VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Name :- Akshat Agarwal

Registration No. :- 23BKT0003

Faculty :- Maheshwari B

Course Name :- DBMS Lab L57+L58

Course Code :- BCSE302P

Project : Vehicle Parking Application

Team Member :

1. Akshat Agarwal

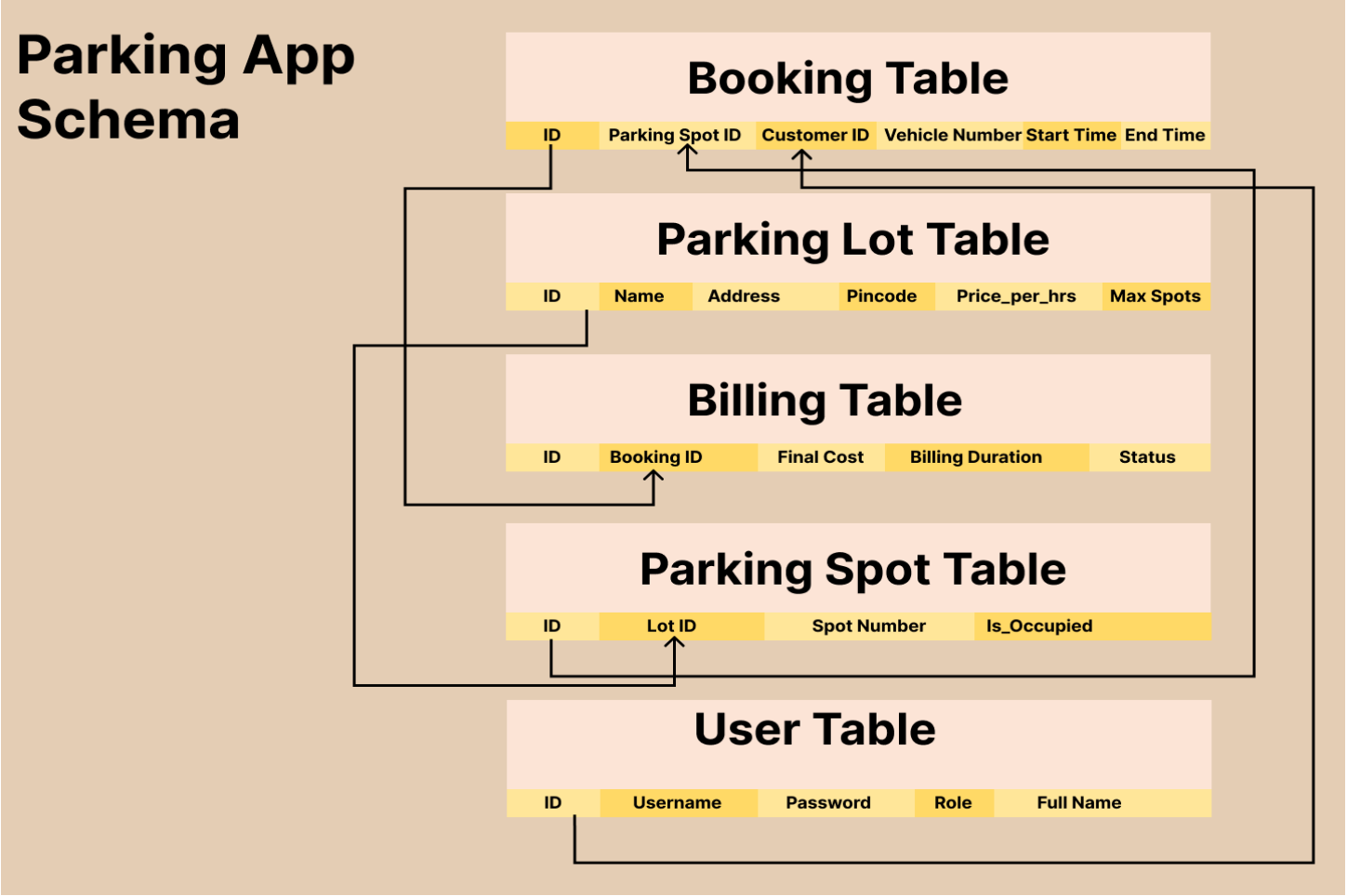
1. Introduction

- **1.1 Problem Statement:** Traditional parking management often relies on manual processes, leading to inefficiencies in tracking available spots, managing user bookings, calculating charges accurately, and providing oversight for administrators. This project aims to address these issues by developing a database-driven web application to streamline parking operations.
- **1.2 Objectives:** The primary database-related objectives were to:
    - Design and implement a normalized relational database schema suitable for managing users, parking lots, individual spots, bookings, and billing records.
    - Utilize an Object-Relational Mapper (SQLAlchemy with Flask-SQLAlchemy) to effectively map Python application objects to the underlying database tables.
    - Implement robust Create, Read, Update, and Delete (CRUD) operations for core entities, specifically Users (registration), Parking Lots, and Parking Spots (via Lot creation).
    - Implement transactional business logic involving multiple tables, such as creating linked Booking and Billing records simultaneously upon reservation, and updating related records (Booking, Billing, ParkingSpot) upon releasing a spot, including cost calculation. Ensure data integrity through cascading deletes when removing parent records like Parking Lots.
    - Develop a clear API layer using Flask to handle data retrieval (fetching lots, user bookings, summaries) and manipulation requests from the frontend, ensuring separation between the database logic and the user interface.
- **1.3 Technologies Used:**
    - **Database:** SQLite (chosen for simplicity in development and deployment for this project scale).
    - **Backend:** Python 3, Flask (web framework), Flask-SQLAlchemy (ORM), Werkzeug (for password hashing).

- **Frontend:** HTML5, CSS3, JavaScript (using fetch API for backend communication, localStorage for client-side user ID storage, Chart.js for admin visualizations).

2. Database Design

- **2.1 ER Schema:**



- **2.2 Schema Description:** The database consists of five main tables:
  - **User**: Stores information about registered users (both Admins and regular Users). Columns include id (Primary Key), username (unique, used for email login), password_hash (stores hashed password for security), role ('Admin' or 'User'), and full_name.
  - **ParkingLot**: Represents distinct parking areas. Columns include id (PK), name, address, pincode, price_per_hour, and max_spots.

- **ParkingSpot**: Represents individual spots within a lot. Columns include id (PK), lot_id (Foreign Key referencing ParkingLot), spot_number, and is_occupied (Boolean flag).
- **Booking**: Records each instance of a user parking in a spot. Columns include id (PK), spot_id (FK referencing ParkingSpot), customer_id (FK referencing User), vehicle_number, start_time, end_time (nullable), and status ('Active' or 'Completed').
- **Billing**: Stores the financial record associated with each booking. Columns include id (PK), booking_id (Unique FK referencing Booking), final_cost (nullable until checkout), billing_time (nullable until checkout), and status ('Reserved', 'Completed', 'Paid').

- **2.3 Relationships and Constraints:** Key relationships ensure data integrity:
  - **One-to-Many:** ParkingLot to ParkingSpot (lot_id in ParkingSpot).
  - **One-to-Many:** User to Booking (customer_id in Booking).
  - **One-to-Many:** ParkingSpot to Booking (spot_id in Booking).
  - **One-to-One:** Booking to Billing (booking_id in Billing, unique).
  - **Cascade Rules:**
    - **ondelete='CASCADE'** is set on Booking.spot_id. This ensures that if a ParkingSpot is deleted (which happens if its ParkingLot is deleted via SQLAlchemy cascade), all associated Booking records are automatically deleted by the database. SQLAlchemy's cascade="all, delete-orphan" on Booking.billing further ensures the linked Billing record is also removed. This maintains referential integrity when a parking lot is removed.
    - **ondelete='SET NULL'** is set on Booking.customer_id. If a User record is deleted, associated Booking records are kept, but their customer_id is set to NULL, preserving historical booking data without requiring the user to exist.

3. Implementation (Database Interaction)

- **3.1 ORM Models (SQLAlchemy):** Python classes (User, ParkingLot, ParkingSpot, Booking, Billing) were defined using Flask-SQLAlchemy, inheriting from **db.Model**.
    - **db.Column** was used to define table columns and data types, including primary keys (primary_key=True) and nullability (nullable=False).
    - **db.ForeignKey** was used to establish links between tables, incorporating ondelete rules for database-level cascades.
    - **db.relationship** defined the object-oriented connections between models, using back_populates for bidirectional navigation and cascade="all, delete-orphan" for SQLAlchemy-level object deletion logic (e.g., deleting a ParkingLot object triggers deletion of its ParkingSpot objects).
    - Helper methods like to_dict were added for easy JSON serialization, and set_password/check_password handle security.
- **3.2 Key API Routes & Business Logic:** Flask routes were created to handle interactions:
    - **Lot Management (/api/lots POST, PUT, DELETE):** The POST route creates a ParkingLot and then iterates to create associated ParkingSpot records. The PUT route updates lot details. The DELETE route relies on SQLAlchemy's cascade on the ParkingLot.spots relationship and the database's ondelete='CASCADE' from ParkingSpot to Booking (and subsequently Billing) to remove all related data.
    - **Booking a Spot (/api/book-spot POST):** This route performs checks (spot validity, availability via is_occupied). Within a try...except block (implicitly managed by Flask-SQLAlchemy session commits), it creates a Booking instance and a Billing instance, links them via the relationship (new_booking.billing =

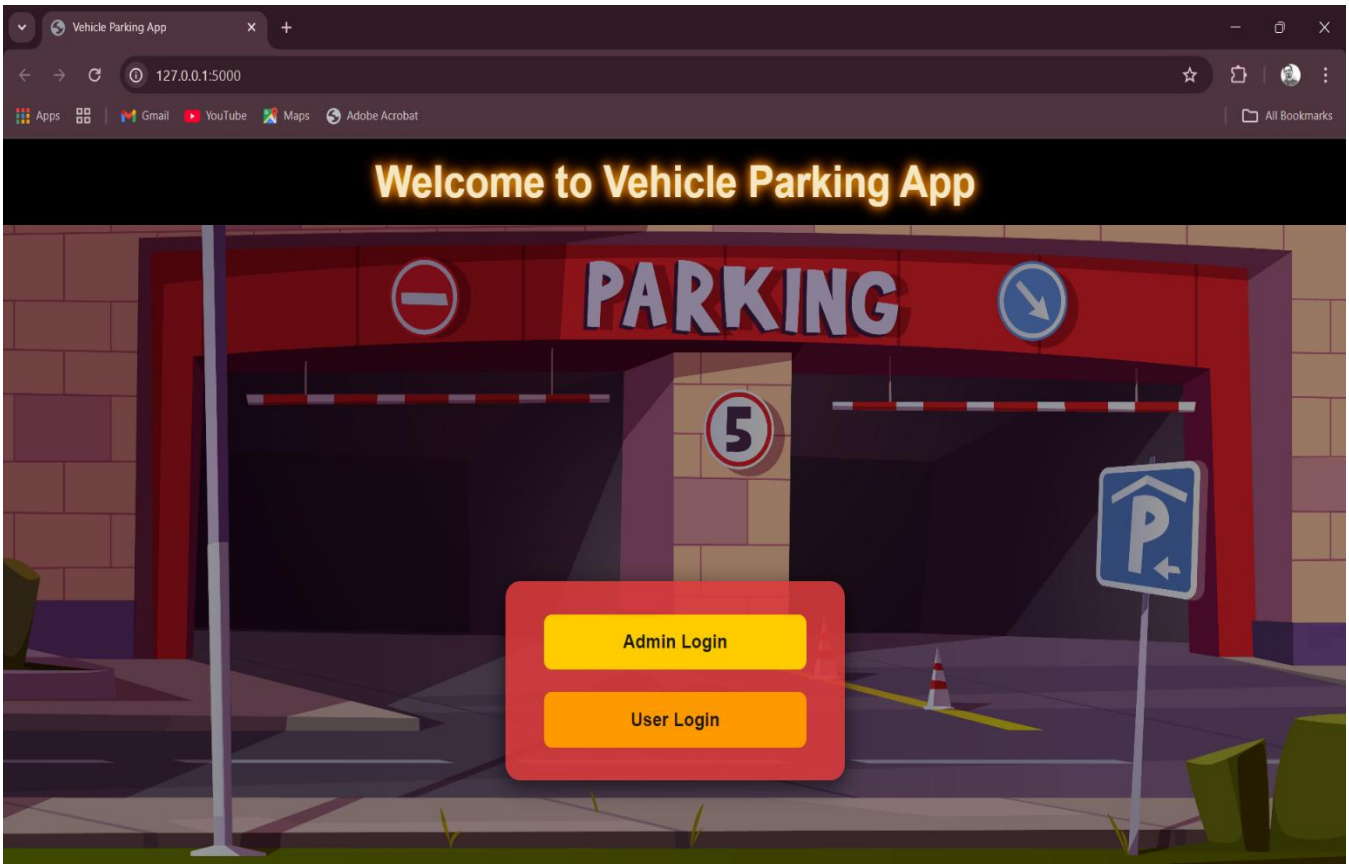
new_billing), updates ParkingSpot.is_occupied = True, and commits the transaction.

- **Releasing a Spot (/api/release-spot/<booking_id> POST):** Finds the Booking and related Billing, Spot, and Lot. Calculates duration (end_time - start_time) and final_cost (duration * price_per_hour, minimum 1 hour). Updates Booking (end_time, status='Completed'), Billing (final_cost, billing_time, status='Completed'), and ParkingSpot (is_occupied=False). Commits the changes.
- **Fetching Data (/api/my-bookings, /api/user-summary, /api/summary/profit-by-lot):** These GET routes perform database queries. /api/my-bookings and /api/user-summary use db.session.query(...).join(...) to combine data from multiple tables (Booking, ParkingSpot, ParkingLot or Billing, Booking) based on user ID. /api/summary/profit-by-lot uses db.func.sum() and .group_by() for aggregation to calculate total profit per lot from Billing records linked through Booking, ParkingSpot to ParkingLot. Data is formatted into JSON for the frontend.

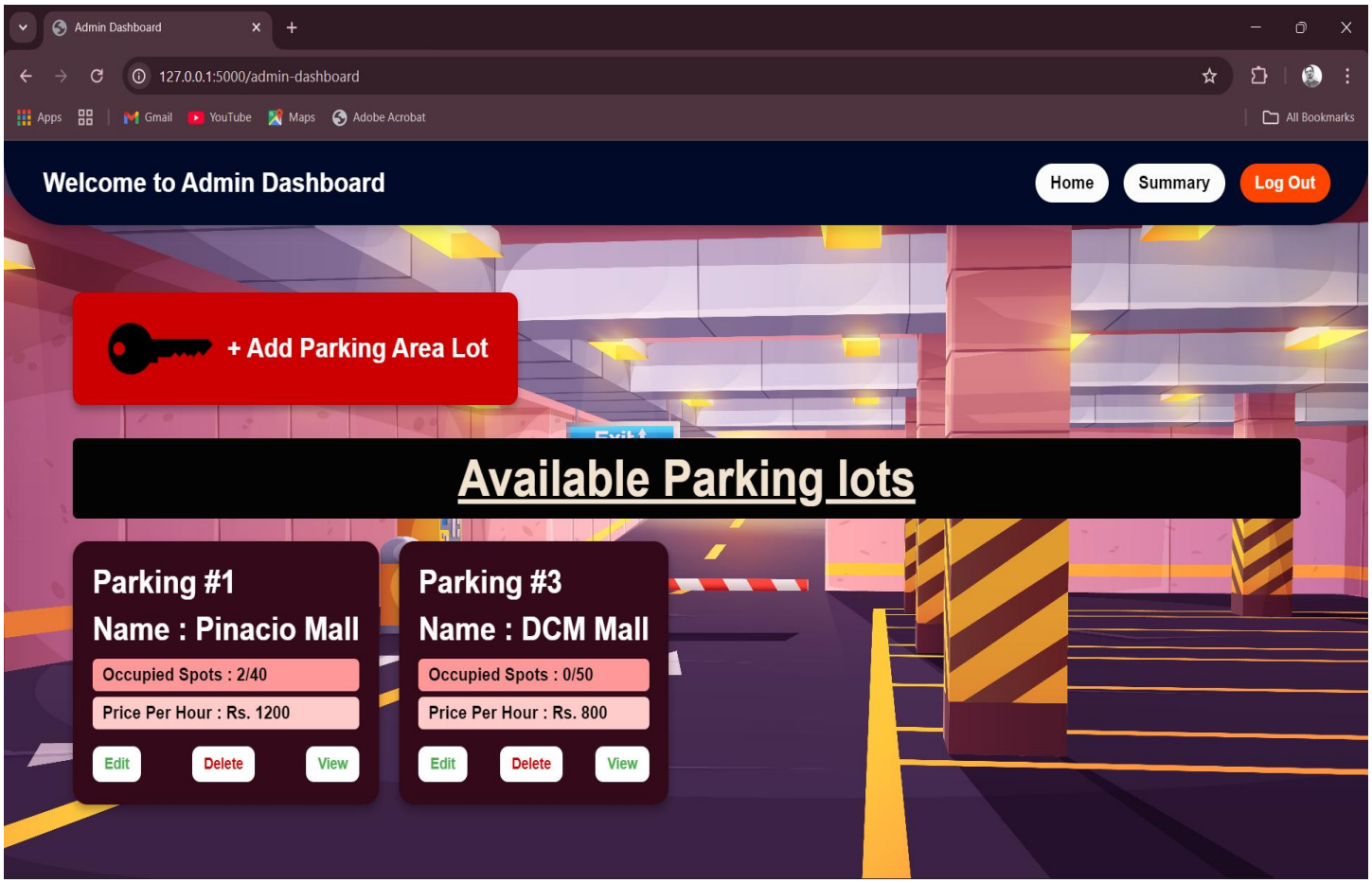
- **3.3 Frontend Interaction:** JavaScript files use the fetch API to make asynchronous requests to these backend endpoints (e.g., GET to fetch data, POST to create/update). User input (like vehicle number) is collected, validated client-side, and sent as JSON. Responses (data or error messages) are used to dynamically update the HTML DOM, showing tables, charts, or feedback messages. localStorage is used to store the user_id after login for subsequent API calls.

4. Website Screenshots/ Results :

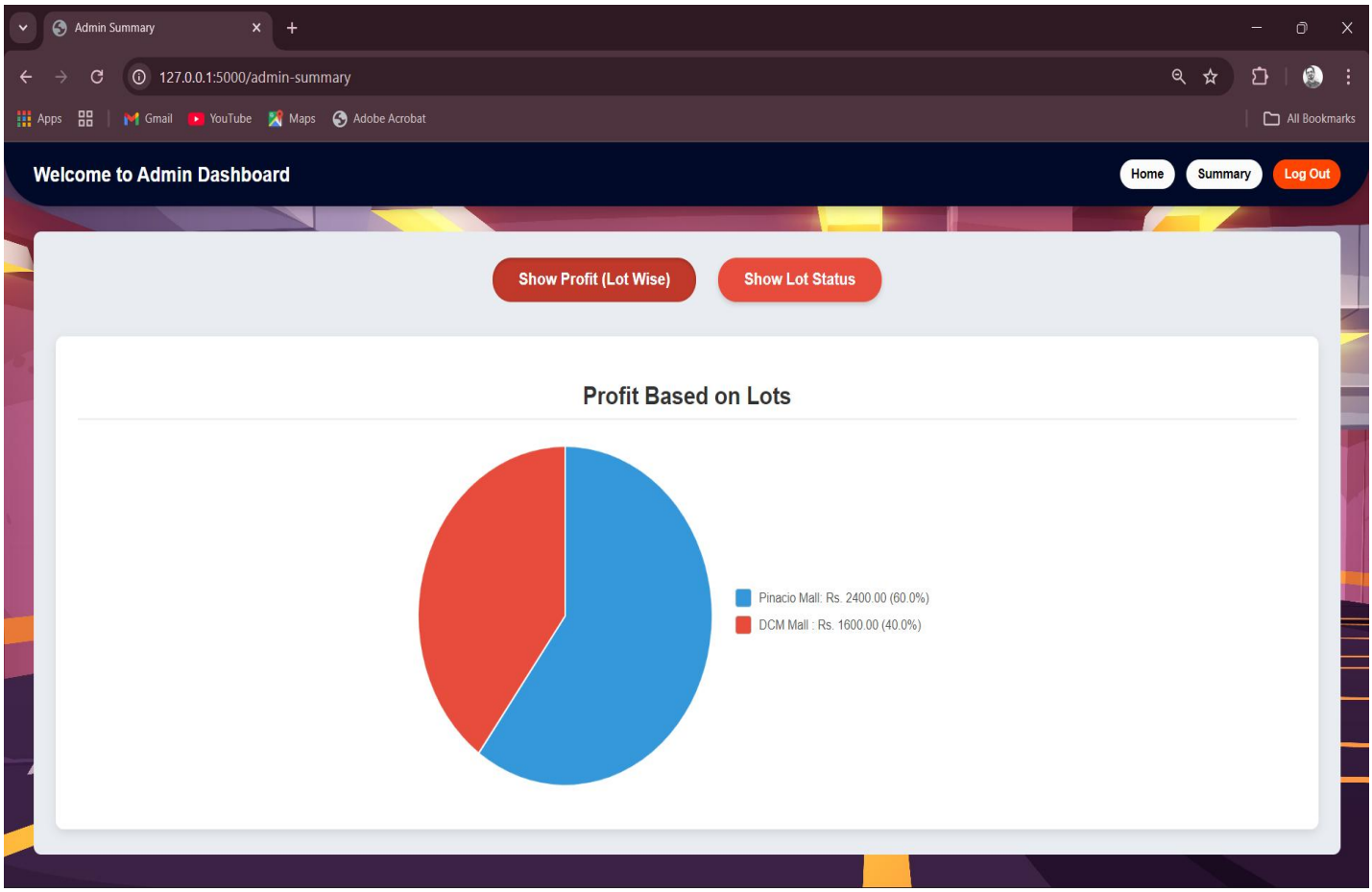
1. Index Page



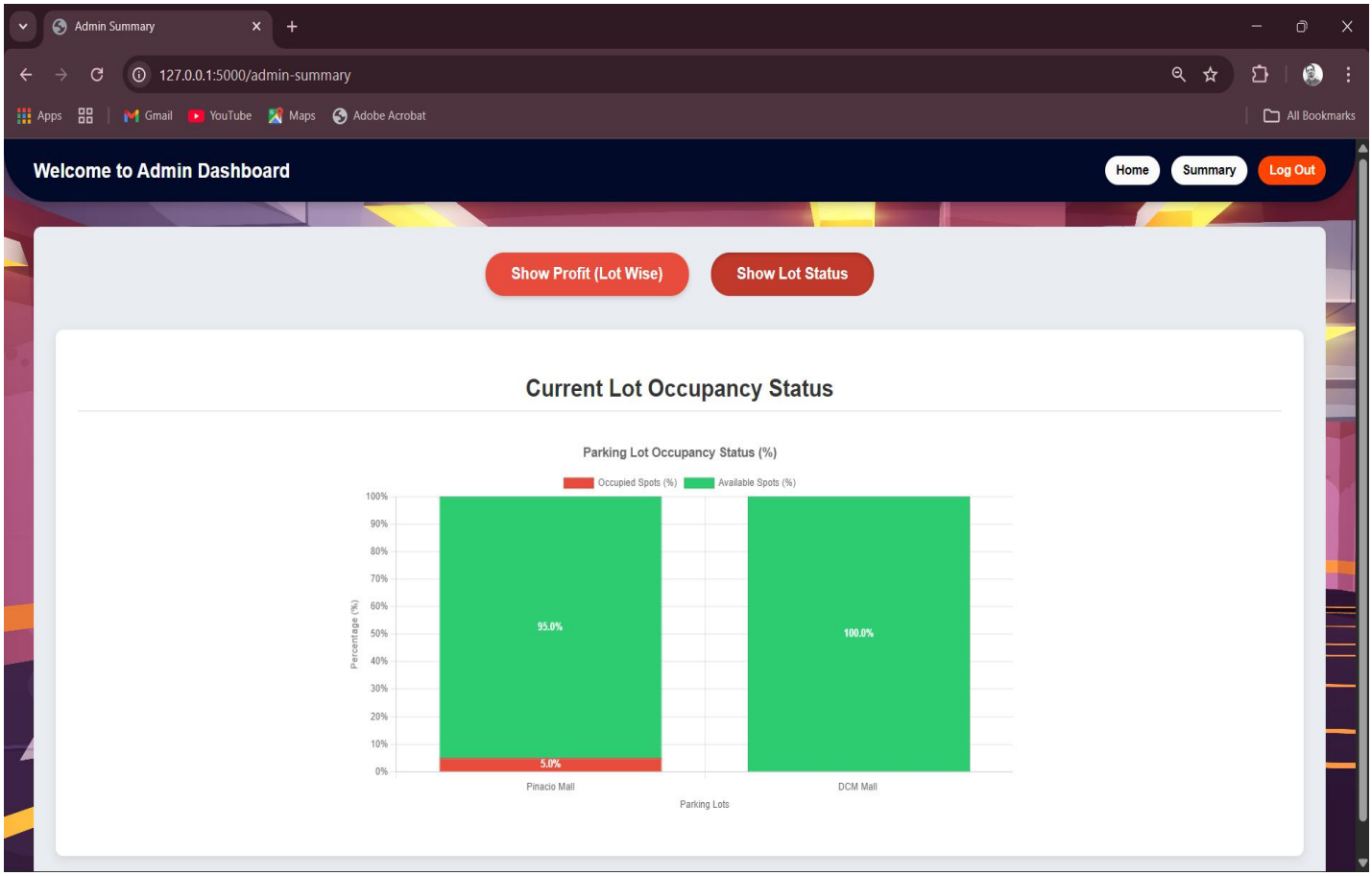
2. Admin Dashboard



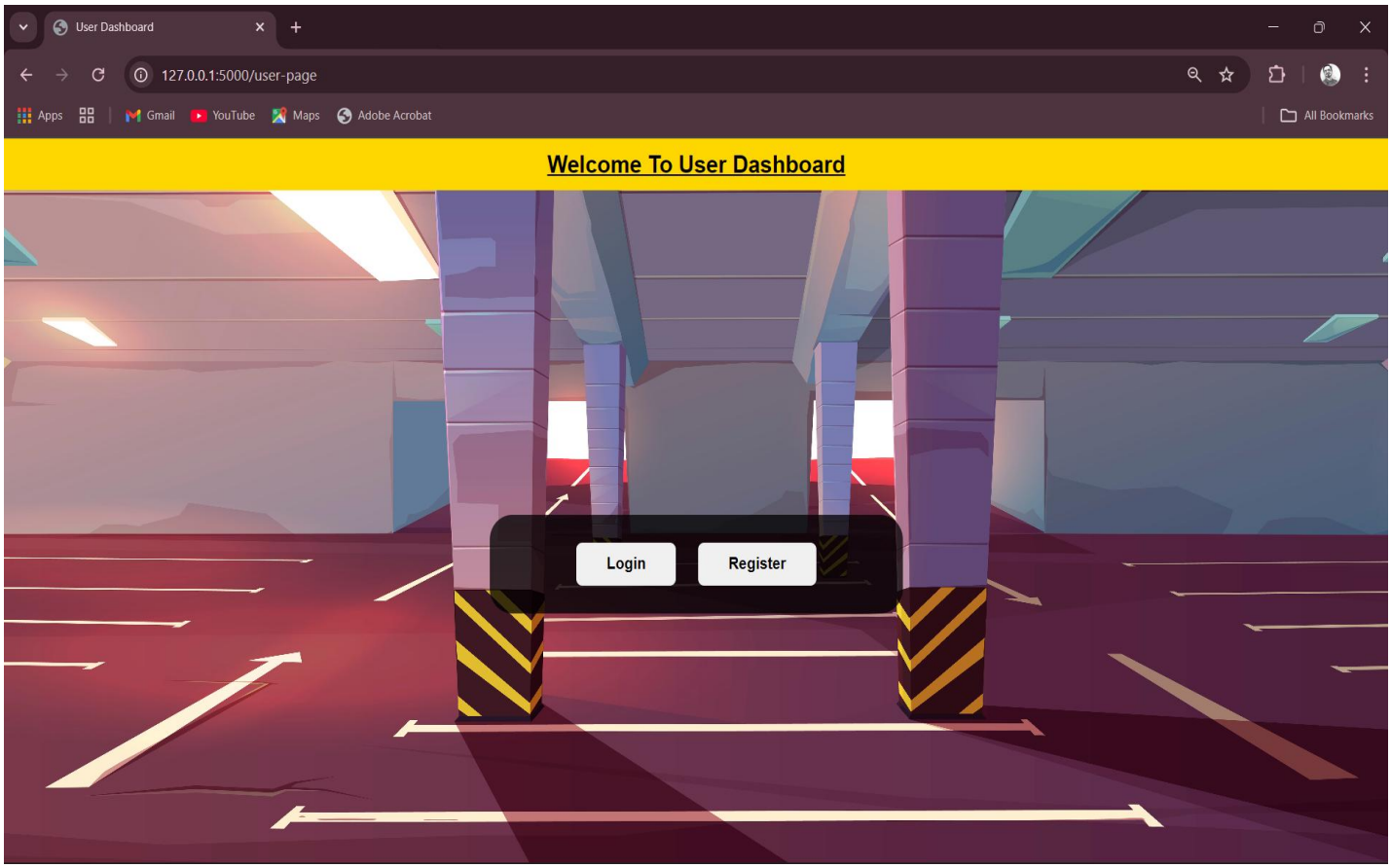
3. Admin Summary Page (Parking Lot Wise Profit)



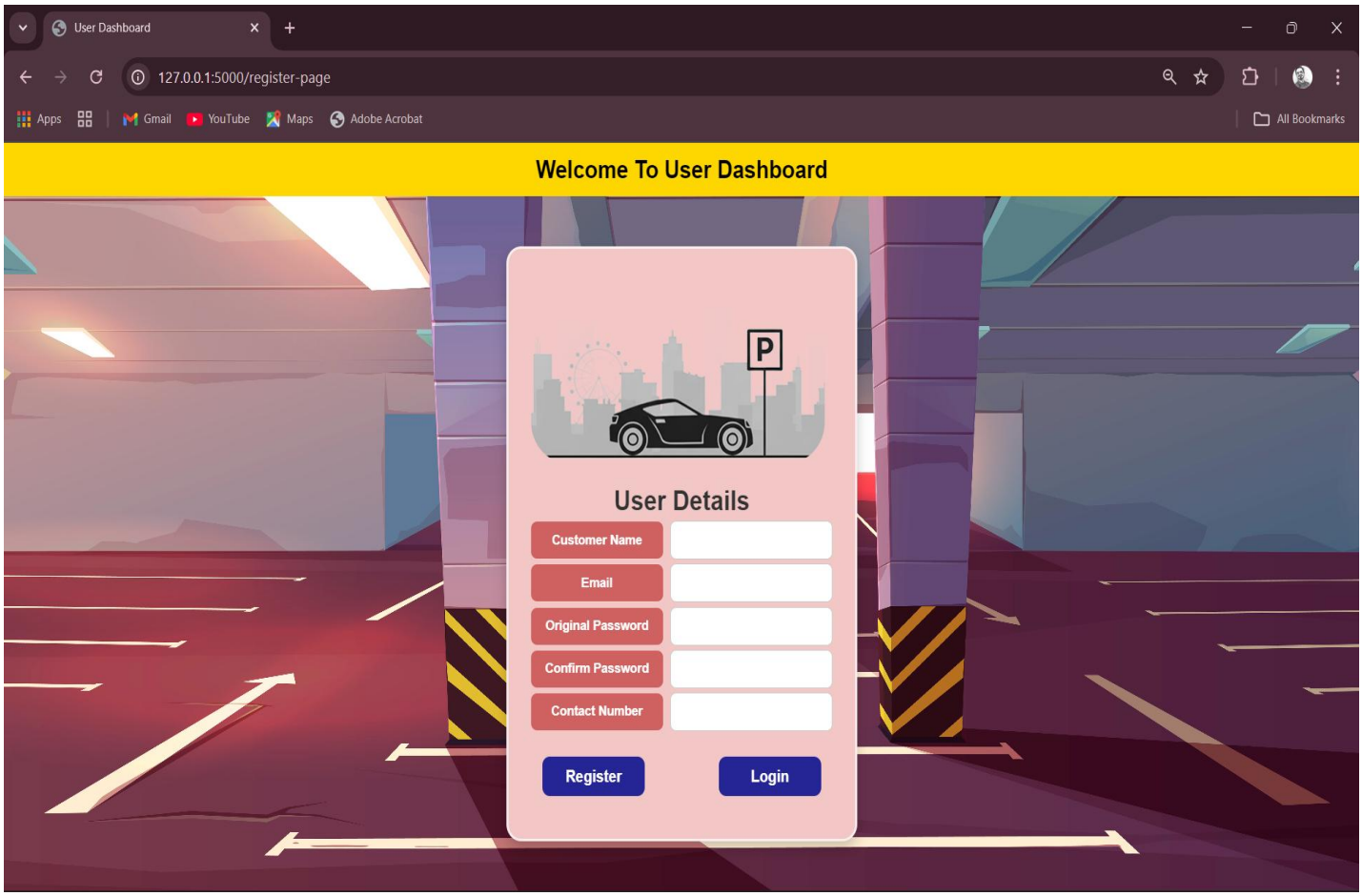
4. Admin Summary Page (Parking Lot Status)



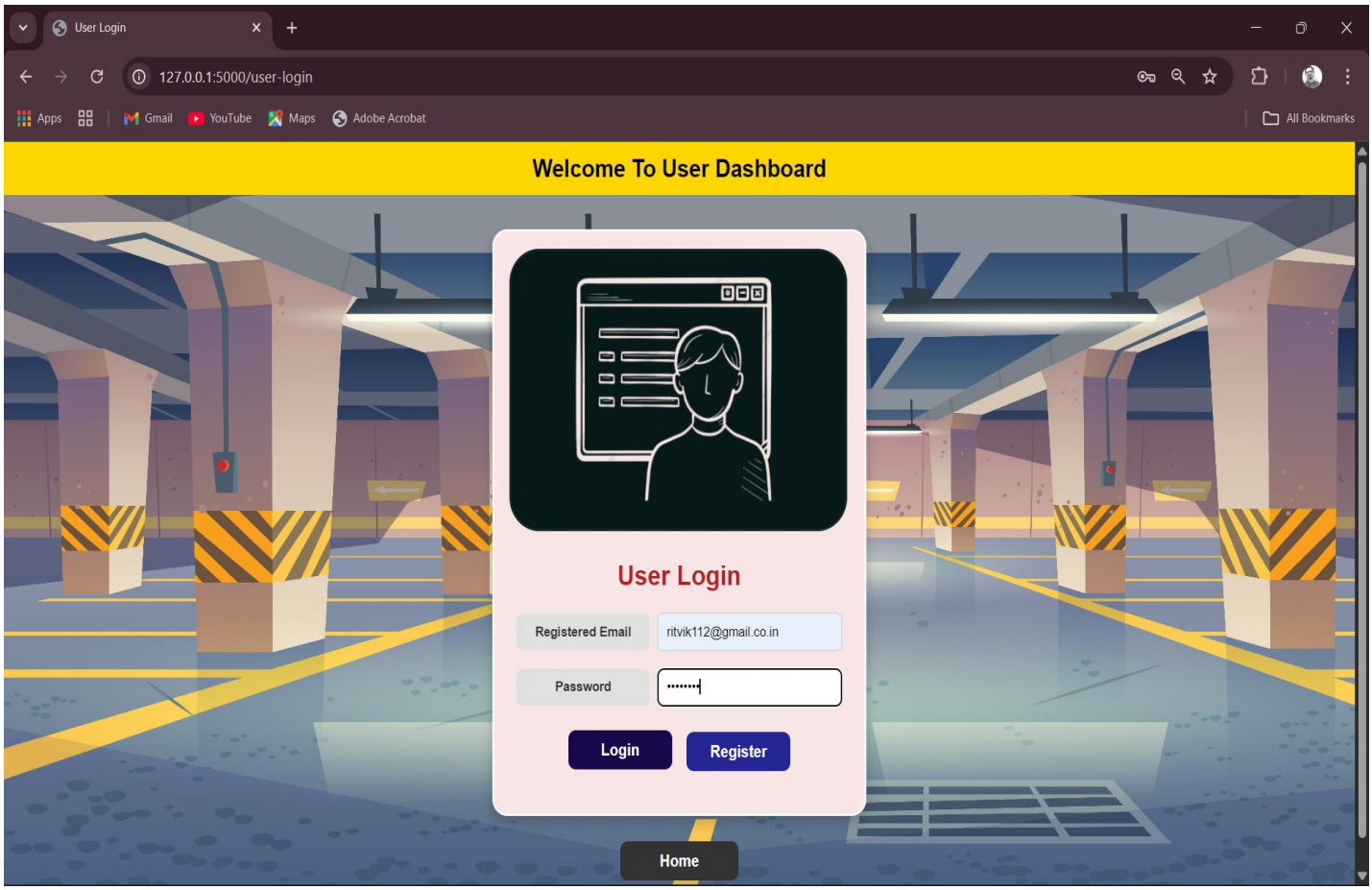
5. User Index Page



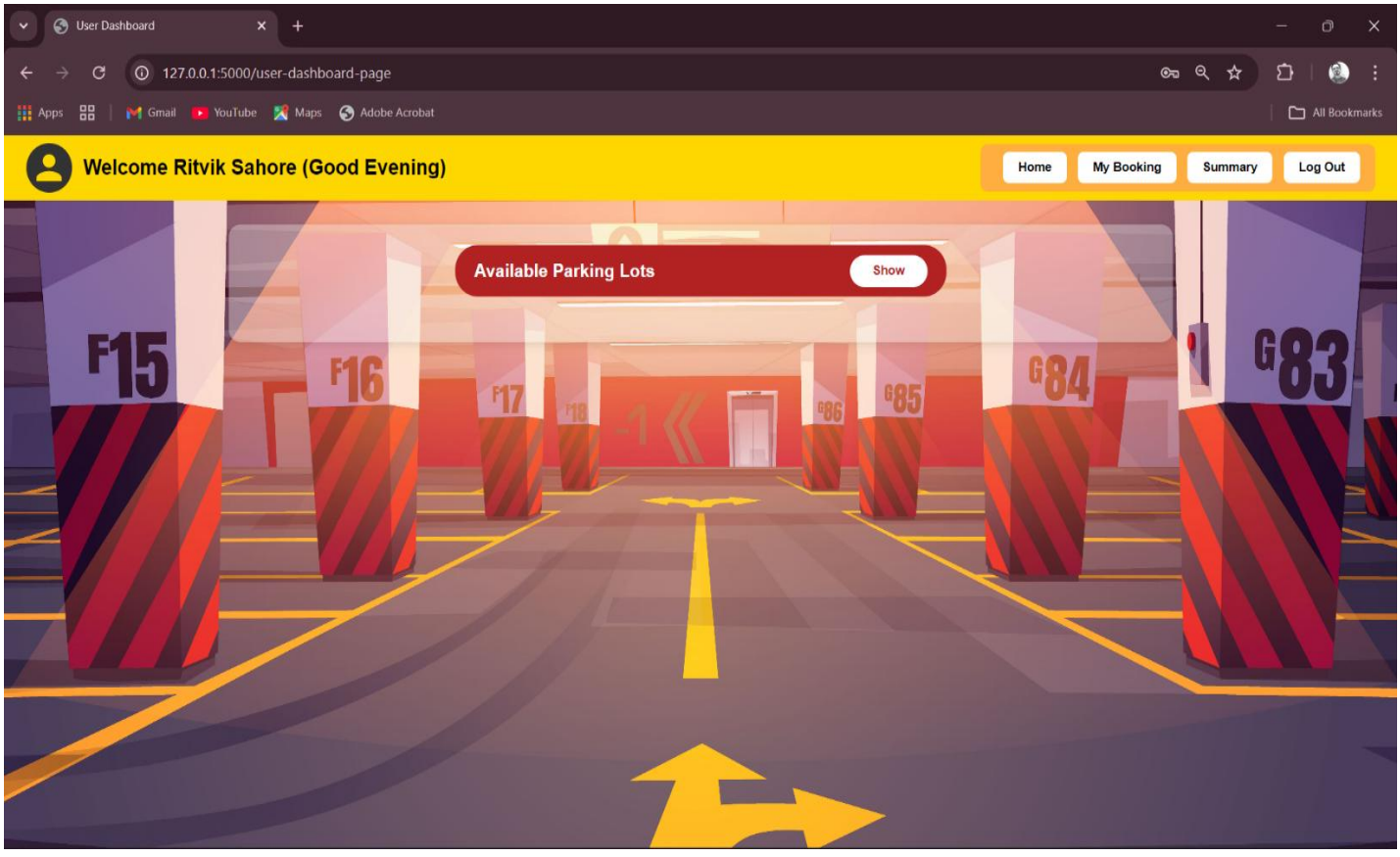
6. User Registration Page

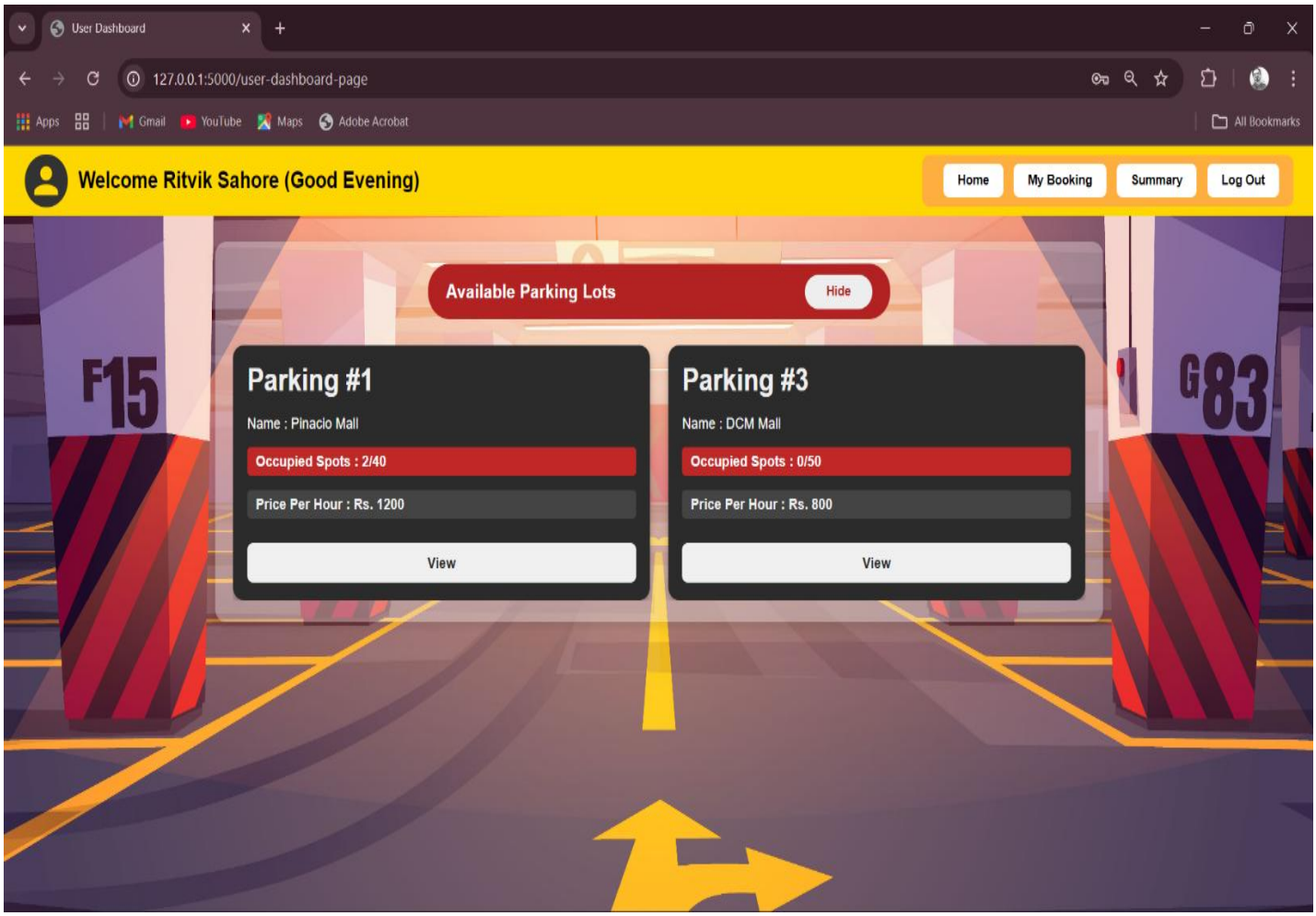


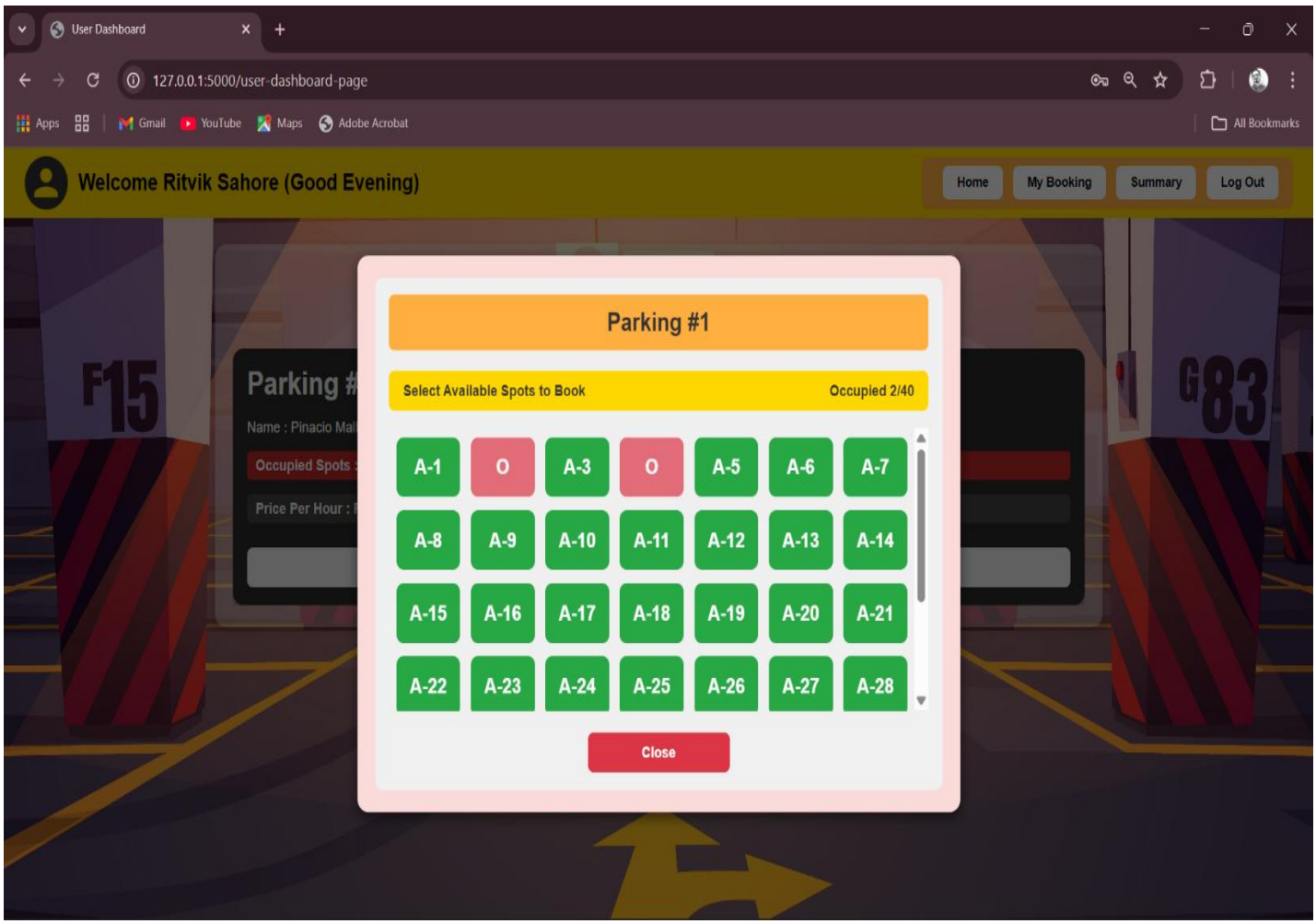
7. User Login Page



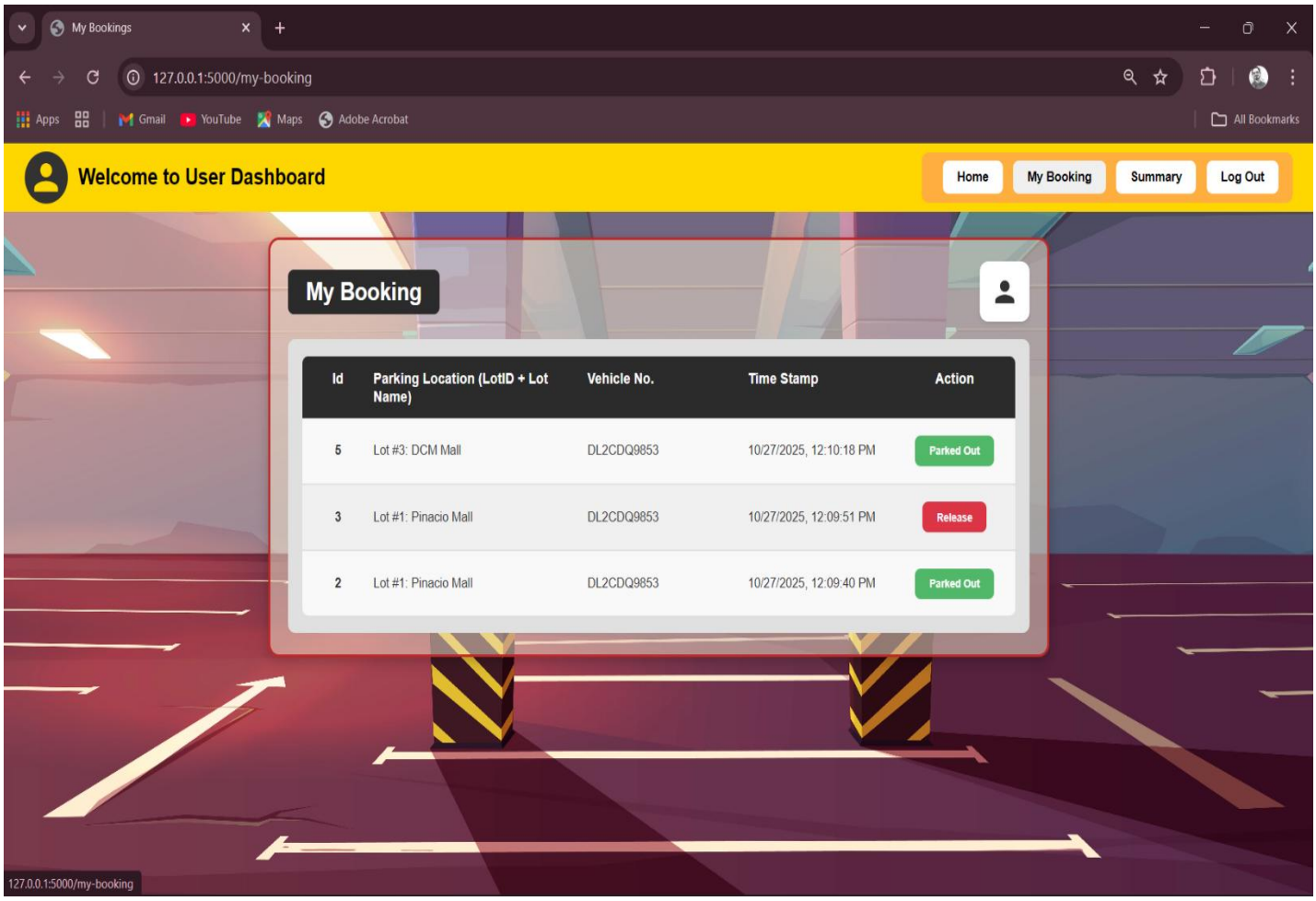
8. User Dashboard



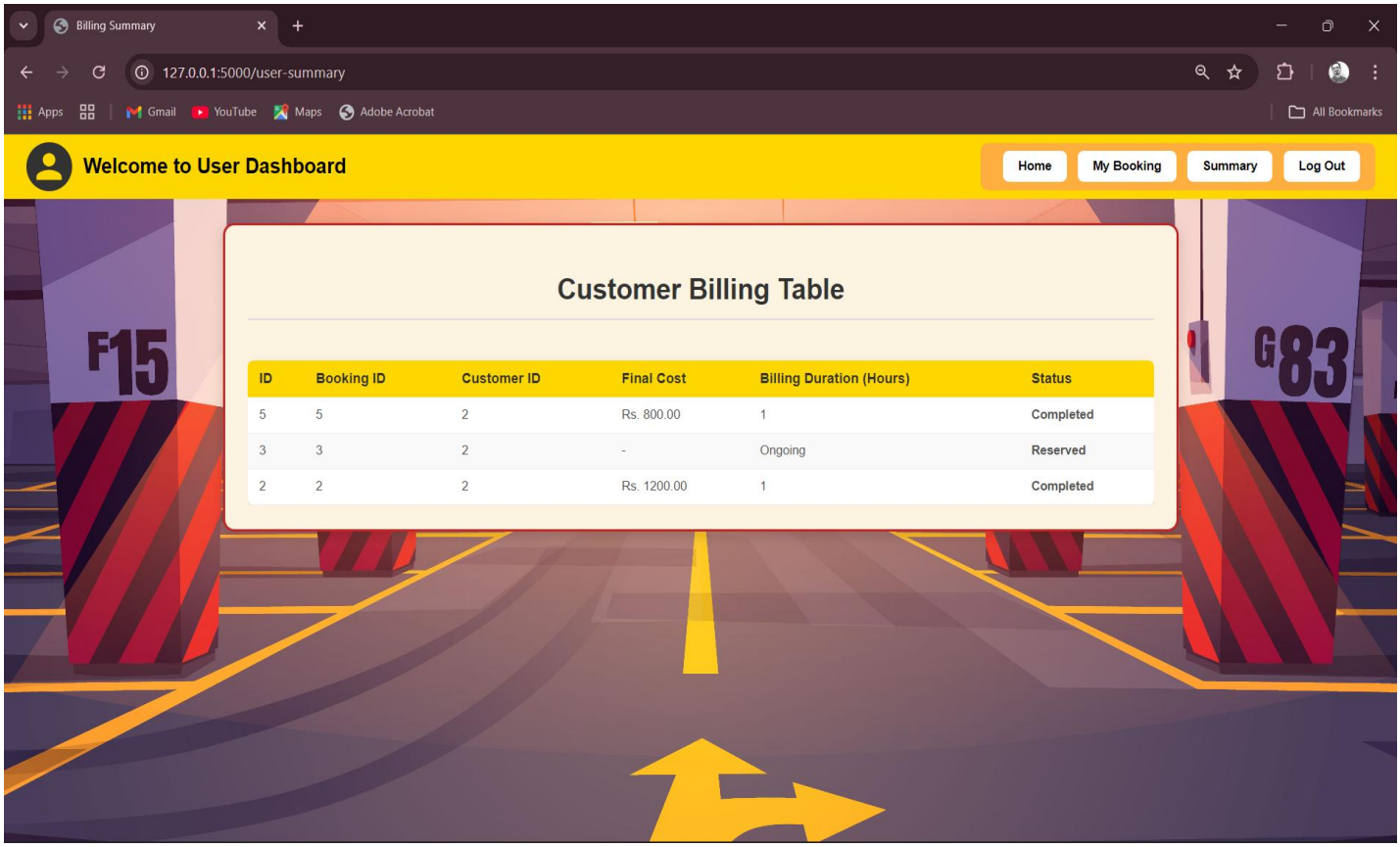

User Dashboard
127.0.0.1:5000/user-dashboard-page
Apps
Gmail
YouTube
Maps
Adobe Acrobat
All Bookmarks
Welcome Ritvik Sahore (Good Evening)
Home
My Booking
Summary
Log Out
Available Parking Lots
Hide
Parking #1
Name : Pinacio Mall
Occupied Spots : 2/40
Price Per Hour : Rs. 1200
View
Parking #3
Name : DCM Mall
Occupied Spots : 0/50
Price Per Hour : Rs. 800
View
F15
G83


User Dashboard
127.0.0.1:5000/user-dashboard-page
Apps
Gmail
YouTube
Maps
Adobe Acrobat
All Bookmarks
Welcome Ritvik Sahore (Good Evening)
Home
My Booking
Summary
Log Out
Parking #1
Select Available Spots to Book
Occupied 2/40
A-1
O
A-3
O
A-5
A-6
A-7
A-8
A-9
A-10
A-11
A-12
A-13
A-14
A-15
A-16
A-17
A-18
A-19
A-20
A-21
A-22
A-23
A-24
A-25
A-26
A-27
A-28
Close
F15
G83

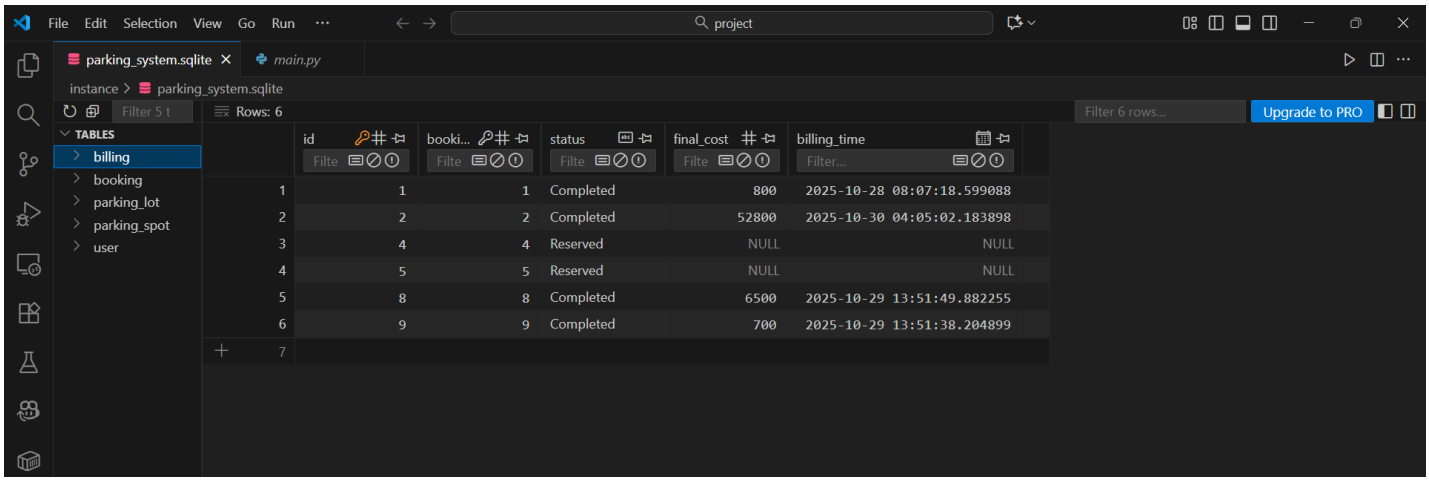
9. User My Booking Page

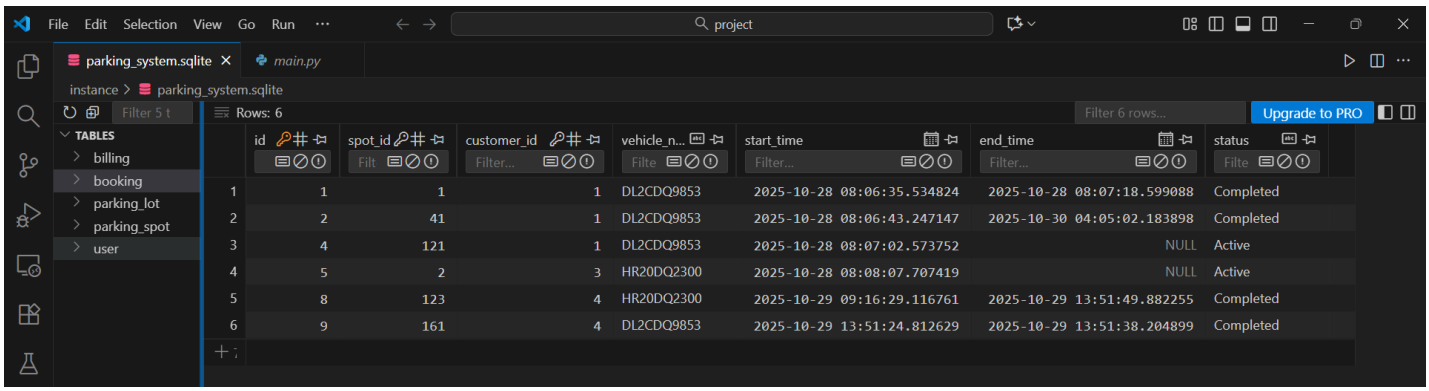


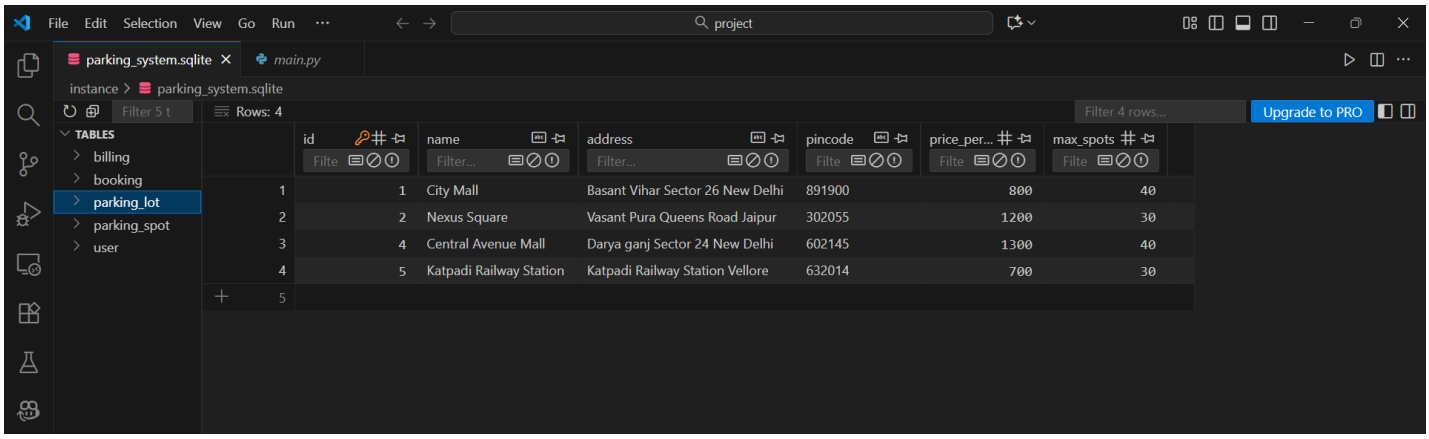
10. User Summary Page

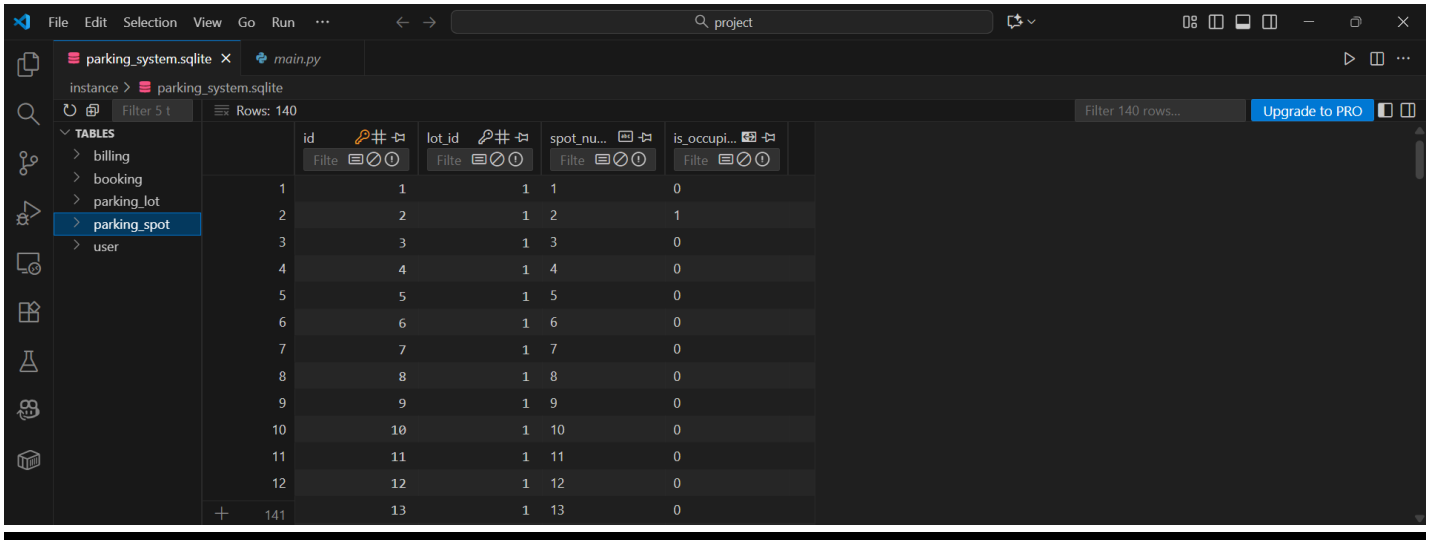


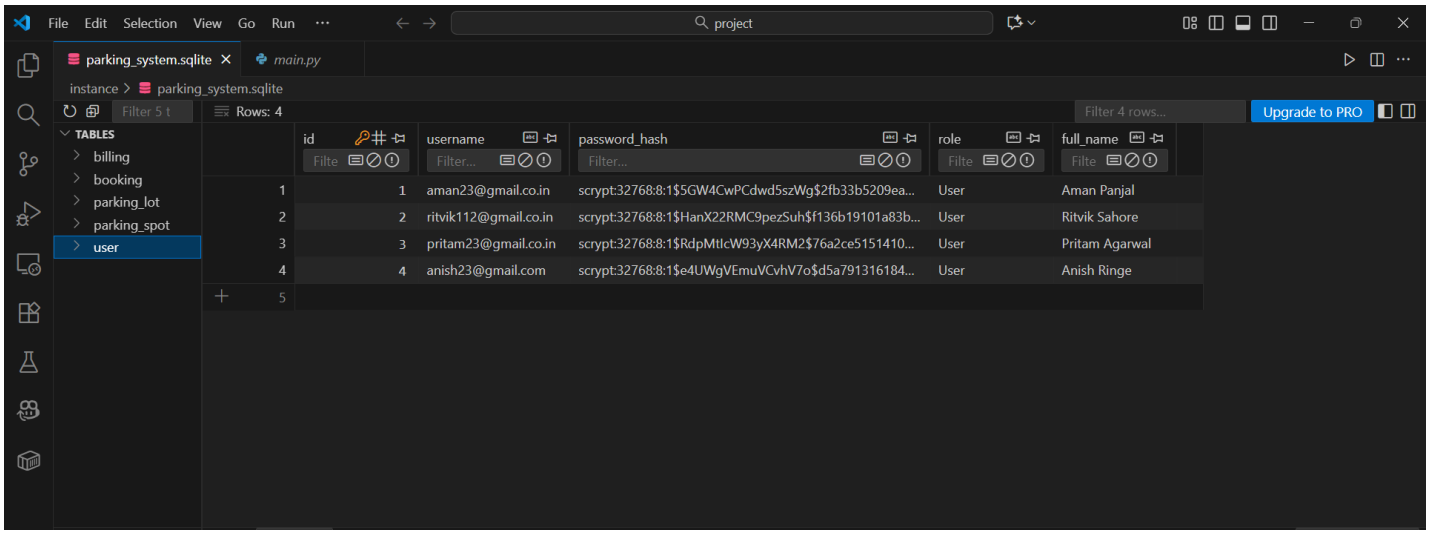
11. SqlAlchemy Database Screenshots











5. Conclusion

- The project successfully implemented a database-driven web application for vehicle parking management, meeting the primary objectives.
- A normalized relational schema was designed and realized using SQLite and Flask-SQLAlchemy, establishing clear relationships and constraints between users, lots, spots, bookings, and billing.
- The ORM facilitated the implementation of key business logic, including transactional operations for booking/releasing spots and ensuring data integrity through cascading deletes.

Thank You