



**VIT<sup>®</sup>**  
**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**

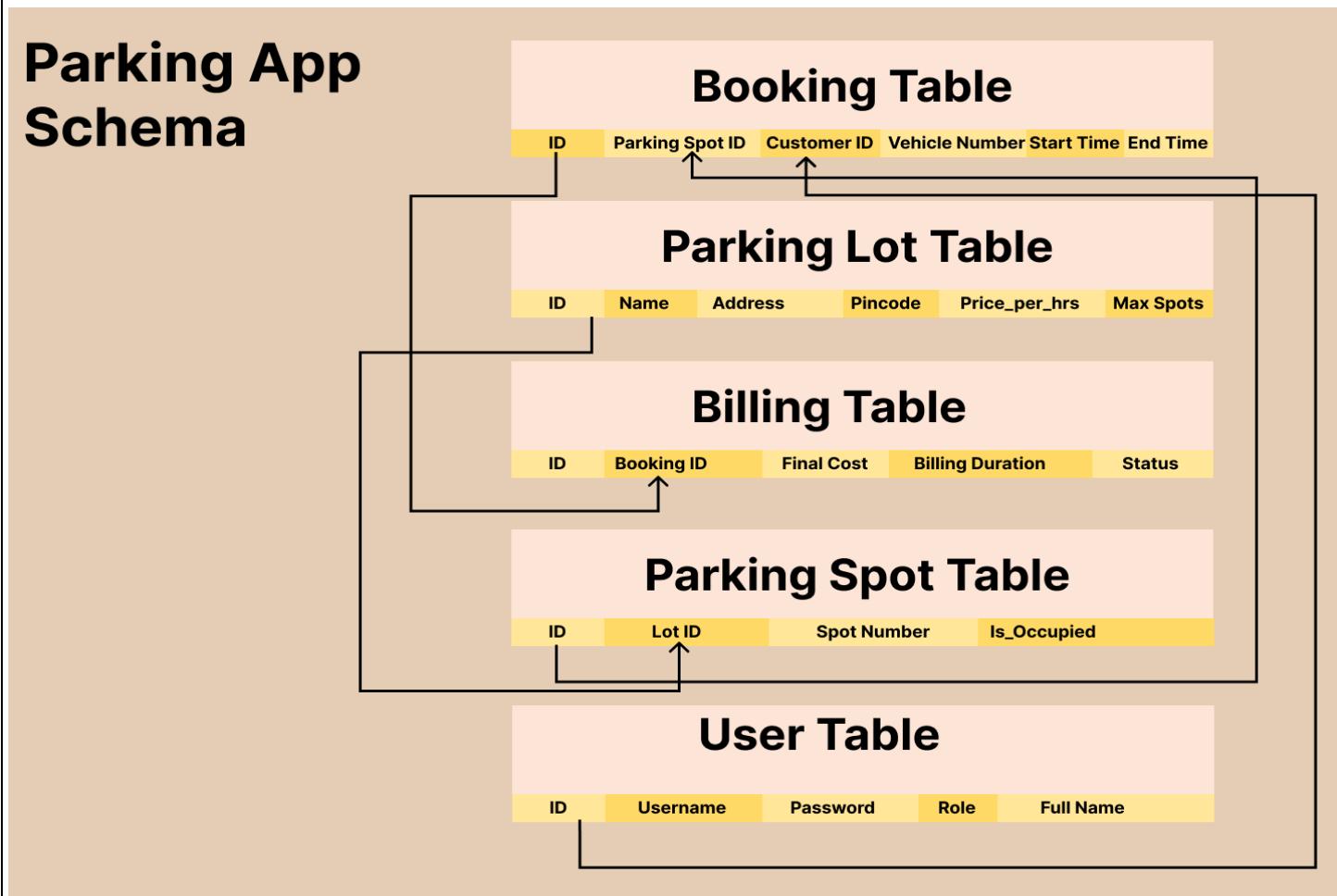
# 1. Introduction

- **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)**

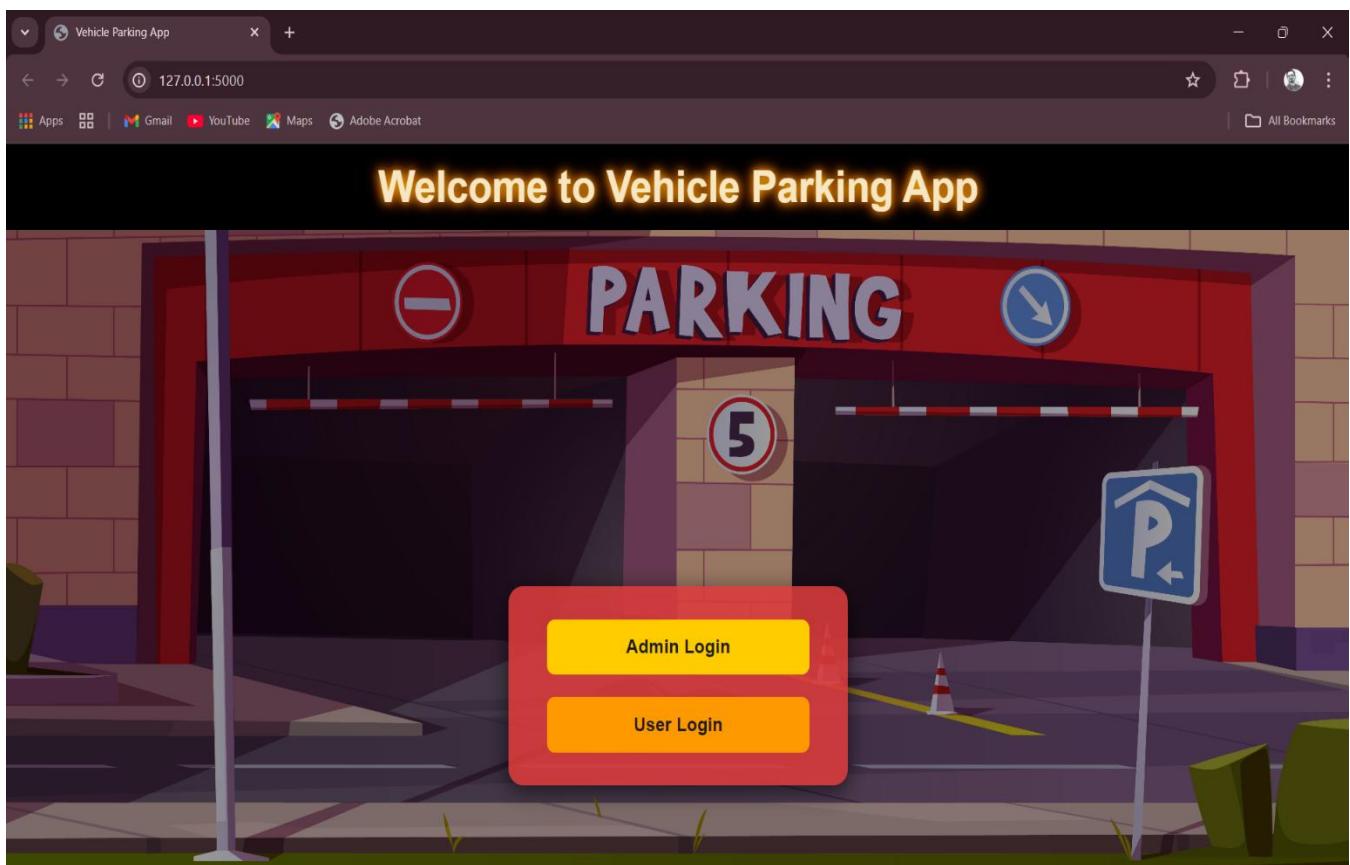
- **3.1 ORM Models (SQLAlchemy):** Python classes (User, ParkingLot, ParkingSpot, Booking, Billing) were defined using Flask-SQLAlchemy, inheriting from **db.Model**.
  - **db.Column** was used to define table columns and data types, including primary keys (`primary_key=True`) and nullability (`nullable=False`).
  - **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 :**

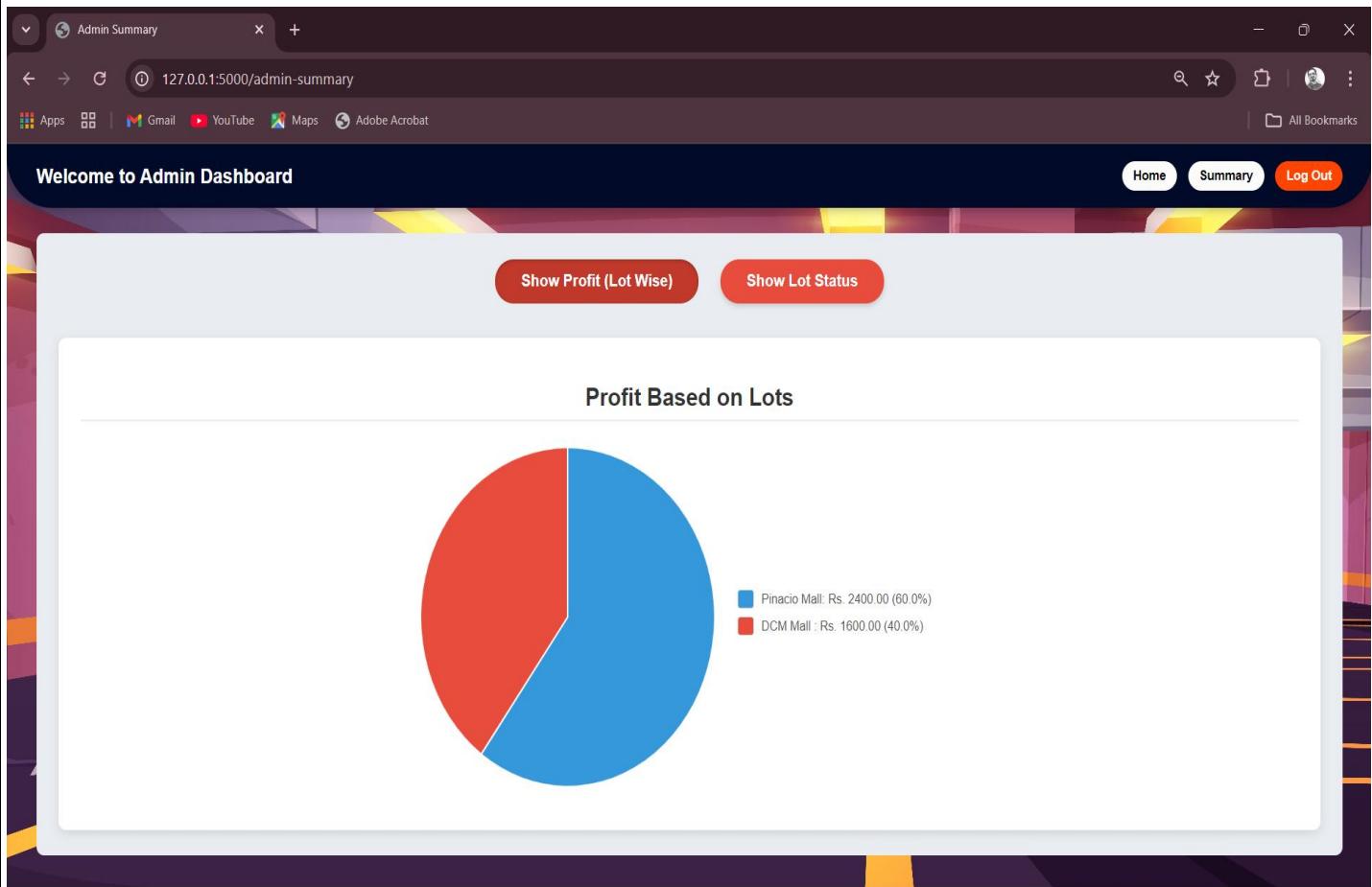
### **1. Index Page**



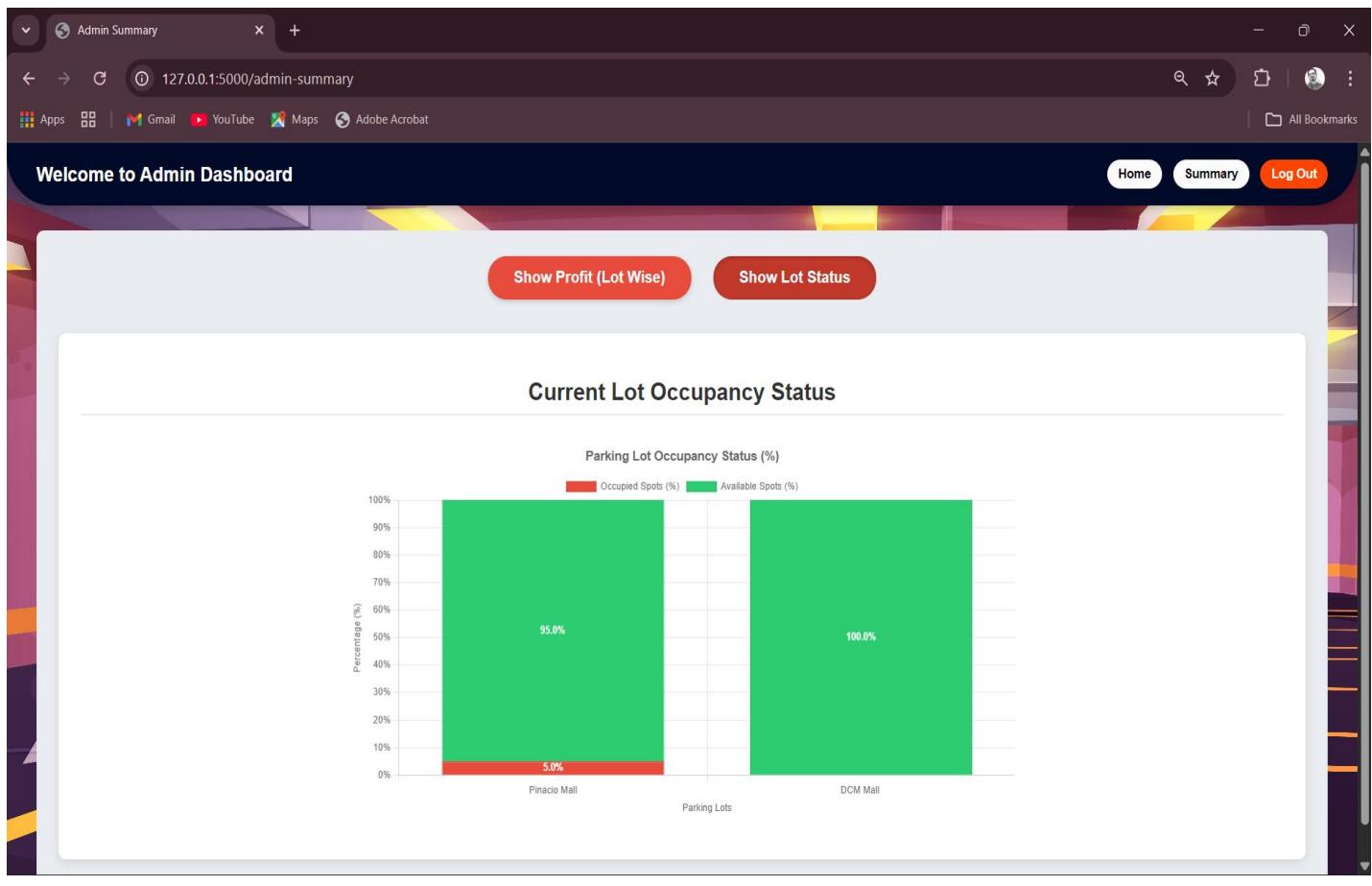
### **2. Admin Dashboard**

A screenshot of the 'Admin Dashboard' page. The URL in the address bar is 127.0.0.1:5000/admin-dashboard. The header says 'Welcome to Admin Dashboard' with buttons for 'Home', 'Summary', and 'Log Out'. Below the header is a red button with a key icon and the text '+ Add Parking Area Lot'. The main area features a large black banner with the text 'Available Parking lots'. Below the banner are two dark purple cards. The first card is for 'Parking #1' at 'Pinacio Mall', showing 'Occupied Spots : 2/40' and 'Price Per Hour : Rs. 1200', with 'Edit', 'Delete', and 'View' buttons. The second card is for 'Parking #3' at 'DCM Mall', showing 'Occupied Spots : 0/50' and 'Price Per Hour : Rs. 800', also with 'Edit', 'Delete', and 'View' buttons. The background of the dashboard shows a stylized building with a red and white striped barrier gate.

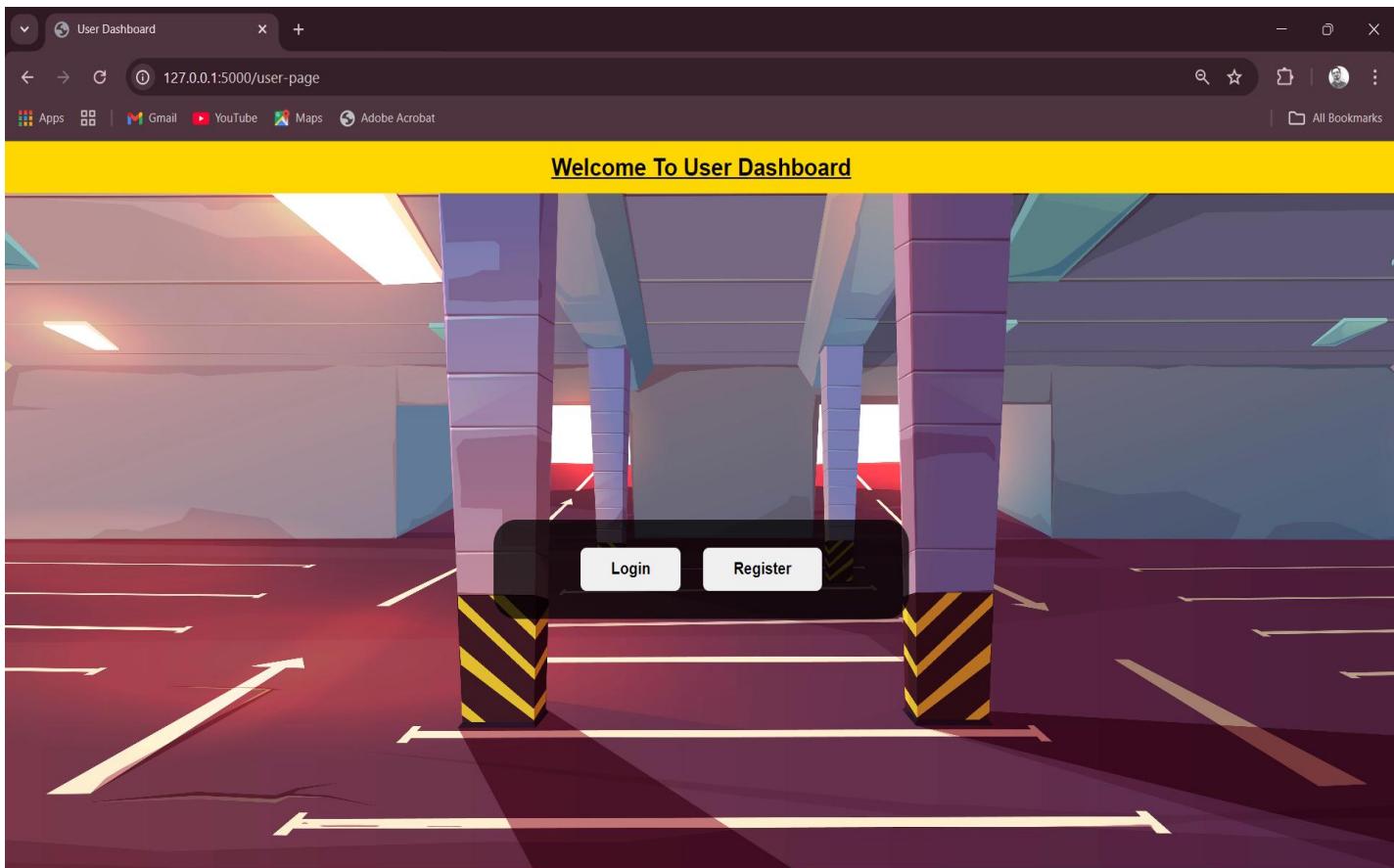
### 3. Admin Summary Page (Parking Lot Wise Profit)



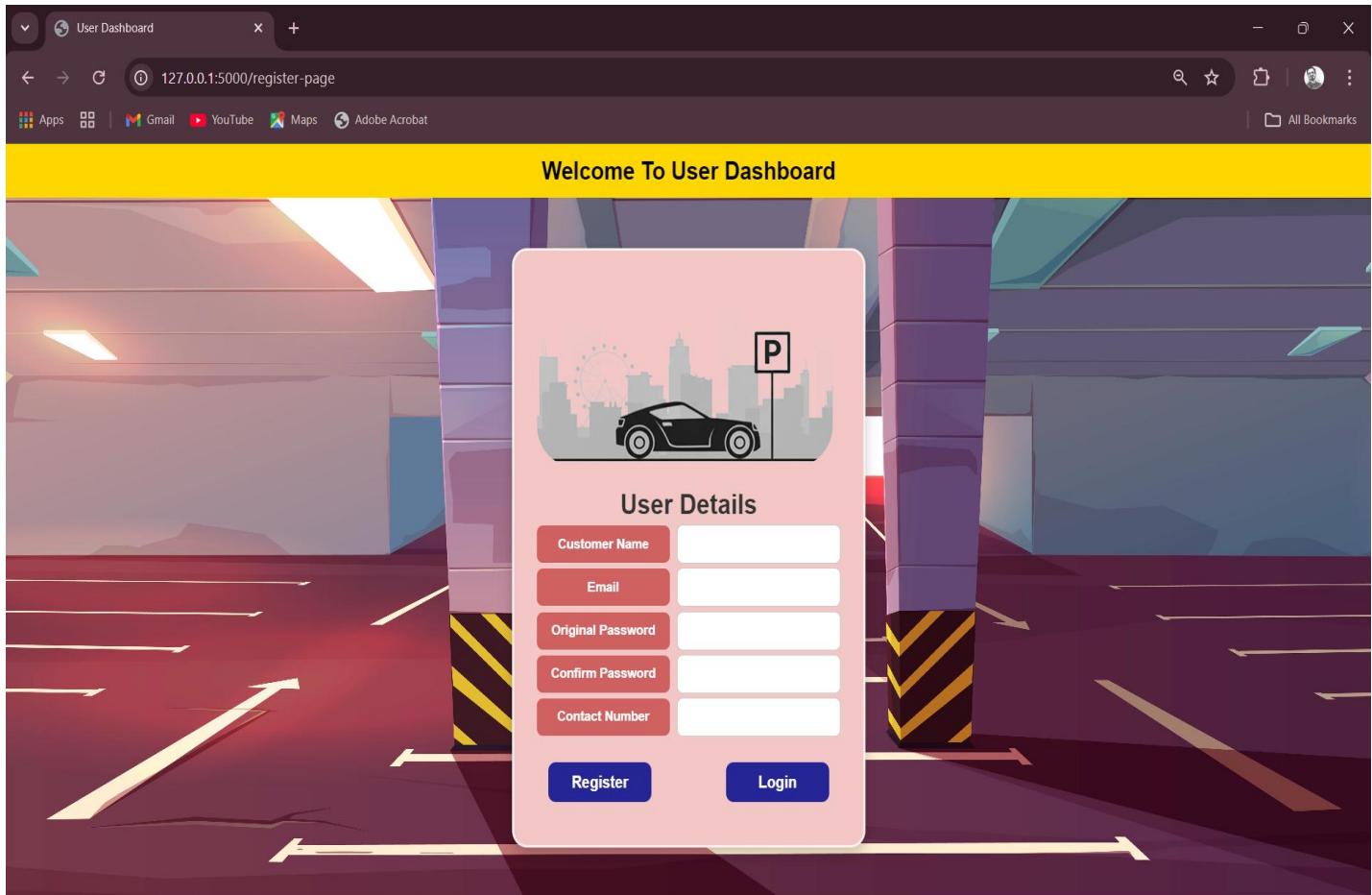
### 4. Admin Summary Page (Parking Lot Status)



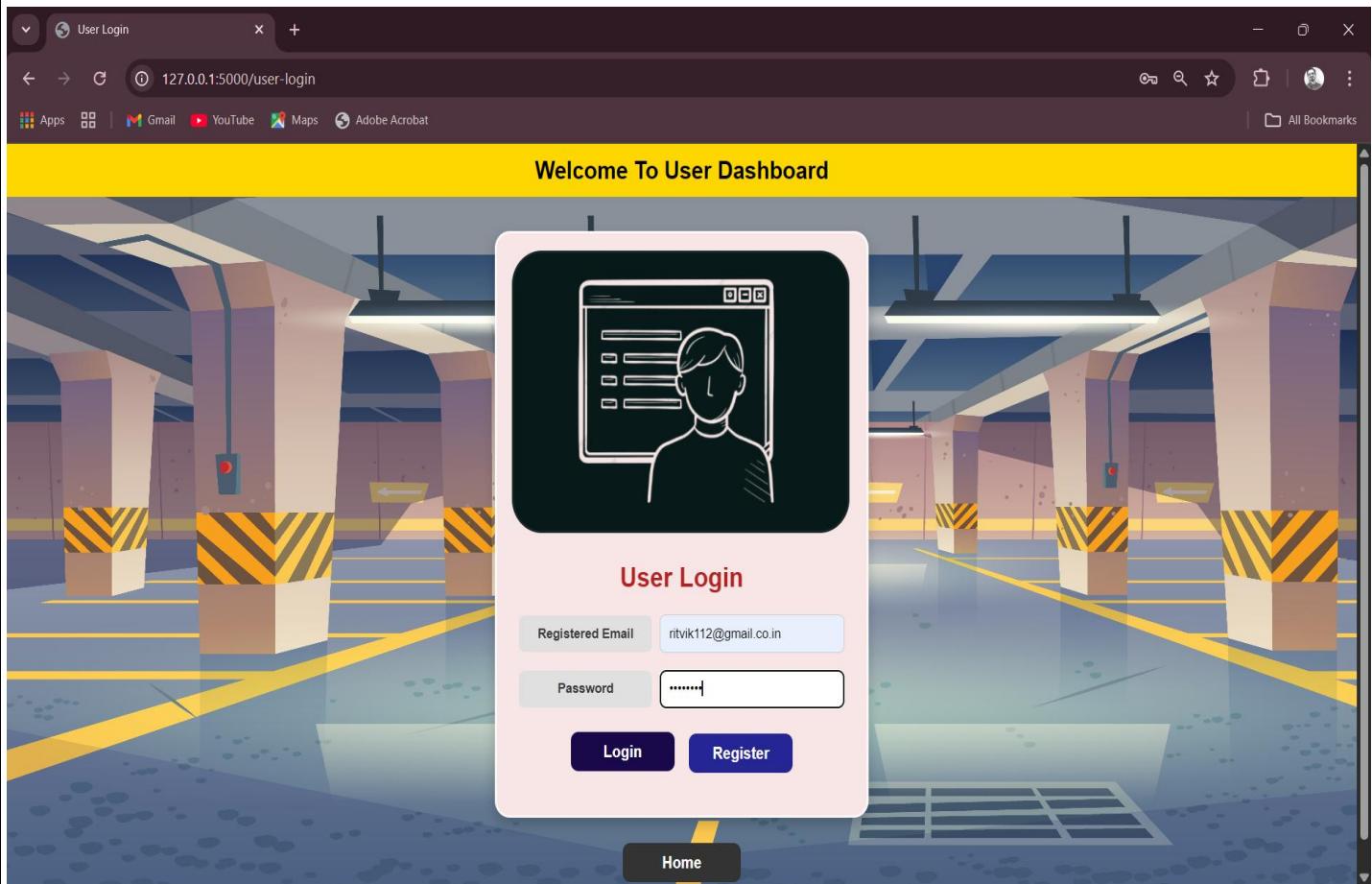
## 5. User Index Page



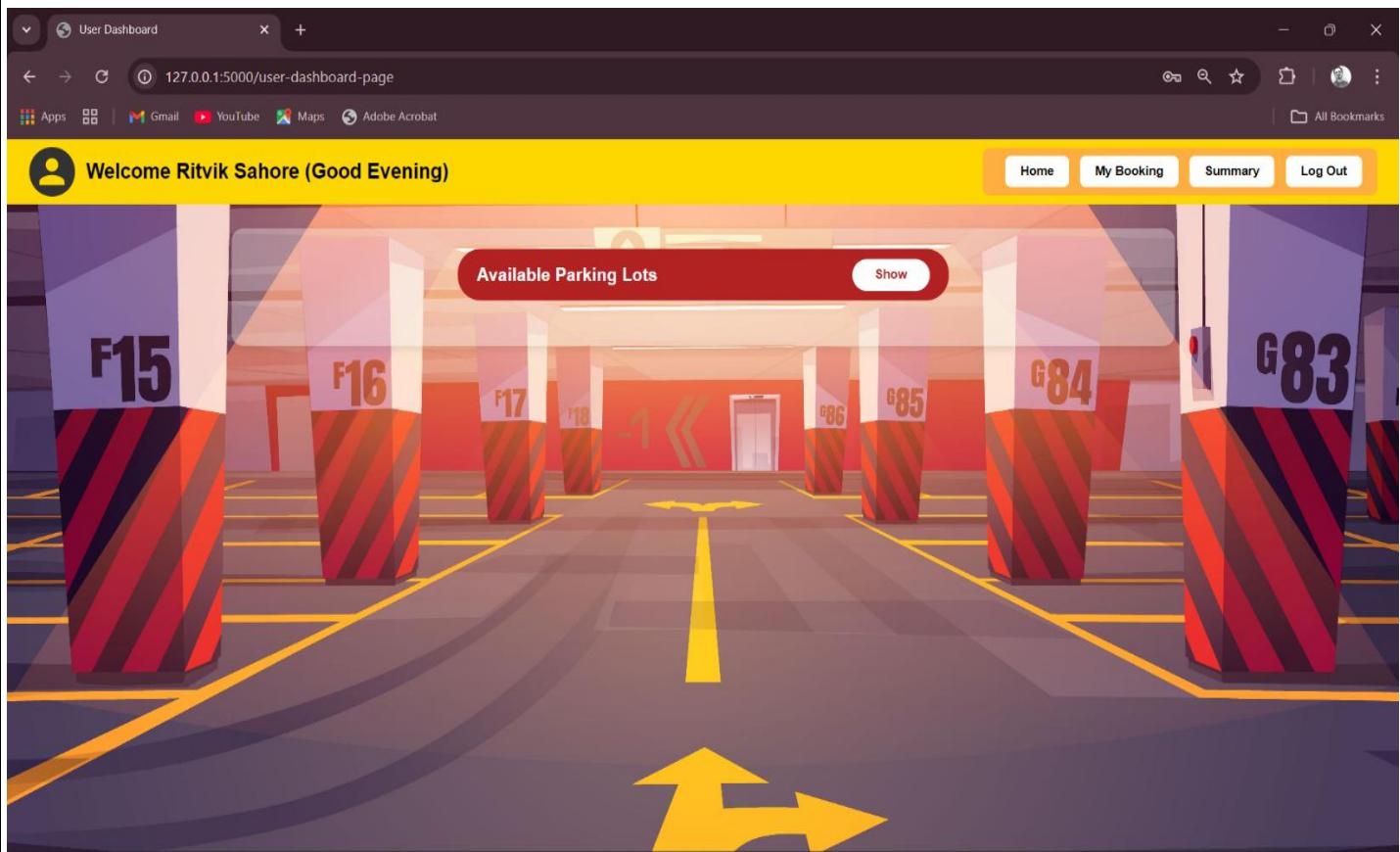
## 6. User Registration Page

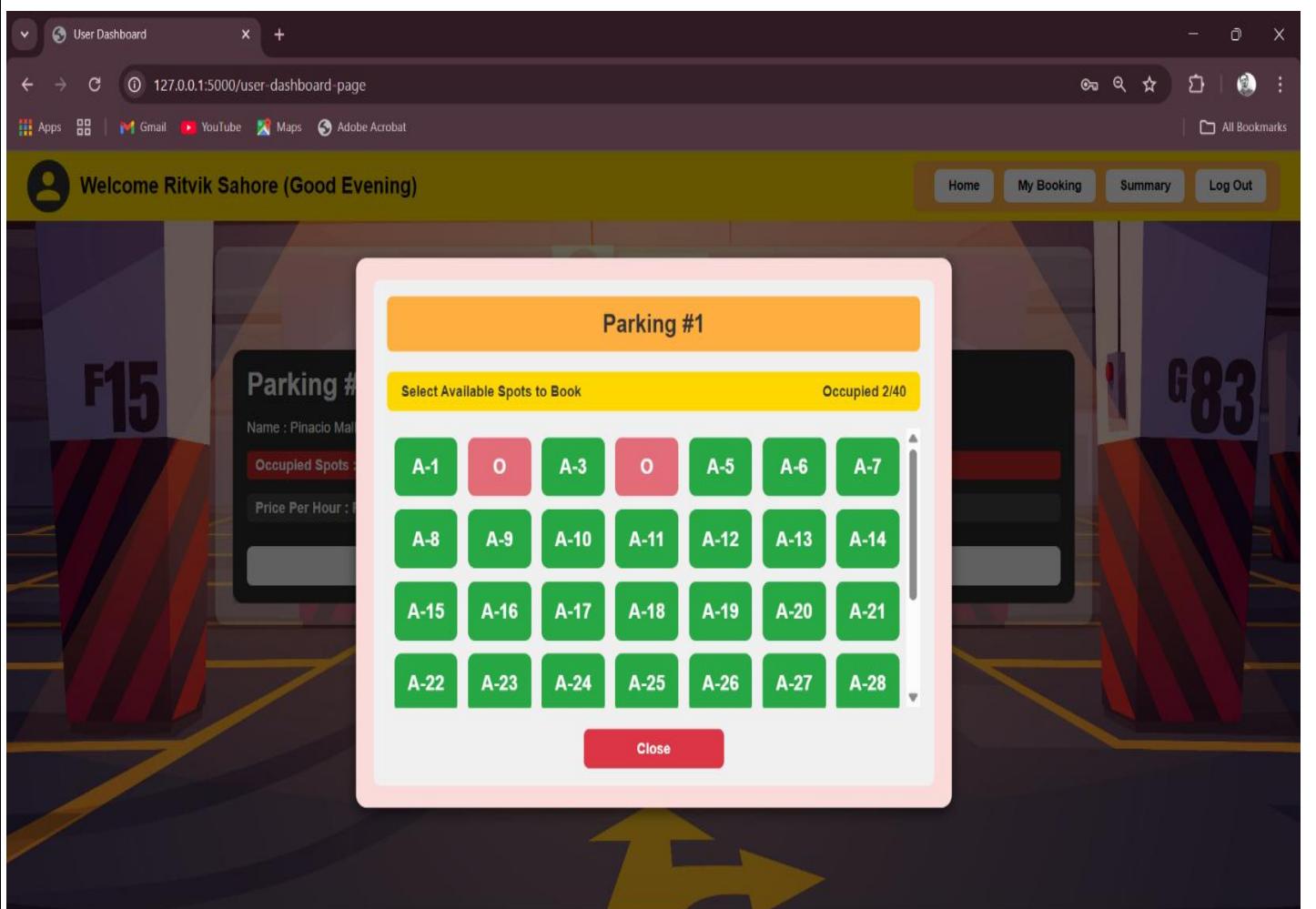
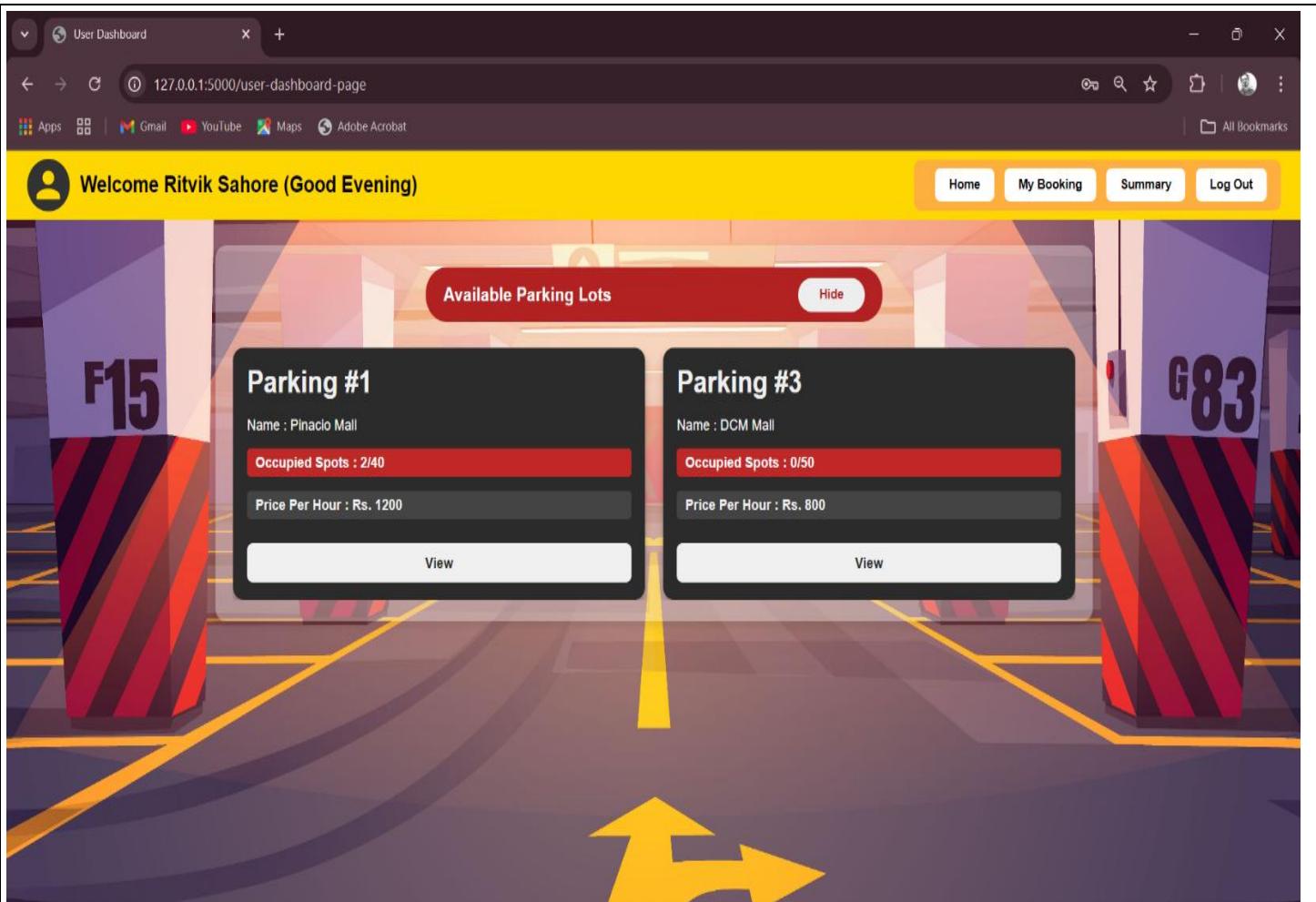


## 7. User Login Page



## 8. User Dashboard





## 9. User My Booking Page

The screenshot shows a web browser window titled "My Bookings" with the URL "127.0.0.1:5000/my-booking". The page has a yellow header bar with the text "Welcome to User Dashboard" and navigation buttons for "Home", "My Booking", "Summary", and "Log Out". The main content area is titled "My Booking" and contains a table with three rows of booking information. The table columns are "Id", "Parking Location (LotID + Lot Name)", "Vehicle No.", "Time Stamp", and "Action". The first row (Id 5) shows a booking for "Lot #3: DCM Mall" with vehicle number "DL2CDQ9853" at "10/27/2025, 12:10:18 PM" with a green "Parked Out" button. The second row (Id 3) shows a booking for "Lot #1: Pinacio Mall" with vehicle number "DL2CDQ9853" at "10/27/2025, 12:09:51 PM" with a red "Release" button. The third row (Id 2) shows a booking for "Lot #1: Pinacio Mall" with vehicle number "DL2CDQ9853" at "10/27/2025, 12:09:40 PM" with a green "Parked Out" button. The background of the page features a stylized parking garage with red and purple lighting.

ID	Parking Location (LotID + Lot Name)	Vehicle No.	Time Stamp	Action
5	Lot #3: DCM Mall	DL2CDQ9853	10/27/2025, 12:10:18 PM	Parked Out
3	Lot #1: Pinacio Mall	DL2CDQ9853	10/27/2025, 12:09:51 PM	Release
2	Lot #1: Pinacio Mall	DL2CDQ9853	10/27/2025, 12:09:40 PM	Parked Out

## 10. User Summary Page

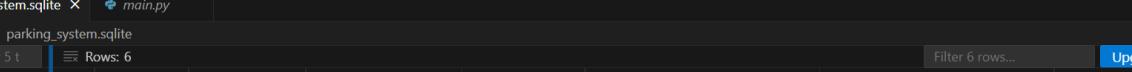
The screenshot shows a web browser window titled "Billing Summary" with the URL "127.0.0.1:5000/user-summary". The page has a yellow header bar with the text "Welcome to User Dashboard" and navigation buttons for "Home", "My Booking", "Summary", and "Log Out". The main content area is titled "Customer Billing Table" and contains a table with three rows of billing information. The table columns are "ID", "Booking ID", "Customer ID", "Final Cost", "Billing Duration (Hours)", and "Status". The first row (ID 5) shows a booking with "Booking ID 5", "Customer ID 2", "Rs. 800.00", "1" hour, and "Completed" status. The second row (ID 3) shows a booking with "Booking ID 3", "Customer ID 2", "-", "Ongoing" status. The third row (ID 2) shows a booking with "Booking ID 2", "Customer ID 2", "Rs. 1200.00", "1" hour, and "Completed" status. The background of the page features a stylized parking garage with red and purple lighting.

ID	Booking ID	Customer ID	Final Cost	Billing Duration (Hours)	Status
5	5	2	Rs. 800.00	1	Completed
3	3	2	-	Ongoing	Reserved
2	2	2	Rs. 1200.00	1	Completed

# 11. SQLAlchemy Database Screenshots

The screenshot shows a DBeaver interface with a SQLite database named 'parking\_system.sqlite'. The 'billing' table is selected, displaying the following data:

	id	booki...	status	final_cost	billing_time
	1	1	Completed	800	2025-10-28 08:07:18.599088
	2	2	Completed	52800	2025-10-30 04:05:02.183898
	3	4	Reserved	NULL	NULL
	4	5	Reserved	NULL	NULL
	5	8	Completed	6500	2025-10-29 13:51:49.882255
	6	9	Completed	700	2025-10-29 13:51:38.204899
	+	7			



The screenshot shows a SQLite database browser interface with the following details:

- Database:** parking\_system.sqlite
- Tables:**
  - booking:** Contains 6 rows of data.
  - user:** Contains 1 row of data.
  - billing:** Contains 1 row of data.
  - parking\_lot:** Contains 1 row of data.
  - parking\_spot:** Contains 1 row of data.

**booking Table Data:**

id	spot_id	customer_id	vehicle_no	start_time	end_time	status
1	1	1	DL2CDQ9853	2025-10-28 08:06:35.534824	2025-10-28 08:07:18.599088	Completed
2	2	41	DL2CDQ9853	2025-10-28 08:06:43.247147	2025-10-30 04:05:02.183898	Completed
3	4	121	DL2CDQ9853	2025-10-28 08:07:02.573752		NULL
4	5	2	HR2DDQ2300	2025-10-28 08:08:07.707419		NULL
5	8	123	HR2DDQ2300	2025-10-29 09:16:29.116761	2025-10-29 13:51:49.882255	Completed
6	9	161	DL2CDQ9853	2025-10-29 13:51:24.812629	2025-10-29 13:51:38.204899	Completed

The screenshot shows a SQLite database browser interface with the following details:

- File Edit Selection View Go Run ...**
- project** search bar
- parking\_system.sqlite** is the current database.
- main.py** is the current script.
- instance > parking\_system.sqlite**
- Tables:** **billing**, **booking**, **parking\_lot** (selected), **parking\_spot**, **user**.
- parking\_lot Table Data:**

id	name	address	pincode	price_per...	max_spots
1	City Mall	Basant Vihar Sector 26 New Delhi	891900	800	40
2	Nexus Square	Vasant Pura Queens Road Jaipur	302055	1200	30
3	Central Avenue Mall	Darya ganj Sector 24 New Delhi	602145	1300	40
4	Katpadi Railway Station	Katpadi Railway Station Vellore	632014	700	30
5					
- Toolbar:** Filter 5 t, Rows: 4, Filter 4 rows..., Upgrade to PRO.

The screenshot shows the DBeaver interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Toolbar:** Back, Forward, Project search bar, and various icons.
- Project Tree:** Shows the database instance as "parking\_system.sqlite" and the main file as "main.py".
- Table List:** A tree view under "TABLES" showing tables: billing, booking, parking\_lot, parking\_spot, and user. The "parking\_spot" table is currently selected.
- Table View:** A grid showing the "parking\_spot" table with 140 rows. The columns are: id, lot\_id, spot\_number, and is\_occupied. The data shows 141 rows from 1 to 141.
- Bottom Bar:** Filter 140 rows..., Upgrade to PRO, and other icons.

	id	lot_id	spot_number	is_occupied
1	1	1	1	0
2	2	1	2	1
3	3	1	3	0
4	4	1	4	0
5	5	1	5	0
6	6	1	6	0
7	7	1	7	0
8	8	1	8	0
9	9	1	9	0
10	10	1	10	0
11	11	1	11	0
12	12	1	12	0
141	13	1	13	0

	id	username	password_hash	role	full_name
	1	aman23@gmail.co.in	scrypt32768:8:1\$5GW4CwPCdwd5szWg\$2fb33b5209ea...	User	Aman Panjal
	2	ritvik12@gmail.co.in	scrypt32768:8:1\$HanX22RMC9pezSuh\$136b19101a83b...	User	Ritvik Sahore
	3	pritam23@gmail.co.in	scrypt32768:8:1\$RdpMtIcW93yX4RM2\$76a2ce5151410...	User	Pritam Agarwal
	4	anish23@gmail.com	scrypt32768:8:1\$e4UWgYEmuVChV7o\$d5a791316184...	User	Anish Ringe
	5				

## 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