Vehicle Parking Application

Final Project Report

BY

Jayesh Bansal

24f1001895 | Modern Application Development - I | May 2025 Term

Author:

Name: Jayesh Bansal
Hallticket No: 24f1001895
Email: 24f1001895@ds.study.iitm.ac.in
About: Hi, I am Jayesh Bansal, a Data Science undergraduate with a deep interest in backend development and intelligent/automation systems. I enjoy building user-focused tools by combining intuitive design with clean backend logic.

Description:

The Vehicle Parking Management App is a comprehensive web application designed to efficiently manage parking lots and facilitate parking spot reservations. It supports multiple user roles, specifically "Admin" and "User," providing tailored functionalities for each. The application aims to streamline parking operations, allowing administrators to manage parking infrastructure and users to easily find, book, and release parking spots.

Technologies used:

Backend:
- Framework: Flask
- Database: SQLite (for local development and easy setup)
- Authentication: Flask-Login
- Password Hashing: Werkzeug (used for generate_password_hash and check_password_hash)

Frontend:
- Templating Engine: Jinja2
- CSS Framework: Bootstrap 5
- Icons: Font Awesome 6
- Charting: Chart.js (for dashboard analytics)

AI Used:

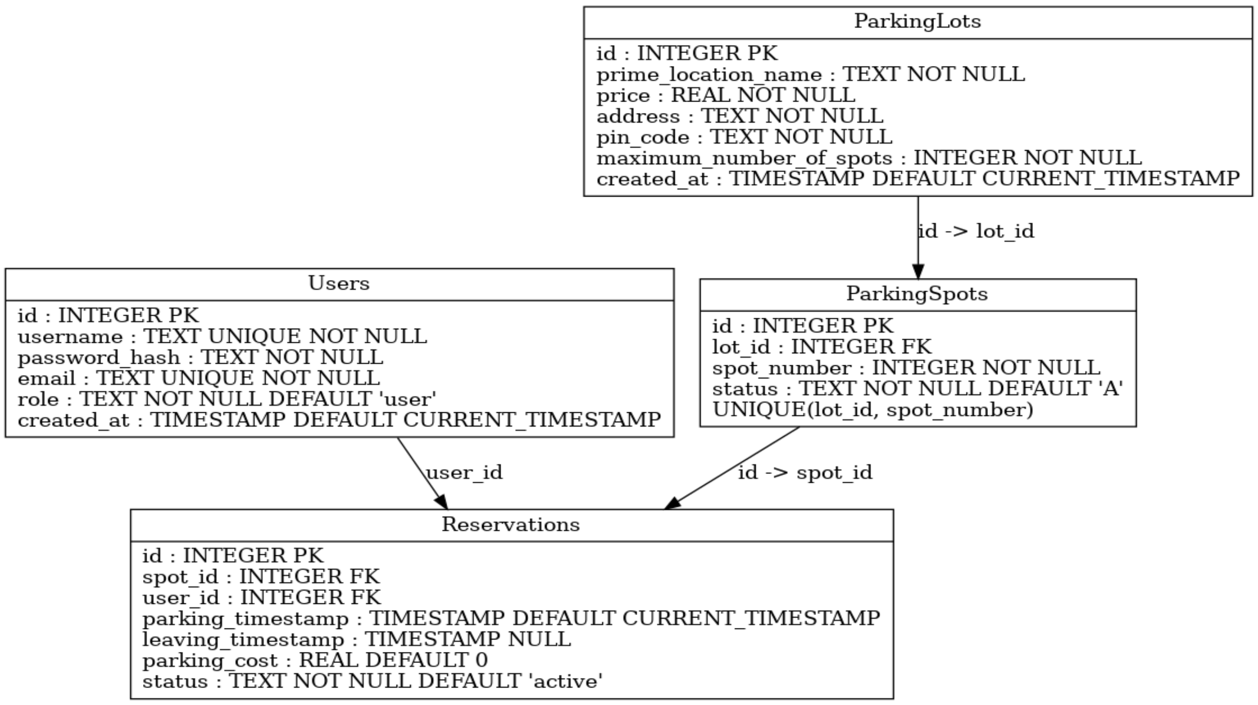
Category	Percentage Used	Notes
Backend(Flask,DB)	25%	Used to get the proper structure on how to make flask applications and sqlalchemy to extract details using queries
Frontend(HTML,JS)	17%	Used HTMl to get a sample template so that it can be copies and modified. Used JS for password check
Styling/UI	4%	Get the base styling
Auth/CRUD Logic	16%	Learned/Used Werkzeug for authorization
Extras(Charts)	4%	Used chart.js to keep charts

API Design:

/api/parking_stats (GET)
Purpose: Provides statistical data for charts on both admin and user dashboards.
Authentication: Requires user to be logged in (@login_required).
Behavior:
- If current_user.role == 'admin': Returns data for parking lot occupancy
- If current_user.role == 'user': Returns user-specific booking statistics (labels: date, data: bookings).

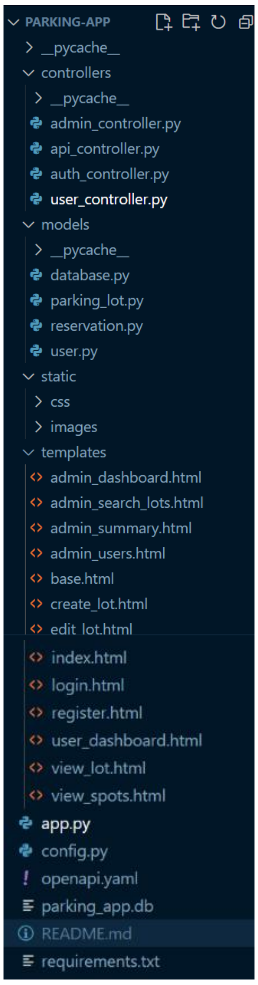
Response: JSON object containing labels, available (or bookings), and occupied arrays.

DB Schema Design:


ParkingLots
id : INTEGER PK
prime_location_name : TEXT NOT NULL
price : REAL NOT NULL
address : TEXT NOT NULL
pin_code : TEXT NOT NULL
maximum_number_of_spots : INTEGER NOT NULL
created_at : TIMESTAMP DEFAULT CURRENT_TIMESTAMP
id -> lot_id
Users
id : INTEGER PK
username : TEXT UNIQUE NOT NULL
password_hash : TEXT NOT NULL
email : TEXT UNIQUE NOT NULL
role : TEXT NOT NULL DEFAULT 'user'
created_at : TIMESTAMP DEFAULT CURRENT_TIMESTAMP
ParkingSpots
id : INTEGER PK
lot_id : INTEGER FK
spot_number : INTEGER NOT NULL
status : TEXT NOT NULL DEFAULT 'A'
UNIQUE(lot_id, spot_number)
user_id
id -> spot_id
Reservations
id : INTEGER PK
spot_id : INTEGER FK
user_id : INTEGER FK
parking_timestamp : TIMESTAMP DEFAULT CURRENT_TIMESTAMP
leaving_timestamp : TIMESTAMP NULL
parking_cost : REAL DEFAULT 0
status : TEXT NOT NULL DEFAULT 'active'

Architecture and Features

- app.py: The main Flask application file.
models/: This directory contains the data models and database interaction logic.
- database.py: Handles SQLite database connection and initialization (init_db).
- user.py: Defines the User class for user management
- parking_lot.py: Defines the ParkingLot class for managing parking lots and their spots.
- reservation.py: Defines the Reservation class for managing parking reservations.
controllers/: This directory contains Flask Blueprints, which group related routes and their logic.
- auth_controller.py: Handles user authentication (login, registration, logout).
- admin_controller.py: Manages routes and logic for admin functionalities. Includes an @admin_required decorator for access control.
- user_controller.py: Manages routes and logic for regular user functionalities (dashboard, book/release spots). Includes a @user_required decorator.
- api_controller.py: Contains API endpoints, currently for parking statistics.
templates/: This directory holds the Jinja2 HTML templates for rendering the user interface.
- base.html: The base template providing common structure (navbar, footer, flash messages).
- index.html: The public home page.
- login.html, register.html: Authentication forms.
- admin_dashboard.html, create_lot.html, edit_lot.html, view_spots.html, admin_users.html: Admin-specific views.
- user_dashboard.html: User-specific view.
static/: Contains static asset.
- **images/:** Contains images
- **css/:** Contains style.css which manages the styles of the template
- config.py: Manages application configurations for different environments
- requirement.txt: Required python dependencies
- openapi.yaml: Defines the structure and endpoints of the RESTful API


```
PARKING-APP
  > __pycache__
  v controllers
    > __pycache__
    admin_controller.py
    api_controller.py
    auth_controller.py
    user_controller.py
  v models
    > __pycache__
    database.py
    parking_lot.py
    reservation.py
    user.py
  v static
    > css
    > images
  v templates
    admin_dashboard.html
    admin_search_lots.html
    admin_summary.html
    admin_users.html
    base.html
    create_lot.html
    edit_lot.html
    index.html
    login.html
    register.html
    user_dashboard.html
    view_lot.html
    view_spots.html
  app.py
  config.py
  openapi.yaml
  parking_app.db
  README.md
  requirements.txt
```

Features:

- Multi-Part Dashboard: The admin dashboard is logically divided into three main sections for improved navigation and focus:
- Create Lot: Admins can easily add new parking lots by providing details like location name, hourly price, address, pin code, and the maximum number of spots. The system automatically generates individual parking spots for the new lot.
- Edit Lot: Existing parking lot details can be updated, including the maximum number of spots. The system intelligently adjusts the number of physical spots if the maximum capacity changes.
- Delete Lot: Parking lots can be deleted, but only if there are no currently occupied spots within that lot, preventing accidental data loss and ensuring operational integrity.
- Parking Spot Monitoring: Admins can view a detailed layout of all spots within a specific parking lot, visually indicating available ('A') and occupied ('O') statuses. For occupied spots, current reservation details (user, parking time, duration) are displayed.
- User Management: Admins have a dedicated page to view all registered users, their email addresses, and current roles.
- Role Management: Admins can update the role of any user (from 'user' to 'admin' and vice-versa), providing granular control over system access.
- Secure Authentication: Users can register for new accounts and securely log in using a username and password.
- Active Reservations: A clear list of their currently active parking reservations, showing the lot name, spot number, parking start time, and hourly rate.
- Available Parking Lots: A list of all parking lots with available spots, including their address, pin code, price per hour, and the number of available spots.
- Parking History: Users can view a comprehensive history of their past parking sessions, including details like lot name, spot number, parked on/left on timestamps, total cost, and reservation status.
- Responsive Design: The application utilizes Bootstrap 5 to ensure a responsive and mobile-friendly user interface across various devices.
- Flash Messages: Provides informative feedback to users for actions like successful logins, registrations, or errors.
- Visual Enhancements: Strategic integration of images on the landing page, login/register forms, and dashboards to create a more attractive and user-friendly experience.
- API Endpoint: A dedicated API endpoint (`/api/parking_stats`) provides structured data for dynamic charts and potential future integrations.

Video:

https://drive.google.com/file/d/1Til750ykdoJc7C6GR5T-xqHiXOLz50cR/view?usp=drive_link