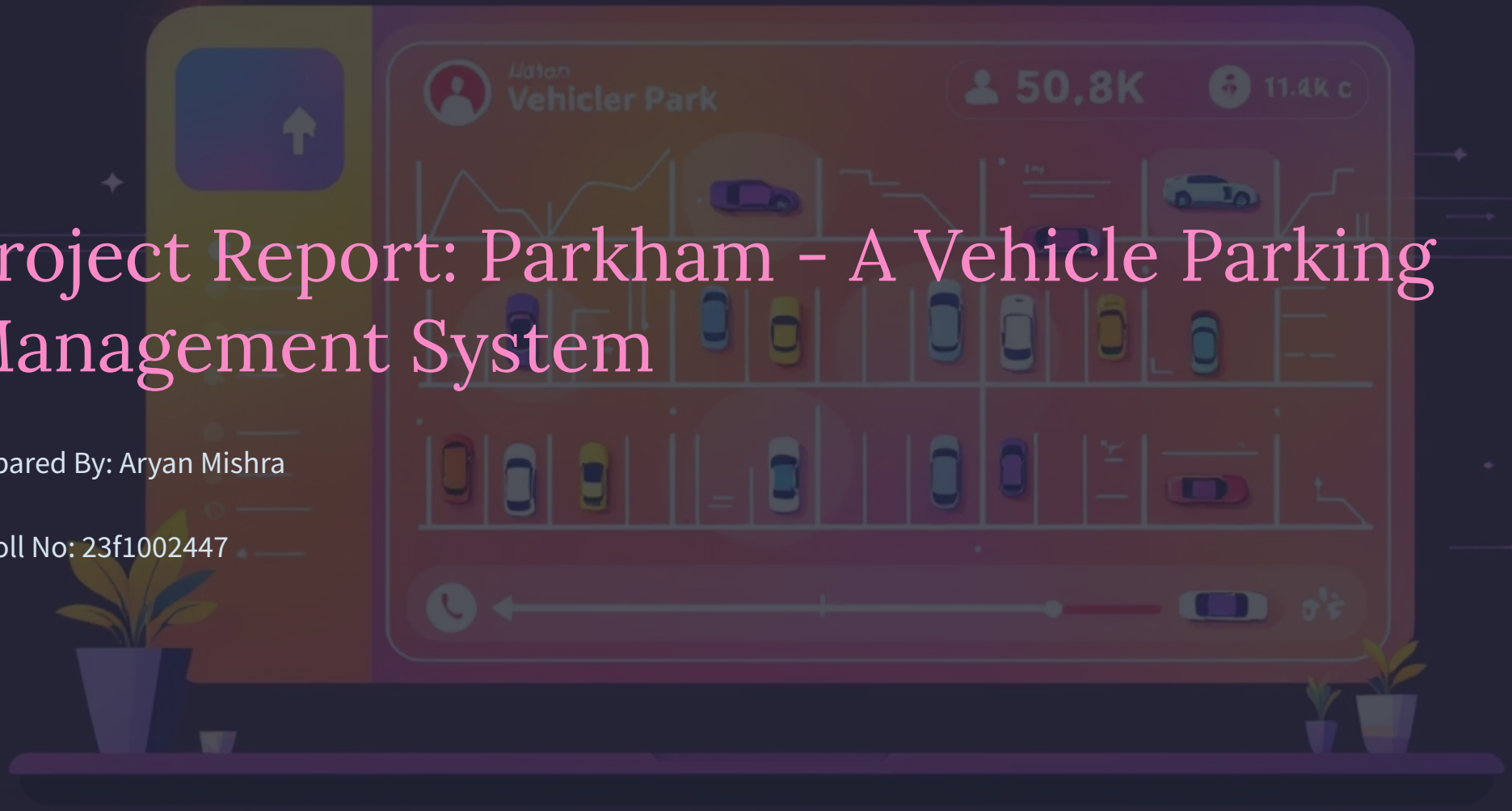


Project Report: Parkham - A Vehicle Parking Management System

Prepared By: Aryan Mishra

 Roll No: 23f1002447

1. Project Details

1.1 Problem Statement

The objective of this project is to develop "Parkham," a multi-user web application designed for the efficient management of 4-wheeler vehicle parking. The system caters to two distinct roles: Administrators and Users.

Admin

A superuser with root access who can manage the entire parking infrastructure. This includes creating, editing, and deleting parking lots, defining the number of spots within each lot, and setting pricing. The admin has a global view of all parking spots, their occupancy status, and registered user data.

User

A registered member who can browse available parking lots, book a parking spot (which is automatically allocated by the system), and subsequently release the spot upon departure.

The application aims to provide a seamless and automated experience for both managing parking facilities and finding and reserving parking spaces.

1.2 Approach

The application is developed using a monolithic architecture with a clear separation between the backend logic and the frontend presentation.

Backend

Built with the Flask framework in Python, handling all business logic, database interactions, and API endpoints.

Frontend

 Developed using Jinja2 for server-side templating, styled with HTML, CSS, and Bootstrap to ensure a responsive and user-friendly interface.

Database

SQLite is used as the database, created and managed programmatically through Flask-SQLAlchemy models to ensure data integrity and consistency.

The system is designed with distinct dashboards for each role, providing tailored functionalities and data visualizations to meet their specific needs.

2. Frameworks, Libraries, and Database

2.1 Frameworks and Libraries Used

Flask

A lightweight WSGI web application framework in Python used for the core backend logic.

Jinja2

A modern and designer-friendly templating engine for Python, used to render the frontend pages.

Bootstrap

A popular CSS framework for developing responsive and mobile-first websites.

SQLite

A C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.

Flask-SQLAlchemy

An extension for Flask that adds support for SQLAlchemy, simplifying database operations and object-relational mapping (ORM).

Chart.js

A JavaScript library used to create simple yet flexible and interactive charts for data visualization on the admin and user dashboards.

2.2 Database ER Diagram

The database consists of five main tables to model the application's data structure:

User

Stores information about registered users.

- id (Primary Key)
- username (String, Unique)
- password_hash (String)
- role_id (Foreign Key -> Role.id)

Role

Defines the roles within the system (Admin, User).

- id (Primary Key)
- name (String, Unique)

ParkingLot

Contains details about each parking facility.

- id (Primary Key)
- prime_location_name (String)
- price (Float)
- address (String)
- pin_code (String)
- maximum_number_of_spots (Integer)

ParkingSpot

Represents an individual parking spot within a lot.

- id (Primary Key)
- lot_id (Foreign Key -> ParkingLot.id)
- status (String, 'Available' or 'Occupied')

Booking

Records the details of each parking reservation.

- id (Primary Key)
- spot_id (Foreign Key -> ParkingSpot.id)
- user_id (Foreign Key -> User.id)
- parking_timestamp (DateTime)
- leaving_timestamp (DateTime, Nullable)
- parking_cost (Float)

Relationships:

- A ParkingLot has many ParkingSpots (One-to-Many).
- A User can have many Bookings (One-to-Many).
- A ParkingSpot can be part of many Bookings over time (One-to-Many).

3. Core Functionalities and API Endpoints

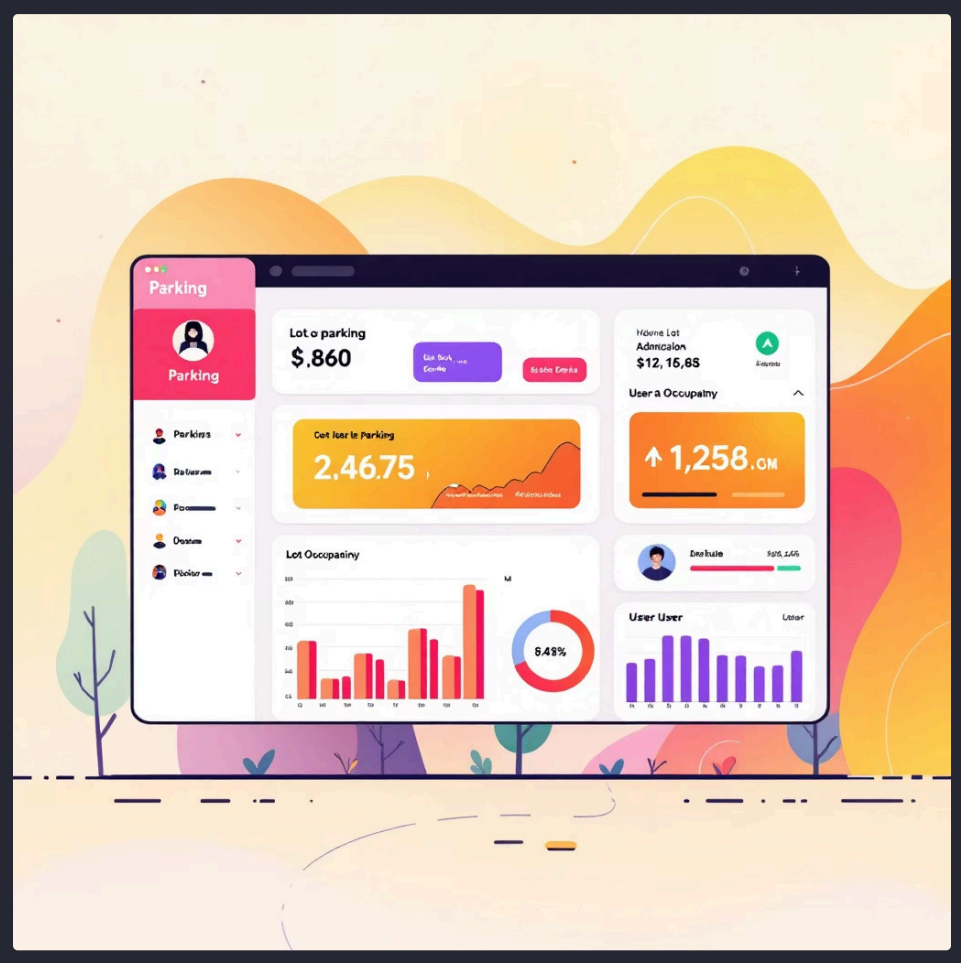
3.1 Core Functionalities Implemented

Authentication

Secure login/registration system for users and a dedicated login for the pre-existing admin.

Admin Dashboard

- Full CRUD (Create, Read, Update, Delete) functionality for Parking Lots.
- Automatic creation of Parking Spots based on the lot's capacity.
- Live view of all parking spots and their occupancy status.
- Ability to view details of all registered users.
- Summary charts displaying parking lot usage and revenue.



User Dashboard

- View available parking lots.
- Book a parking spot, which is automatically allocated on a first-available basis.
- View personal booking history and summary charts of parking expenses.
- Functionality to "release" a spot, which records the leaving timestamp and calculates the final cost.

3.2 API Resource Endpoints

The following RESTful API endpoints were created to allow for programmatic interaction with the application's data:

1

GET /api/lots

Retrieve a list of all available parking lots.

2

GET /api/lots/int:lot_id

Get detailed information for a specific parking lot.

3

GET /api/lots/int:lot_id/spots

View the status of all spots within a specific lot.

4

POST /api/bookings

Create a new parking reservation for a user.

5

GET /api/users/int:user_id/bookings

Fetch the booking history for a specific user.