# Design and Implementation of a Parking Management System for Vehicle Entry/Exit and Receipt Generation

Andrés Felipe Mateus Saavedra - 20201020119* Anderson Jefrey López Jiménez - 20162020424* Juan Esteban Oviedo Sandoval - 20192020064* *Software Engineering Seminar, Faculty of Engineering, Francisco José de Caldas District University
Emails: afmateuss@udistrital.edu.co, ajlopezj@udistrital.edu.co¡ jeoviedos@udistrital.edu.co

***Abstract*—Parking management is a daily challenge in urban and corporate spaces. Traditional methods, which rely on manual records, often result in errors, delays, and limited vehicle traceability. To address these challenges, we propose the development of a low-cost digital application that optimizes the vehicle entry and exit registration process. This solution improves internal organization, reduces human error, and facilitates vehicle flow monitoring, leading to more efficient management of available parking spaces. The system was implemented and tested, demonstrating its potential for enhancing parking lot management and reducing operational inefficiencies.**

***Index Terms*—Parking Management, Vehicle Registration, Receipt Generation, Operational Efficiency, Real-time Tracking, PostgreSQL, React, Flask, Docker, Performance Testing, System Design.**

## I. Introduction

Parking management is a significant challenge in urban and corporate spaces. Traditional methods, which often rely on manual records, can lead to inefficiencies, errors, and delays. These issues affect the accessibility and availability of parking spaces, leading to poor operational management. Furthermore, manually recording vehicle entries and exits makes it difficult to trace vehicles accurately, impacting security and billing.[1].

In response to this problem, we propose the development of a low-cost digital application designed to optimize the vehicle entry and exit registration process. The proposed solution is designed to be cost-effective, allowing smaller parking lots to benefit from technology without a large financial investment. By streamlining the process of vehicle registration and receipt generation, our system can improve operational efficiency, reduce human error, and provide real-time tracking of vehicles within the parking lot.

This paper describes the design and implementation of the proposed parking management system. We outline its architecture, the technology stack used, and the testing methods employed to evaluate the system's effectiveness. The results show that the system successfully automates parking management and reduces errors compared to traditional manual methods.

## II. Methods and Materials

### A. Design of the Solution

The Parking Management System was designed to automate and optimize the process of managing parking space availability, vehicle registration, and receipt generation. The solution aims to reduce human error, enhance operational efficiency, and offer a user-friendly interface for parking attendants.

The system follows a modular design approach, dividing it into several core components. The primary components include the frontend user interface (UI), the backend API (business logic), and the database. The frontend is responsible for interacting with parking attendants, allowing them to register vehicle entries and exits as well as generate receipts. The backend API processes the vehicle data, calculates parking fees, and communicates with the database to store or retrieve information. The database, based on PostgreSQL, stores data related to vehicles, parking slots, and receipts.

The technical decisions made during the design of the system were driven by the need for cost-effectiveness, scalability, and real-time data handling. The system was built to be low-cost, so that smaller parking lots could adopt it without significant financial investments. Additionally, scalability was considered in the design, allowing the system to handle increased usage if deployed in larger parking facilities. Real-time data handling ensures that parking attendants have up-to-date information on vehicle entry, exit, and parking space availability, contributing to improved operational control.[2]

### B. Technical Decisions

Several key technologies were chosen for the development of the system. React was selected for the frontend because of its flexibility and ability to manage dynamic content efficiently. React allows for the creation of an interactive user interface where parking attendants can easily register vehicles, view parking space availability, and generate receipts.

For the backend, Flask was chosen due to its lightweight nature, scalability, and excellent support for building REST APIs. Flask's simplicity made it an ideal choice to handle requests from the frontend, manage the business logic, and communicate with the database.

PostgreSQL was chosen as the database management system due to its robustness, ability to handle complex queries, and strong data integrity features. It was used to store vehicle data (such as license plates and entry/exit times) and receipt information (such as total parking duration and fees).

The CI/CD pipeline was implemented using GitHub Actions, automating the testing, building, and deployment of the system. Docker was used to containerize the system, ensuring consistency across development, testing, and production environments. The use of Docker Compose allowed for easy orchestration of the frontend, backend, and database containers, simplifying the deployment process.

### C. User Flow

The system's user flow is designed to be intuitive and efficient. When parking attendants log into the system, they are presented with the main interface. The primary tasks they will perform include registering vehicle entries, recording exits, and generating receipts.

Upon vehicle entry, the attendant enters the vehicle's license plate number into the system, which logs the entry time and updates the parking space availability in real time. When the vehicle exits, the attendant registers the exit, and the system calculates the total parking duration and generates a receipt. All vehicle and receipt data is stored in the PostgreSQL database, ensuring data consistency and reliability.

### D. System Analysis and Architecture Design

The system follows a client-server architecture. The client-side is the web-based interface that interacts with the parking attendants, and the server-side consists of the backend API and database. The frontend sends requests to the backend via REST APIs, which processes the data and returns the necessary information. The backend API interacts with the PostgreSQL database to store and retrieve vehicle and receipt data.

*1) Diagrams:* To enhance the understanding of the system's design and its components, several diagrams were created to visualize the system:

*a) Business Model Canvas::* The Business Model Canvas was used to map out the value proposition, customer segments, revenue streams, channels, and key activities of the Parking Management System. This diagram helped align the system's objectives with the overall business goals.

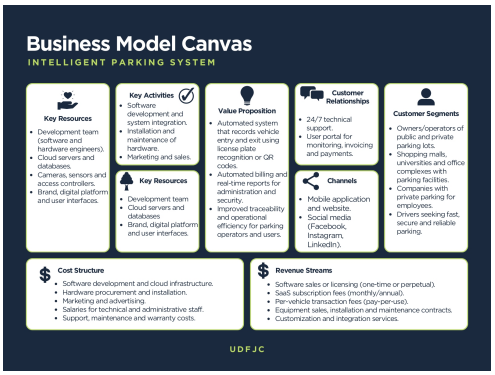


Fig. 1. Business Model Canvas for the Parking Management System

*b) User Story Mapping::* User Story Mapping was utilized to break down and prioritize the functionalities of the system from the perspective of parking attendants. It provided a visual representation of the system's features and their importance, helping to align development with user needs.

Fig. 2. User Story Mapping for the Parking Management System

*c) Class Diagram::* The Class Diagram illustrates the object-oriented structure of the system. It shows the relationships between primary classes such as Vehicle, Receipt, and ParkingSlot. These classes represent the main entities in the system and their interactions.

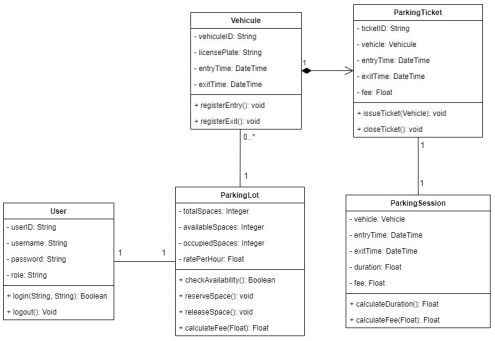


Fig. 3. Class Diagram for the Parking Management System

*d) System Architecture Diagram::* The System Architecture Diagram shows how the different components of the system interact with each other. It illustrates the flow of data between the frontend, backend, and database.

*e) Deployment Diagram::* The Deployment Diagram illustrates the physical deployment of the system across servers. It shows how the frontend, backend, and

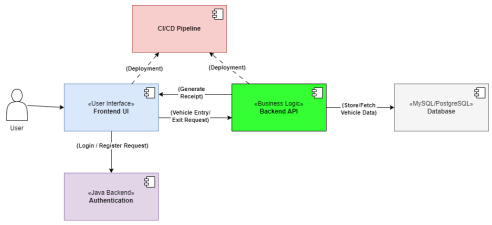


Fig. 4. System Architecture Diagram for the Parking Management System

database are hosted and interact within the infrastructure, whether on-premises or in the cloud.

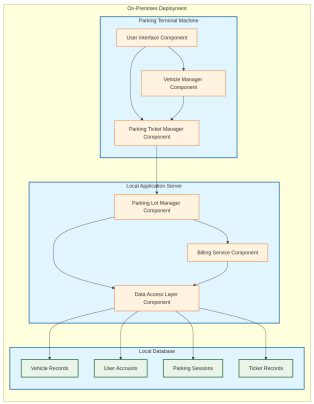


Fig. 5. Deployment Diagram for the Parking Management System

*f) Business Process Diagram::* The Business Process Diagram provides a visual representation of the steps involved in the parking management process, from vehicle entry to receipt generation. It helps visualize the flow of data and the interactions between the parking attendant and the system.

*g) Mockups::* Three mockups were created to illustrate the key pages of the system's frontend:

- **Vehicle Registration Page**: Allows attendants to input vehicle details and register entries or exits.
- **Login/Register Page User**: Validates access credentials to manage and register vehicles entering and leaving the parking lot..
- **Dashboard for Parking Space Availability**: Shows real-time availability of parking spaces.

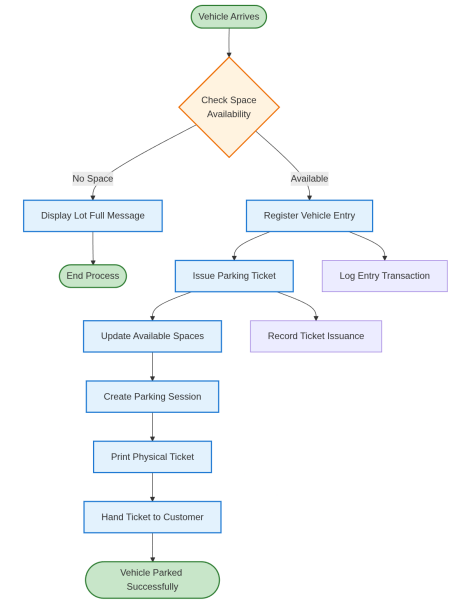


Fig. 6. Business Process Diagram for the Parking Management System

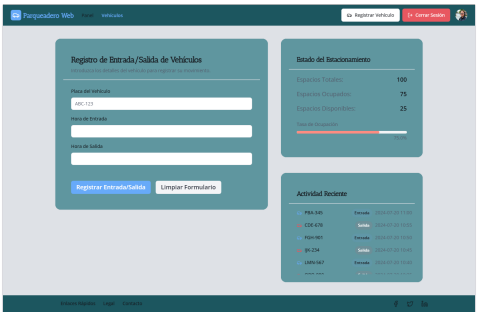


Fig. 7. Mockup of the Vehicle Registration Page

## *E. CRC Cards*

| **Class** | Vehicle |
|---|---|
| **Responsibilities** | **Collaborators** |
| Known its own license plate | |
| Known its own car type | Slot |
| Known its own brand | User |
| Known its user owner | Register |

| **Class** | User |
|---|---|
| **Responsibilities** | **Collaborators** |
| Known its own id | |
| Known its user type | Slot |
| Manage associated cars | Vehicle |
| Known its own brand | Register |
| Known its user owner | |


Fig. 8. Mockup of the Login/Register Page User

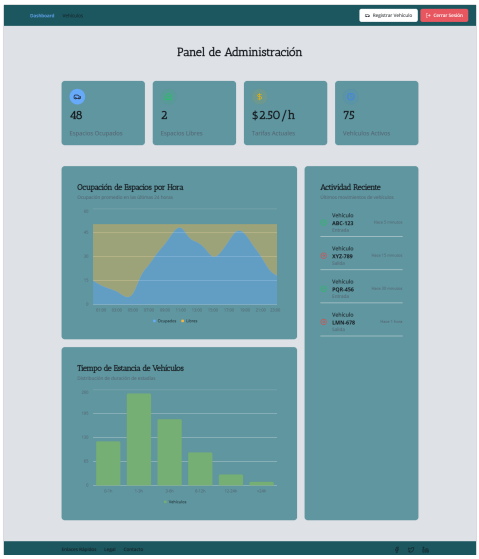


Fig. 9. Mockup of Dashboard for Parking Space

| **Class** | Fee |
|---|---|
| **Responsibilities** | **Collaborators** |
| Known prices per hour | |
| Known prices by vehicle type | User |
| Known special fees by user | Register |
| Apply discounts by user type | |

| **Class** | Register |
|---|---|
| **Responsibilities** | **Collaborators** |
| Register vehicule entry | |
| Register vehicule departure | Vehicle |
| Calculate vehicule parking time | User |
| Calculete cost | Fee |
| Apply user discounts | slot |
| Generate ticket | |

| **Class** | Slot |
|---|---|
| **Responsibilities** | **Collaborators** |
| Know its own id | |
| Know if it's free | Vehicle |
| Know its own slot type | Register |
| Reserve | Area |
| Get empty | |

| **Class** | Area |
|---|---|
| **Responsibilities** | **Collaborators** |
| Know its own id | |
| Know its own name | Slot |
| Know its own capacity | Register |
| Know its own slots disponibility | |

### *F. Conclusion of Methods and Materials*

The Parking Management System was designed with simplicity, scalability, and usability in mind. The system was divided into modular components to ensure that each aspect of the parking process — from vehicle registration to receipt generation — is streamlined and automated. By using modern technologies such as Flask, PostgreSQL, and Docker, the system ensures reliability, scalability, and ease of deployment. The diagrams and mockups provided help visualize the design, user flow, and architecture of the system.

## III. Results and Discussion

Since the project is still in the development phase, it has not yet been tested or trialed in a real-world setting. However, the testing will be conducted in several stages to ensure that the system meets the requirements for performance, usability, and reliability. This section outlines the proposed approach for the evaluation of the system.

### *A. Testing Methodology*

To ensure the effectiveness of the parking management system, the following testing phases will be carried out:

- **Unit Testing**:
  - **Objective**: Verify that individual components (both frontend and backend) work as expected in isolation. Each function, such as vehicle registration, receipt generation, and database communication, will be tested separately.
  - **Approach**: Automated tests will be written to test the core functionalities of each module. For example, testing the logic behind vehicle entry and exit times, ensuring that the receipts are calculated correctly, and verifying database interactions (insert and retrieve operations).

- **Integration Testing**:
  - **Objective**: Test the interactions between the frontend, backend, and database to ensure seamless communication and data consistency.
  - **Approach**: The system's major workflows, such as registering a vehicle entry or generating a receipt, will be tested end-to-end. The backend and frontend will be tested together to check if the data flows correctly between them and if the database is updated in real time.
- **Acceptance Testing**:
  - **Objective**: Ensure that the system meets the expected user requirements and performs well under real-world conditions.
  - **Approach**: This phase will involve testing the system with parking attendants (the primary users). The system's usability will be assessed based on how easy it is for the attendants to:
    - Register vehicle entries and exits.
    - Generate and print receipts.
    - View parking space availability in real-time.

    Feedback from the attendants will be gathered to identify any issues related to the user interface or the system's functionality.

### *B. Performance Testing*

After the system has been integrated and tested for functionality, the following performance tests will be conducted to ensure that the system is efficient and scalable:

- **Load Testing**:
  - **Objective**: Simulate a high volume of vehicle entries and exits to assess the system's ability to handle traffic without degradation in performance.
  - **Approach**: The system will be tested with varying levels of concurrent users and data input (e.g., multiple vehicle registrations at the same time). We will monitor response times and server load under different scenarios to ensure that the system performs well under stress.
- **Response Time Measurement**:
  - **Objective**: Measure the time it takes for the system to process key operations, such as registering an entry, generating a receipt, and querying the database for vehicle data.
  - **Approach**: Average response times will be calculated for critical actions, and the system will be optimized for minimal latency.
- **Error Handling and Recovery**:
  - **Objective**: Ensure that the system can recover gracefully from unexpected issues, such as database failures, network disruptions, or invalid input.
  - **Approach**: Simulate various failure scenarios to verify that the system can handle errors without crashing and that it can recover quickly.

### *C. Usability Testing*

- **User Experience (UX)**:
  - **Objective**: Evaluate how intuitive and easy the system is to use for parking attendants.
  - **Approach**: Conduct usability tests by asking parking attendants to perform typical tasks (e.g., register a vehicle entry, generate a receipt). Their feedback will be used to improve the user interface and workflow.
- **User Interface (UI) Evaluation**:
  - **Objective**: Assess the layout, design, and ease of navigation of the frontend interface.
  - **Approach**: Attendants will rate the clarity of the interface, ease of finding necessary features, and general satisfaction with the visual design. Issues related to the UI will be identified and addressed.

### *D. Future Testing and Rollout*

Upon completion of the initial testing, the system will be deployed in a controlled environment (a small parking lot or a simulated environment) for **pilot testing**. The results from this trial will help to fine-tune the system before it is deployed on a larger scale. This phase will focus on:

- **System stability**: Ensuring that the system can operate continuously for extended periods without failures.
- **Real-world performance**: Measuring how the system handles typical parking lot traffic and various edge cases.
- **User feedback**: Gathering further feedback from parking attendants and facility managers to ensure that the system meets operational needs.

### *E. Conclusion of Testing Approach*

The success of the parking management system will depend on the thoroughness of the testing process. The tests outlined in this section will ensure that the system is functional, scalable, and user-friendly. The feedback obtained from the testing phases will be used to continuously improve the system, addressing any identified issues and refining the user experience. Once the system is fully tested and optimized, it will be ready for broader deployment in a variety of parking lots.

## IV. Conclusions

The parking management system developed in this study provides an efficient, low-cost solution for automating vehicle entry and exit registration and receipt generation. By reducing human error and improving real-time vehicle tracking, the system provides a more efficient and reliable solution for parking lot management. Future improvements will focus on enhancing scalability, integrating payment processing, and adding mobile access for users.

## References

[1] C. F. Chien, Y. C. Ding, C. H. Wei, and C. H. Wei, "A low-cost on-street parking management system based on bluetooth beacons," *Sensors*, vol. 20, no. 6, p. 1701, 2020. DOI: 10.3390/s20061701. [Online]. Available: https://doi.org/10.3390/s20061701.

[2] WebbyLab, *Smart parking system using iot [how to create your own]*, Accessed: 2025-10-24, 2024. [Online]. Available: https://webbylab.com/blog/smart-parking-system-using-iot/.