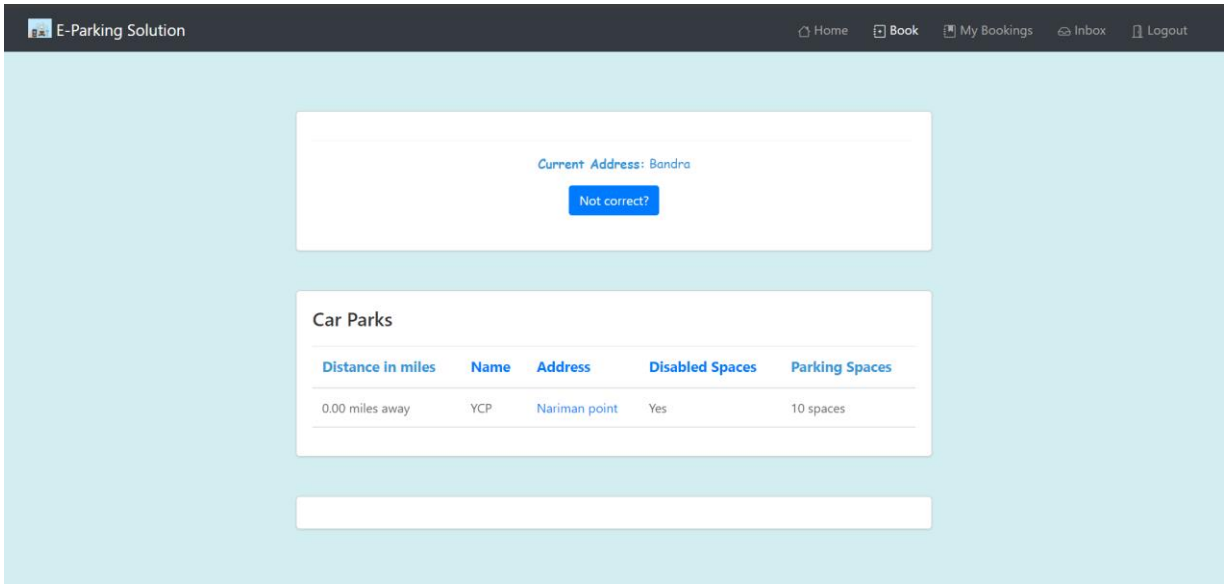
Login Page :



Screenshot-2

This is user Login page, if user already has an account then he/she can simply Login otherwise he/she can create a new account by clicking on Signup button which will directly redirect to registration page"

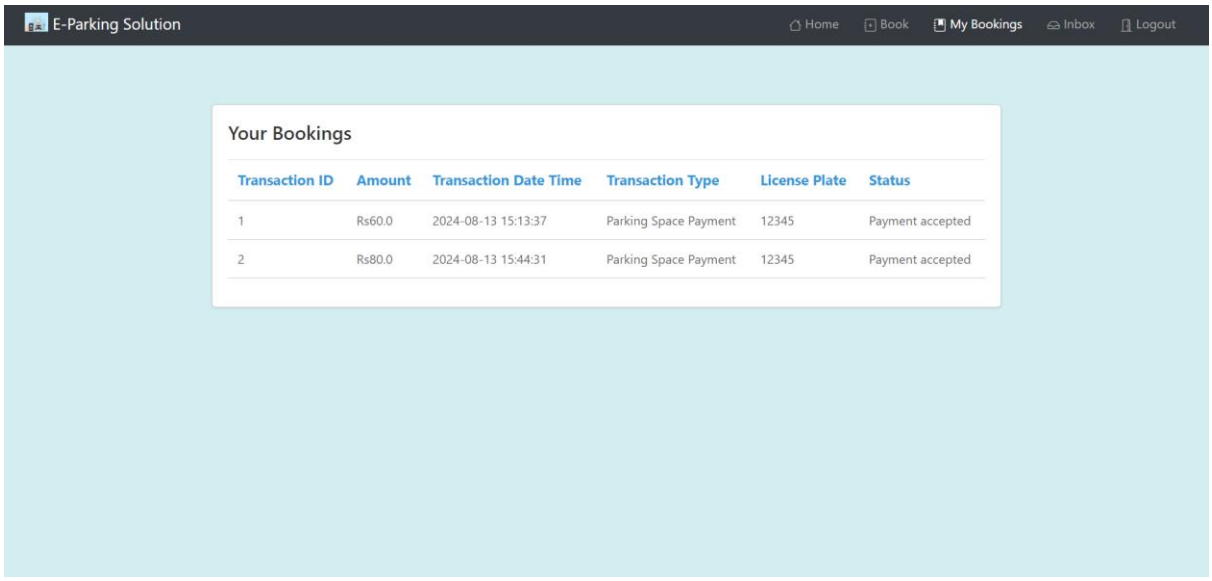
Parking Space Page :



Screenshot-3

This is the Parking-space page on which user can see parking space. Further by clicking on his/her desired parking space he/she will be redirect to page where they can book their desired parking spot.

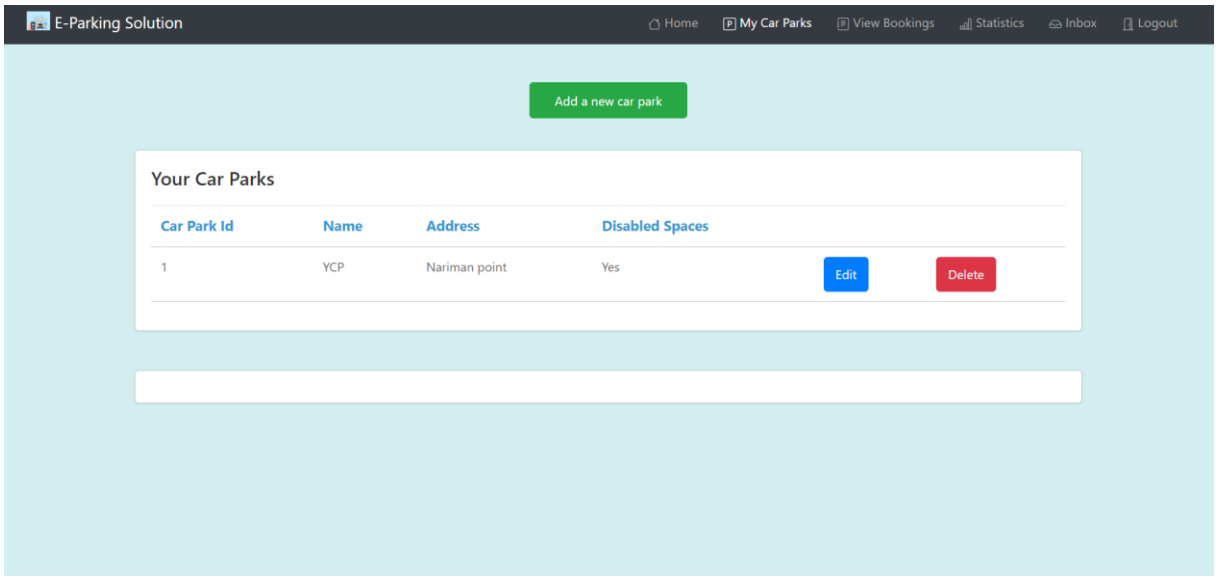
My Bookings Page :



Screenshot-4

This is the My Bookings page where users see the history of their all bookings.

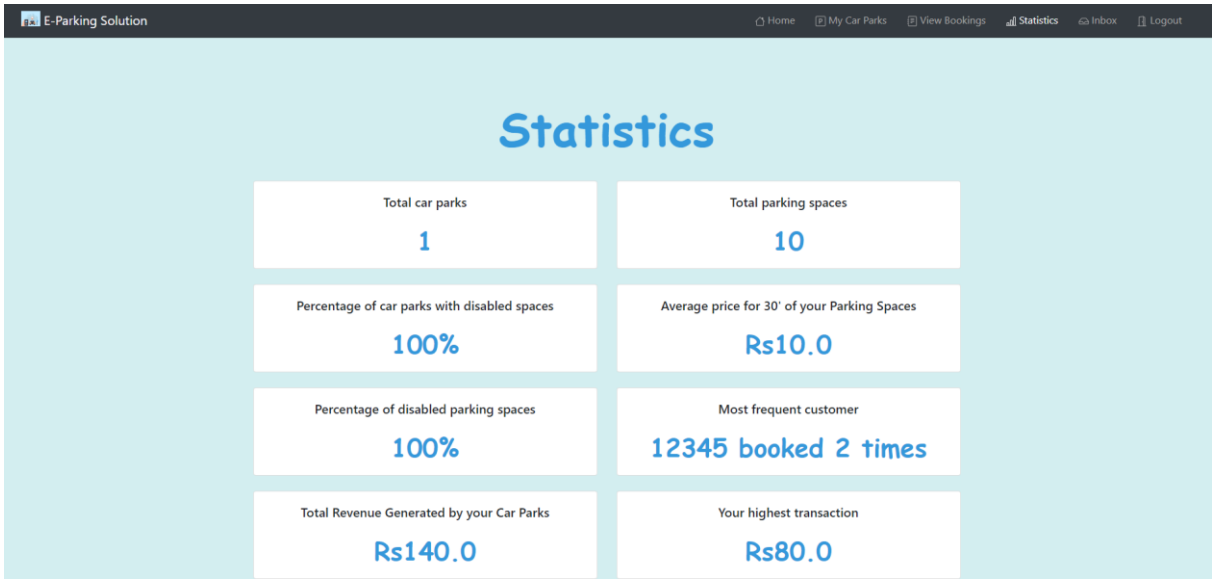
Manage Car Parks Page :



Screenshot-5

This is the page where parking owner can create new car parks, as well as edit the existing car parks, also delete the existing car park.

Parking Owner Statistic Page :



Screenshot-6

This is the statistics page for the parking owner where he can see all the insights for his parking spaces.

Coding :

Functions:

The Pay and Park is built on the Model-View-Controller (MVC) architectural pattern, which provides a clear separation of concerns and promotes modularity, maintainability, and scalability. The system architecture consists of three interconnected layers:

1. Model:

➢ The Model layer represents the application's data and business logic.

➢ It includes entities such as User, Car Park, Parking Space, Transaction, Card, Message encapsulating the data-related operations.

➢Business logic related to User validation rules, Booking parking spaces , Managing transactions etc. processing workflows is implemented in this layer.

2. View:

➢The View layer is responsible for presenting the user interface to the end-user.

➢It includes HTML templates, CSS stylesheets, and JavaScript code for renderingthe UI components.

➢User interfaces such as the home page, login page, user page, and bookings are part ofthe View layer.

3. Controller:

➢The Controller layer acts as an intermediary between the Model and View layers, handling user requests and orchestrating the flow of data.

➢It contains controller classes that receive incoming HTTP requests, process them, invoke appropriate business logic in the Model layer, and select the appropriate view to render the response.

➢Controllers manage the navigation logic, form submissions, and interaction between the user interface and the backend system.

Main Component of Backend:

Entry Point :

```
package com.eparkingsolution;

import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import com.eparkingsolution.model.Card;
import com.eparkingsolution.model.User;
import com.eparkingsolution.repository.CardRepository;
import com.eparkingsolution.repository.UserRepository;

@SpringBootApplication
public class EParkingSolutionApplication implements ApplicationRunner {
   private final CardRepository cardRepository;
   private final UserRepository userRepository;

   public EParkingSolutionApplication(CardRepository cardRepository,
                              UserRepository userRepository) {
      this.cardRepository = cardRepository;
      this.userRepository = userRepository;
   }

   public static void main(String[] args) {
      SpringApplication.run(EParkingSolutionApplication.class, args);
   }

   @Override
   public void run(ApplicationArguments args) {

      Card card1 = new Card();
      card1.setId(1L);
      card1.setCardNumber("4242 4242 4242 4242");
      card1.setExpirationDate("24 24");
      card1.setSecurityCode("123");
      card1.setAccepted(true);
      cardRepository.save(card1);

      Card card2 = new Card();
```

```
        card2.setId(2L);
        card2.setCardNumber("2424 2424 2424 2424");
        card2.setExpirationDate("24 24");
        card2.setSecurityCode("123");
        card2.setAccepted(false);
        cardRepository.save(card2);

        User driver = new User();
        driver.setId(1);
        driver.setUsername("driver");
        String password = "driver1";
        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        String hashedPassword = passwordEncoder.encode(password);
        driver.setPassword(hashedPassword);
        driver.setAddress("Bandra");
        driver.setRole("Driver");
        userRepository.save(driver);

        User parkingOwner = new User();
        parkingOwner.setId(2);
        parkingOwner.setUsername("Parking Owner");
        String password2 = "po1";
        String hashedPassword2 = passwordEncoder.encode(password2);
        parkingOwner.setPassword(hashedPassword2);
        parkingOwner.setAddress("Marine Drive");
        parkingOwner.setRole("Parking Owner");
        userRepository.save(parkingOwner);

        User siteAdministrator = new User();
        siteAdministrator.setId(3);
        siteAdministrator.setUsername("Site Admin");
        String password3 = "sa1";
        String hashedPassword3 = passwordEncoder.encode(password3);
        siteAdministrator.setPassword(hashedPassword3);
        siteAdministrator.setAddress("YCP Mumbai");
        siteAdministrator.setRole("Site Administrator");
        userRepository.save(siteAdministrator);

    }
}
```

Configurations :

```
#Database connection
spring.datasource.url=jdbc:mysql://localhost:3306/eParking
spring.datasource.username=root
spring.datasource.password=Devam@2702@@
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.main.allow-bean-definition-overriding=true

server.port=2702
```

Index Controller:

```
package com.eparkingsolution.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class IndexController {

   @RequestMapping("/")
   public String cons(){

      return "indexLandingPage";
   }

   @RequestMapping("indexDriver")
   public String indexDriver(){
      return "indexDriver";
   }

   @RequestMapping("indexParkingOwner")
   public String indexParkingOwner(){
      return "indexParkingOwner";
   }

   @RequestMapping("indexSiteAdministrator")
   public String indexSiteAdministrator(){
      return "indexSiteAdministrator";
   }
}
```

Register Controller :

```
package com.eparkingsolution.controller;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.eparkingsolution.model.User;
import com.eparkingsolution.service.UserDetailsServiceImpl;

@Controller
```

```
public class RegisterController {

    @Autowired
    private UserDetailsServiceImpl userService;



    @GetMapping("/register")
    public String showRegistrationForm(Model model) {
        model.addAttribute("user", new User());
        return "register";
    }

    @PostMapping("/register")
    public String processRegistrationForm(@ModelAttribute User user, Model model, RedirectAttributes
redirectAttributes) {
        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        String encodedPass = passwordEncoder.encode(user.getPassword());

        if(user.getUsername().isEmpty()){
            model.addAttribute("emptyUser","Email can't be empty");
            return "register";
        }
        if (userService.isUsernameTaken(user.getUsername())) {
            model.addAttribute("errorTaken", "Username already exists");
            return "register";
        }
        if(user.getPassword().isEmpty()){
            model.addAttribute("emptyPassword","Password can't be empty");
            return "register";
        }

        userService.registerUser(user.getUsername(),
passwordEncoder.encode(user.getPassword()),user.getAddress());
        redirectAttributes.addFlashAttribute("messageSent", "Registered successfully!");

        return "redirect:/login";
    }


}
```

Login Controller :

```
package com.eparkingsolution.controller;

importorg.springframework.stereotype.Control

ler;

import

org.springframework.web.bind.annotation.Req

uestMapping;

@Controller

public class LoginController {

    @RequestMapping("/login")

    public String loginPage(){

        return "login";

    }

}
```

Booking Controller:

```
package com.eparkingsolution.controller;RE

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.format.annotation.DateTimeFormat;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.eparkingsolution.model.*;
import com.eparkingsolution.repository.CardRepository;
import com.eparkingsolution.repository.ParkingSpaceRepository;
import com.eparkingsolution.repository.TransactionRepository;
import com.eparkingsolution.repository.UserRepository;
import com.eparkingsolution.service.CarParkService;
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;


import java.security.Principal;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.format.DateTimeFormatter;
import java.time.temporal.ChronoUnit;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

@Controller
public class BookingController {
```

```
@Autowired
private CarParkService carParkService;

@Autowired
private UserRepository userRepository;

@Autowired
private ParkingSpaceRepository parkingSpaceRepository;

@Autowired
private TransactionRepository transactionRepository;

@Autowired
private CardRepository cardRepository;



@GetMapping("/booking")
public String viewHomePage(Model model, Principal principal) {
   String username = principal.getName();
   User user = userRepository.findByUsername(username);
   model.addAttribute("user", user);
   model.addAttribute("address", user.getAddress());

   return findPaginated1(1, "name", "asc", model, principal);
}

@GetMapping("/page/{pageNo}")
public String findPaginated1(@PathVariable(value = "pageNo") int pageNo,
                        @RequestParam("sortField") String sortField,
                        @RequestParam("sortDir") String sortDir,
                        Model model,
                        Principal principal) {
   int pageSize = 5;

   Page<CarPark> page = carParkService.findPaginated(pageNo, pageSize, sortField, sortDir);

   List<CarPark> carPark = page.getContent();

   String username = principal.getName();
```

```
        User user = userRepository.findByUsername(username);



        model.addAttribute("user", user);
        model.addAttribute("address", user.getAddress());
        model.addAttribute("currentPage", pageNo);
        model.addAttribute("totalPages", page.getTotalPages());
        model.addAttribute("totalItems", page.getTotalElements());

        model.addAttribute("sortField", sortField);
        model.addAttribute("sortDir", sortDir);
        model.addAttribute("reverseSortDir", sortDir.equals("asc") ? "desc" : "asc");

        model.addAttribute("carPark", carPark);
        return "booking";
    }

    // handle the form submission to update the user's address
    @PostMapping("/update-address")
    public String updateAddress(Principal principal, @RequestParam("address") String address,
RedirectAttributes redirectAttributes) {
        User existingUser = userRepository.findByUsername(principal.getName());
        existingUser.setAddress(address);
        userRepository.save(existingUser);
        redirectAttributes.addFlashAttribute("message", "Address updated successfully");
        return "redirect:/booking";
    }



    @GetMapping("/booking/{id}")
    public String listObjectsByForeignKey(@PathVariable("id") int id,
                                @RequestParam(value = "pageNo", defaultValue = "1") int pageNo,
                                @RequestParam(value = "pageSize", defaultValue = "5") int pageSize,
                                @RequestParam(value = "sortField", defaultValue = "id") String sortField,
                                @RequestParam(value = "sortDir", defaultValue = "asc") String sortDir,
                                Model model) {
        Pageable pageable = PageRequest.of(pageNo - 1, pageSize, Sort.Direction.fromString(sortDir),
sortField);
        Page<ParkingSpace> page = parkingSpaceRepository.findByCarParkId(id, pageable);

        model.addAttribute("listparkingspace", page.getContent());
```

```
        model.addAttribute("currentPage", pageNo);
        model.addAttribute("totalPages", page.getTotalPages());
        model.addAttribute("totalItems", page.getTotalElements());
        model.addAttribute("sortField", sortField);
        model.addAttribute("sortDir", sortDir);
        model.addAttribute("reverseSortDir", sortDir.equals("asc") ? "desc" : "asc");
        model.addAttribute("carParkId", id);

        return "bookingPS";

    }



        @GetMapping("/booking/parking-space/{id_ps}")
    public String getParkingSpaceById(@PathVariable("id_ps") long id_ps, Model model) {
        ParkingSpace parkingSpace = parkingSpaceRepository.findById(id_ps).orElse(null);
        List<Transaction> transactions = transactionRepository.findByParkingSpace(parkingSpace);
        model.addAttribute("parkingSpace", parkingSpace);
        if (transactions.isEmpty()) {
            model.addAttribute("message", "No bookings yet for this parking space.");
        } else {
            model.addAttribute("transactions", transactions);
        }
        return "displayPS";
    }

    //    Takes input form the User and calculates the total bill to be paid
    // Check if the parking space is available during the selected time frame
    @PostMapping("/booking/parking-space/{id_ps}")
    public String bookParkingSpace(@PathVariable("id_ps") long id_ps,
                                   @RequestParam("startDate") @DateTimeFormat(pattern = "yyyy-MM-dd")
LocalDate startDate,
                                   @RequestParam("startTime") @DateTimeFormat(pattern = "HH:mm") LocalTime
startTime,
                                   @RequestParam("endDate") @DateTimeFormat(pattern = "yyyy-MM-dd")
LocalDate endDate,
                                   @RequestParam("endTime") @DateTimeFormat(pattern = "HH:mm") LocalTime
endTime,
                                   @RequestParam("licensePlate") String licensePlate,
                                   @ModelAttribute Transaction transaction,
```

```
                        Model model) {
    ParkingSpace parkingSpace = parkingSpaceRepository.findById(id_ps).orElse(null);
    LocalDateTime start = LocalDateTime.of(startDate, startTime);
    LocalDateTime end = LocalDateTime.of(endDate, endTime);

    List<Transaction> existingTransactions = transactionRepository.findByParkingSpace(parkingSpace);

    LocalDateTime newStart = LocalDateTime.of(startDate, startTime);
    LocalDateTime newEnd = LocalDateTime.of(endDate, endTime);

    for (Transaction existingTransaction : existingTransactions) {
        LocalDateTime existingStart =
existingTransaction.getStartDate().atTime(existingTransaction.getStartTime());
        LocalDateTime existingEnd =
existingTransaction.getEndDate().atTime(existingTransaction.getEndTime());

        if (newStart.isAfter(existingStart) && newStart.isBefore(existingEnd) ||
            newEnd.isAfter(existingStart) && newEnd.isBefore(existingEnd) ||
            newStart.isBefore(existingStart) && newEnd.isAfter(existingEnd)) {
          model.addAttribute("errorMessage", "Sorry, the parking space is already booked for the selected
time. Please choose a different time.");
          return "doubleBookingError";
        }
    }


    // handle the booking
    double pricePer30Minutes = parkingSpace.getPrice();
    long minutes = ChronoUnit.MINUTES.between(start, end);
    double totalCost = (minutes / 30) * pricePer30Minutes;
    totalCost = Math.round(totalCost * 100.0) / 100.0;
    model.addAttribute("totalCost", totalCost);
    model.addAttribute("startDate", startDate);
    model.addAttribute("startTime", startTime);
    model.addAttribute("endDate", endDate);
    model.addAttribute("endTime", endTime);
    model.addAttribute("transaction", transaction);
    model.addAttribute("licensePlate", licensePlate);
    return "bookingCheckout";
  }
```

```
    //    Process the payment and display the result
    @PostMapping("/payment/{id_ps}")
    public String processPayment(@RequestParam("cardNumber") String cardNumber,
                                 @PathVariable("id_ps") long id_ps,
                                 @ModelAttribute("startDate") String startDate,
                                 @ModelAttribute("endDate") String endDate,
                                 @ModelAttribute("startTime") String startTime,
                                 @ModelAttribute("endTime") String endTime,
                                 @ModelAttribute("totalCost") double totalCost,
                                 @ModelAttribute("licensePlate") String licensePlate,
                                 Authentication authentication,
                                 Model model) {
        Card card = cardRepository.findByCardNumber(cardNumber);
        ParkingSpace parkingSpace = parkingSpaceRepository.findById(id_ps).orElse(null);
        User user = userRepository.findByUsername(authentication.getName());
        if (card != null && card.isAccepted()) {
            Transaction transaction = new Transaction();
            transaction.setStartDate(LocalDate.parse(startDate));
            transaction.setEndDate(LocalDate.parse(endDate));
            transaction.setStartTime(LocalTime.parse(startTime));
            transaction.setEndTime(LocalTime.parse(endTime));
            transaction.setAmount(totalCost);
            transaction.setTransactionType("Parking Space Payment");
            transaction.setLicensePlate(licensePlate);
            transaction.setCard(card);
            transaction.setStatus("Payment accepted");

            // Create a DateTimeFormatter object with the corrected format
            DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

// Get the current date and time
            LocalDateTime transactionDateTime = LocalDateTime.now();

// Format the transactionDateTime using the formatter
            String formattedTransactionDateTime = transactionDateTime.format(formatter);

// Set the formatted transactionDateTime to the transaction object
            transaction.setTransactionDateTime(formattedTransactionDateTime);

            transaction.setCardNumber(cardNumber);
```

```
            transaction.setUser(user);
            double randomNumber = Math.random() * 100000000;
            int receiptNumber = (int) randomNumber;
            transaction.setReceiptNumber(String.valueOf(receiptNumber));

            transaction.setParkingSpace(parkingSpace);

            // save the transaction to the database
            transactionRepository.save(transaction);
            model.addAttribute("transaction", transaction);
            return "displayReceipt";
        } else {
            model.addAttribute("errorMessage", "The payment could not be processed. Please check the card number and try again.");
            return "paymentError";
        }
    }

}
```

RESTful API:

```
package com.eparkingsolution.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

import com.eparkingsolution.model.User;
import com.eparkingsolution.repository.UserRepository;

import java.util.*;

@RestController
@RequestMapping("/api/users")
public class RESTfulAPI {

    @Autowired
    private UserRepository userRepository;

    @PostMapping
    public User createUser(@RequestBody User user) {
        user.setRole("Parking Owner");
        return userRepository.save(user);
    }


    @GetMapping("/viewAll")
    public List<Map<String, Object>> getUsers() {
        List<User> users = userRepository.findAll();
        List<Map<String, Object>> result = new ArrayList<>();
        for (User user : users) {
            Map<String, Object> userMap = new LinkedHashMap<>();
            userMap.put("id", user.getId());
            userMap.put("username", user.getUsername());
            userMap.put("password", user.getPassword());
            userMap.put("address", user.getAddress());
            userMap.put("role", user.getRole());
            userMap.put("enabled", user.isEnabled());
            result.add(userMap);
        }
        return result;
    }
```

```
@PutMapping("/enable/{id}")
public User enableUser(@PathVariable Integer id) {
    User user = userRepository.findById(id)
            .orElseThrow(() -> new UserNotFoundException(id));
    user.setEnabled(true);
    return userRepository.save(user);
}

@PutMapping("/disable/{id}")
public User disableUser(@PathVariable Integer id) {
    User user = userRepository.findById(id)
            .orElseThrow(() -> new UserNotFoundException(id));
    user.setEnabled(false);
    return userRepository.save(user);
}




@ResponseStatus(HttpStatus.NOT_FOUND)
public class UserNotFoundException extends RuntimeException {

    public UserNotFoundException(Integer id) {
       super("User not found with ID " + id);
    }
}


@GetMapping("/{id}")
public User getUserById(@PathVariable Integer id) {
    return userRepository.findById(id).orElseThrow(() -> new UserNotFoundException(id));
}

@PutMapping("/{id}")
public User updateUser(@PathVariable Integer id, @RequestBody User user) {
    User existingUser = userRepository.findById(id).orElseThrow(() -> new
UserNotFoundException(id));
    existingUser.setUsername(user.getUsername());
    existingUser.setPassword(user.getPassword());
    existingUser.setAddress(user.getAddress());
    existingUser.setRole(user.getRole());
    return userRepository.save(existingUser);
}

@DeleteMapping("/{id}")
public void deleteUser(@PathVariable Integer id) {
```

```
        userRepository.deleteById(id);
    }
}
```

FUTURE SCOPE

" The "Pay and Park" application has significant potential for growth and evolution, offering opportunities to enhance its features and expand its reach. One key area of future development is the integration of advanced technologies such as machine learning and artificial intelligence to predict parking availability based on historical data and real-time conditions. This could further optimize the user experience by providing even more accurate and personalized parking suggestions."

CONCLUSION

The "Pay and Park" application successfully addresses the growing challenges of urban parking by offering a streamlined solution for reserving, booking, and paying for parking slots in real-time. Through its user-friendly interface and real-time updates, the application simplifies the parking process, reducing the stress and time associated with finding parking in crowded areas.

By leveraging modern web technologies and robust backend infrastructure, "Pay and Park" ensures a reliable and scalable platform that can meet the needs of both users and parking facility owners. The system not only improves the user experience but also optimizes the utilization of parking spaces, contributing to more efficient urban traffic management.

The project highlights the potential of integrating technology with urban infrastructure to create smarter, more efficient cities. As cities continue to grow and the demand for parking increases, applications like "Pay and Park" will play a crucial role in enhancing urban mobility and reducing congestion.

Looking ahead, the future scope of "Pay and Park" includes expanding its features and integrating with other smart city initiatives. With continuous development and adaptation to emerging technologies, the application has the potential to become a key component of sustainable urban transportation systems worldwide.

Bibliography:

We kept window open for further additional development.

- https://getbootstrap.com/
- https://spring.io/projects/spring-boot

https://www.w3schools.com/