



**SE3050 – Enterprise Software Development – 2025**

**BSc (Hons) in Software Engineering**

## **Assignment**

**Group: Y4S1-SE-WE-01**

**Group ID: 10**

Student ID	Name	Email
IT22125248	Annesiyani Srikanthan	it22125248@my.sliit.lk
IT22099518	Hemapriya H. A . N. S	it22099518@my.sliit.lk
IT22167378	H.I.G.Amith Hasintha	it22167378@my.sliit.lk
IT22083296	E.K.K.Tharushi Nethmini Edirisinghe	it22083296@my.sliit.lk

## Contribution

Student ID	Name	Contribution
IT22125248	Annesiyani Srikanthan	<ul style="list-style-type: none"><li>• Build and integrate APIs for users and bookings from the Web Application side (React).</li><li>• Handles Backoffice &amp; Operator operations.</li></ul>
IT22099518	Hemapriya H. A . N. S	<ul style="list-style-type: none"><li>• Implement EV Owner-related APIs on the Android app.</li><li>• Works with SQLite sync and booking logic.</li></ul>
IT22167378	H.I.G.Amith Hasintha	<ul style="list-style-type: none"><li>• Lead backend setup.</li><li>• Implement JWT-based authentication.</li><li>• Role and notification management, and security logic.</li></ul>
IT22083296	E.K.K.Tharushi Nethmini Edirisinghe	<ul style="list-style-type: none"><li>• Handle station APIs and DB schema design.</li><li>• Overall documentation and reporting.</li></ul>

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>3</b>
<b>2</b>	<b>System Overview Diagram .....</b>	<b>4</b>
<b>3</b>	<b>Use Case Diagram .....</b>	<b>5</b>
<b>4</b>	<b>Data Flow Diagram (DFD) .....</b>	<b>6</b>
4.1	DFD0.....	6
4.2	DFD1.....	7
<b>5</b>	<b>Database Design .....</b>	<b>8</b>
<b>6</b>	<b>UI Screenshots .....</b>	<b>11</b>
6.1	EV Charging Mobile Application UI Screenshots.....	11
6.2	EV Charging Web Application UI Screenshots.....	20
<b>7</b>	<b>Source Codes.....</b>	<b>25</b>
7.1	EV Charging Mobile Application Codes.....	25
7.2	EV Charging Web Application Codes.....	175
7.3	EV Charging Backend Codes .....	312
<b>8</b>	<b>Challenges Faced.....</b>	<b>496</b>

# 1 Introduction

**The EV Charging Station Booking System** is developed to provide a seamless and efficient platform for managing electric vehicle (EV) charging operations. With the rapid growth of EV adoption, the system ensures that EV owners, charging station operators, and back-office staff can collaborate within a centralized and reliable environment.

The solution follows a client–server architecture, where a central web service built on C# Web API handles all business logic and data processing. The server connects to a NoSQL database hosted on IIS, ensuring scalability, faster access, and high availability. Both the web application and mobile application act purely as user interfaces that interact with the web service for data retrieval and updates.

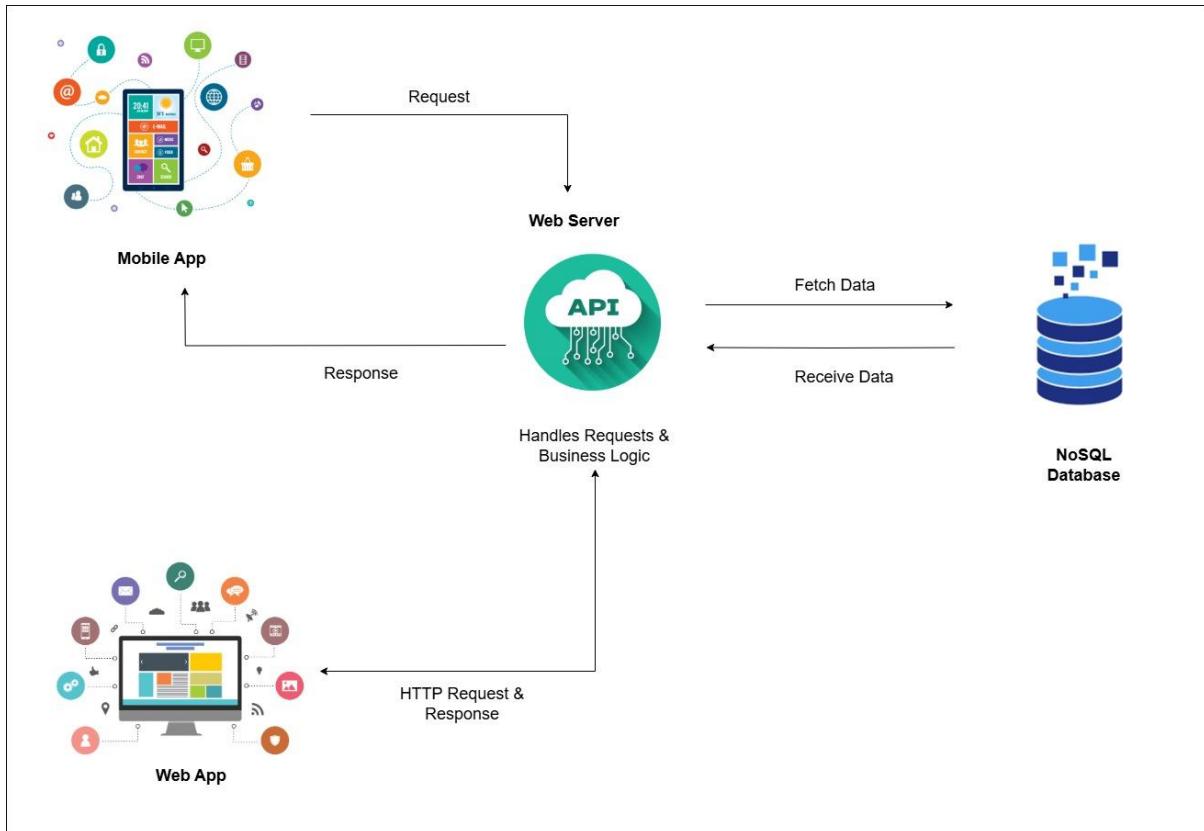
The web application primarily serves back-office users and station operators. Back-office users are responsible for creating and managing system users, EV owner accounts, and charging station details. They also monitor operations and can activate or deactivate accounts and stations. Station operators use the web or mobile interface to update slot availability, monitor bookings, and finalize charging sessions once services are delivered.

The mobile application is designed for EV owners and station operators. EV owners can create accounts, book charging slots, modify or cancel reservations, and view past charging history. Bookings generate a QR code, which station operators scan to verify and confirm the service. The mobile app also integrates Google Maps API to display nearby charging stations, enhancing convenience for users. Operators can log in through the mobile app to validate reservations and confirm service completion.

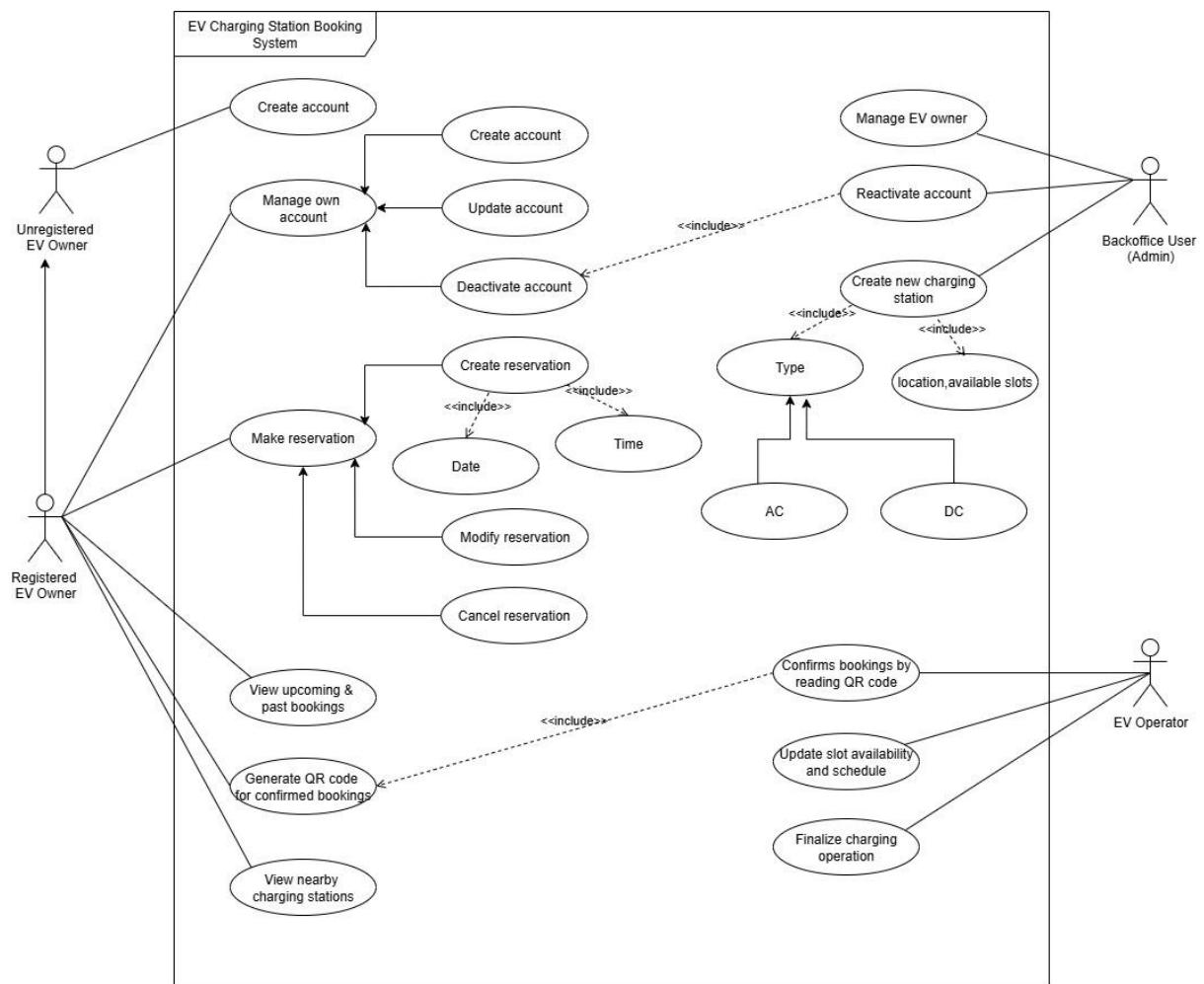
Key processes are organized into three main areas: User Management (managing back-office, operators, and EV owners), Station Management (managing station details, schedules, and slots), and Booking Management (creating, updating, and canceling reservations). This ensures proper role-based access control and smooth coordination between stakeholders.

Overall, the EV Charging Station Booking System delivers a modern, scalable, and secure platform that improves the EV charging experience for all stakeholders while ensuring smooth system administration and reliable data flow across applications.

## 2 System Overview Diagram

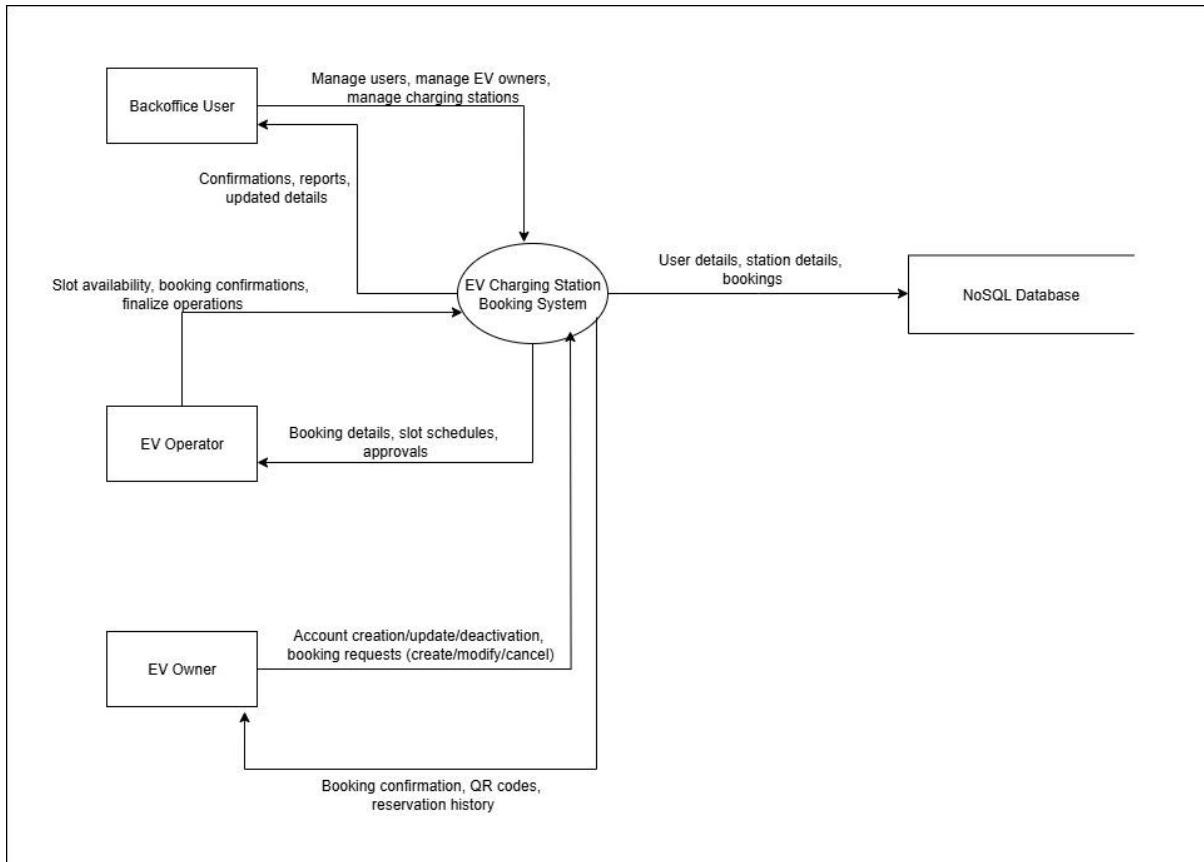


### 3 Use Case Diagram

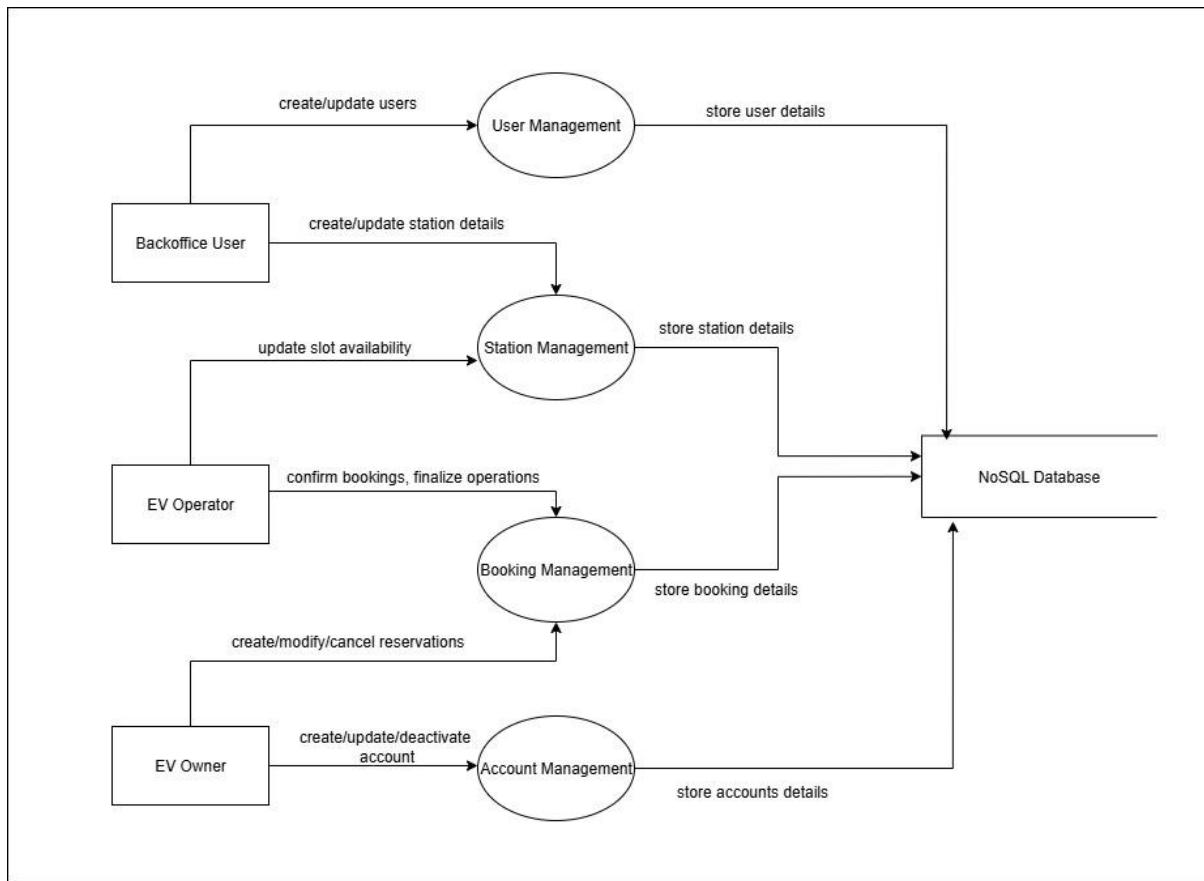


## 4 Data Flow Diagram (DFD)

4.1 DFDO



## 4.2 DFD1



# 5 Database Design

## 1. User Collection

Handles different user roles: Backoffice user(admin), EV operator, EV owner

- NIC (String)
- firstName (String)
- lastName (String)
- Email (String)
- Password (String)
- Role (Enum: Backoffice user, EV operator, EV owner)
- isActive (bool)
  - Whether the account is currently active.
- phoneNumber (String)
- createdAt (DateTime)
  - Timestamp when the account was created.
- updatedAt (DateTime)
  - Timestamp when the account was last updated.

## 2. Booking Collection

- ownerNIC (String)
- stationId (String)
- startTime (DateTime)
- endTime (DateTime)
- Status (Enum: Active, Confirmed, Completed, Cancelled, NoShow)
- QRcode (String)
- totalAmount (Decimal, e.g., 4.5)
- createdAt (DateTime)
  - Timestamp when the booking was created.
- updatedAt (DateTime)
  - Timestamp when the booking was last updated.
- confirmedAt (DateTime)
  - Timestamp when the booking was confirmed (nullable).
- cancelledAt (DateTime)
  - Timestamp when the booking was cancelled (nullable).

### 3. Charging Station Collection

- Name (String)
- Location (String)
- Type (Enum: AC, DC)
  - Type of station.
- totalSlots (int)
  - Total number of charging slots available at the station.
- availableSlots (int)
- Status (Enum: Active, Inactive, Maintenance)
- pricePerHour (Decimal, e.g., 4.5)
  - Price per hour for charging at this station.
- createdAt (DateTime)
  - Timestamp when the station record was created.
- updatedAt (DateTime)
  - Timestamp when the station record was last updated.

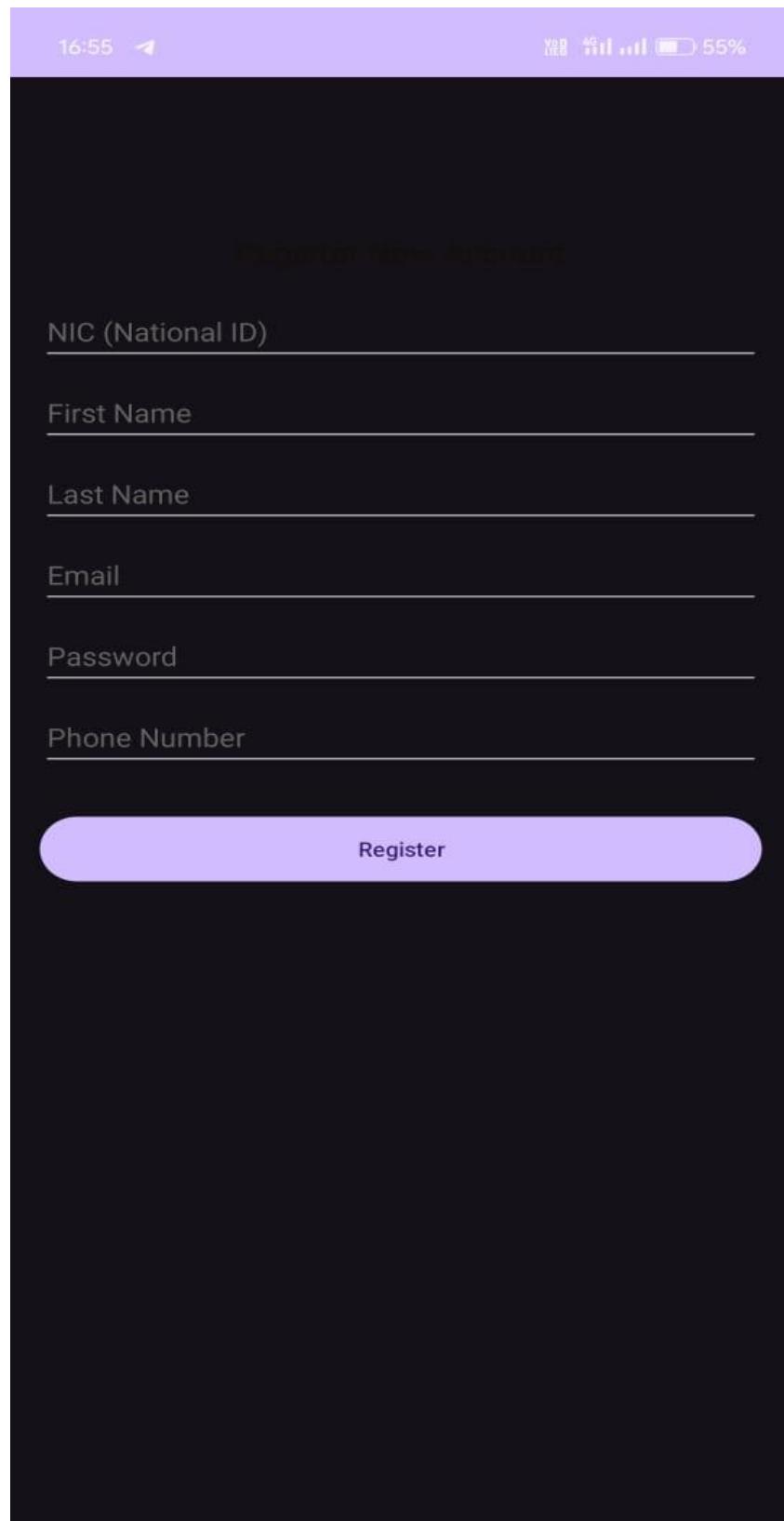
### 4. Notification Collection

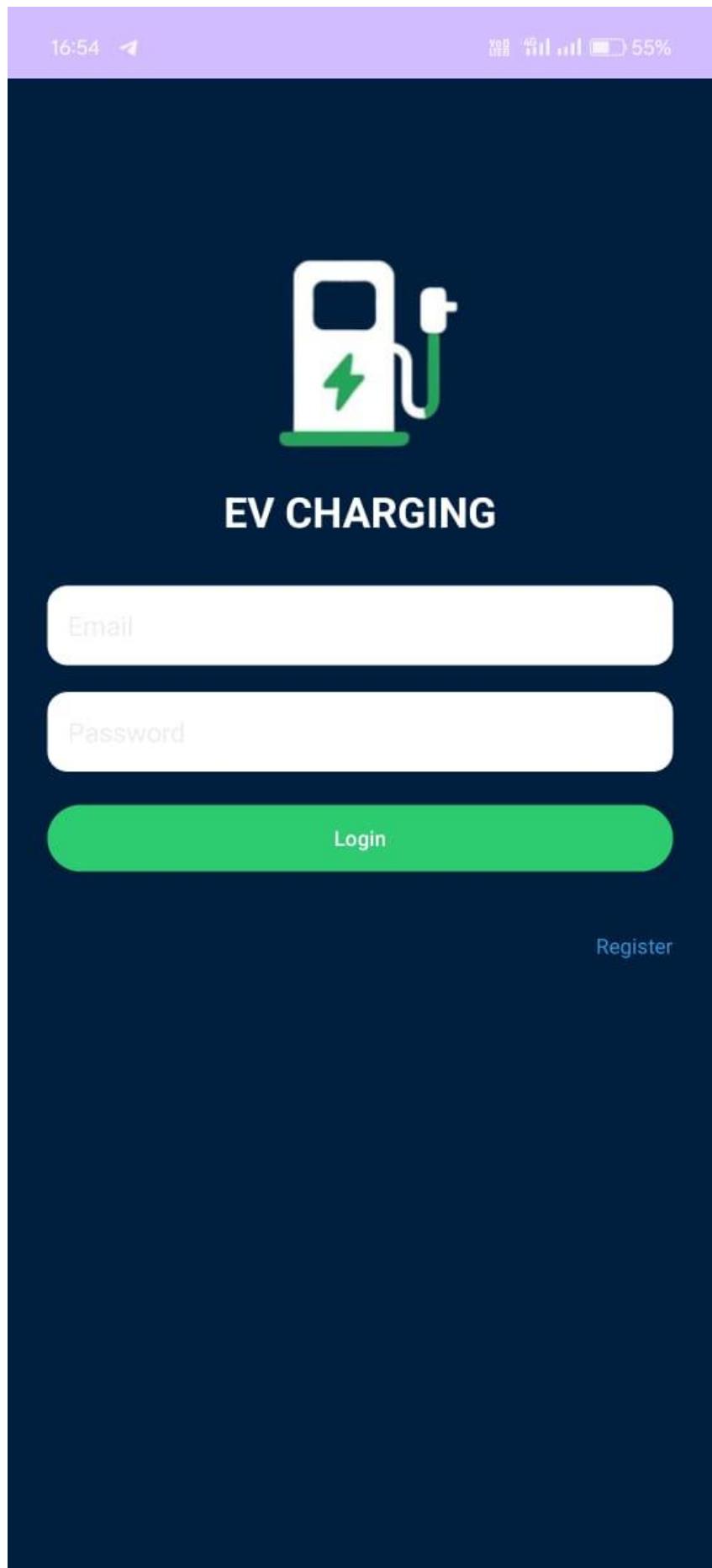
- recipientNIC (String)
- Title (String)
  - Short title of the notification
- Message (String)
  - Body of the notification.
- Type (Enum)
  - Type of notification.
- relatedEntityId (String)
  - ID of the related entity (e.g., booking ID or station ID).
- relatedEntityType (String)
  - Type of the related entity (Booking, Station)
- isRead (bool)
  - Whether the notification has been read by the recipient.
- isDelivered (bool)
  - Whether the notification has been successfully delivered.
- Priority (Enum: Low, Normal, High, Critical)
  - Notification importance level.
- createdAt (DateTime)
  - Timestamp when the notification was created.
- readAt (DateTime)
  - Timestamp when the notification was read.
- deliveredAt (DateTime)
  - Timestamp when the notification was delivered.

- `expiresAt (DateTime)`
  - Expiry date for time-sensitive notifications.
- `metadata (Dictionary<string,object>)`
  - Optional extra data related to the notification (flexible structure).

## 6 UI Screenshots

### 6.1 EV Charging Mobile Application UI Screenshots





22:25 82%

## Dashboard

0 Pending 0 Approved 0 Nearby

### Upcoming Bookings

**Station:** 68e673760e78f5875a255240 **CONFIRMED**

Oct 12, 2025 | 04:34 PM - 04:34 PM

### Nearby Stations

A map of Sri Lanka with various stations marked by green circles and labeled with codes such as A1, A2, A3, A4, A5, A6, A7, A9, A10, A21, A25, A27, A17, A18, E01, and E04. Major cities like Colombo, Negombo, Kandy, and Galle are also labeled. The map includes a legend for Google and a scale bar.

Google

Dashboard

22:24 82%

## Dashboard



 Station: 68e673760e78f5875a255240 CONFIRMED

 Oct 12, 2025 | 04:34 PM - 04:34 PM



 Station: 68e673760e78f5875a255240 CONFIRMED

 Oct 11, 2025 | 11:21 AM - 11:20 AM



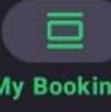
 Station: 68e673770e78f5875a255243 CANCELLED

 Oct 11, 2025 | 11:21 AM - 11:20 AM

 Station: 68e673770e78f5875a255242 ACTIVE

 Oct 11, 2025 | 11:21 AM - 11:20 AM

[CANCEL BOOKING](#)

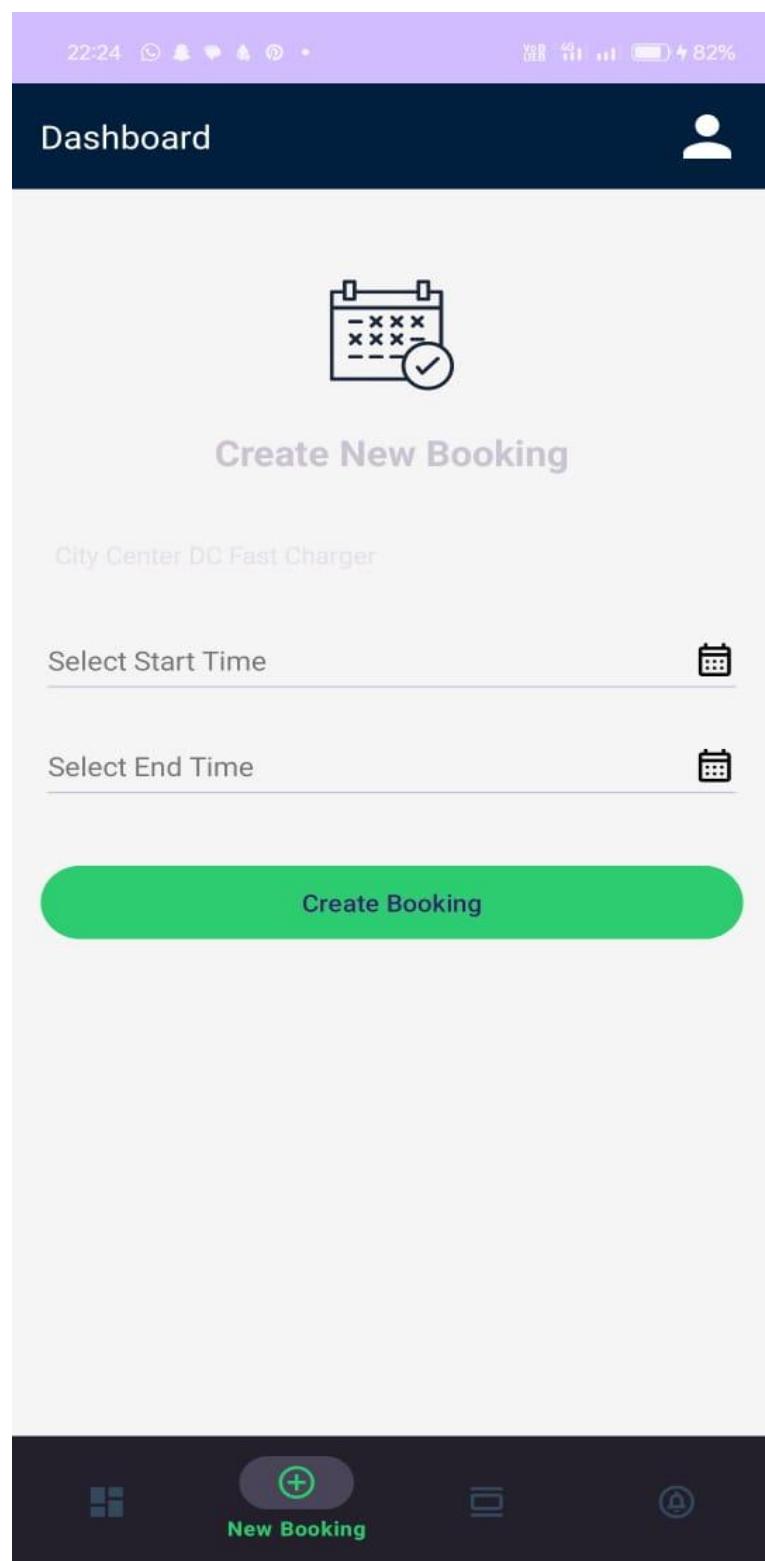
   

The image shows a smartphone screen displaying a mobile application dashboard. At the top, there is a purple status bar with the time "22:24", several icons (including signal strength, battery level at 82%, and a power symbol), and a "Dashboard" title. Below the status bar is a dark blue header with the word "Dashboard" and a user profile icon.

The main content area consists of seven rectangular notification cards, each containing a blue circular icon with a white bell symbol and a title followed by a descriptive message:

- Booking Confirmed**  
Your booking has been confirmed by the operator
- Booking Confirmed**  
Your booking at City Center DC Fast Charger has been confirmed for 2025-10-12 16:34 - 2025-10-13 16:34. Your charging session is ready!
- Booking Cancelled**  
Your booking at University Campus Charger has been cancelled. Reason: Cancelled by user. You can make a new booking anytime.
- Booking Cancelled**  
Your booking at City Center DC Fast Charger has been cancelled. Reason: adghfghfhgf
- Booking Cancelled**  
Your booking at City Center DC Fast Charger has been cancelled. Reason: Cancelled by station operator - adghfghfhgf. You can make a new booking anytime.
- Booking Confirmed**  
Your booking at City Center DC Fast Charger has been confirmed for 2025-10-09 14:21 - 2025-10-09 17:21. Your charging session is ready!
- Booking Reminder**  
Reminder: Your charging session at City Center DC Fast Charger starts at 2025-10-09 14:21. Don't forget to arrive

At the bottom of the screen is a dark footer bar with three icons: a square grid, a plus sign, and a rectangle. To the right of these icons is a green oval button labeled "Notifications" with a green bell icon.



22:25 ⓘ ⓘ ⓘ ⓘ ⓘ ⓘ

82%

## Bookings



### All Customer Bookings

**Booking ID: #1b0712b4**

CONFIRMED

Station: 68e673760e78f5875a255240

Oct 12, 2025 | 04:34 PM - 04:34 PM

User NIC: 789123456V

**Booking ID: #1b0712af**

CONFIRMED

Station: 68e673760e78f5875a255240

Oct 11, 2025 | 11:21 AM - 11:20 AM

User NIC: 789123456V

**Booking ID: #5a255246**

CANCELLED

Station: 68e673760e78f5875a255240

Oct 09, 2025 | 02:21 PM - 05:21 PM

User NIC: 789123456V

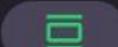
**Booking ID: #5a255248**

COMPLETED

Station: 68e673760e78f5875a255240

Oct 08, 2025 | 08:21 PM - 10:21 PM

User NIC: 789123456V



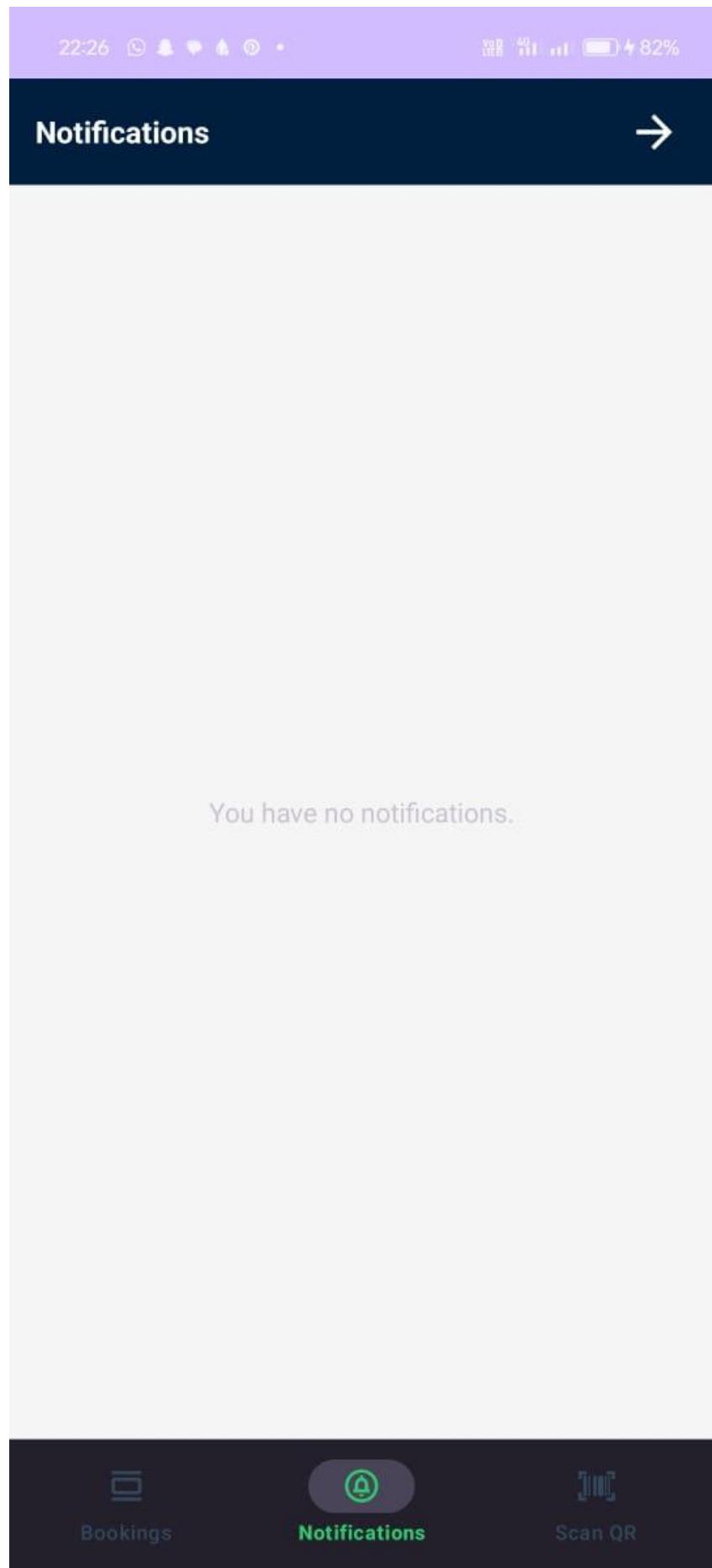
Bookings

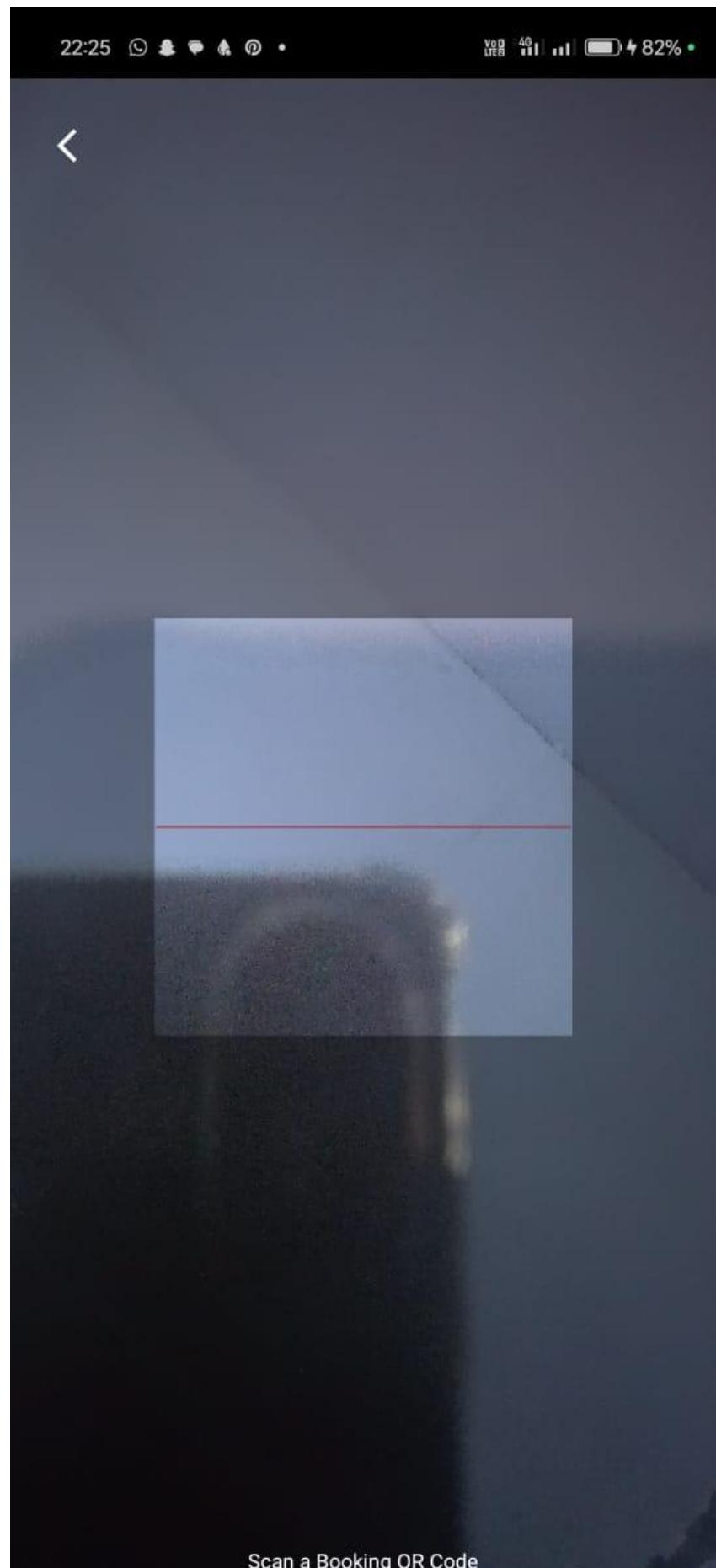


Notifications

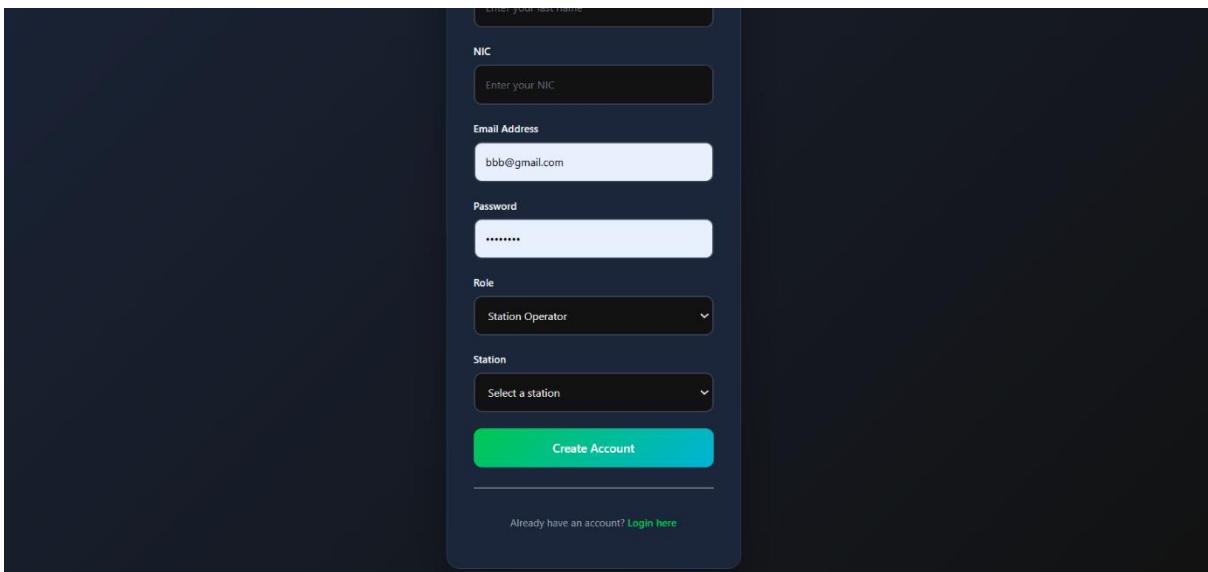
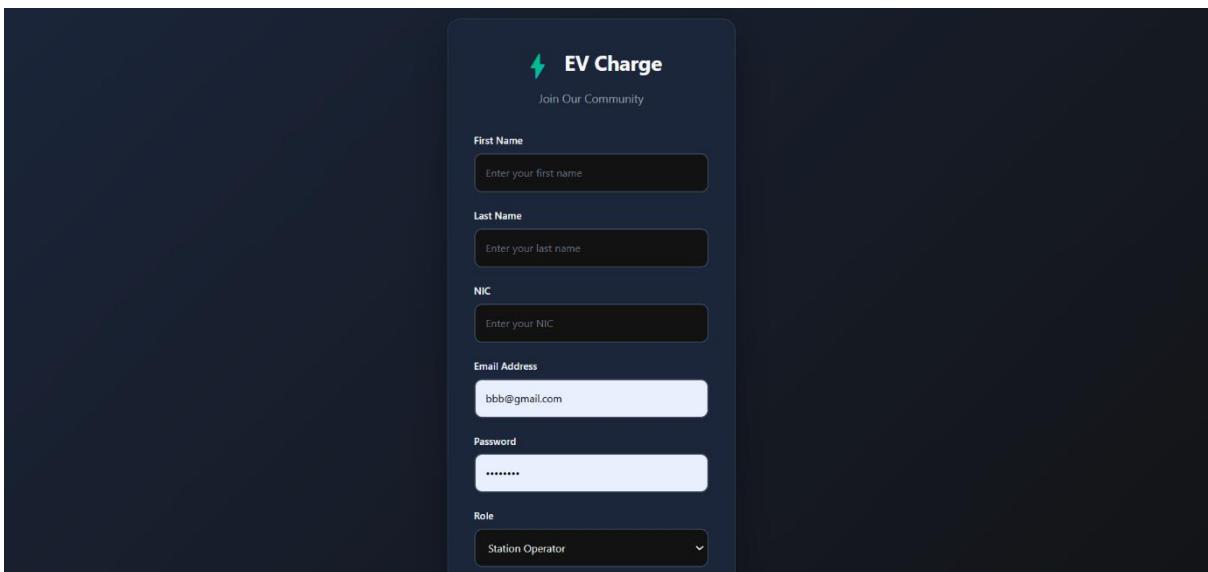
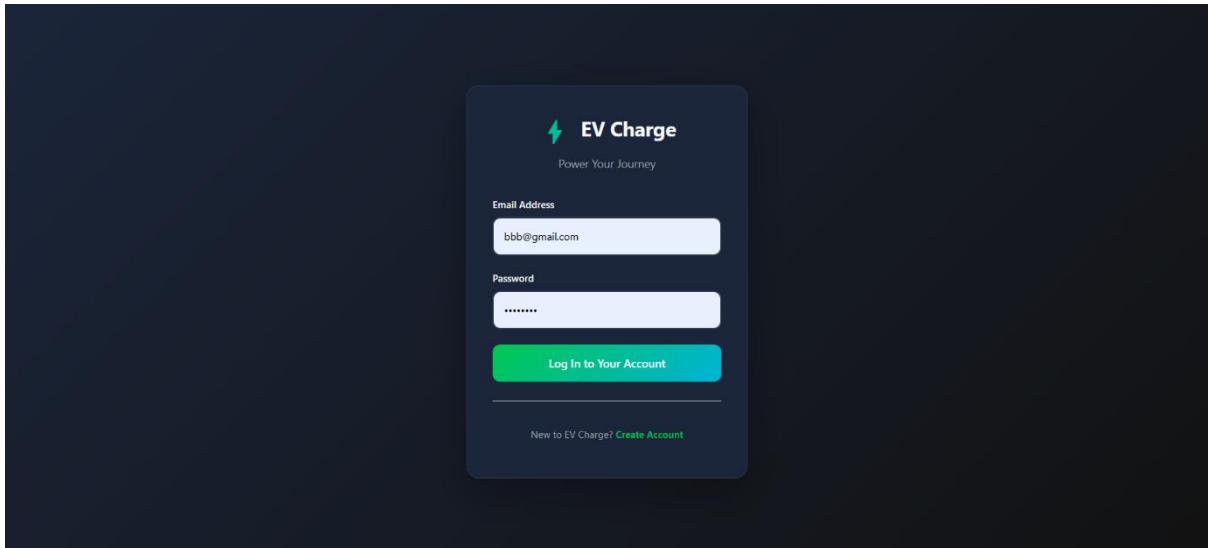


Scan QR





## 6.2 EV Charging Web Application UI Screenshots



**EV Charge - Backoffice**

**Admin Panel**

- Manage Users** (selected)
- Manage Stations
- Manage EV Owners

**Manage Users**  
View and manage all system users

NIC	Name	Email	Phone	Role	Status	Actions			
123456789V	Admin User	admin@evcharging.com	+94771234567	Backoffice	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
987654321V	Station Operator	operator@evcharging.com	+94777654321	Operator	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
456789123V	John Doe	john.doe@example.com	+94774567891	EV Owner	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
789123456V	Jane Smith	jane.smith@example.com	+94777891234	EV Owner	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
321654987V	Bob Wilson	bob.wilson@example.com	+94773216549	EV Owner	Inactive	<span>View</span>	<span>Edit</span>	<span>Activate</span>	<span>Delete</span>
200212232345	aaa aaa	aaa@gmail.com	N/A	Operator	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
245678954	sam perera	sam@gmail.com	0782588558	Operator	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
20023112345	bbb bbb	bbb@gmail.com	N/A	Backoffice	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>

Logout

**Manage Users**

- Manage Stations
- Manage EV Owners

NIC	Name	Email	Phone	Role	Status	Actions			
123456789V	Admin User	admin@evcharging.com	+94771234567	Backoffice	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
987654321V	Station Operator	operator@evcharging.com	+94777654321	Operator	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
456789123V	John Doe	john.doe@example.com	+94774567891	EV Owner	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
789123456V	Jane Smith	jane.smith@example.com	+94777891234	EV Owner	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
321654987V	Bob Wilson	bob.wilson@example.com	+94773216549	EV Owner	Inactive	<span>View</span>	<span>Edit</span>	<span>Activate</span>	<span>Delete</span>
200212232345	aaa aaa	aaa@gmail.com	N/A	Operator	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
245678954	sam perera	sam@gmail.com	0782588558	Operator	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
20023112345	bbb bbb	bbb@gmail.com	N/A	Backoffice	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>

Logout

Power Your Journey

**EV Charge**  
© 2025 EV Charging Station Booking System  
Power Your Journey

**EV Charge - Backoffice**

**Admin Panel**

- Manage Users** (selected)
- Manage Stations
- Manage EV Owners

**Manage Users**  
View and manage all system users

NIC	Name	Email	Phone Number	Role	Status	Actions			
123456789V	Admin User	admin@evcharging.com	+94771234567	Backoffice	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
987654321V	Station Operator	operator@evcharging.com	+94777654321	Operator	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
456789123V	John Doe	john.doe@example.com	+94774567891	EV Owner	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
789123456V	Jane Smith	jane.smith@example.com	+94777891234	EV Owner	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
321654987V	Bob Wilson	bob.wilson@example.com	+94773216549	EV Owner	Inactive	<span>View</span>	<span>Edit</span>	<span>Activate</span>	<span>Delete</span>
200212232345	aaa aaa	aaa@gmail.com	N/A	Operator	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
245678954	sam perera	sam@gmail.com	0782588558	Operator	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>
20023112345	bbb bbb	bbb@gmail.com	N/A	Backoffice	Active	<span>View</span>	<span>Edit</span>	<span>Deactivate</span>	<span>Delete</span>

Logout

20° 11:17 PM

**EV Charge - Backoffice**

**Admin Panel**

- Manage Users
- Manage Stations
- Manage EV Owners

**Manage Users**  
View and manage all system users

NIC	Name	Status	Actions
123456789V	Admin User	Active	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
987654321V	Station Operator	Active	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
456789123V	John Doe	Active	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
789123456V	Jane Smith	Active	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
321654987V	Bob Wilson	Inactive	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Activate</a> <a href="#">Delete</a>
200212232345	aaa aaa	Active	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
245678954	sam perera	Active	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
20023112345	bbb bbb	N/A	<a href="#">Backoffice</a> <a href="#">Active</a>

[Logout](#)

**Edit User**

**NIC**  
123456789V

**First Name** Admin **Last Name** User

**Email** admin@evcharging.com

**Phone Number** +94771234567

**Role** Backoffice **Status** Active

[Close](#) [Save Changes](#)

**EV Charge - Backoffice**

**Admin Panel**

- Manage Users
- Manage Stations
- Manage EV Owners

**Charging Stations**  
Manage all charging stations and locations

Total Stations: 5

Station Name	Location	Price/Hour	Status	Actions
City Center DC Fast Charger	Colombo City Center, Main Street	500.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
Shopping Mall AC Charger	Kandy Shopping Mall	300.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
Highway Rest Stop Charger	South Coast Highway	600.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
University Campus Charger	University of Colombo	250.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
Business District Charger	Working Class District	550.00	Maintenance	<a href="#">Activate</a> <a href="#">Edit</a> <a href="#">Delete</a>

[Add New Station](#)

**+ Add New Station**

**Station Name**  
Enter station name

**Location**  
Enter station location

**Charger Type** AC Charger **Total Slots** 1

**Price Per Hour (\$)** 0.01 **Status** Active

[Cancel](#) [+ Add Station](#)

**EV Charge - Backoffice**

**Admin Panel**

- Manage Users
- Manage Stations
- Manage EV Owners

**Charging Stations**  
Manage all charging stations and locations

Total Stations: 5

Station Name	Location	Price/Hour	Status	Actions
City Center DC Fast Charger	Colombo City Center, Main Street	500.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
Shopping Mall AC Charger	Kandy Shopping Mall	300.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
Highway Rest Stop Charger	South Coast Highway	600.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
University Campus Charger	University of Colombo	250.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
Business District Charger	Working Class District	550.00	Maintenance	<a href="#">Activate</a> <a href="#">Edit</a> <a href="#">Delete</a>

**Edit Station**

**Station Name**  
City Center DC Fast Charger

**Location**  
Colombo City Center, Main Street

**Charger Type** DC Fast Charger **Total Slots** 4

**Price Per Hour (\$)** 500 **Status** Active

[Cancel](#) [Save Changes](#)

### EV Charge - Backoffice

**Admin Panel**

- Manage Users
- Manage Stations
- Manage EV Owners**

### Charging Stations

Manage all charging stations and their availability

Total Stations: 5

[+ Add New Station](#)

Station Name	Location	Type	Slots	Price/Hour	Status	Actions
City Center DC Fast Charger	Colombo City Center, Main Street	+ DC	4	\$500.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
Shopping Mall AC Charger	Kandy Shopping Complex, Level B1	↓ AC	8	\$300.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
Highway Rest Stop Charger	Southern Expressway, Rest Area 1	+ DC	6	\$600.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
University Campus Charger	University of Colombo, Parking Area C	↓ AC	12	\$250.00	Active	<a href="#">Deactivate</a> <a href="#">Edit</a> <a href="#">Delete</a>
Business District Charger	World Trade Center, Underground Parking	+ DC	3	\$550.00	Maintenance	<a href="#">Activate</a> <a href="#">Edit</a> <a href="#">Delete</a>

[Logout](#)

### EV Charge - Backoffice

**Admin Panel**

- Manage Users
- Manage Stations
- Manage EV Owners**

### EV Owner Management

Manage all electric vehicle owners and their accounts

Total EV Owners: 3 Active: 2

NIC	Name	Email	Phone	Status	Actions
456789123V	John Doe	john.doe@example.com	+94774567891	Active	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
789123456V	Jane Smith	jane.smith@example.com	+94777891234	Active	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Deactivate</a> <a href="#">Delete</a>
321654987V	Bob Wilson	bob.wilson@example.com	+94773216549	Inactive	<a href="#">View</a> <a href="#">Edit</a> <a href="#">Activate</a> <a href="#">Delete</a>

[Logout](#)

### EV Charge - Backoffice

**Admin Panel**

- Manage Users
- Manage Stations
- Manage EV Owners**

### EV Owner Management

Manage all electric vehicle owners

Total EV Owners: 3 Active: 2

NIC	Name
456789123V	John Doe
789123456V	Jane Smith
321654987V	Bob Wilson

**EV Owner Details**

NIC  
456789123V

First Name      Last Name  
John      Doe

Email  
john.doe@example.com

Phone Number  
+94774567891

Status  
 Active

Role  
EV Owner

[Close](#)

[Logout](#)

**EV Charge - Backoffice**

**Admin Panel**

- Manage Users
- Manage Stations
- Manage EV Owners**

Total EV Owners: 3 Active: 2

NIC	Name
456789123V	John Doe
789123456V	Jane Smith
321654987V	Bob Wilson

**Edit EV Owner**

NIC: 456789123V

First Name: John Last Name: Doe

Email: john.doe@example.com

Phone Number: +94774567891

Status:  Active

Role:  EV Owner

**Actions**

- View Edit Deactivate Delete
- View Edit Deactivate Delete
- View Edit Activate Delete

**Logout**

**Station Panel**  
Operator Dashboard

**Bookings**

**Operator Dashboard**

**Station Bookings**  
Manage and monitor charging station bookings

Status From Date To Date Refresh

ID	User NIC	Start Time	End Time	Total Amount	QR Code	Status	Actions
...	789123456V	12 Oct 2025, 22:04	13 Oct 2025, 22:04	\$12000.00	5E77D9D8184542098AF4C88799677ED7	<span>Confirmed</span>	<span>Cancel</span>
...	789123456V	11 Oct 2025, 16:51	12 Oct 2025, 16:50	\$11991.67	1606AA20A0E6437FA807844B72460DD5	<span>Confirmed</span>	<span>Cancel</span>
...	789123456V	9 Oct 2025, 19:51	9 Oct 2025, 22:51	\$900.00	DE718A5499214443B1C9F9C7F7F59041	<span>Cancelled</span>	
...	789123456V	9 Oct 2025, 01:51	9 Oct 2025, 03:51	\$500.00	0603023FFD07459C885EEEAB38F9A59A	<span>Completed</span>	
...	789123456V	8 Oct 2025, 21:51	8 Oct 2025, 23:51	\$1000.00	D04C493C7C314D83AD6E3B7D9F5C4487	<span>Pending</span>	<span>Confirm</span> <span>Cancel</span>
...	456789123V	7 Oct 2025, 19:51	7 Oct 2025, 21:51	\$1200.00	2E197EC54C4D4FEC9957F4223C9337ED	<span>Cancelled</span>	

Station ID: 6Be673760e78f5875a255240

**Operator Dashboard**

**Bookings**

**Station Bookings**  
Manage and monitor charging station bookings

Status From Date To Date Refresh

ID	User NIC	Start Time	End Time	Total Amount	QR Code	Status	Actions
...	789123456V	12 Oct 2025, 22:04	13 Oct 2025, 22:04	\$12000.00	5E77D9D8184542098AF4C88799677ED7	<span>Confirmed</span>	<span>Cancel</span>
...	789123456V	11 Oct 2025, 16:51	12 Oct 2025, 16:50	\$11991.67	1606AA20A0E6437FA807844B72460DD5	<span>Confirmed</span>	<span>Cancel</span>
...	789123456V	9 Oct 2025, 19:51	9 Oct 2025, 22:51	\$900.00	DE718A5499214443B1C9F9C7F7F59041	<span>Cancelled</span>	
...	789123456V	9 Oct 2025, 01:51	9 Oct 2025, 03:51	\$500.00	0603023FFD07459C885EEEAB38F9A59A	<span>Completed</span>	
...	789123456V	8 Oct 2025, 21:51	8 Oct 2025, 23:51	\$1000.00	D04C493C7C314D83AD6E3B7D9F5C4487	<span>Pending</span>	<span>Confirm</span> <span>Cancel</span>
...	456789123V	7 Oct 2025, 19:51	7 Oct 2025, 21:51	\$1200.00	2E197EC54C4D4FEC9957F4223C9337ED	<span>Cancelled</span>	

Station ID: 6Be673760e78f5875a255240

**Logout**

## 7 Source Codes

GitHub Link - <https://github.com/amith-hasintha/-EV-Charging-Station-Booking-System>

### 7.1 EV Charging Mobile Application Codes

```
package com.example.evcharging.activities;

import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.ContextCompat;

import com.example.evcharging.R;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;
import com.example.evcharging.models.BookingApi;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class ConfirmBookingActivity extends AppCompatActivity {

    TextView tvConfirmBookingId, tvConfirmStationId, tvConfirmStatus;
    Button btnConfirmBooking;
    ApiService api;
```

```
String authToken, bookingId;

/**
 * Called when the activity is first created.
 * Initializes UI components, retrieves intent extras, and fetches booking details.
 */
@Override

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_confirm_booking);

    authToken = getIntent().getStringExtra("token");
    bookingId = getIntent().getStringExtra("bookingId");

    tvConfirmBookingId = findViewById(R.id.tvConfirmBookingId);
    tvConfirmStationId = findViewById(R.id.tvConfirmStationId);
    tvConfirmStatus = findViewById(R.id.tvConfirmStatus);
    btnConfirmBooking = findViewById(R.id.btnConfirmBooking);

    api = ApiClient.getApiService();

    fetchBookingDetails();

    btnConfirmBooking.setOnClickListener(v -> confirmBooking());
}

/**
 * Fetches booking details from the backend using the booking ID and auth token.
 * Updates UI fields based on the response and sets button state based on booking status.
```

```

*/



private void fetchBookingDetails() {

    api.getBookingById(authToken, bookingId).enqueue(new Callback<BookingApi>() {

        @Override

        public void onResponse(Call<BookingApi> call, Response<BookingApi> response) {

            if (response.isSuccessful() && response.body() != null) {

                BookingApi bookingApi = response.body();

                tvConfirmBookingId.setText("Booking ID: " + bookingApi.id);

                tvConfirmStationId.setText("Station ID: " + bookingApi.stationId);

                // --- THIS IS THE FIX ---

                // Backend Enum: Active=0, Confirmed=1

                int status = bookingApi.status;

                String statusText = "UNKNOWN";



                // Set status text and color based on the numeric status

                switch (status) {

                    case 0: // Active

                        statusText = "ACTIVE";

                        tvConfirmStatus.setTextColor(ContextCompat.getColor(ConfirmBookingActivity.this, R.color.orange_soda));

                        break;

                    case 1: // Confirmed

                        statusText = "CONFIRMED";

                        tvConfirmStatus.setTextColor(ContextCompat.getColor(ConfirmBookingActivity.this, R.color.emerald_green));

                        break;

                    case 3: // Cancelled

                        statusText = "CANCELLED";

```

```

        tvConfirmStatus.setTextColor(ContextCompat.getColor(ConfirmBookingActivity.this,
R.color.red_error));

        break;

    default:

        tvConfirmStatus.setTextColor(android.graphics.Color.GRAY);

        break;

    }

    tvConfirmStatus.setText("Status: " + statusText);

}

// Only allow confirmation if the booking has been approved by an operator (status 1)

if (status == 1) { // <-- Compare integers, not strings

    btnConfirmBooking.setEnabled(true);

} else {

    btnConfirmBooking.setEnabled(false);

    btnConfirmBooking.setText("Cannot Confirm (Status: " + statusText + ")");

}

}

Toast.makeText(ConfirmBookingActivity.this, "Failed to fetch booking details",
Toast.LENGTH_SHORT).show();

}

}

@Override

public void onFailure(Call<BookingApi> call, Throwable t) {

    Toast.makeText(ConfirmBookingActivity.this, "Network Error", Toast.LENGTH_SHORT).show();

}

});

}

```

```

/**
 * Confirms the booking by calling the backend API.
 * Updates the UI based on the success or failure of the confirmation request.
 */

private void confirmBooking() {

    api.confirmBooking(authToken, bookingId).enqueue(new Callback<Void>() {

        @Override
        public void onResponse(Call<Void> call, Response<Void> response) {
            if (response.isSuccessful()) {
                Toast.makeText(ConfirmBookingActivity.this, "Booking Confirmed Successfully!",
                        Toast.LENGTH_LONG).show();
                btnConfirmBooking.setEnabled(false);
                btnConfirmBooking.setText("Confirmed");
                tvConfirmStatus.setText("Status: confirmed");
            } else {
                Toast.makeText(ConfirmBookingActivity.this, "Confirmation Failed",
                        Toast.LENGTH_SHORT).show();
            }
        }

        @Override
        public void onFailure(Call<Void> call, Throwable t) {
            Toast.makeText(ConfirmBookingActivity.this, "Network Error", Toast.LENGTH_SHORT).show();
        }
    });
}

```

```
package com.example.evcharging.activities;

import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;

import androidx.appcompat.app.AppCompatActivity;

import com.example.evcharging.R;
import com.journeyapps.barcodescanner.CaptureManager;
import com.journeyapps.barcodescanner.DecoratedBarcodeView;

public class CustomScannerActivity extends AppCompatActivity {

    private CaptureManager capture;
    private DecoratedBarcodeView barcodeScannerView;

    /**
     * Called when the activity is first created.
     *
     * Initializes the barcode scanner view and sets up the back button functionality.
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_custom_scanner);

        barcodeScannerView = findViewById(R.id.zxing_barcode_scanner);
    }
}
```

```
// --- THIS IS THE FIX FOR THE BACK BUTTON ---
ImageButton backButton = findViewById(R.id.btn_back);
backButton.setOnClickListener(v -> onBackPressed());
// --- END OF FIX ---

capture = new CaptureManager(this, barcodeScannerView);
capture.initializeFromIntent(getIntent(), savedInstanceState);
capture.decode();
}

/**
 * Called when the activity is resumed.
 * Resumes the barcode scanner capture process.
 */
@Override
protected void onResume() {
    super.onResume();
    capture.onResume();
}

/**
 * Called when the activity is paused.
 * Pauses the barcode scanner capture process.
 */
@Override
protected void onPause() {
    super.onPause();
    capture.onPause();
}
```

```
/**  
 * Called when the activity is destroyed.  
 * Releases resources used by the CaptureManager.  
 */  
  
@Override  
protected void onDestroy() {  
    super.onDestroy();  
    capture.onDestroy();  
}  
  
/**  
 * Called to save the instance state of the activity.  
 * Ensures the CaptureManager state is preserved during configuration changes.  
 */  
  
@Override  
protected void onSaveInstanceState(Bundle outState) {  
    super.onSaveInstanceState(outState);  
    capture.onSaveInstanceState(outState);  
}  
}  
  
package com.example.evcharging.activities;  
  
import android.content.Intent;  
import android.os.Bundle;  
import android.view.MenuItem;  
import android.widget.ImageView;
```

```
import android.widget.Toast;

import androidx.annotation.NonNull;import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;

import com.example.evcharging.R;
import com.example.evcharging.fragments.CreateBookingFragment;
import com.example.evcharging.fragments.DashboardFragment;
import com.example.evcharging.fragments.MyBookingsFragment;
import com.example.evcharging.fragments.NotificationsFragment;
import com.google.android.material.bottomnavigation.BottomNavigationView;

public class DashboardActivity extends AppCompatActivity {

    private String authToken;
    private final FragmentManager fm = getSupportFragmentManager();

    // 1. Declare fragments but DO NOT initialize them here.
    // They will be created and managed properly by the FragmentManager.

    private DashboardFragment dashboardFragment;
    private CreateBookingFragment createBookingFragment;
    private MyBookingsFragment myBookingsFragment;
    private NotificationsFragment notificationsFragment;
    private Fragment activeFragment;

    /**
     * Called when the activity is first created.
     * Initializes authentication, UI elements, fragments, and bottom navigation.
    
```

```
*/\n\n@Override\n\nprotected void onCreate(Bundle savedInstanceState) {\n    super.onCreate(savedInstanceState);\n    setContentView(R.layout.activity_dashboard);\n\n    authToken = getIntent().getStringExtra("token");\n\n    if (authToken == null || authToken.isEmpty()) {\n        Toast.makeText(this, "Authentication Failed. Please login again.", Toast.LENGTH_LONG).show();\n        navigateToLogin();\n        return;\n    }\n\n    ImageView ivProfile = findViewById(R.id.ivProfile);\n    ivProfile.setOnClickListener(v -> {\n        Intent intent = new Intent(DashboardActivity.this, ProfileActivity.class);\n        intent.putExtra("token", authToken);\n        startActivity(intent);\n    });\n\n    BottomNavigationView bottomNavigationView = findViewById(R.id.bottom_navigation);\n    bottomNavigationView.setOnItemSelectedListener(this::onNavigationItemSelected);\n\n    // This is the key change: Handle both first-time creation and recreation.\n    if (savedInstanceState == null) {\n        setupFragments();\n        // Set the starting tab\n        bottomNavigationView.setSelectedItemId(R.id.navigation_dashboard);\n    }\n}
```

```

} else {

    // On recreation, find the fragments by their tag instead of creating new ones.

    dashboardFragment = (DashboardFragment) fm.findFragmentByTag("1");

    createBookingFragment = (CreateBookingFragment) fm.findFragmentByTag("2");

    myBookingsFragment = (MyBookingsFragment) fm.findFragmentByTag("3");

    notificationsFragment = (NotificationsFragment) fm.findFragmentByTag("4");




    // Find which fragment was last active to restore it.

    // This loop is a robust way to find the visible fragment.

    for (Fragment fragment : fm.getFragments()) {

        if (fragment.isVisible()) {

            activeFragment = fragment;

            break;

        }

    }

}

}

/**



 * Initializes all fragments using the safe newInstance() pattern.

 * Adds all fragments to the FragmentManager and sets the initial active fragment.

 */

private void setupFragments() {

    // 2. Use the safe `newInstance()` pattern we created before for ALL fragments.

    // This ensures the token is always passed correctly.

    dashboardFragment = DashboardFragment.newInstance(authToken);

    createBookingFragment = CreateBookingFragment.newInstance(authToken);

    myBookingsFragment = MyBookingsFragment.newInstance(authToken);

    notificationsFragment = NotificationsFragment.newInstance(authToken);
}

```

```

// 3. Add all fragments in a single transaction.

// This is more efficient and ensures they all exist before being shown.

fm.beginTransaction()

    .add(R.id.fragment_container, notificationsFragment, "4").hide(notificationsFragment)

    .add(R.id.fragment_container, myBookingsFragment, "3").hide(myBookingsFragment)

    .add(R.id.fragment_container, createBookingFragment, "2").hide(createBookingFragment)

    .add(R.id.fragment_container, dashboardFragment, "1")

    .commit();

// 4. Set the initial active fragment.

activeFragment = dashboardFragment;

}

/***
 * Handles bottom navigation item selection.
 * Switches the visible fragment based on the selected item.
 */

private boolean onNavigationItemSelected(@NonNull MenuItem item) {

    Fragment selectedFragment = null;

    int itemId = item.getItemId();

    if (itemId == R.id.navigation_dashboard) {

        selectedFragment = dashboardFragment;

    } else if (itemId == R.id.navigation_create_booking) {

        selectedFragment = createBookingFragment;

    } else if (itemId == R.id.navigation_my_bookings) {

        selectedFragment = myBookingsFragment;

    } else if (itemId == R.id.navigation_notifications) {


```

```

        selectedFragment = notificationsFragment;
    }

    // Only perform a transaction if the selected fragment is valid and not already active.
    if (selectedFragment != null && selectedFragment != activeFragment) {

        fm.beginTransaction().hide(activeFragment).show(selectedFragment).commit();

        activeFragment = selectedFragment;
    }

    return true;
}

/**
 * Navigates the user back to the login screen in case of authentication failure.
 * Clears the activity stack to prevent returning to Dashboard without login.
 */
private void navigateToLogin() {

    Intent intent = new Intent(this, LoginActivity.class);

    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);

    startActivity(intent);

    finish();
}

}

package com.example.evcharging.activities;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Context;
import android.content.Intent;

```

```
import android.content.SharedPreferences;
import android.os.Bundle;
import android.text.TextUtils;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

import com.example.evcharging.R;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;

import java.util.HashMap;
import java.util.Map;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class LoginActivity extends AppCompatActivity {

    private EditText etEmail, etPassword;
    private Button btnLogin;
    private TextView tvRegister;
    private ApiService api;

    public static final String PREFS_NAME = "EV_CHARGING_PREFS";
    public static final String AUTH_TOKEN_KEY = "AUTH_TOKEN_KEY";
    public static final String STATION_ID_KEY = "STATION_ID_KEY"; // Key for saving stationId
```

```

/**
 * Called when the activity is first created.
 * Initializes UI components, API service, and button click listeners.
 */
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

    etEmail = findViewById(R.id.etEmail);
    etPassword = findViewById(R.id.etPassword);
    btnLogin = findViewById(R.id.btnLogin);
    tvRegister = findViewById(R.id.tvRegister);

    api = ApiClient.getApiService();

    btnLogin.setOnClickListener(v -> doLogin());
    tvRegister.setOnClickListener(v -> startActivity(new Intent(LoginActivity.this, RegisterActivity.class)));
}

/**
 * Collects user input for email and password.
 * Validates input and triggers API login if fields are not empty.
 */
private void doLogin() {
    String email = etEmail.getText().toString().trim();
    String pwd = etPassword.getText().toString().trim();
}

```

```

        if (TextUtils.isEmpty(email) || TextUtils.isEmpty(pwd)) {
            Toast.makeText(this, "Enter Email and Password", Toast.LENGTH_SHORT).show();
            return;
        }

        loginViaApi(email, pwd);
    }

    /**
     * Calls the backend API to perform login using provided credentials.
     * Handles response by saving auth token and navigating to the correct dashboard based on user role.
     */
    private void loginViaApi(String email, String password) {
        Map<String, String> body = new HashMap<>();
        body.put("email", email);
        body.put("password", password);

        api.login(body).enqueue(new Callback<Map<String, Object>>() {
            @Override
            public void onResponse(Call<Map<String, Object>> call, Response<Map<String, Object>> response) {
                if (response.isSuccessful() && response.body() != null) {
                    Map<String, Object> responseBody = response.body();
                    String authToken = (String) responseBody.get("token");
                    int role = -1;
                    String stationId = null;

                    if (responseBody.containsKey("user") && responseBody.get("user") instanceof Map) {
                        Map<String, Object> userMap = (Map<String, Object>) responseBody.get("user");
                        Object roleObj = userMap.get("role");
                    }
                }
            }
        });
    }
}

```

```

        if (roleObj instanceof Number) {

            role = ((Number) roleObj).intValue();

        }

        // If it's an operator (role 1), get their stationId

        if (role == 1 && userMap.containsKey("stationId")) {

            stationId = (String) userMap.get("stationId");

        }

    }

    if (authToken != null) {

        String fullToken = "Bearer " + authToken;

        saveAuthData(fullToken, stationId); // Save both token and stationId

        Toast.makeText(LoginActivity.this, "Login successful!", Toast.LENGTH_SHORT).show();

        navigateToDashboardByRole(fullToken, stationId, role);

    } else {

        Toast.makeText(LoginActivity.this, "Login failed: Token is missing.",

        Toast.LENGTH_LONG).show();

    }

} else {

    Toast.makeText(LoginActivity.this, "Login failed: Invalid credentials.",

    Toast.LENGTH_LONG).show();

}

}

@Override

public void onFailure(Call<Map<String, Object>> call, Throwable t) {

    Toast.makeText(LoginActivity.this, "Network error: " + t.getMessage(),

    Toast.LENGTH_LONG).show();

}

```

```

    });

}

/** 
 * Saves the authentication token and stationId (if available) in SharedPreferences.
 */
private void saveAuthData(String token, String stationId) {
    SharedPreferences.Editor editor = getSharedPreferences(PREFS_NAME,
    Context.MODE_PRIVATE).edit();

    editor.putString(AUTH_TOKEN_KEY, token);

    if (stationId != null) {
        editor.putString(STATION_ID_KEY, stationId);
    } else {
        editor.remove(STATION_ID_KEY); // Clear stationId if user is not an operator
    }
    editor.apply();
}

/** 
 * Navigates the user to the correct dashboard based on their role.
 * Operators (role 1) go to OperatorDashboard, others go to DashboardActivity.
 */
private void navigateToDashboardByRole(String authToken, String stationId, int role) {
    Intent intent;
    if (role == 1) { // Role 1 is Operator
        intent = new Intent(LoginActivity.this, OperatorDashboardActivity.class);
        intent.putExtra("stationId", stationId); // Pass stationId to Operator Dashboard
    } else { // Role 2 (or any other) is EV Owner
        intent = new Intent(LoginActivity.this, DashboardActivity.class);
    }
}

```

```
        }

        intent.putExtra("token", authToken);

        startActivity(intent);

        finish();

    }

}

package com.example.evcharging.activities;

import android.content.Context;

import android.content.Intent;

import android.content.SharedPreferences;

import android.os.Bundle;

import android.util.Log;

import android.view.MenuItem;

import android.widget.ImageView;

import android.widget.TextView;

import android.widget.Toast;

import androidx.activity.result.ActivityResultLauncher;

import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;

import androidx.fragment.app.Fragment;

import androidx.fragment.app.FragmentManager;

import com.example.evcharging.R;

import com.example.evcharging.fragments.NotificationsFragment;

import com.example.evcharging.fragments.OperatorBookingsFragment;

import com.google.android.material.bottomnavigation.BottomNavigationView;
```

```
import com.journeyapps.barcodescanner.ScanContract;
import com.journeyapps.barcodescanner.ScanOptions;

public class OperatorDashboardActivity extends AppCompatActivity {

    private String authToken;
    private final FragmentManager fm = getSupportFragmentManager();

    // Declare fragments but DO NOT initialize them here
    private OperatorBookingsFragment operatorBookingsFragment;
    private NotificationsFragment notificationsFragment;
    private Fragment activeFragment;

    private TextView toolbarTitle;
    private BottomNavigationView bottomNavigationView;

    // ActivityResultLauncher for the QR Code Scanner (this is correct)
    private final ActivityResultLauncher<ScanOptions> barcodeLauncher = registerForActivityResult(new
ScanContract(),
        result -> {
            if(result.getContents() == null) {
                Toast.makeText(OperatorDashboardActivity.this, "Scan cancelled",
Toast.LENGTH_LONG).show();
            } else {
                Log.d("QR_SCAN", "Scanned: " + result.getContents());
                Intent intent = new Intent(OperatorDashboardActivity.this, ConfirmBookingActivity.class);
                intent.putExtra("token", authToken);
                intent.putExtra("bookingId", result.getContents());
                startActivity(intent);
            }
        }
    );
}
```

```

        bottomNavigationView.setSelectedItemId(R.id.navigation_operator_bookings);

    });

    /**
     * Called when the activity is first created.
     *
     * Initializes authentication, toolbar, bottom navigation, and fragments.
     *
     * Restores fragments on configuration change.
     */
}

@Override

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_operator_dashboard);

    authToken = getIntent().getStringExtra("token");
    if (authToken == null || authToken.isEmpty()) {
        Toast.makeText(this, "Authentication Error", Toast.LENGTH_SHORT).show();
        finish();
        return;
    }

    toolbarTitle = findViewById(R.id.toolbar_title);
    ImageView ivLogout = findViewById(R.id.ivLogout);
    ivLogout.setOnClickListener(v -> logoutUser());

    bottomNavigationView = findViewById(R.id.operator_bottom_navigation);
    bottomNavigationView.setOnItemSelectedListener(this::onNavigationItemSelected);

    // Setup fragments only if the activity is newly created
    if (savedInstanceState == null) {

```

```

        setupFragments();

        bottomNavigationView.setSelectedItemId(R.id.navigation_operator_bookings);

    } else {

        // Re-find fragments on configuration change

        operatorBookingsFragment = (OperatorBookingsFragment) fm.findFragmentByTag("1");

        notificationsFragment = (NotificationsFragment) fm.findFragmentByTag("2");

        // Determine active fragment after recreation

        if (operatorBookingsFragment != null && !operatorBookingsFragment.isHidden()) {

            activeFragment = operatorBookingsFragment;

        } else if (notificationsFragment != null && !notificationsFragment.isHidden()) {

            activeFragment = notificationsFragment;

        }

    }

}

/***
 * Initializes and adds the operator and notification fragments.
 *
 * Sets the initial active fragment.
 */
private void setupFragments() {

    // 1. Retrieve the token and stationId ONCE from the Intent.

    String stationId = getIntent().getStringExtra("stationId");

    // 2. Use the safe newInstance factory method to create each fragment.

    operatorBookingsFragment = OperatorBookingsFragment.newInstance(authToken, stationId);

    notificationsFragment = NotificationsFragment.newInstance(authToken);

    // 3. Add the fragments to the FragmentManager with unique tags.

    fm.beginTransaction()

```

```

    .add(R.id.operator_fragment_container, notificationsFragment, "2")
    .hide(notificationsFragment)
    .add(R.id.operator_fragment_container, operatorBookingsFragment, "1")
    .commit();

    // 4. Set the initial active fragment.

    activeFragment = operatorBookingsFragment;
}

/**
 * Handles bottom navigation item clicks.
 * Switches between operator bookings, notifications, and launches QR scanner.
 */
private boolean onNavigationItemSelected(@NonNull MenuItem item) {
    int itemId = item.getItemId();

    if (itemId == R.id.navigation_operator_bookings) {
        toolbarTitle.setText("Bookings");
        if(activeFragment != operatorBookingsFragment) {
            fm.beginTransaction().hide(activeFragment).show(operatorBookingsFragment).commit();
            activeFragment = operatorBookingsFragment;
        }
        return true;
    } else if (itemId == R.id.navigation_operator_notifications) {
        toolbarTitle.setText("Notifications");
        if(activeFragment != notificationsFragment) {
            fm.beginTransaction().hide(activeFragment).show(notificationsFragment).commit();
            activeFragment = notificationsFragment;
        }
    }
}

```

```

        return true;

    } else if (itemId == R.id.navigation_scan_qr) {

        launchQrScanner();

        return false; // Return false so the item doesn't stay selected

    }

    return false;

}

// --- THIS IS THE FIX FOR THE ORIENTATION ---

// Replace the entire launchQrScanner method with this new version

/***
 * Launches the custom QR scanner with orientation lock and other options.
 * Ensures the scanned booking ID is sent to ConfirmBookingActivity.
 */

private void launchQrScanner() {

    ScanOptions options = new ScanOptions();

    options.setOrientationLocked(true); // Locks orientation to prevent rotation issues

    options.setCaptureActivity(CustomScannerActivity.class); // Use our custom activity

    options.setDesiredBarcodeFormats(ScanOptions.QR_CODE);

    options.setPrompt("Scan a Booking QR Code");

    options.setCameraId(0);

    options.setBeepEnabled(true);

    options.setBarcodeImageEnabled(true);

    barcodeLauncher.launch(options);

}

/***
 * Logs out the user by clearing stored auth token and stationId.
*/

```

```
* Redirects to LoginActivity and clears the back stack.

*/
private void logoutUser() {

    SharedPreferences prefs = getSharedPreferences(LoginActivity.PREFS_NAME,
Context.MODE_PRIVATE);

prefs.edit().remove(LoginActivity.AUTH_TOKEN_KEY).remove(LoginActivity.STATION_ID_KEY).apply();

    Intent intent = new Intent(this, LoginActivity.class);

    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);

    startActivity(intent);

    finish();

}

}

package com.example.evcharging.activities;

import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;

import com.example.evcharging.R;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;
```

```
import com.example.evcharging.models.User;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class ProfileActivity extends AppCompatActivity {

    EditText etProfileFirstName, etProfileLastName, etProfilePhone;
    Button btnUpdateProfile, btnDeactivate;
    ApiService api;
    String authToken;

    /**
     * Called when the activity is first created.
     * Initializes UI components, API service, authentication token,
     * and sets up click listeners for profile update and deactivation.
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_profile);

        etProfileFirstName = findViewById(R.id.etProfileFirstName);
        etProfileLastName = findViewById(R.id.etProfileLastName);
        etProfilePhone = findViewById(R.id.etProfilePhone);
        btnUpdateProfile = findViewById(R.id.btnUpdateProfile);
        btnDeactivate = findViewById(R.id.btnDeactivate);
    }
}
```

```

api = ApiClient.getApiService();

authToken = getIntent().getStringExtra("token");

fetchProfile();

btnUpdateProfile.setOnClickListener(v -> updateProfile());

btnDeactivate.setOnClickListener(v -> showDeactivationConfirmDialog());

}

/***
 * Fetches the current user's profile from the backend API.
 * Populates the UI fields with the retrieved data.
 */

private void fetchProfile() {

    api.getMyProfile(authToken).enqueue(new Callback<User>() {

        @Override

        public void onResponse(Call<User> call, Response<User> response) {

            if (response.isSuccessful() && response.body() != null) {

                User user = response.body();

                etProfileFirstName.setText(user.firstName);

                etProfileLastName.setText(user.lastName);

                etProfilePhone.setText(user.phoneNumber);

            } else {

                Toast.makeText(ProfileActivity.this, "Failed to load profile", Toast.LENGTH_SHORT).show();

            }

        }

        @Override

        public void onFailure(Call<User> call, Throwable t) {

```

```

        Toast.makeText(ProfileActivity.this, "Network Error", Toast.LENGTH_SHORT).show();
    }

});

}

/***
 * Updates the user's profile with the values entered in the UI.
 * Sends the updated profile to the backend API.
 */
private void updateProfile() {

    String firstName = etProfileFirstName.getText().toString();

    String lastName = etProfileLastName.getText().toString();

    String phone = etProfilePhone.getText().toString();

    User userToUpdate = new User();

    userToUpdate.firstName = firstName;

    userToUpdate.lastName = lastName;

    userToUpdate.phoneNumber = phone;

    api.updateMyProfile(authToken, userToUpdate).enqueue(new Callback<User>() {

        @Override

        public void onResponse(Call<User> call, Response<User> response) {

            if (response.isSuccessful()) {

                Toast.makeText(ProfileActivity.this, "Profile updated successfully!",
                Toast.LENGTH_SHORT).show();

                finish();

            } else {

                Toast.makeText(ProfileActivity.this, "Update failed", Toast.LENGTH_SHORT).show();

            }
        }
    });
}

```

```

    }

    @Override
    public void onFailure(Call<User> call, Throwable t) {
        Toast.makeText(ProfileActivity.this, "Network Error", Toast.LENGTH_SHORT).show();
    }
});

}

/***
 * Displays a confirmation dialog to the user before deactivating their account.
 */
private void showDeactivationConfirmDialog() {
    new AlertDialog.Builder(this)
        .setTitle("Deactivate Account")
        .setMessage("Are you sure? This action cannot be undone from the app.")
        .setPositiveButton("Deactivate", (dialog, which) -> deactivateAccount())
        .setNegativeButton("Cancel", null)
        .show();
}

/***
 * Deactivates the user's account via the backend API.
 * Clears stored authentication data and navigates back to the login screen upon success.
 */
private void deactivateAccount() {
    api.deactivateMyAccount(authToken).enqueue(new Callback<Void>() {
        @Override
        public void onResponse(Call<Void> call, Response<Void> response) {

```

```

        if (response.isSuccessful()) {

            Toast.makeText(ProfileActivity.this, "Account deactivated. Logging out.",
Toast.LENGTH_LONG).show();

            // Clear token and navigate to login

            SharedPreferences prefs = getSharedPreferences(LoginActivity.PREFS_NAME,
MODE_PRIVATE);

            prefs.edit().clear().apply();

            Intent intent = new Intent(ProfileActivity.this, LoginActivity.class);

            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_NEW_TASK |
Intent.FLAG_ACTIVITY_CLEAR_TASK);

            startActivity(intent);

            finish();

        } else {

            Toast.makeText(ProfileActivity.this, "Deactivation failed", Toast.LENGTH_SHORT).show();

        }

    }

    @Override

    public void onFailure(Call<Void> call, Throwable t) {

        Toast.makeText(ProfileActivity.this, "Network Error", Toast.LENGTH_SHORT).show();

    }

});



}

```

```
package com.example.evcharging.activities;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.text.TextUtils;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.example.evcharging.R;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;
import com.example.evcharging.db.AppDatabase;
import com.example.evcharging.models.User;

import java.util.Map;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class RegisterActivity extends AppCompatActivity {

    EditText etNic, etFirstName, etLastName, etEmail, etPassword, etPhoneNumber;
    Button btnRegister;
    ApiService api;
```

```
/**  
 * Called when the activity is first created.  
 * Initializes UI components, API service, and sets click listener for registration button.  
 */  
  
@Override  
  
protected void onCreate(Bundle savedInstanceState) {  
  
    super.onCreate(savedInstanceState);  
  
    setContentView(R.layout.activity_register);  
  
    etNic = findViewById(R.id.etNic);  
  
    etFirstName = findViewById(R.id.etFirstName);  
  
    etLastName = findViewById(R.id.etLastName);  
  
    etEmail = findViewById(R.id.etEmail);  
  
    etPassword = findViewById(R.id.etPassword);  
  
    etPhoneNumber = findViewById(R.id.etPhoneNumber);  
  
    btnRegister = findViewById(R.id.btnRegister);  
  
    api = ApiClient.getApiService();  
  
    btnRegister.setOnClickListener(v -> registerUser());  
}  
  
/**  
 * Handles the user registration process.  
 * 1. Validates input fields locally.  
 * 2. Sends registration data to the backend API.  
 * 3. Stores the registered user in local database upon success.  
 */  
  
private void registerUser() {
```

```

String nic = etNic.getText().toString().trim();

String firstName = etFirstName.getText().toString().trim();

String lastName = etLastName.getText().toString().trim();

String email = etEmail.getText().toString().trim();

String password = etPassword.getText().toString().trim();

String phoneNumber = etPhoneNumber.getText().toString().trim();

// --- Client-Side Validation ---

if (TextUtils.isEmpty(nic) || TextUtils.isEmpty(firstName) || TextUtils.isEmpty(lastName) ||
    TextUtils.isEmpty(email) || TextUtils.isEmpty(password) || TextUtils.isEmpty(phoneNumber)) {
    Toast.makeText(this, "All fields are required", Toast.LENGTH_SHORT).show();
    return;
}

// Added password length check based on backend requirements

if (password.length() < 6) {
    Toast.makeText(this, "Password must be at least 6 characters long",
    Toast.LENGTH_SHORT).show();
    return; // Stop the registration process
}

// Role for EV Owner is 2 as per API spec

int role = 2;

User userToRegister = new User(nic, firstName, lastName, email, password, role, phoneNumber);

Call<Map<String, Object>> call = api.register(userToRegister);

call.enqueue(new Callback<Map<String, Object>>() {

    @Override

    public void onResponse(Call<Map<String, Object>> call, Response<Map<String, Object>>
    response) {

```

```

        if (response.isSuccessful() && response.body() != null) {

            ExecutorService executor = Executors.newSingleThreadExecutor();

            executor.execute(() -> {

                AppDatabase db = AppDatabase.getDatabase(getApplicationContext());

                db.userDao().insert(userToRegister);

            });

            Toast.makeText(RegisterActivity.this, "Registration successful!", Toast.LENGTH_LONG).show();

            finish();

        } else {

            // You can add more specific error handling here later if needed

            String errorMessage = "Registration failed: " + response.code() + " " + response.message();

            Toast.makeText(RegisterActivity.this, errorMessage, Toast.LENGTH_LONG).show();

        }

    }

    @Override

    public void onFailure(Call<Map<String, Object>> call, Throwable t) {

        Toast.makeText(RegisterActivity.this, "Network error: " + t.getMessage(), Toast.LENGTH_LONG).show();

    }

});
}
}

package com.example.evcharging.activities;

import android.content.Context;
import android.content.Intent;

```

```
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.text.TextUtils;
import android.widget.Toast;

import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;

import com.example.evcharging.R;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;
import com.example.evcharging.models.User;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class SplashActivity extends AppCompatActivity {

    /**
     * Called when the activity is first created.
     *
     * Displays the splash screen for a short duration and then checks user authentication status.
     */
    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
    }
}
```

```

// Wait a moment to show the splash screen, then check the user's status.

new Handler(Looper.getMainLooper()).postDelayed(this::checkUserStatus, 1500);

}

/***
 * Checks if the user has a saved authentication token.
 * If no token exists, navigates to login.
 * If a token exists, fetches the user profile to determine the role.
 */
private void checkUserStatus() {

    SharedPreferences prefs = getSharedPreferences(LoginActivity.PREFS_NAME,
Context.MODE_PRIVATE);

    String authToken = prefs.getString(LoginActivity.AUTH_TOKEN_KEY, null);

    if (TextUtils.isEmpty(authToken)) {

        // No token saved, user must log in.

        navigateToLogin();

    } else {

        // Token found, verify it and get user role.

        fetchProfileAndNavigate(authToken);

    }

}

/***
 * Fetches the user's profile from the backend to verify the token and retrieve the user role.
 * The authentication token of the user.
 */
private void fetchProfileAndNavigate(String authToken) {

```

```

 ApiService apiService = ApiClient.getApiService();

apiService.getMyProfile(authToken).enqueue(new Callback<User>() {

    @Override

    public void onResponse(Call<User> call, Response<User> response) {

        if (response.isSuccessful() && response.body() != null) {

            User user = response.body();

            // Successfully fetched profile, now navigate based on the role.

            navigateToDashboardByRole(authToken, user.role);

        } else {

            // Token might be expired or invalid. Clear it and go to login.

            Toast.makeText(SplashActivity.this, "Session expired. Please login again.",

Toast.LENGTH_SHORT).show();

            clearTokenAndNavigateToLogin();

        }

    }

    @Override

    public void onFailure(Call<User> call, Throwable t) {

        // Network error. Can't verify token, so go to login.

        Toast.makeText(SplashActivity.this, "Network error. Please try again.",

Toast.LENGTH_SHORT).show();

        navigateToLogin();

    }

});

}

private void navigateToDashboardByRole(String authToken, int role) {

Intent intent;

if (role == 1) {

    // Role 1 is Operator.

```

```

        intent = new Intent(SplashActivity.this, OperatorDashboardActivity.class);
    } else {
        // Role 2 (or any other) is EV Owner.

        intent = new Intent(SplashActivity.this, DashboardActivity.class);
    }

    intent.putExtra("token", authToken);
    startActivity(intent);
    finish(); // Close SplashActivity.

}

private void navigateToLogin() {
    startActivity(new Intent(SplashActivity.this, LoginActivity.class));
    finish(); // Close SplashActivity.

}

private void clearTokenAndNavigateToLogin() {
    SharedPreferences prefs = getSharedPreferences(LoginActivity.PREFS_NAME,
Context.MODE_PRIVATE);

    prefs.edit().remove(LoginActivity.AUTH_TOKEN_KEY).apply();
    navigateToLogin();
}

package com.example.evcharging.adapters;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.os.Build;

```

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.core.content.ContextCompat;
import androidx.recyclerview.widget.RecyclerView;

import com.example.evcharging.R;
import com.example.evcharging.api.ApiService;
import com.example.evcharging.models.BookingApi;
import com.google.zxing.BarcodeFormat;
import com.google.zxing.WriterException;
import com.google.zxing.common.BitMatrix;
import com.google.zxing.qrcode.QRCodeWriter;

import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Locale;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;
```

```
public class BookingAdapter extends RecyclerView.Adapter<BookingAdapter.BookingViewHolder> {

    private final List<BookingApi> bookingApiList;
    private final ApiService apiService;
    private final String authToken;
    private Context context; // Keep context for resources

    // Constructor: Initializes the adapter with booking data, API service, and auth token
    public BookingAdapter(List<BookingApi> bookingApiList, ApiService apiService, String authToken) {
        this.bookingApiList = bookingApiList;
        this.apiService = apiService;
        this.authToken = authToken;
    }

    // Inflates the item layout and creates a ViewHolder
    @NonNull
    @Override
    public BookingViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        this.context = parent.getContext();
        View view = LayoutInflater.from(context).inflate(R.layout.item_booking, parent, false);
        return new BookingViewHolder(view);
    }

    // Binds data to the ViewHolder and sets click listeners
    @Override
    public void onBindViewHolder(@NonNull BookingViewHolder holder, int position) {
        BookingApi bookingApi = bookingApiList.get(position);
        holder.bind(bookingApi, context);
    }
}
```

```

// Cancellation logic is now handled inside the ViewHolder's bind method
// to simplify state management and ensure buttons are only active when they should be.

holder.btnCancelBooking.setOnClickListener(v -> {
    // Only allow cancellation if the booking status is 'Active' (0)
    if (bookingApi.status == 0) {
        cancelBooking(bookingApi, holder.getAdapterPosition());
    }
});

// Returns the number of items in the list
@Override
public int getItemCount() {
    return bookingApiList.size();
}

// Cancels a booking via the API and updates UI on success
private void cancelBooking(final BookingApi bookingApi, final int position) {
    if (position == RecyclerView.NO_POSITION) return;

    // Use the correct API endpoint for user cancellation
    apiService.cancelBooking(authToken, bookingApi.id).enqueue(new Callback<Void>() {
        @Override
        public void onResponse(@NonNull Call<Void> call, @NonNull Response<Void> response) {
            if (response.isSuccessful()) {
                Toast.makeText(context, "Booking successfully cancelled.", Toast.LENGTH_SHORT).show();

                // Update status locally to 'Cancelled' (3) and refresh the item
                bookingApi.status = 3;
                notifyDataSetChanged();
            }
        }
    });
}

```

```

    } else {
        Toast.makeText(context, "Failed to cancel booking (Error: " + response.code() + ")",
        Toast.LENGTH_SHORT).show();
    }
}

@Override
public void onFailure(@NonNull Call<Void> call, @NonNull Throwable t) {
    Toast.makeText(context, "Network Error: " + t.getMessage(), Toast.LENGTH_SHORT).show();
}
});

}

// Generates a QR code bitmap from booking ID and sets it in the ImageView
private void generateAndSetQrCode(ImageView imageView, String bookingId) {
    QRCodeWriter writer = new QRCodeWriter();
    try {
        BitMatrix bitMatrix = writer.encode(bookingId, BarcodeFormat.QR_CODE, 512, 512);
        int width = bitMatrix.getWidth();
        int height = bitMatrix.getHeight();
        Bitmap bmp = Bitmap.createBitmap(width, height, Bitmap.Config.RGB_565);
        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                bmp.setPixel(x, y, bitMatrix.get(x, y) ? Color.BLACK : Color.WHITE);
            }
        }
        imageView.setImageBitmap(bmp);
    } catch (WriterException e) {
        e.printStackTrace();
    }
}

```

```

        }

    }

// --- ViewHolder ---

public class BookingViewHolder extends RecyclerView.ViewHolder {

    TextView tvStationName, tvBookingTime, tvStatus;
    ImageView ivQrCode;
    Button btnCancelBooking;

    public BookingViewHolder(@NonNull View itemView) {
        super(itemView);
        // These IDs now EXACTLY match the item_booking.xml layout file
        tvStationName = itemView.findViewById(R.id.tvBookingStationName);
        tvBookingTime = itemView.findViewById(R.id.tvBookingTime);
        tvStatus = itemView.findViewById(R.id.tvBookingStatus);
        ivQrCode = itemView.findViewById(R.id.ivQrCode);
        btnCancelBooking = itemView.findViewById(R.id.btnCancelBooking);
    }

    public void bind(final BookingApi bookingApi, Context context) {
        tvStationName.setText("Station: " + bookingApi.stationId);
        tvBookingTime.setText(formatDateTimeRange(bookingApi.startTime, bookingApi.endTime));
        // Hide everything by default, then show based on status
        ivQrCode.setVisibility(View.GONE);
        btnCancelBooking.setVisibility(View.GONE);

        // --- THE FINAL NUMERIC STATUS FIX ---
        // Backend Enum: Active=0, Confirmed=1, Completed=2, Cancelled=3, NoShow=4
    }
}

```

```
switch (bookingApi.status) {  
    case 0: // Active (User sees this as "Pending Confirmation")  
        tvStatus.setText("ACTIVE");  
        tvStatus.setBackground(ContextCompat.getDrawable(context,  
R.drawable.status_background_pending));  
        btnCancelBooking.setVisibility(View.VISIBLE); // Show cancel button for active bookings  
        break;  
  
    case 1: // Confirmed (User sees this as "Approved")  
        tvStatus.setText("CONFIRMED");  
        tvStatus.setBackground(ContextCompat.getDrawable(context,  
R.drawable.status_background_approved));  
        ivQrCode.setVisibility(View.VISIBLE); // Show QR code for confirmed bookings  
        generateAndSetQrCode(ivQrCode, bookingApi.id);  
        break;  
  
    case 2: // Completed  
        tvStatus.setText("COMPLETED");  
        tvStatus.setBackground(ContextCompat.getDrawable(context,  
R.drawable.status_background_completed)); // Assumes a blue-ish color  
        break;  
  
    case 3: // Cancelled  
        tvStatus.setText("CANCELLED");  
        tvStatus.setBackground(ContextCompat.getDrawable(context,  
R.drawable.status_background_rejected));  
        break;  
  
    case 4: // NoShow  
    default:
```

```

        tvStatus.setText("UNKNOWN");

        tvStatus.setBackground(ContextCompat.getDrawable(context,
R.drawable.status_background_rejected));

        break;

    }

}

private String formatDateTimeRange(String start, String end) {

    try {

        DateTimeFormatter inputFormatter = null;

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

            inputFormatter = DateTimeFormatter.ISO_ZONED_DATE_TIME;

        }

        DateTimeFormatter dateFormatter = null;

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

            dateFormatter = DateTimeFormatter.ofPattern("MMM dd, yyyy", Locale.getDefault());

        }

        DateTimeFormatter timeFormatter = null;

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

            timeFormatter = DateTimeFormatter.ofPattern("hh:mm a", Locale.getDefault());

        }

        ZonedDateTime startTime = null;

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

            startTime = ZonedDateTime.parse(start, inputFormatter);

        }

        ZonedDateTime endTime = null;

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

            endTime = ZonedDateTime.parse(end, inputFormatter);

        }

    }

}

```

```
    }

    String date = null;
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        date = startTime.format(dateFormatter);
    }

    String startTimeStr = null;
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        startTimeStr = startTime.format(timeFormatter);
    }

    String endTimeStr = null;
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
        endTimeStr = endTime.format(timeFormatter);
    }

    return String.format("%s | %s - %s", date, startTimeStr, endTimeStr);
} catch (Exception e) {
    return "Invalid Date";
}
}

package com.example.evcharging.adapters;

import android.content.Context;
import android.graphics.drawable.GradientDrawable;
import android.os.Build;
import android.view.LayoutInflater;
```

```
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.core.content.ContextCompat;
import androidx.recyclerview.widget.RecyclerView;
import com.example.evcharging.R;
import com.example.evcharging.models.BookingApi;

import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Locale;

public class DashboardBookingAdapter extends
RecyclerView.Adapter<DashboardBookingAdapter.ViewHolder> {

    private final List<BookingApi> bookingApiList;
    private Context context; // Context for accessing resources

    public DashboardBookingAdapter(List<BookingApi> bookingApiList) {
        this.bookingApiList = bookingApiList;
    }

    /**
     * Updates the adapter's data set and refreshes the RecyclerView.
     * @param newBookingApis The new list of bookings to display.
     */
    public void updateBookings(List<BookingApi> newBookingApis) {
```

```
        this.bookingApiList.clear();
        this.bookingApiList.addAll(newBookingApis);
        notifyDataSetChanged();
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        this.context = parent.getContext();
        View view = LayoutInflater.from(context).inflate(R.layout.item_dashboard_booking, parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        BookingApi bookingApi = bookingApiList.get(position);
        holder.bind(bookingApi, context); // Pass context to the bind method
    }

    @Override
    public int getItemCount() {
        return bookingApiList.size();
    }

    static class ViewHolder extends RecyclerView.ViewHolder {
        TextView tvStationName, tvTime, tvStatus;

        ViewHolder(@NonNull View itemView) {
            super(itemView);
        }
    }
}
```

```

// These IDs must match your item_dashboard_booking.xml

tvStationName = itemView.findViewById(R.id.tvCardStationName);

tvTime = itemView.findViewById(R.id.tvCardTime);

tvStatus = itemView.findViewById(R.id.tvCardStatus);

}

void bind(BookingApi bookingApi, Context context) {

    tvStationName.setText(bookingApi.stationId != null ? bookingApi.stationId : "N/A");

    tvTime.setText(formatSimpleTime(bookingApi.startTime));

    int status = bookingApi.status;

    String statusText;

    int statusColor;

    // --- THIS IS THE FINAL FIX ---

    // Backend Enum: Active=0, Confirmed=1, Completed=2, Cancelled=3

    switch (status) {

        case 0: // Active

            statusText = "ACTIVE";

            statusColor = ContextCompat.getColor(context, R.color.orange_soda);

            break;

        case 1: // Confirmed

            statusText = "CONFIRMED";

            statusColor = ContextCompat.getColor(context, R.color.emerald_green);

            break;

        case 3: // Cancelled

            statusText = "CANCELLED";

            statusColor = ContextCompat.getColor(context, R.color.red_error);

            break;
    }
}

```

```

        case 2: // Completed
    default:
        statusText = "COMPLETED";
        statusColor = ContextCompat.getColor(context, R.color.cyan_blue);
        break;
    }

    tvStatus.setText(statusText);

    // Dynamically change status color safely
    if (tvStatus.getBackground() instanceof GradientDrawable) {
        GradientDrawable background = (GradientDrawable) tvStatus.getBackground().mutate();
        background.setColor(statusColor);
    }
}

private String formatSimpleTime(String isoString) {
    if (isoString == null) return "N/A";
    try {
        // The java.time APIs are cleaner and safer than SimpleDateFormat
        DateTimeFormatter inputFormatter = null;
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            inputFormatter = DateTimeFormatter.ISO_ZONED_DATE_TIME;
        }
        DateTimeFormatter timeFormatter = null;
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            timeFormatter = DateTimeFormatter.ofPattern("hh:mm a", Locale.getDefault());
        }
        ZonedDateTime zonedDateTime = null;

```

```
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
            zonedDateTime = ZonedDateTime.parse(isoString, inputFormatter);  
        }  
  
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {  
            return zonedDateTime.format(timeFormatter);  
        }  
  
    } catch (Exception e) {  
  
        // Fallback in case of parsing error  
        return "Invalid Time";  
    }  
  
    return isoString;  
}  
}  
  
}  
  
package com.example.evcharging.adapters;  
  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.ImageView;  
import android.widget.TextView;  
import androidx.annotation.NonNull;  
import androidx.recyclerview.widget.RecyclerView;  
import com.example.evcharging.R;  
import com.example.evcharging.models.Notification; // You need to create this model  
import java.util.List;  
  
public class NotificationAdapter extends RecyclerView.Adapter<NotificationAdapter.ViewHolder> {
```

```
private final List<Notification> notificationList;

public NotificationAdapter(List<Notification> notificationList) {
    this.notificationList = notificationList;
}

@NonNull
@Override
public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_notification, parent,
false);
    return new ViewHolder(view);
}

@Override
public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
    Notification notification = notificationList.get(position);
    holder.bind(notification);
}

@Override
public int getItemCount() {
    return notificationList.size();
}

public void updateData(List<Notification> newNotifications) {
    notificationList.clear();
    notificationList.addAll(newNotifications);
}
```

```
        notifyDataSetChanged();

    }

    static class ViewHolder extends RecyclerView.ViewHolder {

        ImageView ivNotificationIcon;
        TextView tvNotificationTitle, tvNotificationMessage;

        ViewHolder(@NonNull View itemView) {
            super(itemView);
            ivNotificationIcon = itemView.findViewById(R.id.ivNotificationIcon);
            tvNotificationTitle = itemView.findViewById(R.id.tvNotificationTitle);
            tvNotificationMessage = itemView.findViewById(R.id.tvNotificationMessage);
        }

        void bind(Notification notification) {
            tvNotificationTitle.setText(notification.title);
            tvNotificationMessage.setText(notification.message);
            // You can set different icons based on notification.type
            ivNotificationIcon.setImageResource(R.drawable.ic_notifications);
        }
    }
}

package com.example.evcharging.adapters;

import android.content.Context;
import android.os.Build;
import android.view.LayoutInflater;
import android.view.View;
```

```
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.core.content.ContextCompat;
import androidx.recyclerview.widget.RecyclerView;
import com.example.evcharging.R;
import com.example.evcharging.models.BookingApi;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Locale;

public class OperatorBookingAdapter extends
RecyclerView.Adapter<OperatorBookingAdapter.ViewHolder> {

    private final List<BookingApi> bookingApiList;
    private final BookingListener listener;
    private final Context context;

    public interface BookingListener {
        void onConfirm(String bookingId); // Renamed for clarity
        void onCancelByOperator(String bookingId); // Renamed for clarity
    }

    public OperatorBookingAdapter(Context context, List<BookingApi> bookingApiList, BookingListener
listener) {
        this.context = context;
        this.bookingApiList = bookingApiList;
    }
}
```

```
        this.listener = listener;
    }

    @NonNull
    @Override
    public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_operator_booking,
                parent, false);
        return new ViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull ViewHolder holder, int position) {
        BookingApi bookingApi = bookingApiList.get(position);
        holder.bind(bookingApi, listener, context);
    }

    @Override
    public int getItemCount() {
        return bookingApiList.size();
    }

    public void updateData(List<BookingApi> newBookingApis) {
        bookingApiList.clear();
        bookingApiList.addAll(newBookingApis);
        notifyDataSetChanged();
    }

    static class ViewHolder extends RecyclerView.ViewHolder {
```

```

TextView tvBookingId, tvStatus, tvStationId, tvTime, tvUserId;
LinearLayout actionButtonsLayout;
Button btnApprove, btnReject;

ViewHolder(@NonNull View itemView) {
    super(itemView);
    tvBookingId = itemView.findViewById(R.id.tvBookingId);
    tvStatus = itemView.findViewById(R.id.tvBookingStatus);
    tvStationId = itemView.findViewById(R.id.tvStationId);
    tvTime = itemView.findViewById(R.id.tvBookingTime);
    tvUserId = itemView.findViewById(R.id.tvUserId);
    actionButtonsLayout = itemView.findViewById(R.id.action_buttons_layout);
    btnApprove = itemView.findViewById(R.id.btnApprove);
    btnReject = itemView.findViewById(R.id.btnReject);
}

void bind(final BookingApi bookingApi, final BookingListener listener, Context context) {
    tvBookingId.setText("Booking ID: #" + formatId(bookingApi.id));
    tvStationId.setText("Station: " + bookingApi.stationId);
    tvUserId.setText("User NIC: " + bookingApi.ownerNIC);
    tvTime.setText(formatDateTimeRange(bookingApi.startTime, bookingApi.endTime));

    // --- STATUS AND BUTTON VISIBILITY FIX ---
    // Backend Enum: Active = 0, Confirmed = 1, Completed = 2, Cancelled = 3

    switch (bookingApi.status) {
        case 0: // Active (This is what an operator acts on)
            tvStatus.setText("ACTIVE");

```

```

        tvStatus.setBackground(ContextCompat.getDrawable(context,
R.drawable.status_background_pending));

        actionButtonsLayout.setVisibility(View.VISIBLE); // SHOW buttons

        btnApprove.setText("Confirm"); // Set button text to "Confirm"

        btnReject.setText("Cancel"); // Set button text to "Cancel"

        break;

    case 1: // Confirmed

        tvStatus.setText("CONFIRMED");

        tvStatus.setBackground(ContextCompat.getDrawable(context,
R.drawable.status_background_approved));

        actionButtonsLayout.setVisibility(View.GONE); // HIDE buttons

        break;

    case 3: // Cancelled

        tvStatus.setText("CANCELLED");

        tvStatus.setBackground(ContextCompat.getDrawable(context,
R.drawable.status_background_rejected));

        actionButtonsLayout.setVisibility(View.GONE); // HIDE buttons

        break;

    case 2: // Completed

        default:

            tvStatus.setText("COMPLETED");

            tvStatus.setBackground(ContextCompat.getDrawable(context,
R.drawable.status_background_completed));

            actionButtonsLayout.setVisibility(View.GONE); // HIDE buttons

            break;

    }

// --- CORRECTED LISTENERS ---

btnApprove.setOnClickListener(v -> listener.onConfirm(bookingApi.id));

btnReject.setOnClickListener(v -> listener.onCancelByOperator(bookingApi.id));

```

```

}

// Helper functions remain the same

private String formatId(String id) {

    if (id == null || id.length() < 8) return "N/A";

    return id.substring(id.length() - 8);

}

private String formatDateTimeRange(String start, String end) {

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

        try {

            DateTimeFormatter inputFormatter = DateTimeFormatter.ISO_ZONED_DATE_TIME;

            DateTimeFormatter dateFormatter = DateTimeFormatter.ofPattern("MMM dd, yyyy",
Locale.getDefault());

            DateTimeFormatter timeFormatter = DateTimeFormatter.ofPattern("hh:mm a",
Locale.getDefault());


            ZonedDateTime startTime = ZonedDateTime.parse(start, inputFormatter);

            ZonedDateTime endTime = ZonedDateTime.parse(end, inputFormatter);

            String date = startTime.format(dateFormatter);

            String startTimeStr = startTime.format(timeFormatter);

            String endTimeStr = endTime.format(timeFormatter);

            return String.format("%s | %s - %s", date, startTimeStr, endTimeStr);

        } catch (Exception e) {

            return "Invalid Date Format";

        }

    }

}

```

```

        return "Date formatting unavailable";
    }
}

}

/*
 * File: ApiClient.java
 * Purpose: Retrofit client singleton
 */

package com.example.evcharging.api;

import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;
import okhttp3.OkHttpClient;
import okhttp3.logging.HttpLoggingInterceptor;

public class ApiClient {

    // This URL points to your publicly hosted backend.

    private static final String BASE_URL = "http://13.62.48.213:5000/";

    private static ApiService apiService;

    public static ApiService getApiService(){

        if (apiService == null) {

            // add logging

            HttpLoggingInterceptor logging = new HttpLoggingInterceptor();
            logging.setLevel(HttpLoggingInterceptor.Level.BODY);

            OkHttpClient client = new OkHttpClient.Builder().addInterceptor(logging).build();

            Retrofit retrofit = new Retrofit.Builder()

```

```
        .baseUrl(BASE_URL)
        .client(client)
        .addConverterFactory(GsonConverterFactory.create())
        .build();

    apiService = retrofit.create(ApiService.class);

}

return apiService;

}

}

package com.example.evcharging.api;

import com.example.evcharging.models.BookingApi;
import com.example.evcharging.models.CancellationReason;
import com.example.evcharging.models.Station;
import com.example.evcharging.models.User;
import com.example.evcharging.models.Notification;

import java.util.List;
import java.util.Map;

import retrofit2.Call;
import retrofit2.http.Body;
import retrofit2.http.GET;
import retrofit2.http.Header;
import retrofit2.http.POST;
import retrofit2.http.PUT;
import retrofit2.http.Path;
import retrofit2.http.Query;
```

```
public interface ApiService {

    // --- Authentication ---

    @POST("api/auth/login")
    Call<Map<String, Object>> login(@Body Map<String, String> body);

    @POST("api/auth/register")
    Call<Map<String, Object>> register(@Body User user);

    // --- User Profile ---

    @GET("api/users/me")
    Call<User> getMyProfile(@Header("Authorization") String token);

    @PUT("api/users/me")
    Call<User> updateMyProfile(@Header("Authorization") String token, @Body User user);

    @POST("api/users/deactivate")
    Call<Void> deactivateMyAccount(@Header("Authorization") String token);

    // --- Stations ---

    @GET("api/chargingstations/active")
    Call<List<Station>> getActiveStations(@Header("Authorization") String token);

    @GET("api/chargingstations")
    Call<List<Station>> getAllStations(@Header("Authorization") String token);
```

```

// --- EV Owner Bookings ---

@GET("api/bookings/my-bookings")
Call<List<BookingApi>> getMyBookings(@Header("Authorization") String token);

@POST("api/bookings")
Call<Map<String, Object>> createBooking(@Header("Authorization") String token, @Body Map<String, String> body);

@POST("api/bookings/{id}/cancel")
Call<Void> cancelBooking(@Header("Authorization") String token, @Path("id") String bookingId);

// --- Operator/Admin Bookings ---

@GET("api/bookings")
Call<List<BookingApi>> getAllBookings(@Header("Authorization") String token);

@GET("api/bookings/{id}")
Call<BookingApi> getBookingById(@Header("Authorization") String token, @Path("id") String bookingId);

@GET("api/bookings/station/{stationId}")
Call<List<BookingApi>> getStationBookings(@Header("Authorization") String token, @Path("stationId") String stationId);

@POST("api/bookings/{id}/finalize")
Call<Void> finalizeBooking(@Header("Authorization") String token, @Path("id") String bookingId);

// --- START: CORRECTED OPERATOR ACTIONS ---

// The old "approveBooking" and "rejectBooking" methods have been completely removed.

```

```

    @POST("api/bookings/{id}/confirm")
    Call<Void> confirmBooking(@Header("Authorization") String token, @Path("id") String bookingId);

    @POST("api/bookings/{id}/cancel-by-operator")
    Call<Void> cancelBookingByOperator(
        @Header("Authorization") String token,
        @Path("id") String bookingId,
        @Body CancellationReason reason
    );
    // --- END: CORRECTED OPERATOR ACTIONS ---

    // --- Notifications ---
    @GET("api/notifications/my-notifications")
    Call<List<Notification>> getMyNotifications(@Header("Authorization") String token);

    @GET("api/notifications/my-notifications")
    Call<List<Notification>> getMyNotificationsWithQuery(
        @Header("Authorization") String token,
        @Query("includeRead") boolean includeRead,
        @Query("limit") int limit
    );
}

package com.example.evcharging.dao;

import androidx.room.Dao;
import androidx.room.Insert;

```

```
import androidx.room.OnConflictStrategy;
import androidx.room.Query;
import com.example.evcharging.models.Booking;

import java.util.List;

@Dao
public interface BookingDao {

    // This now works because Booking is a valid @Entity
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void insert(Booking booking);

    // This now works because we are querying the 'bookings' table for the Booking entity
    @Query("SELECT * FROM bookings WHERE ownerNIC = :nic")
    List<Booking> getBookingsByUserNic(String nic);

    // This now works because it queries the correct 'bookings' table
    @Query("DELETE FROM bookings")
    void deleteAll();

    // REMOVED: All methods that were trying to use or return 'BookingApi'.
    // The DAO should not know about the API model.

}

package com.example.evcharging.dao;

import androidx.room.Dao;
import androidx.room.Insert;
```

```
import androidx.room.OnConflictStrategy;
import androidx.room.Query;
import androidx.room.Update;

import com.example.evcharging.models.User;

@Dao
public interface UserDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void insert(User user);

    @Update
    void update(User user);

    @Query("SELECT * FROM users WHERE id = :id LIMIT 1")
    User getUserId(String id);

    @Query("SELECT * FROM users WHERE email = :email LIMIT 1")
    User getUserByEmail(String email);

    @Query("DELETE FROM users")
    void deleteAll();

}

package com.example.evcharging.db;

import android.content.Context;
import androidx.room.Database;
```

```

import androidx.room.Room;

import androidx.room.RoomDatabase;

import com.example.evcharging.dao.BookingDao;
import com.example.evcharging.dao.UserDao;
import com.example.evcharging.models.Booking;
import com.example.evcharging.models.BookingApi;
import com.example.evcharging.models.User;

// Add your entities to the entities array
@Database(entities = {User.class, Booking.class}, version = 2, exportSchema = false) // <-- ADD
Booking.class AND INCREMENT version

public abstract class AppDatabase extends RoomDatabase {

    // Define your DAOs here
    public abstract UserDao userDao();
    public abstract BookingDao bookingDao();

    private static volatile AppDatabase INSTANCE;

    public static AppDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (AppDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                        AppDatabase.class, "ev_charging_db")
                    // Add migrations here if you change the schema in the future
                    .fallbackToDestructiveMigration()
                    .build();
                }
            }
        }
        return INSTANCE;
    }
}

```

```
        }

    }

}

return INSTANCE;

}

}

package com.example.evcharging.fragments;

import android.app.DatePickerDialog;
import android.app.TimePickerDialog;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import com.example.evcharging.R;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;
import com.example.evcharging.models.Station;
import java.text.SimpleDateFormat;
```

```
import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
import java.util.List;
import java.util.Locale;
import java.util.Map;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class CreateBookingFragment extends Fragment {

    private static final String TAG = "CreateBookingFragment";
    private static final String ARG_TOKEN = "ARG_TOKEN";

    private Spinner spinnerStation;
    private EditText etStartTime, etEndTime;
    private ApiService apiService;
    private String authToken;
    private List<Station> stationList = new ArrayList<>();
    private Calendar startCalendar = Calendar.getInstance();
    private Calendar endCalendar = Calendar.getInstance();

    public static CreateBookingFragment newInstance(String token) {
        CreateBookingFragment fragment = new CreateBookingFragment();
        Bundle args = new Bundle();
        args.putString(ARG_TOKEN, token);
        fragment.setArguments(args);
        return fragment;
    }
}
```

```

}

@Override
public void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    if (getArguments() != null) {
        authToken = getArguments().getString(ARG_TOKEN);
    }
}

@Nullable
@Override
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,
@Nullable Bundle savedInstanceState) {
    // --- START: THIS IS THE FIX ---
    // The layout file is named 'activity_booking.xml', not 'fragment_create_booking.xml'.
    return inflater.inflate(R.layout.activity_booking, container, false);
    // --- END: FIX ---
}

@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);

    apiService = ApiClient.getApiService();
    spinnerStation = view.findViewById(R.id.spinnerStation);
    etStartTime = view.findViewById(R.id.etStartTime);
    etEndTime = view.findViewById(R.id.etEndTime);
    Button btnCreateBooking = view.findViewById(R.id.btnCreateBooking);
}

```

```

        etStartTime.setOnClickListener(v -> showDateTimePicker(startCalendar, etStartTime));
        etEndTime.setOnClickListener(v -> showDateTimePicker(endCalendar, etEndTime));
        btnCreateBooking.setOnClickListener(v -> createBooking());

        fetchStations();
    }

private void fetchStations() {
    if (authToken == null) return;

    apiService.getActiveStations(authToken).enqueue(new Callback<List<Station>>() {
        @Override
        public void onResponse(@NonNull Call<List<Station>> call, @NonNull Response<List<Station>> response) {
            if (isAdded() && response.isSuccessful() && response.body() != null) {
                stationList = response.body();
                List<String> stationNames = new ArrayList<>();
                for (Station station : stationList) {
                    stationNames.add(station.name);
                }
                if (getContext() != null) {
                    ArrayAdapter<String> adapter = new ArrayAdapter<>(getContext(),
                            android.R.layout.simple_spinner_item, stationNames);
                    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
                    spinnerStation.setAdapter(adapter);
                }
            }
        }
    });
}

@Override

```

```

public void onFailure(@NonNull Call<List<Station>> call, @NonNull Throwable t) {
    Log.e(TAG, "Failed to fetch stations: " + t.getMessage());
}

});

}

private void showDatePicker(final Calendar calendar, final EditText editText) {
    if (getContext() == null) return;
    new DatePickerDialog(getContext(), (view, year, month, dayOfMonth) -> {
        calendar.set(Calendar.YEAR, year);
        calendar.set(Calendar.MONTH, month);
        calendar.set(Calendar.DAY_OF_MONTH, dayOfMonth);

        new TimePickerDialog(getContext(), (timeView, hourOfDay, minute) -> {
            calendar.set(Calendar.HOUR_OF_DAY, hourOfDay);
            calendar.set(Calendar.MINUTE, minute);

            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'", Locale.getDefault());
            editText.setText(sdf.format(calendar.getTime()));
        }, calendar.get(Calendar.HOUR_OF_DAY), calendar.get(Calendar.MINUTE), false).show();
    }, calendar.get(Calendar.YEAR), calendar.get(Calendar.MONTH),
    calendar.get(Calendar.DAY_OF_MONTH)).show();
}

private void createBooking() {
    if (stationList.isEmpty() || spinnerStation.getSelectedItem() == null) {
        Toast.makeText(getContext(), "Please select a station", Toast.LENGTH_SHORT).show();
        return;
    }
}

```

```

String stationId = stationList.get(spinnerStation.getSelectedItemPosition()).id;
String startTime = etStartTime.getText().toString();
String endTime = etEndTime.getText().toString();

if (startTime.isEmpty() || endTime.isEmpty()) {
    Toast.makeText(getApplicationContext(), "Please select start and end times", Toast.LENGTH_SHORT).show();
    return;
}

Map<String, String> body = new HashMap<>();
body.put("stationId", stationId);
body.put("startTime", startTime);
body.put("endTime", endTime);

apiService.createBooking(authToken, body).enqueue(new Callback<Map<String, Object>>() {

    @Override
    public void onResponse(@NonNull Call<Map<String, Object>> call, @NonNull Response<Map<String, Object>> response) {
        if (isAdded() && response.isSuccessful()) {
            Toast.makeText(getApplicationContext(), "Booking created successfully!", Toast.LENGTH_SHORT).show();
        } else if(isAdded()) {
            Toast.makeText(getApplicationContext(), "Failed to create booking: " + response.code(),
                    Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onFailure(@NonNull Call<Map<String, Object>> call, @NonNull Throwable t) {
        if (isAdded())
    }
})
}

```

```
        Toast.makeText(getApplicationContext(), "Network error: " + t.getMessage(),
        Toast.LENGTH_SHORT).show();
    }
}
});
```

```
package com.example.evcharging.fragments;

import android.Manifest;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.core.app.ActivityCompat;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.example.evcharging.R;
import com.example.evcharging.adapters.BookingAdapter;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;
import com.example.evcharging.models.BookingApi;
```

```
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import java.util.ArrayList;
import java.util.List;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class DashboardFragment extends Fragment implements OnMapReadyCallback {

    private static final String TAG = "DashboardFragment";
    private static final String ARG_TOKEN = "ARG_TOKEN"; // Key for argument

    private GoogleMap mMap;
    private ApiService apiService;
    private String authToken;
    private BookingAdapter bookingAdapter;
    private final List<BookingApi> bookingList = new ArrayList<>();
    private TextView tvNoUpcomingBookings;
    private RecyclerView rvBookings;

    public static DashboardFragment newInstance(String token) {
        DashboardFragment fragment = new DashboardFragment();
        Bundle args = new Bundle();
        args.putString(ARG_TOKEN, token);
        fragment.setArguments(args);
    }
}
```

```
        return fragment;
    }

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        if (getArguments() != null) {
            authToken = getArguments().getString(ARG_TOKEN);
        }
    }

    @Nullable
    @Override
    public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,
    @Nullable Bundle savedInstanceState) {
        return inflater.inflate(R.layout.fragment_dashboard, container, false);
    }

    @Override
    public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);

        apiService = ApiClient.getApiService();
        tvNoUpcomingBookings = view.findViewById(R.id.tvNoUpcomingBookings);
        rvBookings = view.findViewById(R.id.rvBookings);

        setupRecyclerView();

        SupportMapFragment mapFragment = (SupportMapFragment)
        getChildFragmentManager().findFragmentById(R.id.map);
```

```

        if (mapFragment != null) {
            mapFragment.getMapAsync(this);
        }
    }

    @Override
    public void onResume() {
        super.onResume();
        fetchBookings();
    }

    private void setupRecyclerView() {
        rvBookings.setLayoutManager(new LinearLayoutManager(getContext(),
                LinearLayoutManager.HORIZONTAL, false));
        // --- START: THIS IS THE FIX ---
        // The constructor only takes 3 arguments. The 4th one has been removed.
        bookingAdapter = new BookingAdapter(bookingList, apiService, authToken);
        // --- END: FIX ---
        rvBookings.setAdapter(bookingAdapter);
    }

    private void fetchBookings() {
        if (authToken == null) return;
        apiService.getMyBookings(authToken).enqueue(new Callback<List<BookingApi>>() {
            @Override
            public void onResponse(@NonNull Call<List<BookingApi>> call, @NonNull
                    Response<List<BookingApi>> response) {
                if (isAdded() && response.isSuccessful() && response.body() != null) {
                    List<BookingApi> bookings = response.body();
                    bookingList.clear();
                }
            }
        });
    }
}

```

```

        bookingList.addAll(bookings);

        bookingAdapter.notifyDataSetChanged();

    }

    if (bookings.isEmpty()) {

        rvBookings.setVisibility(View.GONE);

        tvNoUpcomingBookings.setVisibility(View.VISIBLE);

    } else {

        rvBookings.setVisibility(View.VISIBLE);

        tvNoUpcomingBookings.setVisibility(View.GONE);

    }

}

@Override

public void onFailure(@NonNull Call<List<BookingApi>> call, @NonNull Throwable t) {

    Log.e(TAG, "Failed to fetch bookings: " + t.getMessage());

}

});

}

@Override

public void onMapReady(@NonNull GoogleMap googleMap) {

    mMap = googleMap;

    LatLng sriLanka = new LatLng(7.8731, 80.7718);

    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(sriLanka, 8));

    if (getContext() != null && ActivityCompat.checkSelfPermission(getContext(),
Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {

        mMap.setMyLocationEnabled(true);

    }

}

```

```
    }

}

package com.example.evcharging.fragments;

import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import com.example.evcharging.R;
import com.example.evcharging.adapters.BookingAdapter;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;
import com.example.evcharging.models.BookingApi;
import java.util.ArrayList;
import java.util.List;
import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class MyBookingsFragment extends Fragment {

    private static final String TAG = "MyBookingsFragment";
```

```
private static final String ARG_TOKEN = "ARG_TOKEN";\n\nprivate RecyclerView rvMyBookings;\n\nprivate BookingAdapter adapter;\n\nprivate final List<BookingApi> bookingList = new ArrayList<>();\n\nprivate ApiService apiService;\n\nprivate String authToken;\n\n\npublic static MyBookingsFragment newInstance(String token) {\n    MyBookingsFragment fragment = new MyBookingsFragment();\n\n    Bundle args = new Bundle();\n    args.putString(ARG_TOKEN, token);\n\n    fragment.setArguments(args);\n\n    return fragment;\n}\n\n\n@Override\npublic void onCreate(@Nullable Bundle savedInstanceState) {\n    super.onCreate(savedInstanceState);\n\n    if (getArguments() != null) {\n        authToken = getArguments().getString(ARG_TOKEN);\n    }\n}\n\n\n@Nullable\n@Override\npublic View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,\n@Nullable Bundle savedInstanceState) {\n    return inflater.inflate(R.layout.fragment_my_bookings, container, false);
```

```
}

@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);

    apiService = ApiClient.getApiService();
    rvMyBookings = view.findViewById(R.id.rvMyBookings);
    setupRecyclerView();
}

@Override
public void onResume() {
    super.onResume();
    fetchMyBookings();
}

private void setupRecyclerView() {
    rvMyBookings.setLayoutManager(new LinearLayoutManager(getContext()));
    // --- START: THIS IS THE FIX ---
    // The constructor only takes 3 arguments. The 4th one has been removed.
    adapter = new BookingAdapter(bookingsList, apiService, authToken);
    // --- END: FIX ---
    rvMyBookings.setAdapter(adapter);
}

private void fetchMyBookings() {
    if (authToken == null) return;
    apiService.getMyBookings(authToken).enqueue(new Callback<List<BookingApi>>() {
```



```
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;
import androidx.swiperefreshlayout.widget.SwipeRefreshLayout;

import com.example.evcharging.R;
import com.example.evcharging.adapters.NotificationAdapter;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;
import com.example.evcharging.models.Notification;

import java.util.ArrayList;
import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class NotificationsFragment extends Fragment {

    private static final String TAG = "NotificationsFragment";
    private static final String ARG_TOKEN = "ARG_TOKEN"; // Key for argument bundle

    private RecyclerView rvNotifications;
    private TextView tvNoNotifications;
    private ProgressBar progressBar;
    private SwipeRefreshLayout swipeRefreshLayout;
    private NotificationAdapter adapter;
```

```

private ApiService apiService;

private String authToken;

private final List<Notification> notificationList = new ArrayList<>();

// --- THIS IS THE MISSING METHOD THAT FIXES THE BUILD ERROR ---

public static NotificationsFragment newInstance(String token) {

    NotificationsFragment fragment = new NotificationsFragment();

    Bundle args = new Bundle();
    args.putString(ARG_TOKEN, token);
    fragment.setArguments(args);

    return fragment;
}

// --- END OF FIX ---


@Override

public void onCreate(@Nullable Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    // Retrieve arguments here, which is safer for the fragment lifecycle

    if (getArguments() != null) {

        authToken = getArguments().getString(ARG_TOKEN);

    }
}

@Nullable

@Override

public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,
@Nullable Bundle savedInstanceState) {

    return inflater.inflate(R.layout.fragment_notifications, container, false);
}

```

```
@Override  
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {  
    super.onViewCreated(view, savedInstanceState);  
  
    apiService = ApiClient.getApiService();  
    rvNotifications = view.findViewById(R.id.rvNotifications);  
    tvNoNotifications = view.findViewById(R.id.tvNoNotifications);  
    progressBar = view.findViewById(R.id.progressBar);  
    swipeRefreshLayout = view.findViewById(R.id.swipeRefreshLayout);  
  
    setupRecyclerView();  
    setupSwipeRefresh();  
}  
  
@Override  
public void onResume() {  
    super.onResume();  
    if (authToken != null) {  
        fetchNotifications(true); // Show initial loading progress bar  
    }  
}  
  
private void setupRecyclerView() {  
    rvNotifications.setLayoutManager(new LinearLayoutManager(getContext()));  
    adapter = new NotificationAdapter(notificationList);  
    rvNotifications.setAdapter(adapter);  
}
```

```

private void setupSwipeRefresh() {
    swipeRefreshLayout.setOnRefreshListener(() -> fetchNotifications(false)); // Don't show initial
progress bar on refresh
}

private void fetchNotifications(boolean showInitialProgress) {
    if (showInitialProgress) {
        progressBar.setVisibility(View.VISIBLE);
    }

    rvNotifications.setVisibility(View.GONE);
    tvNoNotifications.setVisibility(View.GONE);

    apiService.getMyNotifications(authToken).enqueue(new Callback<List<Notification>>() {
        @Override
        public void onResponse(@NonNull Call<List<Notification>> call, @NonNull
Response<List<Notification>> response) {
            if (!isAdded()) return;
            progressBar.setVisibility(View.GONE);
            swipeRefreshLayout.setRefreshing(false);

            if (response.isSuccessful() && response.body() != null) {
                List<Notification> notifications = response.body();
                if (notifications.isEmpty()) {
                    tvNoNotifications.setText("You have no notifications.");
                    tvNoNotifications.setVisibility(View.VISIBLE);
                } else {
                    rvNotifications.setVisibility(View.VISIBLE);
                    adapter.updateData(notifications);
                }
            } else {

```

```
        String errorText = "Failed to load notifications. Code: " + response.code();
        tvNoNotifications.setText(errorText);
        tvNoNotifications.setVisibility(View.VISIBLE);
        Log.e(TAG, "API Error: " + response.code() + " - " + response.message());
    }
}

@Override
public void onFailure(@NonNull Call<List<Notification>> call, @NonNull Throwable t) {
    if (!isAdded()) return;
    progressBar.setVisibility(View.GONE);
    swipeRefreshLayout.setRefreshing(false);
    tvNoNotifications.setText("Network error. Please try again.");
    tvNoNotifications.setVisibility(View.VISIBLE);
    Log.e(TAG, "Network Failure: ", t);
}
});

}
}
```

```
package com.example.evcharging.fragments;
```

```
import android.app.AlertDialog;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
```

```
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.recyclerview.widget.LinearLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import com.example.evcharging.R;
import com.example.evcharging.adapters.OperatorBookingAdapter;
import com.example.evcharging.api.ApiClient;
import com.example.evcharging.api.ApiService;
import com.example.evcharging.models.BookingApi;
import com.example.evcharging.models.CancellationReason; // <- Import the new model

import java.util.ArrayList;
import java.util.List;

import retrofit2.Call;
import retrofit2.Callback;
import retrofit2.Response;

public class OperatorBookingsFragment extends Fragment implements
OperatorBookingAdapter.BookingListener {

    private static final String TAG = "OperatorBookingsFrag";
```

```
private static final String ARG_TOKEN = "ARG_TOKEN";

private static final String ARG_STATION_ID = "ARG_STATION_ID";


private RecyclerView rvOperatorBookings;

private OperatorBookingAdapter adapter;

private final List<BookingApi> bookingList = new ArrayList<>();

private ApiService apiService;

private String authToken;

private String stationId;


private ProgressBar progressBar;

private TextView tvNoBookings;


public static OperatorBookingsFragment newInstance(String token, String stationId) {

    OperatorBookingsFragment fragment = new OperatorBookingsFragment();

    Bundle args = new Bundle();

    args.putString(ARG_TOKEN, token);

    args.putString(ARG_STATION_ID, stationId);

    fragment.setArguments(args);

    return fragment;

}

@Override

public void onCreate(@Nullable Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    if (getArguments() != null) {

        authToken = getArguments().getString(ARG_TOKEN);

        stationId = getArguments().getString(ARG_STATION_ID);

    }

}
```

```
}

@NoArgsConstructor
@Override
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container,
@Nullable Bundle savedInstanceState) {
    return inflater.inflate(R.layout.fragment_operator_bookings, container, false);
}

@Override
public void onViewCreated(@NonNull View view, @Nullable Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    apiService = ApiClient.getApiService();

    progressBar = view.findViewById(R.id.progressBar);
    tvNoBookings = view.findViewById(R.id.tvNoBookings);
    rvOperatorBookings = view.findViewById(R.id.rvOperatorBookings);
    setupRecyclerView();

    if (TextUtils.isEmpty(stationId)) {
        Log.e(TAG, "Station ID is missing! Cannot fetch bookings.");
        progressBar.setVisibility(View.GONE);
        tvNoBookings.setText("Operator station ID not found.");
        tvNoBookings.setVisibility(View.VISIBLE);
    }
}

@Override
public void onResume() {
```

```

super.onResume();

if (!TextUtils.isEmpty(authToken) && !TextUtils.isEmpty(stationId)) {
    fetchBookingsForStation();
}

}

private void setupRecyclerView() {

    rvOperatorBookings.setLayoutManager(new LinearLayoutManager(getContext()));
    adapter = new OperatorBookingAdapter(getContext(), bookingList, this);
    rvOperatorBookings.setAdapter(adapter);
}

private void fetchBookingsForStation() {

    // This method remains unchanged
    progressBar.setVisibility(View.VISIBLE);
    tvNoBookings.setVisibility(View.GONE);
    rvOperatorBookings.setVisibility(View.GONE);

    apiService.getStationBookings(authToken, stationId).enqueue(new Callback<List<BookingApi>>() {

        @Override
        public void onResponse(@NonNull Call<List<BookingApi>> call, @NonNull Response<List<BookingApi>> response) {
            if (!isAdded()) return;
            progressBar.setVisibility(View.GONE);

            if (response.isSuccessful() && response.body() != null) {
                List<BookingApi> bookings = response.body();
                if (bookings.isEmpty()) {
                    tvNoBookings.setText("No bookings found for this station.");
                }
            }
        }

        @Override
        public void onFailure(Call<List<BookingApi>> call, Throwable t) {
            Log.e("OperatorBookingsFragment", "Error fetching bookings: " + t.getMessage());
        }
    });
}

```

```

        tvNoBookings.setVisibility(View.VISIBLE);

    } else {
        rvOperatorBookings.setVisibility(View.VISIBLE);
        adapter.updateData(bookings);
    }
} else {
    tvNoBookings.setText("Failed to load bookings. Code: " + response.code());
    tvNoBookings.setVisibility(View.VISIBLE);
    Log.e(TAG, "API Error on getStationBookings. Code: " + response.code() + " Message: " +
response.message());
}

}

@Override
public void onFailure(@NonNull Call<List<BookingApi>> call, @NonNull Throwable t) {
    if (!isAdded()) return;
    progressBar.setVisibility(View.GONE);
    tvNoBookings.setText("Network Error. Please try again.");
    tvNoBookings.setVisibility(View.VISIBLE);
    Log.e(TAG, "Network failure on getStationBookings: ", t);
}
});

}

@Override
public void onConfirm(String bookingId) {
    // This method remains unchanged
    apiService.confirmBooking(authToken, bookingId).enqueue(createActionCallback("confirmed"));
}

```

```

// --- START: NEW onCancelByOperator and showCancelDialog methods ---

@Override

public void onCancelByOperator(String bookingId) {
    showCancelDialog(bookingId);
}

private void showCancelDialog(final String bookingId) {
    if (getContext() == null) return;

    final EditText reasonInput = new EditText(getContext());
    reasonInput.setHint("Enter reason for cancellation");
    reasonInput.setPadding(40, 40, 40, 40);

    new AlertDialog.Builder(getContext())
        .setTitle("Cancel Booking")
        .setMessage("Please provide a reason for cancelling this booking.")
        .setView(reasonInput)
        .setPositiveButton("Submit", (dialog, which) -> {
            String reason = reasonInput.getText().toString().trim();
            if (TextUtils.isEmpty(reason)) {
                Toast.makeText(getContext(), "Reason cannot be empty.", Toast.LENGTH_SHORT).show();
            } else {
                CancellationReason cancellationReason = new CancellationReason(reason);
                apiService.cancelBookingByOperator(authToken, bookingId, cancellationReason)
                    .enqueue(createActionCallback("cancelled"));
            }
        })
        .setNegativeButton("Back", null)
}

```

```

.show();

}

// --- END: NEW METHODS ---


private Callback<Void> createActionCallback(String action) {

    // This method remains unchanged

    return new Callback<Void>() {

        @Override

        public void onResponse(@NonNull Call<Void> call, @NonNull Response<Void> response) {

            if (isAdded() && response.isSuccessful()) {

                Toast.makeText(getApplicationContext(), "Booking successfully " + action + ".",
                    Toast.LENGTH_SHORT).show();

                fetchBookingsForStation(); // Refresh the list

            } else if (isAdded()) {

                Toast.makeText(getApplicationContext(), "Action failed. Code: " + response.code(),
                    Toast.LENGTH_SHORT).show();

            }

        }

    }

    @Override

    public void onFailure(@NonNull Call<Void> call, @NonNull Throwable t) {

        if (isAdded()) {

            Toast.makeText(getApplicationContext(), "Network Error. Please try again.",
                Toast.LENGTH_SHORT).show();

        }

    }

};

}

```

```
package com.example.evcharging.models;

import androidx.annotation.NonNull;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

// This class is ONLY for the local Room database.

@Entity(tableName = "bookings") // This annotation fixes the "no such table" error.

public class Booking {

    @PrimaryKey
    @NonNull
    public String id; // This annotation fixes the "must have @PrimaryKey" error.

    public String ownerNIC;
    public String stationId;
    public String startTime;
    public String endTime;
    public int status;
    public String qrCode;
    public double totalAmount;
    public String createdAt;

    // A default constructor is required by Room.

    public Booking() {
        this.id = ""; // Initialize to a non-null value
    }

    // Optional: A constructor for creating instances manually.
```

```
public Booking(@NonNull String id, String ownerNIC, String stationId, String startTime, String endTime,
int status) {

    this.id = id;
    this.ownerNIC = ownerNIC;
    this.stationId = stationId;
    this.startTime = startTime;
    this.endTime = endTime;
    this.status = status;
}

}
```

```
package com.example.evcharging.models;
```

```
import androidx.annotation.NonNull;
```

```
// Note: I am removing the Room annotations (@Entity, @PrimaryKey, etc.)
// because the fields from the API now differ from what you might store locally.
// If you need Room persistence, a separate "local" model is recommended.
```

```
public class BookingApi {
```

```
    @NonNull
```

```
    public String id;
```

```
// This field now correctly matches the backend's "ownerNIC"
```

```
    public String ownerNIC;
```

```
    public String stationId;
```

```
    public String startTime;
```

```
    public String endTime;
```

```
// This field will now correctly receive the numeric status from the API

public int status;

public String qrCode;

public double totalAmount; // Using double for decimal values

public String createdAt;

public String updatedAt;

public String confirmedAt;

public String cancelledAt;

// A default constructor is good practice for libraries like Gson

public BookingApi() {}

// A full constructor for convenience

public BookingApi(@NonNull String id, String ownerNIC, String stationId, String startTime, String
endTime, int status) {

    this.id = id;

    this.ownerNIC = ownerNIC;

    this.stationId = stationId;

    this.startTime = startTime;

    this.endTime = endTime;

    this.status = status;

}

package com.example.evcharging.models;

import com.google.gson.annotations.SerializedName;
```

```
public class CancellationReason {

    @SerializedName("reason")
    private String reason;

    public CancellationReason(String reason) {
        this.reason = reason;
    }

    public String getReason() {
        return reason;
    }
}

package com.example.evcharging.models;

import com.google.gson.annotations.SerializedName;
import java.util.Date;
import java.util.Map;

// This class now accurately reflects the backend's Notification.cs model
public class Notification {

    @SerializedName("id")
    public String id;

    @SerializedName("recipientNIC")
    public String recipientNIC;
```

```
@SerializedName("title")
public String title;

@SerializedName("message")
public String message;

// Backend sends 'type' as a number (enum)
@SerializedName("type")
public int type;

@SerializedName("relatedEntityId")
public String relatedEntityId;

@SerializedName("isRead")
public boolean isRead;

// Backend sends 'priority' as a number (enum)
@SerializedName("priority")
public int priority;

@SerializedName("createdAt")
public Date createdAt;

// Optional fields
@SerializedName("metadata")
public Map<String, Object> metadata;

// Default constructor for Gson
```

```

public Notification() {}

}

/*
 * File: Station.java
 *
 * Purpose: Model for Charging Station
 */

package com.example.evcharging.models;

public class Station {

    public String id;
    public String name;
    public String location;

    // --- START: ADD THESE TWO FIELDS ---
    public double latitude;
    public double longitude;
    // --- END: ADD THESE TWO FIELDS ---

    public int type; // e.g., 1 for Type 1 AC, 2 for CCS, 3 for CHAdeMO
    public int totalSlots;
    public double pricePerHour;
    public int availableSlots; // Number of currently available slots
    public String status; // e.g., "active", "maintenance", "coming_soon"

    public Station() {
        // Default constructor for Gson
    }
}

```

```
// --- UPDATE THE CONSTRUCTOR TO INCLUDE LATITUDE AND LONGITUDE ---  
  
public Station(String id, String name, String location, double latitude, double longitude, int type, int  
totalSlots, double pricePerHour, int availableSlots, String status) {  
  
    this.id = id;  
  
    this.name = name;  
  
    this.location = location;  
  
    this.latitude = latitude; // Add this line  
  
    this.longitude = longitude; // Add this line  
  
    this.type = type;  
  
    this.totalSlots = totalSlots;  
  
    this.pricePerHour = pricePerHour;  
  
    this.availableSlots = availableSlots;  
  
    this.status = status;  
  
}  
  
// No getters and setters needed since fields are public
```

```
}
```

```
package com.example.evcharging.models;  
  
import androidx.room.ColumnInfo;  
import androidx.room.Entity;  
import androidx.room.PrimaryKey;  
import androidx.annotation.NonNull;  
  
import com.google.gson.annotations.SerializedName;  
  
@Entity(tableName = "users")  
public class User {
```

```
@PrimaryKey  
@NonNull  
@ColumnInfo(name = "id")  
public String nic; // National ID Card - used as primary key  
  
@ColumnInfo(name = "first_name")  
public String firstName;  
  
@ColumnInfo(name = "last_name")  
public String lastName;  
  
@ColumnInfo(name = "email")  
public String email;  
  
@ColumnInfo(name = "password")  
public String password; // Note: Storing plain text passwords is not secure.  
  
@ColumnInfo(name = "role")  
public int role;  
  
@SerializedName("phoneNumber")  
public String phoneNumber;  
  
// --- ADD THIS LINE ---  
@SerializedName("stationId")  
public String stationId; // This will hold the operator's assigned station ID
```

```

    @ColumnInfo(name = "is_active")

    public boolean active;

    // Room requires a public, no-argument constructor
    public User() {}

    // You can keep other constructors for convenience
    public User(@NonNull String nic, String firstName, String lastName, String email, String password, int
role, String phoneNumber) {

        this.nic = nic;
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.password = password;
        this.role = role;
        this.phoneNumber = phoneNumber;
        this.active = true;
    }
}

/*
 * File: QRCodeGenerator.java
 * Purpose: Helper to generate QR bitmap using ZXing
 */
package com.example.evcharging.utils;

import android.graphics.Bitmap;

import com.google.zxing.BarcodeFormat;

```

```

import com.google.zxing.WriterException;
import com.journeyapps.barcodescanner.BarcodeEncoder;

public class QRCodeGenerator {

    /**
     * Generate a QR code bitmap for given text
     *
     * @param text content to encode
     *
     * @param size pixel size
     *
     * @return Bitmap
     */
    public static Bitmap generate(String text, int size) throws WriterException {
        BarcodeEncoder encoder = new BarcodeEncoder();
        return encoder.encodeBitmap(text, BarcodeFormat.QR_CODE, size, size);
    }
}

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    tools:context=".fragments.CreateBookingFragment">

    <!-- ... ImageView, TextView, and Spinner are unchanged ... -->

    <ImageView

```

```
    android:id="@+id/imageView"
    android:layout_width="84dp"
    android:layout_height="80dp"
    android:layout_marginTop="32dp"
    android:contentDescription="Booking Icon"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:srcCompat="@drawable/booking_icon" />
```

```
<TextView
    android:id="@+id/tvBookingTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:text="Create New Booking"
    android:textSize="24sp"
    android:textStyle="bold"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/imageView" />
```

```
<Spinner
    android:id="@+id/spinnerStation"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
    android:background="@android:color/transparent"
    android:minHeight="48dp"
```

```
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tvBookingTitle" />
```

```
<!-- START: MODIFICATION for Start Time -->
```

```
<EditText
    android:id="@+id/etStartTime"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:hint="Select Start Time"
    android:minHeight="48dp"
    android:drawableEnd="@drawable/ic_calendar"
    android:focusable="false"
    android:clickable="true"
    android:textColor="@color/charcoal_black"
    android:textColorHint="#757575"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/spinnerStation"/>
```

```
<!-- END: MODIFICATION for Start Time -->
```

```
<!-- START: MODIFICATION for End Time -->
```

```
<EditText
    android:id="@+id/etEndTime"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
```

```
    android:hint="Select End Time"  
    android:minHeight="48dp"  
    android:drawableEnd="@drawable/ic_calendar"  
    android:focusable="false"  
    android:clickable="true"  
    android:textColor="@color/charcoal_black"  
    android:textColorHint="#757575"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@id/etStartTime" />  
    <!-- END: MODIFICATION for End Time -->
```

```
<Button  
    android:id="@+id	btnCreateBooking"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="32dp"  
    android:backgroundTint="@color/emerald_green"  
    android:padding="12dp"  
    android:text="Create Booking"  
    android:textSize="16sp"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@id/etEndTime" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activities.ConfirmBookingActivity">

    <!-- Consistent Top App Bar -->
    <com.google.android.material.appbar.AppBarLayout
        android:id="@+id/appBarLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent">

        <androidx.appcompat.widget.Toolbar
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="@color/midnight_navy"
            app:title="Confirm Booking"
            app:titleTextColor="@color/snow_white" />

    </com.google.android.material.appbar.AppBarLayout>

    <com.google.android.material.card.MaterialCardView
        android:id="@+id/card_details"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_margin="24dp"
```

```
    app:cardCornerRadius="8dp"
    app:cardElevation="4dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/appBarLayout">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="16dp">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Booking Details"
            android:textColor="@color/midnight_navy"
            android:textSize="20sp"
            android:textStyle="bold" />

        <TextView
            android:id="@+id/tvConfirmBookingId"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="16dp"
            android:textSize="16sp"
            tools:text="Booking ID: bk_123abc456" />

        <TextView
```

```
    android:id="@+id/tvConfirmStationId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:textSize="16sp"
    tools:text="Station ID: st_789xyz" />

<TextView
    android:id="@+id/tvConfirmStatus"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:textSize="18sp"
    android:textStyle="bold"
    tools:text="Status: Approved"
    tools:textColor="@color/emerald_green"/>

</LinearLayout>
</com.google.android.material.card.MaterialCardView>

<Button
    android:id="@+id/btnConfirmBooking"
    android:layout_width="0dp"
    android:layout_height="60dp"
    android:layout_margin="24dp"
    android:backgroundTint="@color/emerald_green"
    android:text="Confirm Booking"
    android:textAllCaps="false"
    android:textColor="@color/snow_white"
    android:textSize="18sp"
```

```
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".activities.CustomScannerActivity">

    <!-- The core scanner view provided by the ZXing library -->

    <!-- Our custom back button, placed at the top-left -->
    <com.journeyapps.barcodescanner.DecoratedBarcodeView
        android:id="@+id/zxing_barcode_scanner"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:zxing_framing_rect_height="250dp"
        app:zxing_framing_rect_width="250dp"
        app:zxing_scanner_layout="@layout/zxing_capture"
        tools:layout_editor_absoluteX="-82dp"
        tools:layout_editor_absoluteY="175dp" />

    <ImageButton
```

```
    android:id="@+id	btn_back"
    android:layout_width="48dp"
    android:layout_height="48dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:background="?attr/selectableItemBackgroundBorderless"
    android:contentDescription="Back"
    android:src="@drawable/ic_arrow_back"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:tint="@android:color/white" />

</androidx.constraintlayout.widget.ConstraintLayout>

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#F5F5F5"
    tools:context=".activities.DashboardActivity">
```

```
<!-- Top App Bar -->
<com.google.android.material.appbar.AppBarLayout
    android:id="@+id/appBarLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
    app:layout_constraintTop_toTopOf="parent">

    <androidx.appcompat.widget.Toolbar
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:background="@color/midnight_navy"
        app:title="Dashboard"
        app:titleTextColor="@color/snow_white" />

</com.google.android.material.appbar.AppBarLayout>

<!-- Profile Icon on Toolbar -->
<ImageView
    android:id="@+id/ivProfile"
    android:layout_width="40dp"
    android:layout_height="40dp"
    android:layout_marginEnd="16dp"
    android:src="@drawable/ic_account_circle"
    app:layout_constraintBottom_toBottomOf="@+id/appBarLayout"
    app:layout_constraintEnd_toEndOf="@+id/appBarLayout"
    app:layout_constraintTop_toTopOf="@+id/appBarLayout"
    app:tint="@color/snow_white" />

<!-- This FrameLayout is the container that will display your different fragments -->
<FrameLayout
    android:id="@+id/fragment_container"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintTop_toBottomOf="@+id/appBarLayout"
```

```
    app:layout_constraintBottom_toTopOf="@+id/bottom_navigation"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent" />

<!-- Bottom Navigation Bar -->
<com.google.android.material.bottomnavigation.BottomNavigationView
    android:id="@+id/bottom_navigation"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?android:attr/windowBackground"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:menu="@menu/bottom_nav_menu"

    app:itemIconTint="@color/bottom_nav_icon_tint_selector"
    app:itemTextColor="@color/bottom_nav_icon_tint_selector" />
</androidx.constraintlayout.widget.ConstraintLayout>

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:background="@color/midnight_navy"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="24dp">
```

```
<ImageView  
    android:id="@+id/imageView2"  
    android:layout_width="130dp"  
    android:layout_height="123dp"  
    android:layout_marginTop="80dp"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent"  
    app:srcCompat="@drawable/ev_logo" />
```

```
<TextView  
    android:id="@+id/title"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="16dp"  
    android:text="EV CHARGING"  
    android:textColor="@color/snow_white"  
    android:textSize="28sp"  
    android:textStyle="bold"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/imageView2" />
```

```
<EditText  
    android:id="@+id/etEmail"  
    android:layout_width="0dp"  
    android:layout_height="48dp"  
    android:layout_marginTop="28dp"  
    android:background="@drawable/rounded_white"
```

```
    android:hint="Email"  
    android:inputType="textEmailAddress"  
    android:padding="12dp"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/title" />
```

```
<EditText  
    android:id="@+id/etPassword"  
    android:layout_width="0dp"  
    android:layout_height="48dp"  
    android:layout_marginTop="16dp"  
    android:background="@drawable/rounded_white"  
    android:hint="Password"  
    android:inputType="textPassword"  
    android:padding="12dp"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/etEmail" />
```

```
<Button  
    android:id="@+id/btnLogin"  
    android:layout_width="0dp"  
    android:layout_height="48dp"  
    android:layout_marginTop="16dp"  
    android:backgroundTint="@color/emerald_green"  
    android:text="Login"  
    android:textAllCaps="false"  
    android:textColor="@color/snow_white"
```

```
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etPassword" />

<TextView
    android:id="@+id/tvRegister"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="32dp"
    android:text="Register"
    android:textColor="@color/cyan_blue"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/btnLogin" />
</androidx.constraintlayout.widget.ConstraintLayout>

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#F5F5F5"
    tools:context=".activities.OperatorDashboardActivity">

<!-- Top App Bar with Title and Logout -->
<com.google.android.material.appbar.AppBarLayout
    android:id="@+id/appBarLayout"
    android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"

        app:layout_constraintTop_toTopOf="parent">

        <androidx.appcompat.widget.Toolbar

            android:layout_width="match_parent"

            android:layout_height="?attr/actionBarSize"

            android:background="@color/midnight_navy">

            <TextView

                android:id="@+id/toolbar_title"

                android:layout_width="wrap_content"

                android:layout_height="wrap_content"

                android:text="Bookings"

                android:textColor="@color/snow_white"

                android:textSize="20sp"

                android:textStyle="bold"

                android:layout_gravity="start" />

        </androidx.appcompat.widget.Toolbar>

    </com.google.android.material.appbar.AppBarLayout>

    <ImageView

        android:id="@+id/ivLogout"

        android:layout_width="40dp"

        android:layout_height="40dp"

        android:layout_marginEnd="16dp"

        android:background="?attr/selectableItemBackgroundBorderless"

        android:clickable="true"

        android:focusable="true"

        android:padding="4dp"

        android:src="@drawable/ic_logout"

        app:layout_constraintBottom_toBottomOf="@+id/appBarLayout"
```

```
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="@+id/appBarLayout"
    app:tint="@color/snow_white" />

    <!-- Fragment container to hold the content -->
    <FrameLayout
        android:id="@+id/operator_fragment_container"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toBottomOf="@+id/appBarLayout"
        app:layout_constraintBottom_toTopOf="@+id/operator_bottom_navigation"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <!-- Bottom Navigation Bar -->
    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/operator_bottom_navigation"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="?android:attr/windowBackground"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:itemIconTint="@color/bottom_nav_icon_tint_selector"
        app:itemTextColor="@color/bottom_nav_icon_tint_selector"
        app:menu="@menu/operator_bottom_nav_menu" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="24dp"
    tools:context=".activities.ProfileActivity">

    <TextView
        android:id="@+id/tvProfileTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="My Profile"
        android:textSize="28sp"
        android:textStyle="bold"
        android:textColor="@color/midnight_navy"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/tilProfileFirstName"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:hint="First Name"
        app:layout_constraintEnd_toEndOf="parent"
```

```
    app:layout_constraintStart_toStartOf="parent"

    app:layout_constraintTop_toBottomOf="@+id/tvProfileTitle">

<com.google.android.material.textfield.TextInputEditText
    android:id="@+id/etProfileFirstName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPersonName" />

</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilProfileLastName"
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:hint="Last Name"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etProfileFirstName">

<com.google.android.material.textfield.TextInputEditText
    android:id="@+id/etProfileLastName"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPersonName" />

</com.google.android.material.textfield.TextInputLayout>

<com.google.android.material.textfield.TextInputLayout
```

```
        android:id="@+id/tilProfilePhone"
        style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:hint="Phone Number"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/etProfileLastName">

    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/etProfilePhone"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="phone" />
</com.google.android.material.textfield.TextInputLayout>

<Button
    android:id="@+id/btnUpdateProfile"
    android:layout_width="0dp"
    android:layout_height="48dp"
    android:layout_marginTop="24dp"
    android:backgroundTint="@color/emerald_green"
    android:text="Update Profile"
    android:textAllCaps="false"
    android:textColor="@color/snow_white"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tilProfilePhone" />
```

```
<Button  
    android:id="@+id	btnDeactivate"  
    style="@style/Widget.MaterialComponents.Button.TextButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="16dp"  
    android:text="Deactivate My Account"  
    android:textColor="@android:color/holo_red_dark"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id	btnUpdateProfile" />  
</androidx.constraintlayout.widget.ConstraintLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>  
<androidx.constraintlayout.widget.ConstraintLayout  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:padding="16dp"  
    tools:context=".activities.RegisterActivity">  
  
<TextView  
    android:id="@+id/tvRegisterTitle"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="80dp"  
    android:text="Register New Account"  
    android:textColor="#140F0F"
```

```
        android:textSize="20sp"  
        android:textStyle="bold"  
        app:layout_constraintEnd_toEndOf="parent"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintTop_toTopOf="parent" />
```

```
<EditText  
        android:id="@+id/etNic"  
        android:layout_width="0dp"  
        android:layout_height="wrap_content"  
        android:layout_marginTop="16dp"  
        android:hint="NIC (National ID)"  
        android:inputType="text"  
        android:textColor="#140F0F"  
        android:textColorHint="#646464"  
        app:layout_constraintEnd_toEndOf="parent"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintTop_toBottomOf="@+id/tvRegisterTitle" />
```

```
<EditText  
        android:id="@+id/etFirstName"  
        android:layout_width="0dp"  
        android:layout_height="wrap_content"  
        android:layout_marginTop="8dp"  
        android:hint="First Name"  
        android:inputType="textPersonName"  
        android:textColor="#140F0F"  
        android:textColorHint="#646464"  
        app:layout_constraintEnd_toEndOf="parent"
```

```
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etNic" />

<EditText
    android:id="@+id/etLastName"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:hint="Last Name"
    android:inputType="textPersonName"
    android:textColor="#140F0F"
    android:textColorHint="#646464"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etFirstName" />

<EditText
    android:id="@+id/etEmail"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:hint="Email"
    android:inputType="textEmailAddress"
    android:textColor="#140F0F"
    android:textColorHint="#646464"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/etLastName" />
```

```
<EditText  
    android:id="@+id/etPassword"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="8dp"  
    android:hint="Password"  
    android:inputType="textPassword"  
    android:textColor="#140F0F"  
    android:textColorHint="#646464"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/etEmail" />
```

```
<EditText  
    android:id="@+id/etPhoneNumber"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="8dp"  
    android:hint="Phone Number"  
    android:inputType="phone"  
    android:textColor="#140F0F"  
    android:textColorHint="#646464"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/etPassword" />
```

```
<Button  
    android:id="@+id/btnRegister"  
    android:layout_width="0dp"
```

```
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:text="Register"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/etPhoneNumber" />
    </androidx.constraintlayout.widget.ConstraintLayout>

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/midnight_navy">

    <ImageView
        android:id="@+id/splash_logo"
        android:layout_width="150dp"
        android:layout_height="150dp"
        app:srcCompat="@drawable/ev_logo"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <ProgressBar
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
    android:layout_marginTop="32dp"

    app:layout_constraintTop_toBottomOf="@+id/splash_logo"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
```

```
    android:background="#F5F5F5"
```

```
    tools:context=".fragments.DashboardFragment">
```

```
<!-- Summary Cards (Unchanged) -->
```

```
<LinearLayout
```

```
    android:id="@+id/cardRow"
```

```
    android:layout_width="0dp"
```

```
    android:layout_height="wrap_content"
```

```
    android:clipToPadding="false"
```

```
    android:orientation="horizontal"
```

```
    android:padding="8dp"
```

```
    app:layout_constraintEnd_toEndOf="parent"
```

```
    app:layout_constraintStart_toStartOf="parent"
```

```
    app:layout_constraintTop_toTopOf="parent">
```

```
<!-- All 3 MaterialCardViews for Pending, Approved, Nearby go here... -->
```

```
<com.google.android.material.card.MaterialCardView  
    android:layout_width="0dp"  
    android:layout_height="100dp"  
    android:layout_margin="8dp"  
    android:layout_weight="1"  
    app:cardBackgroundColor="@color/emerald_green"  
    app:cardCornerRadius="12dp">
```

```
<TextView  
    android:id="@+id/tvPending"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:gravity="center"  
    android:text="0\nPending"  
    android:textColor="@android:color/white"  
    android:textSize="18sp"  
    android:textStyle="bold" />  
</com.google.android.material.card.MaterialCardView>
```

```
<com.google.android.material.card.MaterialCardView  
    android:layout_width="0dp"  
    android:layout_height="100dp"  
    android:layout_margin="8dp"  
    android:layout_weight="1"  
    app:cardBackgroundColor="@color/cyan_blue"  
    app:cardCornerRadius="12dp">
```

```
<TextView  
    android:id="@+id/tvApproved"
```

```
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:text="0\nApproved"
    android:textColor="@android:color/white"
    android:textSize="18sp"
    android:textStyle="bold" />
</com.google.android.material.card.MaterialCardView>

<com.google.android.material.card.MaterialCardView
    android:layout_width="0dp"
    android:layout_height="100dp"
    android:layout_margin="8dp"
    android:layout_weight="1"
    app:cardBackgroundColor="@color/orange_soda"
    app:cardCornerRadius="12dp">

    <TextView
        android:id="@+id/tvNearby"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="0\nNearby"
        android:textColor="@android:color/white"
        android:textSize="18sp"
        android:textStyle="bold" />
</com.google.android.material.card.MaterialCardView>

</LinearLayout>
```

```
<!-- Upcoming Bookings Title -->

<TextView

    android:id="@+id/tvUpcomingTitle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="16dp"
    android:layout_marginTop="16dp"
    android:text="Upcoming Bookings"
    android:textColor="@color/charcoal_black"
    android:textSize="18sp"
    android:textStyle="bold"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/cardRow" />
```

```
<!-- RecyclerView for Bookings -->

<androidx.recyclerview.widget.RecyclerView

    android:id="@+id/rvBookings"
    android:layout_width="0dp"
    android:layout_height="125dp"
    android:layout_marginTop="8dp"
    android:clipToPadding="false"
    android:paddingStart="16dp"
    android:paddingEnd="16dp"
    android:visibility="gone"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tvUpcomingTitle"
    tools:visibility="visible" />
```

```
<!-- **NEW**: Empty State TextView -->

<TextView

    android:id="@+id/tvNoUpcomingBookings"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:text="No upcoming bookingApis found."

    android:textColor="#888"

    android:textSize="16sp"

    android:visibility="visible"

    app:layout_constraintBottom_toBottomOf="@+id/rvBookings"

    app:layout_constraintEnd_toEndOf="@+id/rvBookings"

    app:layout_constraintStart_toStartOf="@+id/rvBookings"

    app:layout_constraintTop_toTopOf="@+id/rvBookings" />
```

```
<!-- Google Map Section -->

<TextView

    android:id="@+id/tvStationsTitle"

    android:layout_width="wrap_content"

    android:layout_height="wrap_content"

    android:layout_marginStart="16dp"

    android:layout_marginTop="16dp"

    android:text="Nearby Stations"

    android:textColor="@color/charcoal_black"

    android:textSize="18sp"

    android:textStyle="bold"

    app:layout_constraintStart_toStartOf="parent"

    app:layout_constraintTop_toBottomOf="@+id/rvBookings" />
```

```
<androidx.cardview.widget.CardView  
    android:layout_width="0dp"  
    android:layout_height="0dp"  
    android:layout_margin="16dp"  
    app:cardCornerRadius="12dp"  
    app:cardElevation="4dp"  
    app:layout_constraintBottom_toBottomOf="parent"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toBottomOf="@+id/tvStationsTitle">  
  
<fragment  
    android:id="@+id/map"  
    android:name="com.google.android.gms.maps.SupportMapFragment"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />  
</androidx.cardview.widget.CardView>  
</androidx.constraintlayout.widget.ConstraintLayout>  
  
<?xml version="1.0" encoding="utf-8"?>  
<androidx.recyclerview.widget.RecyclerView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent">  
  
<androidx.recyclerview.widget.RecyclerView  
    android:id="@+id/rvMyBookings"  
    android:layout_width="match_parent"
```

```
        android:layout_height="match_parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>

    <?xml version="1.0" encoding="utf-8"?>
    <androidx.swiperefreshlayout.widget.SwipeRefreshLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:id="@+id/swipeRefreshLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context="com.example.evcharging.fragments.NotificationsFragment"
        tools:ignore="MissingClass">

        <androidx.constraintlayout.widget.ConstraintLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="#F5F5F5">

            <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/rvNotifications"
                android:layout_width="0dp"
                android:layout_height="0dp"
                app:layout_constraintTop_toTopOf="parent"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                android:paddingTop="8dp"
                android:clipToPadding="false">
        
```

```
tools:listitem="@layout/item_notification"
    android:visibility="gone"
    tools:visibility="visible"/>

<TextView
    android:id="@+id/tvNoNotifications"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="No notifications found."
    android:textSize="18sp"
    android:visibility="gone"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    tools:visibility="visible"/>

<ProgressBar
    android:id="@+id/progressBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:visibility="visible"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

```
</androidx.swiperefreshlayout.widget.SwipeRefreshLayout>

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#F5F5F5"
    android:padding="16dp">

    <TextView
        android:id="@+id/tvTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="All Customer Bookings"
        android:textColor="@color/charcoal_black"
        android:textSize="20sp"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/rvOperatorBookings"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginTop="16dp"
        android:visibility="gone" />

```

```
    app:layout_constraintTop_toBottomOf="@+id/tvTitle"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    tools:listitem="@layout/item_operator_booking"
    tools:visibility="visible"/>

<TextView
    android:id="@+id/tvNoBookings"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="No bookingApis found."
    android:textSize="18sp"
    android:visibility="gone"
    app:layout_constraintTop_toBottomOf="@+id/tvTitle"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"/>

<ProgressBar
    android:id="@+id/progressBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:visibility="visible"
    app:layout_constraintTop_toBottomOf="@+id/tvTitle"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>  
<com.google.android.material.card.MaterialCardView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:layout_marginHorizontal="16dp"  
    android:layout_marginTop="8dp"  
    android:layout_marginBottom="8dp"  
    app:cardBackgroundColor="@android:color/white"  
    app:cardCornerRadius="12dp"  
    app:cardElevation="4dp">
```

```
<androidx.constraintlayout.widget.ConstraintLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:padding="16dp">
```

```
<!-- Station Name/ID -->  
<TextView  
    android:id="@+id/tvBookingStationName"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:drawableStart="@drawable/ic_station"  
    android:drawablePadding="8dp"  
    android:gravity="center_vertical"  
    android:textColor="@color/charcoal_black"  
    android:textSize="16sp"
```

```
        android:textStyle="bold"  
        app:layout_constraintEnd_toStartOf="@+id/tvBookingStatus"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintTop_toTopOf="parent"  
        app:tint="@color/charcoal_black"  
        tools:text="Station: ST001" />
```

```
<!-- Status Pill -->
```

```
<TextView  
    android:id="@+id/tvBookingStatus"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:background="@drawable/status_background_pending"  
    android:paddingHorizontal="12dp"  
    android:paddingVertical="4dp"  
    android:textColor="@android:color/white"  
    android:textSize="12sp"  
    android:textStyle="bold"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintTop_toTopOf="parent"  
    tools:text="CONFIRMED" />
```

```
<!-- Date & Time Range -->
```

```
<TextView  
    android:id="@+id/tvBookingTime"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:layout_marginTop="8dp"  
    android:drawableStart="@drawable/ic_calendar"
```

```
        android:drawablePadding="8dp"
        android:gravity="center_vertical"
        android:textColor="#555"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tvBookingStationName"
        app:tint="@color/charcoal_black"
        tools:text="Oct 28, 2025 | 10:00 AM - 12:00 PM" />

    <!-- QR Code (Initially hidden) -->
    <ImageView
        android:id="@+id/ivQrCode"
        android:layout_width="120dp"
        android:layout_height="120dp"
        android:layout_marginTop="16dp"
        android:visibility="gone"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tvBookingTime"
        tools:src="@tools:sample/avatars"
        tools:visibility="visible" />

    <!-- Cancel Button (Initially hidden) -->
    <com.google.android.material.button.MaterialButton
        android:id="@+id btnCancelBooking"
        style="@style/Widget.MaterialComponents.Button.OutlinedButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
```

```
        android:text="Cancel Booking"  
        android:textColor="@color/red_error"  
        android:visibility="gone"  
        app:layout_constraintEnd_toEndOf="parent"  
        app:layout_constraintStart_toStartOf="parent"  
        app:layout_constraintTop_toBottomOf="@+id/tvBookingTime"  
        app:strokeColor="@color/red_error"  
        tools:visibility="visible" />  
  
</androidx.constraintlayout.widget.ConstraintLayout>  
  
</com.google.android.material.card.MaterialCardView>  
  
  
<?xml version="1.0" encoding="utf-8"?>  
  
<com.google.android.material.card.MaterialCardView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:app="http://schemas.android.com/apk/res-auto"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="180dp"  
    android:layout_height="wrap_content"  
    android:layout_marginEnd="12dp"  
    app:cardBackgroundColor="@color/midnight_navy"  
    app:cardCornerRadius="12dp">  
  
  
<androidx.constraintlayout.widget.ConstraintLayout  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:padding="12dp">  
  
  
<TextView  
    android:id="@+id/tvCardStationName"
```

```
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:maxLines="1"
    android:ellipsize="end"
    android:textColor="@color/snow_white"
    android:textSize="16sp"
    android:textStyle="bold"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    tools:text="Station Name" />
```

```
<TextView
    android:id="@+id/tvCardTime"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:textColor="@color/snow_white"
    android:textSize="12sp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tvCardStationName"
    tools:text="Oct 28, 10:00 AM" />
```

```
<TextView
    android:id="@+id/tvCardStatus"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="12dp"
```

```
    android:paddingHorizontal="8dp"
    android:paddingVertical="2dp"
    android:background="@drawable/status_background"
    android:textColor="@color/snow_white"
    android:textSize="10sp"
    android:textStyle="bold"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tvCardTime"
    tools:text="PENDING"
    tools:backgroundTint="@color/emerald_green" />
</androidx.constraintlayout.widget.ConstraintLayout>
</com.google.android.material.card.MaterialCardView>
```

```
<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginHorizontal="16dp"
    android:layout_marginTop="8dp"
    android:layout_marginBottom="8dp"
    app:cardBackgroundColor="@android:color/white"
    app:cardCornerRadius="8dp"
    app:cardElevation="2dp">

    <androidx.constraintlayout.widget.ConstraintLayout
        android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:padding="16dp">>

    <ImageView
        android:id="@+id/ivNotificationIcon"
        android:layout_width="24dp"
        android:layout_height="24dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:src="@drawable/ic_notifications"
        app:tint="@color/cyan_blue"/>

    <TextView
        android:id="@+id/tvNotificationTitle"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:textColor="@color/charcoal_black"
        android:textSize="16sp"
        android:textStyle="bold"
        app:layout_constraintStart_toEndOf="@+id/ivNotificationIcon"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        tools:text="Booking Confirmed" />

    <TextView
        android:id="@+id/tvNotificationMessage"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
```

```
    android:layout_marginTop="4dp"
    android:textColor="#555555"
    app:layout_constraintStart_toStartOf="@+id/tvNotificationTitle"
    app:layout_constraintTop_toBottomOf="@+id/tvNotificationTitle"
    app:layout_constraintEnd_toEndOf="parent"
    tools:text="Your booking API for station ST001 has been confirmed." />
</androidx.constraintlayout.widget.ConstraintLayout>
</com.google.android.material.card.MaterialCardView>

<?xml version="1.0" encoding="utf-8"?>
<com.google.android.material.card.MaterialCardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginHorizontal="16dp"
    android:layout_marginTop="8dp"
    android:layout_marginBottom="8dp"
    app:cardBackgroundColor="@android:color/white"
    app:cardCornerRadius="12dp"
    app:cardElevation="4dp">

<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="16dp">

    <!-- Booking ID (Top Left) -->
```

```
<TextView  
    android:id="@+id/tvBookingId"  
    android:layout_width="0dp"  
    android:layout_height="wrap_content"  
    android:textColor="@color/charcoal_black"  
    android:textSize="16sp"  
    android:textStyle="bold"  
    app:layout_constraintStart_toStartOf="parent"  
    app:layout_constraintTop_toTopOf="parent"  
    app:layout_constraintEnd_toStartOf="@+id/tvBookingStatus"  
    tools:text="Booking ID: #c31b2195" />  
  
<!-- Status Pill (Top Right) -->  
  
<TextView  
    android:id="@+id/tvBookingStatus"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:paddingHorizontal="12dp"  
    android:paddingVertical="4dp"  
    android:background="@drawable/status_background_pending"  
    android:textColor="@android:color/white"  
    android:textSize="12sp"  
    android:textStyle="bold"  
    app:layout_constraintEnd_toEndOf="parent"  
    app:layout_constraintTop_toTopOf="parent"  
    tools:text="PENDING" />  
  
<!-- This new Guideline acts as a clean divider -->  
<androidx.constraintlayout.widget.Guideline
```

```
        android:id="@+id/guideline_divider"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        app:layout_constraintGuide_begin="50dp" />

    <!-- Station ID -->
    <TextView
        android:id="@+id/tvStationId"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="12dp"
        android:drawableStart="@drawable/ic_station"
        android:drawablePadding="8dp"
        android:textColor="@color/charcoal_black"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tvBookingId"
        app:tint="@color/charcoal_black"
        tools:text="Station: station_001" />

    <!-- Date & Time -->
    <TextView
        android:id="@+id/tvBookingTime"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:drawableStart="@drawable/ic_calendar"
        android:drawablePadding="8dp"
        app:layout_constraintStart_toStartOf="parent"
```

```
    app:layout_constraintTop_toBottomOf="@+id/tvStationId"
    app:tint="@color/charcoal_black"
    tools:text="Oct 07, 2025 | 08:16 AM - 04:16 PM" />

<!-- User ID -->
<TextView
    android:id="@+id/tvUserId"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:drawableStart="@drawable/ic_account_circle"
    android:drawablePadding="8dp"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tvBookingTime"
    app:tint="@color/charcoal_black"
    tools:text="User: perera@email.com" />

<!-- Action buttons layout -->
<!-- THIS IS THE CRITICAL FIX: The top constraint is now to tvUserId, giving it space to appear -->
<LinearLayout
    android:id="@+id/action_buttons_layout"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:layout_marginTop="16dp"
    android:visibility="gone"
    tools:visibility="visible"
    app:layout_constraintTop_toBottomOf="@+id/tvUserId"
    app:layout_constraintStart_toStartOf="parent"
```

```
    app:layout_constraintEnd_toEndOf="parent">

    <com.google.android.material.button.MaterialButton
        android:id="@+id	btnReject"
        style="@style/Widget.MaterialComponents.Button.OutlinedButton"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Cancel"
        android:layout_marginEnd="8dp"
        android:textColor="@color/red_error"
        app:strokeColor="@color/red_error"
    />

    <com.google.android.material.button.MaterialButton
        android:id="@+id	btnApprove"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Confirm"
        android:layout_marginStart="8dp"
        app:backgroundTint="@color/emerald_green"
    />

</LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
</com.google.android.material.card.MaterialCardView>
```

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/navigation_dashboard"
        android:icon="@drawable/ic_dashboard"
        android:title="Dashboard"/>

    <item
        android:id="@+id/navigation_create_booking"
        android:icon="@drawable/ic_add_circle"
        android:title="New Booking"/>

    <!-- CHANGED: Replaced QR Scanner with My Bookings -->
    <item
        android:id="@+id/navigation_my_bookings"
        android:icon="@drawable/ic_view_list"
        android:title="My Bookings"/>

    <!-- ADD THIS NEW ITEM -->
    <item
        android:id="@+id/navigation_notifications"
        android:icon="@drawable/ic_notifications"
        android:title="Notifications" />

</menu>
```

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/navigation_operator_bookings"
        android:icon="@drawable/ic_view_list"
        android:title="Bookings" />

    <!-- ADD THIS NEW ITEM -->

    <item
        android:id="@+id/navigation_operator_notifications"
        android:icon="@drawable/ic_notifications"
        android:title="Notifications" />

    <item
        android:id="@+id/navigation_scan_qr"
        android:icon="@drawable/ic_qr_code_scanner"
        android:title="Scan QR" />

</menu>
```

## 7.2 EV Charging Web Application Codes

```
//Footer.jsx

import { Container } from "react-bootstrap";

export default function Footer() {

    // Main footer container with background, text color, padding, and top border
    return (
        <footer style={{

            background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',
            color: '#F9FAFB',
            textAlign: 'center',
            padding: '20px 0',
            borderTop: '1px solid rgba(255, 255, 255, 0.1)',
            marginTop: 'auto'

        }}>

        <Container>
            <div style={{

                display: 'flex',
                alignItems: 'center',
                justifyContent: 'center',
                gap: '8px',
                marginBottom: '8px'

            }}>

            <div style={{

                fontSize: '1.2rem',
                background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',
                WebkitBackgroundClip: 'text',


```

```
        WebkitTextFillColor: 'transparent',
        backgroundClip: 'text'
    }}>
    
</div>
<span style={{ fontWeight: '600' }}>
    EV Charge
</span>
</div>
<p style={{ margin: 0,
    color: '#9CA3AF',
    fontSize: '0.9rem'
}}>
    &copy; {new Date().getFullYear()} EV Charging Station Booking System
</p>
<p style={{ margin: '4px 0 0 0',
    color: '#6C757D',
    fontSize: '0.8rem'
}}>
    Power Your Journey
</p>
</Container>
</footer>
);
}
```

```
//Header.jsx

import { Navbar, Container } from "react-bootstrap";

import { useNavigate } from "react-router-dom";


export default function Header() {

    // Initialize React Router's navigation hook

    const navigate = useNavigate();


    // Render the header/navbar component

    return (

        <Navbar style={{

            background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',

            borderBottom: '1px solid rgba(255, 255, 255, 0.1)'

        }} variant="dark" expand="lg" className="px-3">

            <Container fluid>

                <Navbar.Brand

                    href="/"

                    style={{

                        display: 'flex',

                        alignItems: 'center',

                        gap: '12px',

                        cursor: 'pointer',

                        fontWeight: '700',

                        fontSize: '1.5rem'

                    }}

                    onClick={(e) => {

                        e.preventDefault();

                        navigate('/');

                    }}

                >


```

```

>

<div style={{

  fontSize: '1.8rem',

  background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',

  WebkitBackgroundClip: 'text',

  WebkitTextFillColor: 'transparent',

  backgroundClip: 'text'

}}>

  ⚡

</div>

EV Charge - Backoffice

</Navbar.Brand>

</Container>

</Navbar>

);

}

//Sidebar.jsx

import { Nav, Button } from "react-bootstrap";

import { Link, useLocation, useNavigate } from "react-router-dom";



export default function Sidebar() {

  // React Router hooks for current path and navigation

  const location = useLocation();

  const navigate = useNavigate();



  // Logout handler: clears authentication data and navigates to home

  const handleLogout = () => {

    localStorage.removeItem("token"); // Clear token

```

```
sessionStorage.clear(); // Clear session storage
navigate("/");
};

// Base style for navigation links
const navLinkStyle = {
  color: "#121212",
  padding: "12px 16px",
  borderRadius: "12px",
  marginBottom: "8px",
  fontWeight: "500",
  transition: "all 0.3s ease",
  border: "none",
  textDecoration: "none",
  display: "block",
};

// Style for active/current navigation link
const activeNavLinkStyle = {
  ...navLinkStyle,
  background: "linear-gradient(135deg, #00C853 0%, #00B4D8 100%)",
  color: "white",
  transform: "translateX(8px)",
};

// Style applied on hover for non-active links
const hoverStyle = {
  background: "linear-gradient(135deg, #00C85320 0%, #00B4D820 100%)",
  color: "#121212",
};
```

```
    transform: "translateX(8px)",  
};  
  
// Sidebar container  
return (  
  <div  
    style={{  
      background: "#F9FAFB",  
      borderRight: "1px solid rgba(255, 255, 255, 0.1)",  
      minHeight: "100vh",  
      padding: "24px",  
      width: "280px",  
      boxShadow: "0 0 20px rgba(0, 0, 0, 0.1)",  
      position: "relative",  
    }}  
  >  
  {/* Header */}  
  <div  
    style={{  
      display: "flex",  
      alignItems: "center",  
      gap: "12px",  
      marginBottom: "32px",  
      paddingBottom: "16px",  
      borderBottom: "1px solid #E5E7EB",  
    }}  
  >  
  <div  
    style={{
```

```
    fontSize: "2rem",
    background: "linear-gradient(135deg, #00C853 0%, #00B4D8 100%)",
    WebkitBackgroundClip: "text",
    WebkitTextFillColor: "transparent",
    backgroundClip: "text",
  }}
>
  ⚡
</div>
<h5
  style={{
    margin: 0,
    color: "#121212",
    fontWeight: "700",
    fontSize: "1.25rem",
  }}
>
  Admin Panel
</h5>
</div>

/* Navigation Links */
<Nav className="flex-column">
  <Nav.Link
    as={Link}
    to="/dashboard/users"
    style={
      location.pathname === "/dashboard/users"
      ? activeNavLinkStyle
    }
  >
```

```

        : navLinkStyle
    }

    onMouseEnter={(e) => {
        if (location.pathname !== "/dashboard/users") {
            e.target.style.background = hoverStyle.background;
            e.target.style.color = hoverStyle.color;
            e.target.style.transform = hoverStyle.transform;
        }
    }}

    onMouseLeave={(e) => {
        if (location.pathname !== "/dashboard/users") {
            e.target.style.background = "";
            e.target.style.color = navLinkStyle.color;
            e.target.style.transform = "";
        }
    }}
}

>

   Manage Users

</Nav.Link>

<Nav.Link
    as={Link}
    to="/dashboard/stations"
    style={
        location.pathname === "/dashboard/stations"
        ? activeNavLinkStyle
        : navLinkStyle
    }
    onMouseEnter={(e) => {

```

```

    if (location.pathname !== "/dashboard/stations") {
      e.target.style.background = hoverStyle.background;
      e.target.style.color = hoverStyle.color;
      e.target.style.transform = hoverStyle.transform;
    }
  }

onMouseLeave={(e) => {
  if (location.pathname !== "/dashboard/stations") {
    e.target.style.background = "";
    e.target.style.color = navLinkStyle.color;
    e.target.style.transform = "";
  }
}
>

■ Manage Stations
</Nav.Link>

```

```

<Nav.Link
  as={Link}
  to="/dashboard/ev-owners"
  style={
    location.pathname === "/dashboard/ev-owners"
    ? activeNavLinkStyle
    : navLinkStyle
  }
  onMouseEnter={(e) => {
    if (location.pathname !== "/dashboard/ev-owners") {
      e.target.style.background = hoverStyle.background;
      e.target.style.color = hoverStyle.color;
    }
  }
}

```

```

    e.target.style.transform = hoverStyle.transform;
}

})

onMouseLeave={(e) => {
  if (location.pathname !== "/dashboard/ev-owners") {
    e.target.style.background = "";
    e.target.style.color = navLinkStyle.color;
    e.target.style.transform = "";
  }
}

>

🚗 Manage EV Owners

</Nav.Link>

</Nav>

```

```

{/* Logout Button */}

<div
  style={{
    position: "absolute",
    bottom: "64px",
    left: "24px",
    right: "24px",
  }}
>

<Button
  variant="outline-danger"
  onClick={handleLogout}
  style={{
    width: "100%",
  }}

```

```
borderRadius: "12px",
padding: "10px 0",
fontWeight: "500",
transition: "all 0.3s ease",
border: "1px solid #ef4444",
}

onMouseEnter={(e) => {
  e.target.style.background =
  "linear-gradient(135deg, #ef4444 0%, #dc2626 100%)";
  e.target.style.color = "white";
}

onMouseLeave={(e) => {
  e.target.style.background = "transparent";
  e.target.style.color = "#ef4444";
}

}
>

█ Logout

</Button>

</div>

/* Footer */

<div
style={{
position: "absolute",
bottom: "24px",
left: "24px",
right: "24px",
paddingTop: "8px",
borderTop: "1px solid #E5E7EB",
```

```

        )}
      >
      <p
        style={{
          color: "#6C757D",
          fontSize: "0.85rem",
          margin: 0,
          textAlign: "center",
        }}
      >
        Power Your Journey
      </p>
    </div>
  </div>
);

}

//chargingStation.jsx

import { useEffect, useState } from "react";
import { Table, Button, Modal, Form, Alert } from "react-bootstrap";

export default function ChargingStations() {
  // --- State variables ---
  const [stations, setStations] = useState([]);
  const [showModal, setShowModal] = useState(false);
  const [editMode, setEditMode] = useState(false);
  const [selectedStation, setSelectedStation] = useState(null);
  const [error, setError] = useState("");
  const [success, setSuccess] = useState("");
}

```

```

const [loading, setLoading] = useState(false);

// Form fields

const [name, setName] = useState("");
const [location, setLocation] = useState("");
const [type, setType] = useState("AC");
const [totalSlots, setSlots] = useState(1);
const [pricePerHour, setPricePerHour] = useState(0.01);
const [status, setStatus] = useState(0);

const token = localStorage.getItem("token");

// --- Fetch all stations from API ---

const fetchStations = async () => {
  setLoading(true);
  try {
    const res = await fetch("http://localhost:5082/api/chargingstations", {
      headers: { "Content-Type": "application/json", Authorization: `Bearer ${token}` },
    });
    if (!res.ok) throw new Error("Failed to fetch stations");
    const data = await res.json();
    setStations(data);
  } catch (err) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
};

```

```

// --- Effect: load stations on component mount ---
useEffect(() => {
  fetchStations();
}, []);

// --- Delete a station ---
const handleDelete = async (id) => {
  if (!window.confirm("Are you sure you want to delete this station?")) return;
  try {
    const res = await fetch(`http://localhost:5082/api/chargingstations/${id}`, {
      method: "DELETE",
      headers: { Authorization: `Bearer ${token}` },
    });
    if (!res.ok) throw new Error("Failed to delete station");
    setSuccess("Station deleted successfully");
    fetchStations();
  } catch (err) {
    setError(err.message);
  }
};

// --- Change station status (activate/deactivate) ---
const handleChangeStatus = async (station, newStatus) => {
  try {
    if (newStatus === 1) {
      // Deactivate station
      const res = await fetch(`http://localhost:5082/api/chargingstations/${station.id}/deactivate`, {
        method: "POST",
        headers: { Authorization: `Bearer ${token}` },
      });
    }
  }
};

```

```

    });

    if (!res.ok) throw new Error("Failed to deactivate station");

    setSuccess("Station deactivated successfully");

} else {

    // Activate station

    const payload = {

        Name: station.name,

        Location: station.location,

        Type: station.type,

        TotalSlots: station.totalSlots,

        PricePerHour: station.pricePerHour,

        Status: 0,
    };

    const res = await fetch(`http://localhost:5082/api/chargingstations/${station.id}`, {

        method: "PUT",

        headers: { "Content-Type": "application/json", Authorization: `Bearer ${token}` },

        body: JSON.stringify(payload),
    });

    if (!res.ok) throw new Error("Failed to activate station");

    setSuccess("Station activated successfully");
}

fetchStations();

} catch (err) {

    setError(err.message);
}

};

// --- Open modal for add/edit ---

const handleOpenModal = (station = null) => {

```

```

if (station) {
    // Edit mode
    setSelectedStation(station);
    setEditMode(true);
    setName(station.name);
    setLocation(station.location);
    setType(station.type === 0 ? "AC" : "DC");
    setSlots(station.totalSlots);
    setPricePerHour(station.pricePerHour || 0.01);
    setStatus(station.status);
} else {
    // Add new mode
    setSelectedStation(null);
    setEditMode(false);
    setName("");
    setLocation("");
    setType("AC");
    setSlots(1);
    setPricePerHour(0.01);
    setStatus(0);
}
setShowModal(true);
};

// --- Save station (create or update) ---
const handleSave = async () => {
    const payload = {
        Name: name,
        Location: location,

```

```

Type: type === "AC" ? 0 : 1,
TotalSlots: totalSlots,
PricePerHour: pricePerHour,
Status: status,
};

const url = editMode
? `http://localhost:5082/api/chargingstations/${selectedStation.id}`
: "http://localhost:5082/api/chargingstations";
const method = editMode ? "PUT" : "POST";

try {
  const res = await fetch(url, {
    method,
    headers: { "Content-Type": "application/json", Authorization: `Bearer ${token}` },
    body: JSON.stringify(payload),
  });
  if (!res.ok) throw new Error("Failed to save station");
  setSuccess(editMode ? "Station updated successfully" : "Station created successfully");
  setShowModal(false);
  fetchStations();
} catch (err) {
  setError(err.message);
}
};

// --- Render status badge for table ---
const renderStatusBadge = (status) => {
  switch (status) {

```

```

    case 0: return <span style={{ padding: '6px 12px', borderRadius: '20px', fontSize: '0.85rem',  

fontWeight: '500', background: 'rgba(0, 200, 83, 0.1)', color: '#00C853' }}> 🔍 Active</span>;  

    case 1: return <span style={{ padding: '6px 12px', borderRadius: '20px', fontSize: '0.85rem',  

fontWeight: '500', background: 'rgba(108, 117, 125, 0.1)', color: '#6C757D' }}> 🛑 Inactive</span>;  

    case 2: return <span style={{ padding: '6px 12px', borderRadius: '20px', fontSize: '0.85rem',  

fontWeight: '500', background: 'rgba(255, 183, 3, 0.1)', color: '#FFB703' }}> 🔧 Maintenance</span>;  

    default: return <span style={{ padding: '6px 12px', borderRadius: '20px', fontSize: '0.85rem',  

fontWeight: '500', background: 'rgba(108, 117, 125, 0.1)', color: '#6C757D' }}> ❓ Unknown</span>;  

  }  

};  

// --- Auto-clear success/error messages ---  

useEffect(() => {  

  if (error || success) {  

    const timer = setTimeout(() => {  

      setError("");  

      setSuccess("");  

    }, 5000);  

    return () => clearTimeout(timer);  

  }
}, [error, success]);  

// --- Render component UI ---  

return (
  <div style={{ padding: '24px' }}>  

    /* Header Section */  

    <div style={{  

      display: 'flex',  

      alignItems: 'center',  

      gap: '16px',  

    }}>

```

```
marginBottom: '32px'

}}>

<div style={{

  fontSize: '2.5rem',

  background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',

  WebkitBackgroundClip: 'text',

  WebkitTextFillColor: 'transparent',

  backgroundClip: 'text'

}}>





</div>

<div>

  <h1 style={{

    color: '#121212',

    margin: 0,

    fontWeight: '700',

    fontSize: '2rem'

}}>

    Charging Stations

  </h1>

  <p style={{

    color: '#6C757D',

    margin: 0,

    fontSize: '1.1rem'

}}>

    Manage all charging stations and their availability

  </p>

</div>

</div>
```

```
/* Alerts */

{error && (
  <div style={{
    background: 'rgba(255, 183, 3, 0.1)',
    color: '#FFB703',
    padding: '16px',
    borderRadius: '12px',
    border: '1px solid rgba(255, 183, 3, 0.3)',
    marginBottom: '24px',
    fontWeight: '500'
  }}>
  {error}
</div>
)})

{success && (
  <div style={{
    background: 'rgba(0, 200, 83, 0.1)',
    color: '#00C853',
    padding: '16px',
    borderRadius: '12px',
    border: '1px solid rgba(0, 200, 83, 0.3)',
    marginBottom: '24px',
    fontWeight: '500'
  }}>
  {success}
</div>
)}
```

```

/* Action Bar */

<div style={{

  display: 'flex',

  justifyContent: 'space-between',

  alignItems: 'center',

  marginBottom: '24px'

}}>

  <div style={{ fontSize: '1rem', color: '#6C757D', fontWeight: '500' }}>

    Total Stations: <strong style={{ color: '#121212' }}>{stations.length}</strong>

  </div>

  <Button

    onClick={() => handleOpenModal()}

    style={{

      background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',
      border: 'none',
      borderRadius: '12px',
      fontWeight: '600',
      padding: '12px 24px',
      fontSize: '1rem'

    }}

    >

      + Add New Station

    </Button>

  </div>

/* Stations Table */

<div style={{

  background: '#F9FAFB',

```

```
borderRadius: '20px',
padding: '32px',
boxShadow: '0 10px 40px rgba(0, 0, 0, 0.1)',
border: '1px solid rgba(255, 255, 255, 0.1'

}>

{loading ? (
<div style={{

  textAlign: 'center',
  padding: '40px',
  color: '#6C757D'

}}>

<div className="spinner" style={{

  width: '40px',
  height: '40px',
  border: '3px solid transparent',
  borderTop: '3px solid #00C853',
  borderRadius: '50%',

  animation: 'spin 1s linear infinite',
  margin: '0 auto 16px'

}}></div>

Loading charging stations...

</div>

):(


<Table hover responsive style={{

  margin: 0,
  border: 'none'

}}>

<thead>

<tr style={{
```

```
background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',
color: '#F9FAFB'

}}>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Station Name</th>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Location</th>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Type</th>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Slots</th>

<th style={{

padding: '16px',
```

```
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Price/Hour</th>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Status</th>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem',
textAlign: 'center'

}}>Actions</th>

</tr>
</thead>
<tbody>
{stations.length > 0 ? (
  stations.map((s) => (
    <tr key={s.id} style={{

borderBottom: '1px solid #E5E7EB',
transition: 'all 0.3s ease'

}}>
<td style={{

padding: '16px',
fontWeight: '600',
```

```

        color: '#121212'

    }}>{s.name}</td>

<td style={{

    padding: '16px',

    color: '#121212'

}}>{s.location}</td>

<td style={{ padding: '16px' }}>

<span style={{

    padding: '6px 12px',

    borderRadius: '20px',

    fontSize: '0.85rem',

    fontWeight: '500',

    background: s.type === 0

    ? 'linear-gradient(135deg, #00C853 20 0%, #00B4D8 20 100%)'

    : 'linear-gradient(135deg, #FFB703 20 0%, #FF9100 20 100%)',

    color: s.type === 0 ? '#00C853' : '#FFB703'

}}>

{s.type === 0 ? "⚡ AC" : "⚡ DC"}

</span>

</td>

<td style={{

    padding: '16px',

    color: '#121212',

    fontWeight: '500',

    textAlign: 'center'

}}>{s.totalSlots}</td>

<td style={{

    padding: '16px',

    color: '#121212',
```

```
fontWeight: '600'

}}>

${s.pricePerHour?.toFixed(2) || '0.00'}
```

</td>

```
<td style={{ padding: '16px' }}>
```

{renderStatusBadge(s.status)}

```
</td>
```

```
<td style={{
```

padding: '16px',

textAlign: 'center'

```
}}>
```

```
<div style={{
```

display: 'flex',

gap: '8px',

justifyContent: 'center',

flexWrap: 'wrap'

```
}}>
```

```
{s.status === 0 && (
```

```
<Button
```

variant="outline-secondary"

size="sm"

onClick={() => handleChangeStatus(s, 1)}

```
style={{
```

border: '2px solid #6C757D',

color: '#6C757D',

borderRadius: '8px',

fontWeight: '500',

padding: '6px 12px'

```
}}
```

```
>

     Deactivate

</Button>

)}

{ (s.status === 1 || s.status === 2) && (

<Button

    variant="outline-success"

    size="sm"

    onClick={() => handleChangeStatus(s, 0)}

    style={{

        border: '2px solid #00C853',

        color: '#00C853',

        borderRadius: '8px',

        fontWeight: '500',

        padding: '6px 12px'

    }}

    >

     Activate

</Button>

)}

<Button

    variant="outline-warning"

    size="sm"

    onClick={() => handleOpenModal(s)}

    style={{

        border: '2px solid #FFB703',

        color: '#FFB703',

        borderRadius: '8px',

        fontWeight: '500'

    }}

    >
```

```
padding: '6px 12px'

})

>

Edit

</Button>

<Button

variant="outline-danger"

size="sm"

onClick={() => handleDelete(s.id)}

style={{

border: '2px solid #DC3545',

color: '#DC3545',

borderRadius: '8px',

fontWeight: '500',

padding: '6px 12px'

} }

>

Delete

</Button>

</div>

</td>

</tr>

))

) : (


<tr>

<td colSpan="7" style={{

padding: '40px',

textAlign: 'center',



color: '#6C757D'


```

```

    > }

    <div style={{ fontSize: '3rem', marginBottom: '16px' }}>  </div>

    <h4 style={{ color: '#121212', marginBottom: '8px' }}>No Stations Found</h4>

    <p>Get started by adding your first charging station.</p>

    <Button

        onClick={() => handleOpenModal()}

        style={{

            background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',

            border: 'none',

            borderRadius: '8px',

            fontWeight: '500',

            padding: '10px 20px',

            marginTop: '16px'

        }}

    >

         Add First Station

    </Button>

    </td>

</tr>

)}

</tbody>

</Table>

)}

</div>

/* Add/Edit Modal */

<Modal show={showModal} onHide={() => setShowModal(false)} centered>

<Modal.Header style={{

    background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',
```

```

        color: '#F9FAFB',
        border: 'none'
    }}>

<Modal.Title style={{ fontWeight: '600' }}>
    {editMode ? "📝 Edit Station" : "➕ Add New Station"}
</Modal.Title>

<button
    onClick={() => setShowModal(false)}
    style={{
        background: 'none',
        border: 'none',
        color: '#F9FAFB',
        fontSize: '1.5rem',
        cursor: 'pointer'
    }}
    >
    ×
</button>
</Modal.Header>

<Modal.Body style={{ padding: '24px', background: '#F9FAFB' }}>
    <Form>
        <Form.Group className="mb-3">
            <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Station Name</Form.Label>
            <Form.Control
                value={name}
                onChange={(e) => setName(e.target.value)}
                style={{
                    padding: '12px',
                    borderRadius: '8px',

```

```

        border: '2px solid #E5E7EB',
        background: '#FFFFFF'
    }
    placeholder="Enter station name"
/>
</Form.Group>

<Form.Group className="mb-3">
<Form.Label style={{ fontWeight: '600', color: '#121212' }}>Location</Form.Label>
<Form.Control
    value={location}
    onChange={(e) => setLocation(e.target.value)}
    style={{
        padding: '12px',
        borderRadius: '8px',
        border: '2px solid #E5E7EB',
        background: '#FFFFFF'
    }}
    placeholder="Enter station location"
/>
</Form.Group>

<div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '16px' }}>
<Form.Group className="mb-3">
<Form.Label style={{ fontWeight: '600', color: '#121212' }}>Charger Type</Form.Label>
<Form.Select
    value={type}
    onChange={(e) => setType(e.target.value)}
    style={{

```

```

padding: '12px',
borderRadius: '8px',
border: '2px solid #E5E7EB',
background: '#FFFFFF'

)}

>

<option value="AC">  AC Charger</option>
<option value="DC">  DC Fast Charger</option>
</Form.Select>
</Form.Group>

<Form.Group className="mb-3">
<Form.Label style={{ fontWeight: '600', color: '#121212' }}>Total Slots</Form.Label>
<Form.Control
  type="number"
  min={1}
  max={50}
  value={totalSlots}
  onChange={(e) => setSlots(Number(e.target.value))}

  style={{
    padding: '12px',
    borderRadius: '8px',
    border: '2px solid #E5E7EB',
    background: '#FFFFFF'

  }}
/>
</Form.Group>
</div>

```

```
<div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '16px' }}>

  <Form.Group className="mb-3">

    <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Price Per Hour ($)</Form.Label>

    <Form.Control

      type="number"

      step={0.01}

      min={0.01}

      max={1000}

      value={pricePerHour}

      onChange={(e) => setPricePerHour(Number(e.target.value))}

      style={{

        padding: '12px',

        borderRadius: '8px',

        border: '2px solid #E5E7EB',

        background: '#FFFFFF'

      }}

    />

  </Form.Group>

  <Form.Group className="mb-3">

    <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Status</Form.Label>

    <Form.Select

      value={status}

      onChange={(e) => setStatus(Number(e.target.value))}

      disabled={editMode}

      style={{

        padding: '12px',

        borderRadius: '8px',

        border: '2px solid #E5E7EB',
```

```
        background: '#FFFFFF'

    }}

>

<option value={0}> ⚡ Active</option>

<option value={1}> 🛑 Inactive</option>

<option value={2}> 🛠 Maintenance</option>

</Form.Select>

</Form.Group>

</div>

</Form>

</Modal.Body>

<Modal.Footer style={{

background: '#F9FAFB',
border: 'none',
padding: '16px 24px 24px'

}}>

<Button

variant="outline-secondary"
onClick={() => setShowModal(false)}
style={{

border: '2px solid #6C757D',
color: '#6C757D',
borderRadius: '8px',
fontWeight: '500',
padding: '8px 20px'

}}


>

Cancel

</Button>
```

```

<Button
  variant="primary"
  onClick={handleSave}
  style={{
    background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',
    border: 'none',
    borderRadius: '8px',
    fontWeight: '600',
    padding: '8px 24px'
  }}
>
  {editMode ? "💾 Save Changes" : "➕ Add Station"}
</Button>
</Modal.Footer>
</Modal>

```

```

<style jsx>{
  @keyframes spin {
    0% { transform: rotate(0deg); }
    100% { transform: rotate(360deg); }
  }

  tr:hover {
    background: rgba(0, 200, 83, 0.02) !important;
    transform: translateY(-1px);
  }

  .btn:hover {
    transform: translateY(-2px);
  }
}

```

```

        box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
    }
`}</style>
</div>
);
}

//Dashboard.jsx

import { Container, Row, Col } from "react-bootstrap";
import Sidebar from "../components/Sidebar";
import Header from "../components/Header";
import Footer from "../components/Footer";
import { Outlet } from "react-router-dom";

export default function Dashboard() {
    // --- Main Dashboard Layout ---
    // Renders Header, Sidebar, Footer, and a content area for nested routes (Outlet)
    return (
        <div className="d-flex flex-column vh-100">
            <Header />
            <Container fluid className="flex-grow-1">
                <Row className="h-100">
                    <Col xs={2} className="p-0">
                        <Sidebar />
                    </Col>
                    <Col xs={10} className="p-4 overflow-auto">
                        <Outlet />
                    </Col>
                </Row>
            </Container>
        </div>
    );
}

```

```

        </Container>
        <Footer />
    </div>
);

}

//EVOwnerManagement.jsx

import { useEffect, useState } from "react";
import { Table, Button, Modal, Form, Alert } from "react-bootstrap";

export default function EVOwnerManagement() {

    // --- EV Owners management ---

    const [users, setUsers] = useState([]);
    const [showModal, setShowModal] = useState(false);
    const [editMode, setEditMode] = useState(false);
    const [selectedUser, setSelectedUser] = useState(null);
    const [error, setError] = useState("");
    const [success, setSuccess] = useState("");
    const [loading, setLoading] = useState(false);

    // User fields

    const [nic, setNic] = useState("");
    const [firstName, setFirstName] = useState("");
    const [lastName, setLastName] = useState("");
    const [email, setEmail] = useState("");
    const [phoneNumber, setPhoneNumber] = useState("");
    const [isActive, setIsActive] = useState(false);

    const token = localStorage.getItem("token");
}

```

```

// ----- Fetch EV Owners -----

const fetchEVOwners = async () => {
    setLoading(true);
    try {
        const res = await fetch("http://localhost:5082/api/users", {
            headers: {
                "Content-Type": "application/json",
                Authorization: `Bearer ${token}`,
            },
        });
        if (!res.ok) throw new Error("Failed to fetch users");
        const data = await res.json();
        setUsers(data.filter(u => u.role === 2)); // Only EV Owners
    } catch (err) {
        setError(err.message);
    } finally {
        setLoading(false);
    }
};

useEffect(() => {
    if (!token) {
        setError("Not authorized. Please login.");
        return;
    }
    fetchEVOwners();
}, [token]);

// ----- Delete EV Owner -----

```

```

const handleDelete = async (id) => {
  if (!window.confirm("Are you sure you want to delete this EV Owner?")) return;

  try {
    const res = await fetch(`http://localhost:5082/api/users/${id}`, {
      method: "DELETE",
      headers: { Authorization: `Bearer ${token}` },
    });
    if (!res.ok) throw new Error("Failed to delete EV Owner");
    setSuccess("EV Owner deleted successfully");
    fetchEVOwners();
  } catch (err) {
    setError(err.message);
  }
};

// ----- View EV Owner -----
const handleView = (user) => {
  setSelectedUser(user);
  setEditMode(false);
  fillUserFields(user);
  setShowModal(true);
};

// ----- Edit EV Owner -----
const handleEdit = (user) => {
  setSelectedUser(user);
  setEditMode(true);
  fillUserFields(user);
};

```

```
        setShowModal(true);

    };

// ----- Fill Form Fields -----

const fillUserFields = (user) => {
    setNic(user.nic);
    setFirstName(user.firstName);
    setLastName(user.lastName);
    setEmail(user.email);
    setPhoneNumber(user.phoneNumber || "");
    setIsActive(user.isActive);
};

// ----- Save Updated EV Owner -----

const handleSave = async () => {
    if (!selectedUser) return;

    const payload = {
        nic,
        firstName,
        lastName,
        email,
        phoneNumber,
        role: 2, // Always EV Owner
        isActive,
    };

    try {
        const res = await fetch(`http://localhost:5082/api/users/${selectedUser.id}`, {
```

```

    method: "PUT",
    headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify(payload),
});

if (!res.ok) throw new Error("Failed to update EV Owner");
setSuccess("EV Owner updated successfully");
setShowModal(false);
fetchEVOwners();
} catch (err) {
    setError(err.message);
}
};

// ----- Toggle Status -----
const handleToggleStatus = async (id, currentStatus) => {
    const confirmMsg = currentStatus
        ? "Deactivate this EV Owner?"
        : "Activate this EV Owner?";
    if (!window.confirm(confirmMsg)) return;

    try {
        const res = await fetch(`http://localhost:5082/api/users/${id}/status`, {
            method: "PATCH",
            headers: {
                "Content-Type": "application/json",
                Authorization: `Bearer ${token}`,
            },
        });
        if (!res.ok) throw new Error("Failed to toggle EV Owner status");
        setSuccess(`EV Owner status toggled successfully`);
        setShowModal(false);
    } catch (err) {
        setError(err.message);
    }
};

```

```

    },
    body: JSON.stringify(!currentStatus),
  });
  if (!res.ok) throw new Error("Failed to update status");
  setSuccess(`EV Owner ${currentStatus ? "deactivated" : "activated"} successfully`);
  fetchEVOwners();
} catch (err) {
  setError(err.message);
}
};

// Clear messages after 5 seconds
useEffect(() => {
  if (error || success) {
    const timer = setTimeout(() => {
      setError("");
      setSuccess("");
    }, 5000);
    return () => clearTimeout(timer);
  }
}, [error, success]);

return (
  <div style={{ padding: '24px' }}>
    {/* Header Section */}
    <div style={{ display: 'flex', alignItems: 'center', gap: '16px' }}>

```

```
marginBottom: '32px'

}}>

<div style={{

  fontSize: '2.5rem',

  background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',

  WebkitBackgroundClip: 'text',

  WebkitTextFillColor: 'transparent',

  backgroundClip: 'text'

}}>


```

```
/* Alerts */

{error && (
  <div style={{
    background: 'rgba(255, 183, 3, 0.1)',
    color: '#FFB703',
    padding: '16px',
    borderRadius: '12px',
    border: '1px solid rgba(255, 183, 3, 0.3)',
    marginBottom: '24px',
    fontWeight: '500'
  }}>
  {error}
</div>
)})

{success && (
  <div style={{
    background: 'rgba(0, 200, 83, 0.1)',
    color: '#00C853',
    padding: '16px',
    borderRadius: '12px',
    border: '1px solid rgba(0, 200, 83, 0.3)',
    marginBottom: '24px',
    fontWeight: '500'
  }}>
  {success}
</div>
)}
```

```

/* Stats Bar */

<div style={{

  display: 'flex',
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: '24px'

}}>

<div style={{ fontSize: '1rem', color: '#6C757D', fontWeight: '500' }}>

  Total EV Owners: <strong style={{ color: '#121212' }}>{users.length}</strong>
  <span style={{ marginLeft: '16px' }}>

    Active: <strong style={{ color: '#00C853' }}>
    {users.filter(u => u.isActive).length}
    </strong>
  </span>
</div>
</div>

```

```

/* EV Owners Table */

<div style={{

  background: '#F9FAFB',
  borderRadius: '20px',
  padding: '32px',
  boxShadow: '0 10px 40px rgba(0, 0, 0, 0.1)',
  border: '1px solid rgba(255, 255, 255, 0.1)'

}}>

  {loading ? (
    <div style={{

      textAlign: 'center',

```

```

padding: '40px',
color: '#6C757D'

}}>

<div className="spinner" style={{

width: '40px',
height: '40px',
border: '3px solid transparent',
borderTop: '3px solid #00C853',
borderRadius: '50%',

animation: 'spin 1s linear infinite',
margin: '0 auto 16px'

}}></div>

Loading EV owners...

</div>

): (
<Table hover responsive style={{

margin: 0,
border: 'none'

}}>

<thead>

<tr style={{

background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',
color: '#F9FAFB'

}}>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

```

```
}}>NIC</th>

<th style={{  
    padding: '16px',  
    border: 'none',  
    fontWeight: '600',  
    fontSize: '0.95rem'  
}}>Name</th>

<th style={{  
    padding: '16px',  
    border: 'none',  
    fontWeight: '600',  
    fontSize: '0.95rem'  
}}>Email</th>

<th style={{  
    padding: '16px',  
    border: 'none',  
    fontWeight: '600',  
    fontSize: '0.95rem'  
}}>Phone</th>

<th style={{  
    padding: '16px',  
    border: 'none',  
    fontWeight: '600',  
    fontSize: '0.95rem'  
}}>Status</th>

<th style={{  
    padding: '16px',  
    border: 'none',  
    fontWeight: '600',  
}}
```

```
    fontSize: '0.95rem',
    textAlign: 'center'
  }}>Actions</th>
</tr>
</thead>
<tbody>
{users.length > 0 ? (
  users.map(u => (
    <tr key={u.id} style={{
      borderBottom: '1px solid #E5E7EB',
      transition: 'all 0.3s ease'
    }}>
    <td style={{
      padding: '16px',
      fontWeight: '500',
      color: '#121212'
    }}>{u.nic}</td>
    <td style={{
      padding: '16px',
      color: '#121212',
      fontWeight: '500'
    }}>
      {u.firstName} {u.lastName}
    </td>
    <td style={{
      padding: '16px',
      color: '#121212'
    }}>{u.email}</td>
    <td style={{

```

```
padding: '16px',
color: '#121212'

}}>{u.phoneNumber || "N/A"}</td>

<td style={{ padding: '16px' }}>
<span style={{

padding: '6px 12px',
borderRadius: '20px',
fontSize: '0.85rem',
fontWeight: '500',
background: u.isActive
? 'rgba(0, 200, 83, 0.1)'
: 'rgba(108, 117, 125, 0.1)',

color: u.isActive ? '#00C853' : '#6C757D'

}}>

{u.isActive ? "☑ Active" : "☒ Inactive"}
</span>
</td>

<td style={{

padding: '16px',
textAlign: 'center'

}}>

<div style={{

display: 'flex',
gap: '8px',
justifyContent: 'center',
flexWrap: 'wrap'

}}>

<Button
variant="outline-info"
```

```
size="sm"

onClick={() => handleView(u)}

style={{

border: '2px solid #00B4D8',
color: '#00B4D8',
borderRadius: '8px',
fontWeight: '500',
padding: '6px 12px'

}}


>

👁 View

</Button>

<Button

variant="outline-warning"

size="sm"

onClick={() => handleEdit(u)}

style={{

border: '2px solid #FFB703',
color: '#FFB703',
borderRadius: '8px',
fontWeight: '500',
padding: '6px 12px'

}}


>

✎ Edit

</Button>

<Button

variant={u.isActive ? "outline-secondary" : "outline-success"}

size="sm"
```

```

        onClick={() => handleToggleStatus(u.id, u.isActive)}
        style={{
          border: u.isActive ? '2px solid #6C757D' : '2px solid #00C853',
          color: u.isActive ? '#6C757D' : '#00C853',
          borderRadius: '8px',
          fontWeight: '500',
          padding: '6px 12px'
        }}
      >
  {u.isActive ? "  Deactivate" : "  Activate"}

```

</Button>

<Button

variant="outline-danger"

size="sm"

onClick={() => handleDelete(u.id)}

style={{
 border: '2px solid #DC3545',
 color: '#DC3545',
 borderRadius: '8px',
 fontWeight: '500',
 padding: '6px 12px'
 }}

>

Delete

</Button>

</div>

</td>

</tr>

)

```

) : (
  <tr>
    <td colSpan="6" style={{

      padding: '40px',
      textAlign: 'center',
      color: '#6C757D'

    }}>

      <div style={{ fontSize: '3rem', marginBottom: '16px' }}> 🚗 </div>
      <h4 style={{ color: '#121212', marginBottom: '8px' }}>No EV Owners Found</h4>
      <p>No electric vehicle owners are registered in the system yet.</p>
    </td>
  </tr>
)
</tbody>
</Table>
)
</div>

```

```

/* View/Edit Modal */

<Modal show={showModal} onHide={() => setShowModal(false)} centered>
  <Modal.Header style={{

    background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',
    color: '#F9FAFB',
    border: 'none'

  }}>
    <Modal.Title style={{ fontWeight: '600' }}>
      {editMode ? " 🖍 Edit EV Owner" : " 🕳 EV Owner Details"}
    </Modal.Title>
    <button

```

```

    onClick={() => setShowModal(false)}

    style={{
        background: 'none',
        border: 'none',
        color: '#F9FAFB',
        fontSize: '1.5rem',
        cursor: 'pointer'
    }}
>
    x
</button>
</Modal.Header>

<Modal.Body style={{ padding: '24px', background: '#F9FAFB' }}>
    {selectedUser && (
        <Form>
            <Form.Group className="mb-3">
                <Form.Label style={{ fontWeight: '600', color: '#121212' }}>NIC</Form.Label>
                <Form.Control
                    value={nic}
                    readOnly
                    style={{
                        padding: '12px',
                        borderRadius: '8px',
                        border: '2px solid #E5E7EB',
                        background: '#FFFFFF'
                    }}
                />
            </Form.Group>
    )}

```

```
<div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '16px' }}>

  <Form.Group className="mb-3">

    <Form.Label style={{ fontWeight: '600', color: '#121212' }}>First Name</Form.Label>

    <Form.Control

      value={firstName}

      onChange={e => setFirstName(e.target.value)}

      readOnly={!editMode}

      style={{

        padding: '12px',

        borderRadius: '8px',

        border: '2px solid #E5E7EB',

        background: '#FFFFFF'

      }}

    />

  </Form.Group>

  <Form.Group className="mb-3">

    <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Last Name</Form.Label>

    <Form.Control

      value={lastName}

      onChange={e => setLastName(e.target.value)}

      readOnly={!editMode}

      style={{

        padding: '12px',

        borderRadius: '8px',

        border: '2px solid #E5E7EB',

        background: '#FFFFFF'

      }}

    />

  </Form.Group>


```

```
</Form.Group>

</div>

<Form.Group className="mb-3">

  <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Email</Form.Label>

  <Form.Control

    value={email}

    onChange={e => setEmail(e.target.value)}

    readOnly={!EditMode}

    style={{

      padding: '12px',

      borderRadius: '8px',

      border: '2px solid #E5E7EB',

      background: '#FFFFFF'

    }}

  />

</Form.Group>

<Form.Group className="mb-3">

  <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Phone Number</Form.Label>

  <Form.Control

    value={phoneNumber}

    onChange={e => setPhoneNumber(e.target.value)}

    readOnly={!EditMode}

    style={{

      padding: '12px',

      borderRadius: '8px',

      border: '2px solid #E5E7EB',

      background: '#FFFFFF'

    }}

  />

</Form.Group>
```

```

        }}

    />

</Form.Group>

<Form.Group className="mb-3">

<Form.Label style={{ fontWeight: '600', color: '#121212' }}>Status</Form.Label>

<Form.Control

    value={isActive ?  Active" :  Inactive"}

    readOnly

    style={{

        padding: '12px',

        borderRadius: '8px',

        border: '2px solid #E5E7EB',

        background: '#FFFFFF'

    }}

    />

</Form.Group>

<Form.Group className="mb-3">

<Form.Label style={{ fontWeight: '600', color: '#121212' }}>Role</Form.Label>

<Form.Control

    value="EV Owner"

    readOnly

    style={{

        padding: '12px',

        borderRadius: '8px',

        border: '2px solid #E5E7EB',

        background: '#FFFFFF'

    }}

    />

```

```
        />

      </Form.Group>

    </Form>

  )}

</Modal.Body>

<Modal.Footer style={{

  background: '#F9FAFB',

  border: 'none',

  padding: '16px 24px 24px'

}}>

<Button

  variant="outline-secondary"

  onClick={() => setShowModal(false)}

  style={{

    border: '2px solid #6C757D',

    color: '#6C757D',

    borderRadius: '8px',

    fontWeight: '500',

    padding: '8px 20px'

  }}

>

  Close

</Button>

{editMode && (

  <Button

    variant="primary"

    onClick={handleSave}

    style={{

      background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',
```

```
        border: 'none',
        borderRadius: '8px',
        fontWeight: '600',
        padding: '8px 24px'

    )}

>

     Save Changes

</Button>

)}

</Modal.Footer>

</Modal>

<style jsx>`  

@keyframes spin {  

    0% { transform: rotate(0deg); }  

    100% { transform: rotate(360deg); }  

}  

tr:hover {  

    background: rgba(0, 200, 83, 0.02) !important;  

    transform: translateY(-1px);  

}  

.btn:hover {  

    transform: translateY(-2px);  

    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);  

}  

`</style>  

</div>
```

```
};

}

//Login.jsx

import { useState } from "react";

import { useNavigate } from "react-router-dom";

import axios from "axios";

import "../styles/Login.css";


export default function Login() {

    // --- State Hooks ---

    const [email, setEmail] = useState("");

    const [password, setPassword] = useState("");

    const [error, setError] = useState("");

    const [isLoading, setIsLoading] = useState(false);

    const navigate = useNavigate();


    // --- Handle form submission for login ---

    const handleLogin = async (e) => {

        e.preventDefault(); // Prevent default form submission

        setError(""); // Clear previous errors

        setIsLoading(true); // Set loading state

        try {

            // --- Send login request to API ---

            const response = await axios.post("http://localhost:5082/api/auth/login", {

                email,

                password,

            });

        
```

```

const { token, user } = response.data;

// --- Store JWT and user info in localStorage ---
localStorage.setItem("token", token);
localStorage.setItem("role", user.role);
if (user.stationId) {
  localStorage.setItem("stationId", user.stationId);
}

// --- Redirect user based on role ---
if (user.role === 0) {
  navigate("/dashboard/users"); // Backoffice starts at Users page
} else if (user.role === 1) {
  navigate("/station-dashboard/bookings"); // Operator dashboard
} else {
  navigate("/dashboard/evowner"); // EV Owner dashboard
}
} catch (err) {
// --- Handle API error response ---
setError(err.response?.data?.message || "Login failed");
} finally {
  setIsLoading(false); // Reset loading state
}
};

return (
<div className="login-container">
<div className="login-card">

```

```

/* Header Section */

<div className="login-header">

  <div className="logo">

    <div className="logo-icon"> ⚡ </div>

    <h1>EV Charge</h1>

  </div>

  <p className="tagline">Power Your Journey</p>

</div>

/* Form Section */

<form onSubmit={handleLogin} className="login-form">

  {error && <div className="alert alert-error">{error}</div>}

  <div className="form-group">

    <label htmlFor="email" className="form-label">Email Address</label>

    <input

      id="email"

      type="email"

      className="form-input"

      value={email}

      onChange={(e) => setEmail(e.target.value)}

      placeholder="Enter your email"

      required

    />

  </div>

  <div className="form-group">

    <label htmlFor="password" className="form-label">Password</label>

    <input

```

```
        id="password"
        type="password"
        className="form-input"
        value={password}
        onChange={(e) => setPassword(e.target.value)}
        placeholder="Enter your password"
        required
      />
    </div>

<button
  className={`login-button ${isLoading ? 'loading' : ''}`}
  type="submit"
  disabled={isLoading}
>
  {isLoading ? (
    <>
    <div className="spinner"></div>
    Signing In...
  </>
  ) : (
    'Log In to Your Account'
  )}
</button>
</form>

{/* Footer Links */}

<div className="login-footer">
  <p className="signup-text">
```

```

        New to EV Charge? <a href="/register" className="signup-link">Create Account</a>
    </p>
    </div>
    </div>
    </div>
);

}

//OperatorBookings.jsx

import { useEffect, useState } from "react";

import { Table, Button, Form, Row, Col, Alert, Modal } from "react-bootstrap";


export default function OperatorBookings() {

    // --- Operator booking management ---

    const [bookings, setBookings] = useState([]);

    const [filteredBookings, setFilteredBookings] = useState([]);

    const [statusFilter, setStatusFilter] = useState("");

    const [dateFrom, setDateFrom] = useState("");

    const [dateTo, setDateTo] = useState("");

    const [error, setError] = useState("");

    const [success, setSuccess] = useState("");

    const [loading, setLoading] = useState(false);

    const [showCancelModal, setShowCancelModal] = useState(false);

    const [cancelReason, setCancelReason] = useState("");

    const [selectedBookingId, setSelectedBookingId] = useState(null);

    const token = localStorage.getItem("token");

    const stationId = localStorage.getItem("stationId");

```

```
const statusMap = {
  0: "Active",
  1: "Confirmed",
  2: "Completed",
  3: "Cancelled",
  4: "NoShow",
};

const fetchBookings = async () => {
  if (!stationId) {
    setError("Station ID is missing");
    return;
  }

  setLoading(true);
  try {
    const res = await fetch(
      `http://localhost:5082/api/bookings/station/${stationId}`,
      {
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${token}`,
        },
      },
    );
    if (!res.ok) throw new Error("Failed to fetch bookings");
    const data = await res.json();
    setBookings(data);
    setFilteredBookings(data);
  }
}
```

```
        } catch (err) {
            setError(err.message);
        } finally {
            setLoading(false);
        }
    };

useEffect(() => {
    fetchBookings();
}, [stationId]);

// Filter bookings
useEffect(() => {
    let filtered = [...bookings];

    if (statusFilter) {
        filtered = filtered.filter(
            (b) =>
                statusMap[b.status]?.toLowerCase() === statusFilter.toLowerCase()
        );
    }

    if (dateFrom) {
        filtered = filtered.filter(
            (b) => new Date(b.startTime) >= new Date(dateFrom)
        );
    }

    if (dateTo) {
```

```

filtered = filtered.filter(
  (b) => new Date(b.startTime) <= new Date(dateTo)
);

}

setFilteredBookings(filtered);
}, [statusFilter, dateFrom, dateTo, bookings]);

// Confirm booking
const handleConfirm = async (booking) => {
  const bookingId = booking.id;

  if (!window.confirm("Confirm this booking?")) return;

  try {
    const res = await fetch(
      `http://localhost:5082/api/bookings/${bookingId}/confirm`,
      {
        method: "POST",
        headers: {
          Authorization: `Bearer ${token}`,
        },
      }
    );

    if (!res.ok) throw new Error("Failed to confirm booking");
  }

  // Send notification
  const notifRes = await fetch(`http://localhost:5082/api/notifications`, {

```

```

    method: "POST",
    headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify({
        title: "Booking Confirmed",
        message: "Your booking has been confirmed by the operator",
        recipientNIC: booking.ownerNIC,
        type: 0,
        relatedEntityId: booking.id,
        relatedEntityType: "Booking"
    })
});

if (!notifRes.ok) {
    const errData = await notifRes.json();
    throw new Error(errData?.title || "Failed to send notification");
}

setSuccess("Booking confirmed and notification sent successfully");
setError("");
fetchBookings();

} catch (err) {
    setError(err.message);
}
};

// Cancel booking (open modal)

```

```

const handleCancelClick = (id) => {
  setSelectedBookingId(id);
  setCancelReason("");
  setShowCancelModal(true);
};

// Confirm cancel booking

const handleCancelBooking = async () => {
  if (!cancelReason.trim()) {
    setError("Please enter a reason for cancellation");
    return;
  }

  try {
    const res = await fetch(
      `http://localhost:5082/api/bookings/${selectedBookingId}/cancel-by-operator`,
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({ reason: cancelReason }),
      }
    );
    if (!res.ok) throw new Error("Failed to cancel booking");
  }

  const booking = bookings.find((b) => b.id === selectedBookingId);

```

```

if (!booking) throw new Error("Booking not found for notification");

// Send cancellation notification

const notifRes = await fetch(`http://localhost:5082/api/notifications`, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    Authorization: `Bearer ${token}`,
  },
  body: JSON.stringify({
    title: "Booking Cancelled",
    message: `Your booking at ${booking.stationName} has been cancelled. Reason: ${cancelReason}`,
    recipientNIC: booking.ownerNIC,
    type: 1,
    relatedEntityId: booking.id,
    relatedEntityType: "Booking",
  }),
});

if (!notifRes.ok) {
  const errData = await notifRes.json();
  throw new Error(errData?.title || "Failed to send notification");
}

setSuccess("Booking cancelled and notification sent successfully");
setError("");
setShowCancelModal(false);
fetchBookings();

} catch (err) {

```

```
setError(err.message);

}

};

// Render booking status badge

const renderStatusBadge = (status) => {

  const statusStr = statusMap[status] || "Unknown";

  switch (statusStr.toLowerCase()) {

    case "active":

      return (
        <span style={{
          padding: '6px 12px',
          borderRadius: '20px',
          fontSize: '0.85rem',
          fontWeight: '500',
          background: 'rgba(0, 200, 83, 0.1)',
          color: '#00C853'
        }}>
           Pending
        </span>
      );
    case "confirmed":
      return (
        <span style={{
          padding: '6px 12px',
          borderRadius: '20px',
          fontSize: '0.85rem',
          fontWeight: '500',
          background: 'rgba(0, 123, 255, 0.1)'
        }}>
      );
  }
}
```

```
        color: '#007BFF'  
    }>  
     Confirmed  
    </span>  
);  
  
case "completed":  
    return (  
        <span style={{  
            padding: '6px 12px',  
            borderRadius: '20px',  
            fontSize: '0.85rem',  
            fontWeight: '500',  
            background: 'rgba(40, 167, 69, 0.1)',  
            color: '#28A745'  
        }}>  
         Completed  
        </span>  
);  
  
case "cancelled":  
    return (  
        <span style={{  
            padding: '6px 12px',  
            borderRadius: '20px',  
            fontSize: '0.85rem',  
            fontWeight: '500',  
            background: 'rgba(108, 117, 125, 0.1)',  
            color: '#6C757D'  
        }}>  
         Cancelled  
    );
```

```
</span>
);

case "noshow":
    return (
        <span style={{{
            padding: '6px 12px',
            borderRadius: '20px',
            fontSize: '0.85rem',
            fontWeight: '500',
            background: 'rgba(220, 53, 69, 0.1)',
            color: '#DC3545'
        }}}>

            🚫 No Show
        </span>
    );
}

default:
    return (
        <span style={{{
            padding: '6px 12px',
            borderRadius: '20px',
            fontSize: '0.85rem',
            fontWeight: '500',
            background: 'rgba(255, 193, 7, 0.1)',
            color: '#FFC107'
        }}}>

            ❓ Unknown
        </span>
    );
}
```

```
};

// Clear messages after 5 seconds

useEffect(() => {
  if (error || success) {
    const timer = setTimeout(() => {
      setError("");
      setSuccess("");
    }, 5000);
    return () => clearTimeout(timer);
  }
}, [error, success]);

return (
  <div style={{ padding: '24px' }}>
    {/* Header Section */}
    <div style={{
      display: 'flex',
      alignItems: 'center',
      gap: '16px',
      marginBottom: '32px'
    }}>
      <div style={{
        fontSize: '2.5rem',
        background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',
        WebkitBackgroundClip: 'text',
        WebkitTextFillColor: 'transparent',
        backgroundClip: 'text'
      }}>
```



```
</div>

<div>

  <h1 style={{

    color: '#000000ff',

    margin: 0,

    fontWeight: '700',

    fontSize: '2rem'

  }}>

    Station Bookings

  </h1>

  <p style={{

    color: '#9CA3AF',

    margin: 0,

    fontSize: '1.1rem'

  }}>

    Manage and monitor charging station bookings

  </p>

</div>

</div>

/* Alerts */

{error && (

  <div style={{

    background: 'rgba(248, 247, 245, 0.1)',

    color: '#FFB703',

    padding: '16px',

    borderRadius: '12px',

    border: '1px solid rgba(255, 183, 3, 0.3)',

    width: '100%', height: '100%'

  }}>
```

```
        marginBottom: '24px',
        fontWeight: '500'
    }}>
    {error}
</div>
)}
```

```
{success && (
<div style={{
    background: 'rgba(0, 200, 83, 0.1)',
    color: '#00C853',
    padding: '16px',
    borderRadius: '12px',
    border: '1px solid rgba(0, 200, 83, 0.3)',
    marginBottom: '24px',
    fontWeight: '500'
}}>
    {success}
</div>
)}
```

```
/* Filters Card */
<div style={{
    background: '#F9FAFB',
    borderRadius: '20px',
    padding: '24px',
    boxShadow: '0 10px 40px rgba(0, 0, 0, 0.1)',
    border: '1px solid rgba(255, 255, 255, 0.1)',
    marginBottom: '24px'
```

```

  }>

  <Form>

    <Row className="align-items-end">
      <Col md={3}>
        <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Status</Form.Label>
        <Form.Select
          value={statusFilter}
          onChange={(e) => setStatusFilter(e.target.value)}
          style={{
            padding: '12px',
            borderRadius: '8px',
            border: '2px solid #E5E7EB',
            background: '#FFFFFF'
          }}
        >
          <option value="">All Status</option>
          <option value="Active"> Pending</option>
          <option value="Confirmed"> Confirmed</option>
          <option value="Completed"> Completed</option>
          <option value="Cancelled"> Cancelled</option>
          <option value="NoShow"> No Show</option>
        </Form.Select>
      </Col>

      <Col md={3}>
        <Form.Label style={{ fontWeight: '600', color: '#121212' }}>From Date</Form.Label>
        <Form.Control
          type="date"
          value={dateFrom}
        >
      </Col>
    </Row>
  </Form>

```

```

        onChange={(e) => setDateFrom(e.target.value)}

        style={{
          padding: '12px',
          borderRadius: '8px',
          border: '2px solid #E5E7EB',
          background: '#FFFFFF'
        }}

      />

    </Col>

    <Col md={3}>
      <Form.Label style={{ fontWeight: '600', color: '#121212' }}>To Date</Form.Label>
      <Form.Control
        type="date"
        value={dateTo}
        onChange={(e) => setDateTo(e.target.value)}
        style={{
          padding: '12px',
          borderRadius: '8px',
          border: '2px solid #E5E7EB',
          background: '#FFFFFF'
        }}
      />
    </Col>

    <Col md={3}>
      <Button
        className="w-100"
        onClick={fetchBookings}
      >

```

```
disabled={loading}

style={{
  background: loading
    ? 'linear-gradient(135deg, #00B4D8 0%, #0096C7 100%)'
    : 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',
  border: 'none',
  borderRadius: '12px',
  fontWeight: '600',
  padding: '12px',
  fontSize: '1rem'
}>

>

{loading ? (
  <>
<div className="spinner" style={{
  width: '16px',
  height: '16px',
  border: '2px solid transparent',
  borderTop: '2px solid white',
  borderRadius: '50%',
  animation: 'spin 1s linear infinite',
  display: 'inline-block',
  marginRight: '8px'
}}></div>
  Refreshing...
</>
) : (
  '↻ Refresh'
)}
```

```
</Button>

</Col>

</Row>

</Form>

</div>

{/* Bookings Table Card */}

<div style={{

background: '#F9FAFB',  
borderRadius: '20px',  
padding: '32px',  
boxShadow: '0 10px 40px rgba(0, 0, 0, 0.1)',  
border: '1px solid rgba(255, 255, 255, 0.1)'

}}>

{loading && filteredBookings.length === 0 ? (

<div style={{

textAlign: 'center',  
padding: '40px',  
color: '#6C757D'

}}>

<div className="spinner" style={{

width: '40px',  
height: '40px',  
border: '3px solid transparent',  
borderTop: '3px solid #00C853',  
borderRadius: '50%',  
animation: 'spin 1s linear infinite',  
margin: '0 auto 16px'

}}></div>
```

```

        Loading bookings...

    </div>

) : (

<Table hover responsive style={{

margin: 0,
border: 'none'

}}>

<thead>

<tr style={{

background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',
color: '#F9FAFB'

}}>

<th style={{ padding: '16px', border: 'none', fontWeight: '600' }}>ID</th>
<th style={{ padding: '16px', border: 'none', fontWeight: '600' }}>User NIC</th>
<th style={{ padding: '16px', border: 'none', fontWeight: '600' }}>Start Time</th>
<th style={{ padding: '16px', border: 'none', fontWeight: '600' }}>End Time</th>
<th style={{ padding: '16px', border: 'none', fontWeight: '600' }}>Total Amount</th>
<th style={{ padding: '16px', border: 'none', fontWeight: '600' }}>QR Code</th>
<th style={{ padding: '16px', border: 'none', fontWeight: '600' }}>Status</th>
<th style={{ padding: '16px', border: 'none', fontWeight: '600', textAlign: 'center' }}>Actions</th>

</tr>
</thead>
<tbody>

{filteredBookings.length > 0 ? (

filteredBookings.map((b) => (

<tr key={b._id} style={{

borderBottom: '1px solid #E5E7EB',
transition: 'all 0.3s ease'

}}>

```

```

<td style={{ padding: '16px', fontWeight: '500', color: '#121212' }}>
  {b._id?.substring(0, 8)}...
</td>

<td style={{ padding: '16px', color: '#121212' }}>{b.ownerNIC}</td>

<td style={{ padding: '16px', color: '#121212' }}>
  {new Date(b.startTime).toLocaleString("en-GB", {
    dateStyle: "medium",
    timeStyle: "short",
  })}
</td>

<td style={{ padding: '16px', color: '#121212' }}>
  {new Date(b.endTime).toLocaleString("en-GB", {
    dateStyle: "medium",
    timeStyle: "short",
  })}
</td>

<td style={{ padding: '16px', color: '#121212', fontWeight: '600' }}>
  ${b.totalAmount?.toFixed(2) || '0.00'}
</td>

<td style={{ padding: '16px', color: '#121212' }}>{b.qrCode}</td>

<td style={{ padding: '16px' }}>
  {renderStatusBadge(b.status)}
</td>

<td style={{ padding: '16px', textAlign: 'center' }}>
  <div style={{ display: 'flex', gap: '8px', justifyContent: 'center', flexWrap: 'wrap' }}>
    {b.status === 0 && (
      <>
        <Button
          size="sm"

```

```
    onClick={() => handleConfirm(b)}
```

```
    style={{
        background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',
        border: 'none',
        borderRadius: '8px',
        fontWeight: '500',
        padding: '6px 12px'
    }}
>
```

```
     Confirm
</Button>
```

```
<Button
    size="sm"
    onClick={() => handleCancelClick(b.id)}
    style={{
        background: 'linear-gradient(135deg, #DC3545 0%, #C82333 100%)',
        border: 'none',
        borderRadius: '8px',
        fontWeight: '500',
        padding: '6px 12px'
    }}
>
```

```
     Cancel
</Button>
</>
)}
```

```
{b.status === 1 && (
    <Button
        size="sm"
```

```

        onClick={() => handleCancelClick(b.id)}
        style={{
          background: 'linear-gradient(135deg, #DC3545 0%, #C82333 100%)',
          border: 'none',
          borderRadius: '8px',
          fontWeight: '500',
          padding: '6px 12px'
        }}
      >
    X Cancel
  </Button>
)
}
</div>
</td>
</tr>
))
):(
<tr>
<td colSpan="7" style={{
  padding: '40px',
  textAlign: 'center',
  color: '#6C757D'
}}>
<div style={{ fontSize: '3rem', marginBottom: '16px' }}>  </div>
<h4 style={{ color: '#121212', marginBottom: '8px' }}>No Bookings Found</h4>
<p>No bookings match your current filters.</p>
</td>
</tr>
)
}

```

```

        </tbody>
    </Table>
)
</div>

/* Cancel Modal */

<Modal show={showCancelModal} onHide={() => setShowCancelModal(false)} centered>
<Modal.Header style={{
    background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',
    color: '#F9FAFB',
    border: 'none'
}}>
<Modal.Title style={{ fontWeight: '600' }}>
    X Cancel Booking
</Modal.Title>
<button
    onClick={() => setShowCancelModal(false)}
    style={{
        background: 'none',
        border: 'none',
        color: '#F9FAFB',
        fontSize: '1.5rem',
        cursor: 'pointer'
    }}
    >
    ×
</button>
</Modal.Header>
<Modal.Body style={{ padding: '24px', background: '#F9FAFB' }}>

```

```
<Form.Group>
  <Form.Label style={{ fontWeight: '600', color: '#121212' }}>
    Reason for Cancellation
  </Form.Label>
  <Form.Control
    as="textarea"
    rows={3}
    value={cancelReason}
    onChange={(e) => setCancelReason(e.target.value)}
    placeholder="Please provide a reason for cancellation..."
    style={{
      padding: '12px',
      borderRadius: '8px',
      border: '2px solid #E5E7EB',
      background: '#FFFFFF'
    }}
  />
</Form.Group>
</Modal.Body>
<Modal.Footer style={{
  background: '#F9FAFB',
  border: 'none',
  padding: '16px 24px 24px'
}}>
  <Button
    variant="outline-secondary"
    onClick={() => setShowCancelModal(false)}
    style={{
      border: '2px solid #6C757D',

```

```
        color: '#6C757D',
        borderRadius: '8px',
        fontWeight: '500',
        padding: '8px 20px'
    )}
>
Close
</Button>
<Button
    onClick={handleCancelBooking}
    style={{
        background: 'linear-gradient(135deg, #DC3545 0%, #C82333 100%)',
        border: 'none',
        borderRadius: '8px',
        fontWeight: '600',
        padding: '8px 24px'
    }}
>
trash Confirm Cancel
</Button>
</Modal.Footer>
</Modal>

<style jsx>{
    @keyframes spin {
        0% { transform: rotate(0deg); }
        100% { transform: rotate(360deg); }
    }
}
```

```

        tr:hover {
            background: rgba(0, 200, 83, 0.02) !important;
            transform: translateY(-1px);
        }

        .btn:hover {
            transform: translateY(-2px);
            box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
        }
    `}</style>
</div>
);

}

//Register.jsx
import { useState, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import "../styles/Login.css"; // Reusing the same CSS file

export default function Register() {
    // --- State Hooks ---
    const [firstName, setFirstName] = useState("");
    const [lastName, setLastName] = useState("");
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [role, setRole] = useState(1); // default = StationOperator
    const [error, setError] = useState("");
    const [success, setSuccess] = useState("");
    const [nic, setNic] = useState("");
}

```

```

const [stations, setStations] = useState([]);
const [stationId, setStationId] = useState("");
const [isLoading, setIsLoading] = useState(false);

const navigate = useNavigate();

// --- Handle registration form submission ---
const handleRegister = async (e) => {
  e.preventDefault(); // Prevent default form submission
  setError(""); // Clear previous error
  setSuccess(""); // Clear previous success message
  setIsLoading(true); // Set loading state

  try {
    // --- Send registration request to API ---
    const response = await fetch("http://localhost:5082/api/auth/register", {
      method: "POST",
      headers: { "Content-Type": "application/json" },
      body: JSON.stringify({ nic, firstName, lastName, email, password, role, stationId }),
    });

    const data = await response.json();

    if (!response.ok) {
      throw new Error(data.message || "Registration failed");
    }

    // --- Set success message and redirect ---
    setSuccess("✓ Registration successful! Redirecting to login...");
  } catch (error) {
    console.error(error);
  }
}

```

```

setTimeout(() => {
  navigate("/");
}, 1500);

} catch (err) {
  // --- Handle API error ---
  setError(err.message);
}

} finally {
  setIsLoading(false); // Reset loading state
}

};

// --- Fetch available charging stations on component mount ---
useEffect(() => {
  const fetchStations = async () => {
    try {
      const res = await fetch("http://localhost:5082/api/chargingstations", {
        method: "GET",
        headers: { "Content-Type": "application/json" },
      });
      if (!res.ok) throw new Error("Failed to fetch stations");
      const data = await res.json();
      setStations(data); // Set fetched stations in state
      console.log("Stations fetched:", data);
    } catch (err) {
      console.error("Failed to fetch stations", err);
    }
  }
}

```

```

};

fetchStations();

}, []);

return (
  <div className="login-container">
    <div className="login-card">
      {/* Header Section */}
      <div className="login-header">
        <div className="logo">
          <div className="logo-icon">⚡</div>
          <h1>EV Charge</h1>
        </div>
        <p className="tagline">Join Our Community</p>
      </div>
      {/* Form Section */}
      <form onSubmit={handleRegister} className="login-form">
        {error && <div className="alert alert-error">{error}</div>}
        {success && <div className="alert alert-success">{success}</div>}

        <div className="form-group">
          <label htmlFor="firstName" className="form-label">First Name</label>
          <input
            id="firstName"
            type="text"
            className="form-input"
          >
        </div>
      </form>
    </div>
  </div>
);

```

```
        value={firstName}

        onChange={(e) => setFirstName(e.target.value)}

        placeholder="Enter your first name"

        required

    />

</div>

<div className="form-group">

    <label htmlFor="lastName" className="form-label">Last Name</label>

    <input

        id="lastName"

        type="text"

        className="form-input"

        value={lastName}

        onChange={(e) => setLastName(e.target.value)}

        placeholder="Enter your last name"

        required

    />

</div>

<div className="form-group">

    <label htmlFor="nic" className="form-label">NIC</label>

    <input

        id="nic"

        type="text"

        className="form-input"

        value={nic}

        onChange={(e) => setNic(e.target.value)}

        placeholder="Enter your NIC"

    />

</div>
```

```
    required
  />
</div>

<div className="form-group">
  <label htmlFor="email" className="form-label">Email Address</label>
  <input
    id="email"
    type="email"
    className="form-input"
    value={email}
    onChange={(e) => setEmail(e.target.value)}
    placeholder="Enter your email"
    required
  />
</div>

<div className="form-group">
  <label htmlFor="password" className="form-label">Password</label>
  <input
    id="password"
    type="password"
    className="form-input"
    value={password}
    onChange={(e) => setPassword(e.target.value)}
    placeholder="Enter your password"
    required
  />
</div>
```

```

<div className="form-group">

  <label htmlFor="role" className="form-label">Role</label>

  <select
    id="role"
    className="form-input"
    value={role}
    onChange={(e) => setRole(Number(e.target.value))}

    required
  >

    <option value={0}>Backoffice</option>
    <option value={1}>Station Operator</option>
  </select>

</div>

{role === 1 && ( // 1 = StationOperator

<div className="form-group">

  <label htmlFor="stationId" className="form-label">Station</label>

  <select
    id="stationId"
    className="form-input"
    value={stationId}
    onChange={(e) => setStationId(e.target.value)}

    required
  >

    <option value="">Select a station</option>
    {stations.map((s) => (
      <option key={s.id} value={s.id}>
        {s.name} ({s.location})
      </option>
    ))}
  </select>
</div>

```

```

        ))}
      </select>
    </div>
  )}

<button
  className={`login-button ${isLoading ? 'loading' : ''}`}
  type="submit"
  disabled={isLoading}
>
  {isLoading ? (
    <>
    <div className="spinner"></div>
    Creating Account...
  </>
  ) : (
    'Create Account'
  )}
</button>
</form>

/* Footer Links */
<div className="login-footer">
  <p className="signup-text">
    Already have an account? <a href="/" className="signup-link">Login here</a>
  </p>
</div>
</div>
</div>

```

```

    );
}

// StationDashboard.jsx

import { Outlet, Link, useLocation, useNavigate } from "react-router-dom";
import { useEffect, useState } from "react";
import { Container, Navbar } from "react-bootstrap";

export default function StationDashboard() {

    // --- State Hooks ---

    const navigate = useNavigate();
    const location = useLocation();
    const [stationId, setStationId] = useState(null);
    const [stationName, setStationName] = useState("");

    // --- Check authentication & load station info on mount ---

    useEffect(() => {
        const token = localStorage.getItem("token");
        const sid = localStorage.getItem("stationId");
        const sname = localStorage.getItem("stationName");

        if (!token) navigate("/"); // Redirect if not logged in
        if (sid) setStationId(sid); // Load station ID
        if (sname) setStationName(sname); // Load station name
    }, [navigate]);

    // --- Handle logout ---

    const handleLogout = () => {
        localStorage.removeItem("token");
    }
}

```

```
localStorage.removeItem("stationId");
localStorage.removeItem("stationName");
localStorage.removeItem("role");
navigate("/"); // Redirect to login page
};

return (
<div style={{
  minHeight: "100vh",
  background: "linear-gradient(135deg, #1B263B 0%, #121212 100%)",
  display: "flex"
}}>
/* Sidebar */
<aside style={{
  width: "280px",
  background: "#F9FAFB",
  borderRight: "1px solid rgba(255, 255, 255, 0.1)",
  padding: "24px",
  display: "flex",
  flexDirection: "column",
  boxShadow: "0 0 20px rgba(0, 0, 0, 0.1)"
}}>
/* Header */
<div style={{
  display: "flex",
  alignItems: "center",
  gap: "12px",
  marginBottom: "32px",
  paddingBottom: "16px",
}}
```

```
borderBottom: "1px solid #E5E7EB"
}}>

<div style={{

  fontSize: "2rem",

  background: "linear-gradient(135deg, #00C853 0%, #00B4D8 100%)",

  WebkitBackgroundClip: "text",

  WebkitTextFillColor: "transparent",

  backgroundClip: "text"

}}>

  ⚡

</div>

<div>

  <h5 style={{

    margin: 0,

    color: "#121212",

    fontWeight: "700",

    fontSize: "1.25rem"

}}>

  Station Panel

</h5>

  <p style={{

    margin: 0,

    color: "#6C757D",

    fontSize: "0.85rem"

}}>

  {stationName || "Operator Dashboard"}

</p>

</div>

</div>
```

```
/* Navigation */

<nav style={{ flex: 1 }}>

  <CustomNavLink
    to="bookings"
    label="Bookings"
    icon="grid"
    isActive={location.pathname.includes('bookings')}
  />

</nav>

/* Footer */

<div style={{

  marginTop: "auto",
  paddingTop: "16px",
  borderTop: "1px solid #E5E7EB"

}}>

  <div style={{

    textAlign: "center",
    marginBottom: "16px",
    color: "#6C757D",
    fontSize: "0.85rem"

}}>

    Station ID: <strong>{stationId || "-"}</strong>

  </div>

  <button
    onClick={handleLogout}
    style={{

      width: "100%",


```

```

background: "linear-gradient(135deg, #DC3545 0%, #C82333 100%)",
color: "white",
border: "none",
padding: "12px",
borderRadius: "12px",
fontWeight: "600",
cursor: "pointer",
transition: "all 0.3s ease"
}}
onMouseEnter={(e) => {
e.target.style.transform = "translateY(-2px)";
e.target.style.boxShadow = "0 5px 15px rgba(220, 53, 69, 0.3)";
}}
onMouseLeave={(e) => {
e.target.style.transform = "translateY(0)";
e.target.style.boxShadow = "none";
}}
>
█ Logout
</button>
</div>
</aside>

{/* Main Content */}

<main style={{
flex: 1,
display: "flex",
flexDirection: "column"
}}>

```

```
<Navbar style={{  
    background: "#F9FAFB",  
    borderBottom: "1px solid #E5E7EB",  
    padding: "16px 24px",  
    boxShadow: "0 2px 10px rgba(0, 0, 0, 0.05)"  
}}>  
  
<Container fluid>  
  
<Navbar.Brand style={{  
    fontWeight: "700",  
    color: "#121212",  
    fontSize: "1.5rem",  
    display: "flex",  
    alignItems: "center",  
    gap: "12px"  
}}>  
  
<div style={{  
    fontSize: "1.8rem",  
    background: "linear-gradient(135deg, #00C853 0%, #00B4D8 100%)",  
    WebkitBackgroundClip: "text",  
    WebkitTextFillColor: "transparent",  
    backgroundClip: "text"  
}}>  
  
      
</div>  
  
    Operator Dashboard  
  
</Navbar.Brand>  
  
</Container>  
  
</Navbar>
```

```

<Container fluid style={{
  padding: "24px",
  flex: 1,
  background: "linear-gradient(135deg, #f8f3f3ff 0%, #F9FAFB 100%)"
}}>

<Outlet context={{ stationId }} />

</Container>

</main>

</div>

);

}

/* Custom Nav Link Component */

function CustomNavLink({ to, label, icon, isActive }) {

  const baseStyle = {

    display: "flex",
    alignItems: "center",
    gap: "12px",
    padding: "12px 16px",
    borderRadius: "12px",
    marginBottom: "8px",
    textDecoration: "none",
    fontWeight: "500",
    transition: "all 0.3s ease",
    border: "none",
    color: "#121212"
  };

  const activeStyle = {

```

```

...baseStyle,
background: "linear-gradient(135deg, #00C853 0%, #00B4D8 100%)",
color: "white",
transform: "translateX(8px)"

};

const hoverStyle = {

background: "linear-gradient(135deg, #00C85320 0%, #00B4D820 100%)",
color: "#121212",
transform: "translateX(8px)"

};

return (
<Link
to={to}
style={isActive ? activeStyle : baseStyle}
onMouseEnter={(e) => {
if (!isActive) {
e.target.style.background = hoverStyle.background;
e.target.style.color = hoverStyle.color;
e.target.style.transform = hoverStyle.transform;
}
}}
onMouseLeave={(e) => {
if (!isActive) {
e.target.style.background = "";
e.target.style.color = baseStyle.color;
e.target.style.transform = "";
}
}}
)

```

```

        }}

      >

      <span style={{ fontSize: "1.2rem" }}>{icon}</span>

      <span>{label}</span>

      </Link>

    );

}

//Users.jsx

import { useEffect, useState } from "react";

import { Table, Button, Modal, Form, Alert } from "react-bootstrap";


export default function Users() {

  // --- State Hooks ---

  const [users, setUsers] = useState([]);

  const [showModal, setShowModal] = useState(false);

  const [editMode, setEditMode] = useState(false);

  const [selectedUser, setSelectedUser] = useState(null);

  const [error, setError] = useState("");

  const [success, setSuccess] = useState("");

  const [loading, setLoading] = useState(false);

  const [nic, setNic] = useState("");

  const [firstName, setFirstName] = useState("");

  const [lastName, setLastName] = useState("");

  const [email, setEmail] = useState("");

  const [phoneNumber, setPhoneNumber] = useState("");

  const [role, setRole] = useState("");

  const [isActive, setIsActive] = useState(false);
}

```

```
const token = localStorage.getItem("token");

// --- Fetch users from API ---
const fetchUsers = async () => {
  setLoading(true);
  try {
    const res = await fetch("http://localhost:5082/api/users", {
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
    });
    if (!res.ok) throw new Error("Failed to fetch users");
    const data = await res.json();
    setUsers(data);
  } catch (err) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
};

// --- Load users on component mount ---
useEffect(() => {
  if (!token) {
    setError("Not authorized. Please login.");
    return;
  }
});
```

```

    }

    fetchUsers();

}, [token]);


// --- Delete a user ---

const handleDelete = async (id) => {
  if (!window.confirm("Are you sure you want to delete this user?")) return;

  try {
    const res = await fetch(`http://localhost:5082/api/users/${id}`, {
      method: "DELETE",
      headers: { Authorization: `Bearer ${token}` },
    });

    if (!res.ok) throw new Error("Failed to delete user");

    setSuccess("User deleted successfully");
    fetchUsers();
  } catch (err) {
    setError(err.message);
  }
};

// --- Toggle user active/inactive status ---

const handleToggleStatus = async (id, currentStatus) => {
  const confirmMsg = currentStatus
    ? "Deactivate this user?"
    : "Activate this user?";

  if (!window.confirm(confirmMsg)) return;

```

```

try {

  const res = await fetch(`http://localhost:5082/api/users/${id}/status`, {
    method: "PATCH",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify(!currentStatus),
  });

  if (!res.ok) throw new Error("Failed to update status");

  setSuccess(`User ${currentStatus ? "deactivated" : "activated"} successfully`);

  fetchUsers();

} catch (err) {

  setError(err.message);

}

};

// --- Open modal to view user details ---

const handleView = (user) => {

  setSelectedUser(user);

  setEditMode(false);

  setNic(user.nic);

  setFirstName(user.firstName);

  setLastName(user.lastName);

  setEmail(user.email);

  setPhoneNumber(user.phoneNumber || "");

  setRole(user.role);

  setIsActive(user.isActive);

}

```

```
        setShowModal(true);

    };

// --- Open modal to edit user ---

const handleEdit = (user) => {
    setSelectedUser(user);
    setEditMode(true);
    setNic(user.nic);
    setFirstName(user.firstName);
    setLastName(user.lastName);
    setEmail(user.email);
    setPhoneNumber(user.phoneNumber || "");
    setRole(user.role);
    setIsActive(user.isActive);
    setShowModal(true);
};

// --- Save updated user information ---

const handleSave = async () => {
    if (!selectedUser) return;

    const payload = {
        nic,
        firstName,
        lastName,
        email,
        phoneNumber,
        role: Number(role),
        isActive,
    };
}
```

```

};

try {
  const res = await fetch(
    `http://localhost:5082/api/users/${selectedUser.id}`,
    {
      method: "PUT",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${token}`,
      },
      body: JSON.stringify(payload),
    }
  );
}

if (!res.ok) throw new Error("Failed to update user");
setSuccess("User updated successfully");
setShowModal(false);
fetchUsers();
} catch (err) {
  setError(err.message);
}
};

// --- Clear error and success messages after 5 seconds ---
useEffect(() => {
  if (error || success) {
    const timer = setTimeout(() => {
      setError("");
    }, 5000);
  }
});

```

```
        setSuccess("");
    }, 5000);

    return () => clearTimeout(timer);

}

}, [error, success]);
```

```
return (

<div style={{ padding: '24px' }}>

{/* Header Section */}

<div style={{

    display: 'flex',

    alignItems: 'center',

    gap: '16px',

    marginBottom: '32px'

}}>

<div style={{

    fontSize: '2.5rem',

    background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',

    WebkitBackgroundClip: 'text',

    WebkitTextFillColor: 'transparent',

    backgroundClip: 'text'

}}>


```

██████████

```
</div>

<div>

<h1 style={{

    color: '#121212',

    margin: 0,
```

██████████

```
    fontWeight: '700',
```

```
    fontSize: '2rem'

  }>

  Manage Users

</h1>

<p style={{

  color: '#6C757D',


  margin: 0,


  fontSize: '1.1rem'

}}>

  View and manage all system users

</p>

</div>

</div>

/* Alerts */

{error && (

<div style={{

  background: 'rgba(255, 183, 3, 0.1)',

  color: '#FFB703',


  padding: '16px',


  borderRadius: '12px',


  border: '1px solid rgba(255, 183, 3, 0.3)',


  marginBottom: '24px',


  fontWeight: '500'

}}>

{error}

</div>

) }
```

```

{success && (
  <div style={{
    background: 'rgba(0, 200, 83, 0.1)',
    color: '#00C853',
    padding: '16px',
    borderRadius: '12px',
    border: '1px solid rgba(0, 200, 83, 0.3)',
    marginBottom: '24px',
    fontWeight: '500'
  }}>
  {success}
</div>
)}

```

```

/* Users Table */
<div style={{
  background: '#F9FAFB',
  borderRadius: '20px',
  padding: '32px',
  boxShadow: '0 10px 40px rgba(0, 0, 0, 0.1)',
  border: '1px solid rgba(255, 255, 255, 0.1)'
}}>
{loading ? (
  <div style={{
    textAlign: 'center',
    padding: '40px',
    color: '#6C757D'
  }}>
    <div className="spinner" style={{

```

```

width: '40px',
height: '40px',
border: '3px solid transparent',
borderTop: '3px solid #00C853',
borderRadius: '50%',
animation: 'spin 1s linear infinite',
margin: '0 auto 16px'

}}></div>

Loading users...

</div>

): (
<Table hover responsive style={{

margin: 0,
border: 'none'

}}>

<thead>

<tr style={{

background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',
color: '#F9FAFB'

}}>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>NIC</th>

<th style={{

padding: '16px',
border: 'none',
```

```
fontWeight: '600',
fontSize: '0.95rem'

}}>Name</th>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Email</th>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Phone</th>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Role</th>

<th style={{

padding: '16px',
border: 'none',
fontWeight: '600',
fontSize: '0.95rem'

}}>Status</th>

<th style={{

padding: '16px',
```

```
border: 'none',
fontWeight: '600',
fontSize: '0.95rem',
textAlign: 'center'

}}>Actions</th>

</tr>
</thead>
<tbody>
{users.length > 0 ? (
  users.map((u) => (
    <tr key={u.id} style={{
      borderBottom: '1px solid #E5E7EB',
      transition: 'all 0.3s ease'
    }}>
    <td style={{
      padding: '16px',
      fontWeight: '500',
      color: '#121212'
    }}>{u.nic}</td>
    <td style={{
      padding: '16px',
      color: '#121212'
    }}>{u.firstName} {u.lastName}</td>
    <td style={{
      padding: '16px',
      color: '#121212'
    }}>{u.email}</td>
    <td style={{
      padding: '16px',

```

```

color: '#121212'

}}>{u.phoneNumber || "N/A"</td>

<td style={{ padding: '16px' }}>

<span style={{

padding: '6px 12px',


borderRadius: '20px',


fontSize: '0.85rem',


fontWeight: '500',


background: u.role === 0

? 'linear-gradient(135deg, #00C853 20 0%, #00B4D8 20 100%)'

: u.role === 1

? 'linear-gradient(135deg, #FFB703 20 0%, #FF9100 20 100%)'

: 'linear-gradient(135deg, #7209B7 20 0%, #560BAD 20 100%)',


color: u.role === 0

? '#00C853'

: u.role === 1

? '#FFB703'

: '#7209B7'

}}>

{u.role === 0 ? "Backoffice" : u.role === 1 ? "Operator" : "EV Owner"}


</span>

</td>

<td style={{ padding: '16px' }}>

<span style={{

padding: '6px 12px',


borderRadius: '20px',


fontSize: '0.85rem',


fontWeight: '500',


background: u.isActive

```

```
? 'rgba(0, 200, 83, 0.1)'  
: 'rgba(108, 117, 125, 0.1)',  
color: u.isActive ? '#00C853' : '#6C757D'  
}}>  
{u.isActive ? "Active" : "Inactive"}  
</span>  
</td>  
<td style={{  
padding: '16px',  
textAlign: 'center'  
}}>  
<div style={{  
display: 'flex',  
gap: '8px',  
justifyContent: 'center',  
flexWrap: 'wrap'  
}}>  
<Button  
variant="outline-info"  
size="sm"  
onClick={() => handleView(u)}  
style={{  
border: '2px solid #00B4D8',  
color: '#00B4D8',  
borderRadius: '8px',  
fontWeight: '500',  
padding: '6px 12px'  
}}>  
>
```

👁 View

```
</Button>

<Button
    variant="outline-warning"
    size="sm"
    onClick={() => handleEdit(u)}
    style={{
        border: '2px solid #FFB703',
        color: '#FFB703',
        borderRadius: '8px',
        fontWeight: '500',
        padding: '6px 12px'
    }}
>

    🖍 Edit
</Button>

<Button
    variant={u.isActive ? "outline-secondary" : "outline-success"}
    size="sm"
    onClick={() => handleToggleStatus(u.id, u.isActive)}
    style={{
        border: u.isActive ? '2px solid #6C757D' : '2px solid #00C853',
        color: u.isActive ? '#6C757D' : '#00C853',
        borderRadius: '8px',
        fontWeight: '500',
        padding: '6px 12px'
    }}
>

{u.isActive ? "🚫 Deactivate" : "➡️ Activate"}
```

```

        </Button>

        <Button
            variant="outline-danger"
            size="sm"
            onClick={() => handleDelete(u.id)}
            style={{
                border: '2px solid #DC3545',
                color: '#DC3545',
                borderRadius: '8px',
                fontWeight: '500',
                padding: '6px 12px'
            }}
        >
             Delete
        </Button>
    </div>
</td>
</tr>
))
) : (
<tr>
<td colSpan="7" style={{
    padding: '40px',
    textAlign: 'center',
    color: '#6C757D'
}}>
<div style={{ fontSize: '3rem', marginBottom: '16px' }}>👤 </div>
<h4 style={{ color: '#121212', marginBottom: '8px' }}>No Users Found</h4>
<p>There are no users in the system yet.</p>

```

```

        </td>
      </tr>
    )}
</tbody>
</Table>
)}
</div>

/* View/Edit Modal */

<Modal show={showModal} onHide={() => setShowModal(false)} centered>
<Modal.Header style={{
  background: 'linear-gradient(135deg, #1B263B 0%, #121212 100%)',
  color: '#F9FAFB',
  border: 'none'
}}>
<Modal.Title style={{ fontWeight: '600' }}>
  {editMode ? "📝 Edit User" : "👁 User Details"}
</Modal.Title>
<button
  onClick={() => setShowModal(false)}
  style={{
    background: 'none',
    border: 'none',
    color: '#F9FAFB',
    fontSize: '1.5rem',
    cursor: 'pointer'
}}
>
  ✕

```

```

        </button>

    </Modal.Header>

    <Modal.Body style={{ padding: '24px', background: '#F9FAFB' }}>
        {selectedUser && (
            <Form>
                <Form.Group className="mb-3">
                    <Form.Label style={{ fontWeight: '600', color: '#121212' }}>NIC</Form.Label>
                    <Form.Control
                        value={nic}
                        readOnly
                        style={{
                            padding: '12px',
                            borderRadius: '8px',
                            border: '2px solid #E5E7EB',
                            background: '#FFFFFF'
                        }}
                    />
                </Form.Group>

                <div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '16px' }}>
                    <Form.Group className="mb-3">
                        <Form.Label style={{ fontWeight: '600', color: '#121212' }}>First Name</Form.Label>
                        <Form.Control
                            value={firstName}
                            onChange={(e) => setFirstName(e.target.value)}
                            readOnly={!editMode}
                            style={{
                                padding: '12px',
                                borderRadius: '8px',
                            }}
                        />
                    </Form.Group>
                </div>
            </Form>
        )}>
    </Modal.Body>

```

```

        border: '2px solid #E5E7EB',
        background: '#FFFFFF'
    )}
/>
</Form.Group>

<Form.Group className="mb-3">
    <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Last Name</Form.Label>
    <Form.Control
        value={lastName}
        onChange={(e) => setLastName(e.target.value)}
        readOnly={!editMode}
        style={{
            padding: '12px',
            borderRadius: '8px',
            border: '2px solid #E5E7EB',
            background: '#FFFFFF'
        }}
    />
</Form.Group>
</div>

<Form.Group className="mb-3">
    <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Email</Form.Label>
    <Form.Control
        value={email}
        onChange={(e) => setEmail(e.target.value)}
        readOnly={!editMode}
        style={{

```

```
padding: '12px',
borderRadius: '8px',
border: '2px solid #E5E7EB',
background: '#FFFFFF'
}}
/>
</Form.Group>

<Form.Group className="mb-3">
<Form.Label style={{ fontWeight: '600', color: '#121212' }}>Phone Number</Form.Label>
<Form.Control
value={phoneNumber}
onChange={(e) => setPhoneNumber(e.target.value)}
readOnly={!editMode}
style={{
padding: '12px',
borderRadius: '8px',
border: '2px solid #E5E7EB',
background: '#FFFFFF'
}}
/>
</Form.Group>

<div style={{ display: 'grid', gridTemplateColumns: '1fr 1fr', gap: '16px' }}>
<Form.Group className="mb-3">
<Form.Label style={{ fontWeight: '600', color: '#121212' }}>Role</Form.Label>
<Form.Control
value={
role === 0
```

```

    ? "Backoffice"
    : role === 1
      ? "Station Operator"
      : "EV Owner"
    }
  readOnly
  style={{
    padding: '12px',
    borderRadius: '8px',
    border: '2px solid #E5E7EB',
    background: '#FFFFFF'
  }}
/>
</Form.Group>

<Form.Group className="mb-3">
  <Form.Label style={{ fontWeight: '600', color: '#121212' }}>Status</Form.Label>
  <Form.Control
    value={isActive ? "Active" : "Inactive"}
    readOnly
    style={{
      padding: '12px',
      borderRadius: '8px',
      border: '2px solid #E5E7EB',
      background: '#FFFFFF'
    }}
  /
>
</Form.Group>
</div>

```

```
</Form>

)}

</Modal.Body>

<Modal.Footer style={{

background: '#F9FAFB',
border: 'none',
padding: '16px 24px 24px'

}}>

<Button

variant="outline-secondary"
onClick={() => setShowModal(false)}
style={{

border: '2px solid #6C757D',
color: '#6C757D',
borderRadius: '8px',
fontWeight: '500',
padding: '8px 20px'

}}
>

Close

</Button>

{editMode && (
<Button

variant="primary"
onClick={handleSave}
style={{

background: 'linear-gradient(135deg, #00C853 0%, #00B4D8 100%)',
border: 'none',
borderRadius: '8px',
```

```
        fontWeight: '600',
        padding: '8px 24px'
    )}
>
     Save Changes
</Button>
)}
```

</Modal.Footer>

</Modal>

```
<style jsx>{`  
@keyframes spin {  
    0% { transform: rotate(0deg); }  
    100% { transform: rotate(360deg); }  
}  
  
tr:hover {  
    background: rgba(0, 200, 83, 0.02) !important;  
    transform: translateY(-1px);  
}  
  
.btn:hover {  
    transform: translateY(-2px);  
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);  
}  
`}</style>
```

</div>

);

}

```
/* Reset and Base Styles */

* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

.login-container {
    min-height: 100vh;
    background: linear-gradient(135deg, #1B263B 0%, #121212 100%);
    display: flex;
    align-items: center;
    justify-content: center;
    padding: 20px;
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
}

.login-card {
    background: #F9FAFB;
    border-radius: 20px;
    padding: 40px;
    box-shadow: 0 20px 60px rgba(0, 0, 0, 0.3);
    width: 100%;
    max-width: 440px;
    border: 1px solid rgba(255, 255, 255, 0.1);
}

/* Header Styles */

.login-header {
```

```
    text-align: center;  
    margin-bottom: 40px;  
}  
  
}
```

```
.logo {  
    display: flex;  
    align-items: center;  
    justify-content: center;  
    gap: 12px;  
    margin-bottom: 8px;  
}
```

```
.logo-icon {  
    font-size: 2.5rem;  
    background: linear-gradient(135deg, #00C853 0%, #00B4D8 100%);  
    -webkit-background-clip: text;  
    -webkit-text-fill-color: transparent;  
    background-clip: text;  
}
```

```
.logo h1 {  
    color: #121212;  
    font-size: 2rem;  
    font-weight: 700;  
    letter-spacing: -0.5px;  
}
```

```
.tagline {  
    color: #6C757D;
```

```
    font-size: 1.1rem;  
    font-weight: 400;  
}  
  
/* Form Styles */
```

```
.login-form {  
    margin-bottom: 30px;  
}
```

```
.form-group {  
    margin-bottom: 24px;  
}
```

```
.form-label {  
    display: block;  
    color: #121212;  
    font-weight: 600;  
    margin-bottom: 8px;  
    font-size: 0.95rem;  
}
```

```
.form-input {  
    width: 100%;  
    padding: 16px;  
    border: 2px solid #E5E7EB;  
    border-radius: 12px;  
    font-size: 1rem;  
    transition: all 0.3s ease;  
    background: #FFFFFF;
```

```
color: #121212;  
}  
  
.form-input:focus {  
    outline: none;  
    border-color: #00C853;  
    box-shadow: 0 0 0 3px rgba(0, 200, 83, 0.1);  
    transform: translateY(-2px);  
}  
  
.form-input::placeholder {  
    color: #9CA3AF;  
}  
  
/* Button Styles */  
.login-button {  
    width: 100%;  
    background: linear-gradient(135deg, #00C853 0%, #00B4D8 100%);  
    color: white;  
    border: none;  
    padding: 16px;  
    border-radius: 12px;  
    font-size: 1.1rem;  
    font-weight: 600;  
    cursor: pointer;  
    transition: all 0.3s ease;  
    display: flex;  
    align-items: center;  
    justify-content: center;
```

```
gap: 10px;  
margin-top: 10px;  
}  
  
.login-button:hover:not(:disabled) {  
    transform: translateY(-2px);  
    box-shadow: 0 10px 25px rgba(0, 200, 83, 0.3);  
}  
  
.login-button:active {  
    transform: translateY(0);  
}  
  
.login-button:disabled {  
    opacity: 0.7;  
    cursor: not-allowed;  
    transform: none;  
}  
  
.login-button.loading {  
    background: linear-gradient(135deg, #00B4D8 0%, #0096C7 100%);  
}  
  
/* Spinner Animation */  
.spinner {  
    width: 20px;  
    height: 20px;  
    border: 2px solid transparent;  
    border-top: 2px solid white;
```

```
border-radius: 50%;  
animation: spin 1s linear infinite;  
}
```

```
@keyframes spin {  
0% { transform: rotate(0deg); }  
100% { transform: rotate(360deg); }  
}
```

```
/* Alert Styles */
```

```
.alert {  
padding: 16px;  
border-radius: 12px;  
margin-bottom: 24px;  
font-weight: 500;  
text-align: center;  
}
```

```
.alert-error {  
background: rgba(255, 183, 3, 0.1);  
color: #FFB703;  
border: 1px solid rgba(255, 183, 3, 0.3);  
}
```

```
/* Footer Styles */
```

```
.login-footer {  
text-align: center;  
border-top: 1px solid #E5E7EB;  
padding-top: 24px;
```

```
}
```

```
.footer-link {  
    color: #00B4D8;  
    text-decoration: none;  
    font-weight: 500;  
    transition: color 0.3s ease;  
}
```

```
.footer-link:hover {  
    color: #00C853;  
    text-decoration: underline;  
}
```

```
.signup-text {  
    margin-top: 16px;  
    color: #6C757D;  
    font-size: 0.95rem;  
}
```

```
.signup-link {  
    color: #00C853;  
    text-decoration: none;  
    font-weight: 600;  
    transition: color 0.3s ease;  
}
```

```
.signup-link:hover {  
    color: #00B4D8;
```

```
text-decoration: underline;  
}
```

```
/* Responsive Design */  
  
@media (max-width: 480px) {  
  
.login-card {  
  
padding: 30px 24px;  
  
margin: 10px;  
}
```

```
.logo h1 {  
  
font-size: 1.75rem;  
}  
  
}
```

```
.logo-icon {  
  
font-size: 2rem;  
}  
  
}
```

```
.tagline {  
  
font-size: 1rem;  
}  
  
}
```

```
.form-input {  
  
padding: 14px;  
}  
  
}
```

```
.login-button {  
  
padding: 14px;  
  
font-size: 1rem;
```

```
}

}

/* Dark mode support */

@media (prefers-color-scheme: dark) {

.login-card {

background: #1B263B;

border: 1px solid rgba(255, 255, 255, 0.1);

}

.logo h1,

.form-label {

color: #F9FAFB;

}

.form-input {

background: #121212;

border-color: #374151;

color: #F9FAFB;

}

.form-input::placeholder {

color: #6B7280;

}

.tagline,

.signup-text {

color: #9CA3AF;

}
```

```
}

/* Add this to your existing CSS */

.alert-success {

background: rgba(0, 200, 83, 0.1);

color: #00C853;

border: 1px solid rgba(0, 200, 83, 0.3);

}

/* Ensure the select element matches the input styling */

.form-input select {

appearance: none;

background-image: url("data:image/svg+xml; charset=US-ASCII,<svg

xmlns='http://www.w3.org/2000/svg' viewBox='0 0 4 5'><path fill='%23666' d='M2 0L0 2h4zm0 5L0

3h4z' /></svg>");

background-repeat: no-repeat;

background-position: right 16px center;

background-size: 12px;

}

.form-input select:focus {

outline: none;

border-color: #00C853;

box-shadow: 0 0 0 3px rgba(0, 200, 83, 0.1);

transform: translateY(-2px);

}
```

```

//App.js

import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Login from "./pages/Login";
import Register from "./pages/Register";
import Dashboard from "./pages/Dashboard";
import Users from "./pages/Users";
import EVOwnerManagement from "./pages/EVOwnerManagement";
import ChargingStations from "./pages/ChargingStations";
import StationDashboard from "./pages/StationDashboard";
import OperatorBookings from "./pages/OperatorBookings";

function App() {
  return (
    <Router>
      <Routes>
        {/* Auth Pages */}
        <Route path="/" element={<Login />} />
        <Route path="/register" element={<Register />} />

        {/* Backoffice Dashboard (Admin) */}
        <Route path="/dashboard" element={<Dashboard />}>
          <Route path="users" element={<Users />} />
          <Route path="ev-owners" element={<EVOwnerManagement />} />
          <Route path="stations" element={<ChargingStations />} />
        </Route>

        {/* ☑ Station Dashboard (for operators) */}
        <Route path="/station-dashboard" element={<StationDashboard />}>

```

```
<Route path="bookings" element={<OperatorBookings />} />  
</Route>  
</Routes>  
</Router>  
);  
}
```

```
export default App;
```

## 7.3 EV Charging Backend Codes

```
/*
 * File: AuthController.cs
 * Project: EV Charging Station Booking System
 * Description: Authentication controller for login and registration
 * Author: EV Charging System
 * Date: September 27, 2025
 */
```

```
using EVChargingBackend.DTOs;
using EVChargingBackend.Services;
using Microsoft.AspNetCore.Mvc;

namespace EVChargingBackend.Controllers
{
    /// <summary>
    /// Controller for user authentication operations (login and registration)
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    public class AuthController : ControllerBase
    {
        private readonly IAuthService _authService;
        private readonly ILogger<AuthController> _logger;

        /// <summary>
        /// Initializes authentication controller with dependencies
        /// </summary>
    }
}
```

```

/// <param name="authService">Authentication service</param>
/// <param name="logger">Logger for controller operations</param>

public AuthController(IAuthService authService, ILogger<AuthController> logger)
{
    _authService = authService;
    _logger = logger;
}

/// <summary>
/// Registers a new user in the system
/// </summary>

/// <param name="registerDto">User registration data</param>
/// <returns>Created user information</returns>
[HttpPost("register")]

public async Task<ActionResult<UserResponseDto>> Register([FromBody] RegisterUserDto
registerDto)
{
    _logger.LogInformation("Registration attempt for email: {Email}", registerDto.Email);

    var user = await _authService.RegisterAsync(registerDto);

    return CreatedAtAction(nameof(Register), new { id = user.Id }, user);
}

/// <summary>
/// Authenticates user login and returns JWT token
/// </summary>

/// <param name="loginDto">Login credentials</param>
/// <returns>Login response with JWT token</returns>
[HttpPost("login")]

public async Task<ActionResult<LoginResponseDto>> Login([FromBody] LoginDto loginDto)

```

```

    {
        _logger.LogInformation("Login attempt for email: {Email}", loginDto.Email);

        var response = await _authService.LoginAsync(loginDto);

        return Ok(response);
    }
}

/*
 * File: BookingsController.cs
 * Project: EV Charging Station Booking System
 * Description: Booking management controller with role-based operations
 * Author: EV Charging System
 * Date: September 27, 2025
 */

using EVChargingBackend.DTOs;
using EVChargingBackend.Models;
using EVChargingBackend.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

namespace EVChargingBackend.Controllers
{
    /// <summary>
    /// Controller for booking management operations with role-based access control

```

```

/// </summary>
[ApiController]
[Route("api/[controller]")]
[Authorize]
public class BookingsController : ControllerBase
{
    private readonly IBookingService _bookingService;
    private readonly ILogger<BookingsController> _logger;

    /// <summary>
    /// Initializes bookings controller with dependencies
    /// </summary>
    /// <param name="bookingService">Booking service for business operations</param>
    /// <param name="logger">Logger for controller operations</param>
    public BookingsController(IBookingService bookingService, ILogger<BookingsController> logger)
    {
        _bookingService = bookingService;
        _logger = logger;
    }

    /// <summary>
    /// Creates a new booking (EV Owner only)
    /// </summary>
    /// <param name="createDto">Booking creation data</param>
    /// <returns>Created booking information</returns>
    [HttpPost]
    [Authorize(Roles = nameof(UserRole.EVOwner))]
    public async Task<ActionResult<BookingResponseDto>> CreateBooking([FromBody]
CreateBookingDto createDto)
    {

```

```

var ownerNic = User.FindFirst("nic")?.Value;

if (string.IsNullOrEmpty(ownerNic))
{
    return BadRequest("Invalid user token");
}

_logger.LogInformation("Creating booking for station {StationId} by user {OwnerNIC}",
createDto.StationId, ownerNic);

var booking = await _bookingService.CreateBookingAsync(createDto, ownerNic);

return CreatedAtAction(nameof(GetBooking), new { id = booking.Id }, booking);
}

/// <summary>
/// Gets all bookings (backoffice and station operators)
/// </summary>
/// <returns>List of all bookings</returns>
[HttpGet]
[Authorize(Roles = $"{nameof(UserRole.Backoffice)},{nameof(UserRole.StationOperator)}")]
public async Task<ActionResult<List<BookingResponseDto>>> GetAllBookings()

{
    _logger.LogInformation("Fetching all bookings");

    var bookings = await _bookingService.GetAllBookingsAsync();

    return Ok(bookings);
}

/// <summary>

```

```

/// Gets current user's bookings (EV Owner only)

/// </summary>

/// <returns>List of user's bookings</returns>

[HttpGet("my-bookings")]

[Authorize(Roles = nameof(UserRole.EVOwner))]

public async Task<ActionResult<List<BookingResponseDto>>> GetMyBookings()

{

    var ownerNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(ownerNic))

    {

        return BadRequest("Invalid user token - NIC not found");

    }

    _logger.LogInformation("Fetching bookings for user: {OwnerNIC}", ownerNic);

    var bookings = await _bookingService.GetUserBookingsAsync(ownerNic);

    return Ok(bookings);

}

/// <summary>

/// Gets bookings for a specific station (station operators)

/// </summary>

/// <param name="stationId">Station ID</param>

/// <returns>List of station bookings</returns>

[HttpGet("station/{stationId}")]

[Authorize(Roles = $"{nameof(UserRole.Backoffice)},{nameof(UserRole.StationOperator)}")]

public async Task<ActionResult<List<BookingResponseDto>>> GetStationBookings(string stationId)

{

```

```

_logger.LogInformation("Fetching bookings for station: {StationId}", stationId);

var bookings = await _bookingService.GetStationBookingsAsync(stationId);

return Ok(bookings);

}

/// <summary>

/// Gets booking by ID

/// </summary>

/// <param name="id">Booking ID</param>

/// <returns>Booking information</returns>

[HttpGet("{id}")]

public async Task<ActionResult<BookingResponseDto>> GetBooking(string id)

{

    var currentUserRole = User.FindFirst(ClaimTypes.Role)?.Value;

    var ownerNic = User.FindFirst("nic")?.Value;

    _logger.LogInformation("Fetching booking: {BookingId}", id);

    var booking = await _bookingService.GetBookingByIdAsync(id);

    // EV Owners can only view their own bookings

    if (currentUserRole == nameof(UserRole.EVOwner) && booking.OwnerNIC != ownerNic)

    {

        return Forbid("You can only view your own bookings");

    }

    return Ok(booking);

}

```

```

/// <summary>
/// Updates booking information (EV Owner only, own bookings)
/// </summary>
/// <param name="id">Booking ID</param>
/// <param name="updateDto">Updated booking data</param>
/// <returns>Updated booking information</returns>
[HttpPut("{id}")]
[Authorize(Roles = nameof(UserRole.EVOwner))]
public async Task<ActionResult<BookingResponseDto>> UpdateBooking(string id, [FromBody]
UpdateBookingDto updateDto)
{
    var ownerNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(ownerNic))
    {
        return BadRequest("Invalid user token - NIC not found");
    }

    _logger.LogInformation("Updating booking: {BookingId}", id);

    var booking = await _bookingService.UpdateBookingAsync(id, updateDto, ownerNic);

    return Ok(booking);
}

/// <summary>
/// Confirms booking (station operator action)
/// </summary>
/// <param name="id">Booking ID</param>
/// <returns>Success status</returns>

```

```

[HttpPost("{id}/confirm")]

[Authorize(Roles = $"{nameof(UserRole.Backoffice)},{nameof(UserRole.StationOperator)}")]

public async Task<ActionResult> ConfirmBooking(string id)

{

    _logger.LogInformation("Confirming booking: {BookingId}", id);

    var result = await _bookingService.ConfirmBookingAsync(id);

    if (!result)

    {

        return NotFound("Booking not found");

    }

    return Ok(new { message = "Booking confirmed successfully" });

}

/// <summary>

/// Cancels booking (EV Owner only, own bookings)

/// </summary>

/// <param name="id">Booking ID</param>

/// <returns>Success status</returns>

[HttpPost("{id}/cancel")]

[Authorize(Roles = nameof(UserRole.EVOwner))]

public async Task<ActionResult> CancelBooking(string id)

{

    var ownerNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(ownerNic))

    {

        return BadRequest("Invalid user token - NIC not found");

    }

```

```

    }

_logger.LogInformation("Cancelling booking: {BookingId}", id);

var result = await _bookingService.CancelBookingAsync(id, ownerNic);

if (!result)
{
    return NotFound("Booking not found");
}

return Ok(new { message = "Booking cancelled successfully" });

}

/// <summary>
/// Cancels booking by operator (Station Operator and Backoffice only)
/// </summary>

/// <param name="id">Booking ID</param>
/// <param name="request">Cancellation request with reason</param>
/// <returns>Success status</returns>
[HttpPost("{id}/cancel-by-operator")]

[Authorize(Roles = $"{nameof(UserRole.Backoffice)},{nameof(UserRole.StationOperator)}")]

public async Task<ActionResult> CancelBookingByOperator(string id, [FromBody]
CancelBookingByOperatorDto request)

{
    _logger.LogInformation("Operator cancelling booking: {BookingId}, Reason: {Reason}", id,
request.Reason);

var result = await _bookingService.CancelBookingByOperatorAsync(id, request.Reason);

if (!result)

```

```

    {
        return NotFound("Booking not found");
    }

    return Ok(new { message = "Booking cancelled successfully by operator" });
}

}

/*
 * File: ChargingStationsController.cs
 * Project: EV Charging Station Booking System
 * Description: Charging station management controller
 * Author: EV Charging System
 * Date: September 27, 2025
 */

```

```

using EVChargingBackend.DTOs;
using EVChargingBackend.Models;
using EVChargingBackend.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace EVChargingBackend.Controllers
{
    /// <summary>
    /// Controller for charging station management operations
    /// </summary>
    [ApiController]

```

```

[Route("api/[controller]")]
[Authorize]

public class ChargingStationsController : ControllerBase
{
    private readonly IChargingStationService _stationService;
    private readonly ILogger<ChargingStationsController> _logger;

    /// <summary>
    /// Initializes charging stations controller with dependencies
    /// </summary>
    /// <param name="stationService">Charging station service</param>
    /// <param name="logger">Logger for controller operations</param>
    public ChargingStationsController(IChargingStationService stationService,
        ILogger<ChargingStationsController> logger)
    {
        _stationService = stationService;
        _logger = logger;
    }

    /// <summary>
    /// Creates a new charging station (backoffice only)
    /// </summary>
    /// <param name="createDto">Station creation data</param>
    /// <returns>Created station information</returns>
    [HttpPost]
    [Authorize(Roles = nameof(UserRole.Backoffice))]

    public async Task<ActionResult<StationResponseDto>> CreateStation([FromBody] CreateStationDto
createDto)
    {
        _logger.LogInformation("Creating new charging station: {Name}", createDto.Name);
    }
}

```

```

        var station = await _stationService.CreateStationAsync(createDto);

        return CreatedAtAction(nameof(GetStation), new { id = station.Id }, station);
    }

    /// <summary>
    /// Gets all charging stations
    /// </summary>
    /// <returns>List of all charging stations</returns>
    [HttpGet]
    [AllowAnonymous] // allow non-logged-in users
    public async Task<ActionResult<List<StationResponseDto>>> GetAllStations()
    {
        _logger.LogInformation("Fetching all charging stations");

        var stations = await _stationService.GetAllStationsAsync();

        return Ok(stations);
    }

    /// <summary>
    /// Gets active charging stations only
    /// </summary>
    /// <returns>List of active charging stations</returns>
    [HttpGet("active")]
    public async Task<ActionResult<List<StationResponseDto>>> GetActiveStations()
    {
        _logger.LogInformation("Fetching active charging stations");

        var stations = await _stationService.GetActiveStationsAsync();
    }
}

```

```

        return Ok(stations);
    }

    /// <summary>
    /// Gets charging station by ID
    /// </summary>

    /// <param name="id">Station ID</param>
    /// <returns>Station information</returns>

    [HttpGet("{id}")]
    public async Task<ActionResult<StationResponseDto>> GetStation(string id)
    {
        _logger.LogInformation("Fetching charging station: {StationId}", id);

        var station = await _stationService.GetStationByIdAsync(id);

        return Ok(station);
    }

    /// <summary>
    /// Updates charging station information (backoffice only)
    /// </summary>

    /// <param name="id">Station ID</param>
    /// <param name="updateDto">Updated station data</param>
    /// <returns>Updated station information</returns>

    [HttpPut("{id}")]
    [Authorize(Roles = nameof(UserRole.Backoffice))]
    public async Task<ActionResult<StationResponseDto>> UpdateStation(string id, [FromBody]
UpdateStationDto updateDto)
    {
        _logger.LogInformation("Updating charging station: {StationId}", id);

```

```

        var station = await _stationService.UpdateStationAsync(id, updateDto);

        return Ok(station);
    }

    /// <summary>
    /// Deactivates a charging station (backoffice only)
    /// </summary>

    /// <param name="id">Station ID</param>
    /// <returns>Success status</returns>

    [HttpPost("{id}/deactivate")]
    [Authorize(Roles = nameof(UserRole.Backoffice))]

    public async Task<ActionResult> DeactivateStation(string id)

    {
        _logger.LogInformation("Deactivating charging station: {StationId}", id);

        var result = await _stationService.DeactivateStationAsync(id);

        if (!result)
        {
            return NotFound("Charging station not found");
        }

        return Ok(new { message = "Charging station deactivated successfully" });
    }

    /// <summary>
    /// Deletes a charging station (backoffice only)
    /// </summary>

    /// <param name="id">Station ID</param>
    /// <returns>Success status</returns>

```

```

[HttpDelete("{id}")]
[Authorize(Roles = nameof(UserRole.Backoffice))]
public async Task<ActionResult> DeleteStation(string id)
{
    _logger.LogInformation("Deleting charging station: {StationId}", id);

    var result = await _stationService.DeleteStationAsync(id);

    if (!result)
    {
        return NotFound("Charging station not found");
    }

    return Ok(new { message = "Charging station deleted successfully" });
}

}

/*
 * File: NotificationsController.cs
 * Project: EV Charging Station Booking System
 * Description: Controller for notification management operations
 * Author: EV Charging System
 * Date: October 7, 2025
 */
using EVChargingBackend.DTOs;
using EVChargingBackend.Models;
using EVChargingBackend.Services;

```

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

namespace EVChargingBackend.Controllers

{
    /// <summary>
    /// Controller for notification management operations
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    [Authorize]
    public class NotificationsController : ControllerBase
    {
        private readonly INotificationService _notificationService;
        private readonly ILogger<NotificationsController> _logger;
        /// <summary>
        /// Initializes notifications controller with dependencies
        /// </summary>
        /// <param name="notificationService">Notification service</param>
        /// <param name="logger">Logger</param>
        public NotificationsController(
            INotificationService notificationService,
            ILogger<NotificationsController> logger)
        {
            _notificationService = notificationService;
            _logger = logger;
        }
        /// <summary>
```

```

/// Creates a notification (Backoffice and Station Operators only)

/// </summary>

/// <param name="createDto">Notification creation data</param>

/// <returns>Created notification</returns>

[HttpPost]

[Authorize(Roles = $"{nameof(UserRole.Backoffice)},{nameof(UserRole.StationOperator)}")]

public async Task<ActionResult<NotificationResponseDto>> CreateNotification([FromBody]
CreateNotificationDto createDto)

{

    _logger.LogInformation("Creating notification for user {RecipientNIC}", createDto.RecipientNIC);

    var notification = await _notificationService.CreateNotificationAsync(createDto);

    return CreatedAtAction(nameof(GetNotification), new { id = notification.Id }, notification);

}

/// <summary>

/// Creates bulk notifications (Backoffice only)

/// </summary>

/// <param name="bulkDto">Bulk notification data</param>

/// <returns>List of created notifications</returns>

[HttpPost("bulk")]

[Authorize(Roles = nameof(UserRole.Backoffice))]

public async Task<ActionResult<List<NotificationResponseDto>>>
CreateBulkNotification([FromBody] BulkNotificationDto bulkDto)

{

    _logger.LogInformation("Creating bulk notifications for {Count} recipients",
bulkDto.RecipientNICs.Count);

    var notifications = await _notificationService.CreateBulkNotificationAsync(bulkDto);

```

```

        return Ok(notifications);

    }

/// <summary>
/// Gets current user's notifications
/// </summary>

/// <param name="includeRead">Include read notifications</param>
/// <param name="limit">Maximum number of notifications</param>
/// <param name="offset">Number of notifications to skip</param>
/// <returns>List of user's notifications</returns>

[HttpGet("my-notifications")]

public async Task<ActionResult<List<NotificationResponseDto>>> GetMyNotifications(
    [FromQuery] bool includeRead = true,
    [FromQuery] int limit = 50,
    [FromQuery] int offset = 0)

{
    var userNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(userNic))
    {
        return BadRequest("Invalid user token - NIC not found");
    }

    _logger.LogInformation("Fetching notifications for user: {UserNIC}", userNic);

    var notifications = await _notificationService.GetUserNotificationsAsync(
        userNic, includeRead, limit, offset);

    return Ok(notifications);
}

```

```

/// <summary>
/// Gets current user's unread notifications
/// </summary>
/// <returns>List of unread notifications</returns>
[HttpGet("unread")]

public async Task<ActionResult<List<NotificationResponseDto>>> GetUnreadNotifications()

{
    var userNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(userNic))
    {
        return BadRequest("Invalid user token - NIC not found");
    }

    _logger.LogInformation("Fetching unread notifications for user: {UserNIC}", userNic);

    var notifications = await _notificationService.GetUnreadNotificationsAsync(userNic);

    return Ok(notifications);
}

/// <summary>
/// Gets notifications by type for current user
/// </summary>
/// <param name="type">Notification type</param>
/// <param name="limit">Maximum number of notifications</param>
/// <returns>List of notifications by type</returns>
[HttpGet("by-type/{type}")]
public async Task<ActionResult<List<NotificationResponseDto>>> GetNotificationsByType(
    NotificationType type,

```

```

[FromQuery] int limit = 20)

{

    var userNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(userNic))

    {

        return BadRequest("Invalid user token - NIC not found");

    }

    _logger.LogInformation("Fetching {Type} notifications for user: {UserNIC}", type, userNic);

    var notifications = await _notificationService.GetNotificationsByTypeAsync(

        userNic, type, limit);

    return Ok(notifications);

}

/// <summary>

/// Gets notification by ID

/// </summary>

/// <param name="id">Notification ID</param>

/// <returns>Notification details</returns>

[HttpGet("{id}")]

public async Task<ActionResult<NotificationResponseDto>> GetNotification(string id)

{

    var userNic = User.FindFirst("nic")?.Value;

    var userRole = User.FindFirst(ClaimTypes.Role)?.Value;

    if (string.IsNullOrEmpty(userNic))

    {

```

```

        return BadRequest("Invalid user token - NIC not found");

    }

    _logger.LogInformation("Fetching notification: {NotificationId}", id);

    var notification = await _notificationService.GetNotificationByIdAsync(id);

    // Users can only view their own notifications (except backoffice)
    if (userRole != nameof(UserRole.Backoffice) && notification.RecipientNIC != userNic)
    {
        return Forbid("You can only view your own notifications");
    }

    return Ok(notification);
}

/// <summary>
/// Marks notifications as read
/// </summary>
/// <param name="markReadDto">Notification IDs to mark as read</param>
/// <returns>Success status with count</returns>
[HttpPost("mark-read")]

public async Task<ActionResult> MarkNotificationsAsRead([FromBody] MarkNotificationReadDto
markReadDto)

{
    var userNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(userNic))
    {
        return BadRequest("Invalid user token - NIC not found");
    }
}

```

```

    }

    _logger.LogInformation("Marking {Count} notifications as read for user: {UserNIC}",
        markReadDto.NotificationIds.Count, userNic);

    var markedCount = await _notificationService.MarkNotificationsAsReadAsync(
        markReadDto.NotificationIds, userNic);

    return Ok(new { message = $"{markedCount} notifications marked as read", count = markedCount
    });
}

/// <summary>
/// Marks all notifications as read for current user
/// </summary>
/// <returns>Success status with count</returns>
[HttpPost("mark-all-read")]

public async Task<ActionResult> MarkAllNotificationsAsRead()
{
    var userNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(userNic))
    {
        return BadRequest("Invalid user token - NIC not found");
    }

    _logger.LogInformation("Marking all notifications as read for user: {UserNIC}", userNic);

    var markedCount = await _notificationService.MarkAllAsReadAsync(userNic);
}

```

```

        return Ok(new { message = $"All {markedCount} notifications marked as read", count =
markedCount });

    }

/// <summary>
/// Deletes a notification
/// </summary>

/// <param name="id">Notification ID</param>
/// <returns>Success status</returns>

[HttpDelete("{id}")]
public async Task<ActionResult> DeleteNotification(string id)

{
    var userNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(userNic))
    {

        return BadRequest("Invalid user token - NIC not found");
    }

    _logger.LogInformation("Deleting notification: {NotificationId} for user: {UserNIC}", id, userNic);

    var result = await _notificationService.DeleteNotificationAsync(id, userNic);

    if (!result)
    {

        return NotFound("Notification not found");
    }

    return Ok(new { message = "Notification deleted successfully" });
}

```

```

/// <summary>
/// Gets notification summary for current user
/// </summary>
/// <returns>Notification summary</returns>
[HttpGet("summary")]

public async Task<ActionResult<NotificationSummaryDto>> GetNotificationSummary()

{
    var userNic = User.FindFirst("nic")?.Value;

    if (string.IsNullOrEmpty(userNic))
    {
        return BadRequest("Invalid user token - NIC not found");
    }

    _logger.LogInformation("Fetching notification summary for user: {UserNIC}", userNic);

    var summary = await _notificationService.GetNotificationSummaryAsync(userNic);

    return Ok(summary);
}

/// <summary>
/// Cleans up expired notifications (Backoffice only)
/// </summary>
/// <returns>Number of deleted notifications</returns>
[HttpPost("cleanup-expired")]
[Authorize(Roles = nameof(UserRole.Backoffice))]

public async Task<ActionResult> CleanupExpiredNotifications()

{
    _logger.LogInformation("Cleaning up expired notifications");
}

```

```

        var deletedCount = await _notificationService.CleanupExpiredNotificationsAsync();

        return Ok(new { message = $"Cleaned up {deletedCount} expired notifications", count =
    deletedCount });
}

}

}

/*
 * File: UsersController.cs
 *
 * Project: EV Charging Station Booking System
 *
 * Description: User management controller with role-based authorization
 *
 * Author: EV Charging System
 *
 * Date: September 27, 2025
 */

using EVChargingBackend.DTOs;
using EVChargingBackend.Models;
using EVChargingBackend.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Security.Claims;

namespace EVChargingBackend.Controllers
{
    /// <summary>
    /// Controller for user management operations with role-based access control
    /// </summary>

```

```

[ApiController]
[Route("api/[controller]")]
[Authorize]

public class UsersController : ControllerBase
{
    private readonly IUserService _userService;
    private readonly ILogger<UsersController> _logger;

    /// <summary>
    /// Initializes users controller with dependencies
    /// </summary>
    /// <param name="userService">User service for business operations</param>
    /// <param name="logger">Logger for controller operations</param>

    public UsersController(IUserService userService, ILogger<UsersController> logger)
    {
        _userService = userService;
        _logger = logger;
    }

    /// <summary>
    /// Gets all users (backoffice only)
    /// </summary>
    /// <returns>List of all users</returns>

    [HttpGet]
    [Authorize(Roles = nameof(UserRole.Backoffice))]

    public async Task<ActionResult<List<UserResponseDto>>> GetAllUsers()
    {
        _logger.LogInformation("Fetching all users");

        var users = await _userService.GetAllUsersAsync();
    }
}

```

```

        return Ok(users);
    }

    /// <summary>
    /// Gets user by ID
    /// </summary>

    /// <param name="id">User ID</param>
    /// <returns>User information</returns>

    [HttpGet("{id}")]
    public async Task<ActionResult<UserResponseDto>> GetUser(string id)
    {
        var currentUserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
        var currentUserRole = User.FindFirst(ClaimTypes.Role)?.Value;

        // Users can only view their own profile unless they are backoffice
        if (currentUserRole != nameof(UserRole.Backoffice) && currentUserId != id)
        {
            return Forbid("You can only view your own profile");
        }

        _logger.LogInformation("Fetching user: {UserId}", id);

        var user = await _userService.GetUserByIdAsync(id);

        return Ok(user);
    }

    /// <summary>
    /// Updates user information
    /// </summary>

    /// <param name="id">User ID</param>

```

```

/// <param name="updateDto">Updated user data</param>
/// <returns>Updated user information</returns>
[HttpPut("{id}")]
public async Task<ActionResult<UserResponseDto>> UpdateUser(string id, [FromBody]
UpdateUserDto updateDto)

{
    var currentUserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;
    var currentUserRole = User.FindFirst(ClaimTypes.Role)?.Value;

    // Users can only update their own profile unless they are backoffice
    if (currentUserRole != nameof(UserRole.Backoffice) && currentUserId != id)
    {
        return Forbid("You can only update your own profile");
    }

    _logger.LogInformation("Updating user: {UserId}", id);

    var user = await _userService.UpdateUserAsync(id, updateDto);

    return Ok(user);
}

/// <summary>
/// Activates or deactivates a user (backoffice only)
/// </summary>
/// <param name="id">User ID</param>
/// <param name="isActive">Active status</param>
/// <returns>Success status</returns>
[HttpPatch("{id}/status")]
[Authorize(Roles = nameof(UserRole.Backoffice))]

```

```

public async Task<ActionResult> SetUserActiveStatus(string id, [FromBody] bool isActive)
{
    _logger.LogInformation("Setting user {UserId} active status to {Status}", id, isActive);

    var result = await _userService.SetUserActiveStatusAsync(id, isActive);

    if (!result)
    {
        return NotFound("User not found");
    }

    return Ok(new { message = $"User {(isActive ? "activated" : "deactivated")}" successfully });
}

/// <summary>
/// Deactivates current user (EV Owner self-deactivation)
/// </summary>
/// <returns>Success status</returns>

[HttpPost("deactivate")]
[Authorize(Roles = nameof(UserRole.EVOwner))]

public async Task<ActionResult> DeactivateCurrentUser()
{
    var currentUserId = User.FindFirst(ClaimTypes.NameIdentifier)?.Value;

    if (string.IsNullOrEmpty(currentUserId))
    {
        return BadRequest("Invalid user token");
    }

    _logger.LogInformation("User self-deactivation: {UserId}", currentUserId);
}

```

```

var result = await _userService.DeactivateUserAsync(currentUserId);

if (!result)
{
    return NotFound("User not found");
}

return Ok(new { message = "Account deactivated successfully" });
}

/// <summary>
/// Deletes a user (backoffice only)
/// </summary>
/// <param name="id">User ID</param>
/// <returns>Success status</returns>
[HttpDelete("{id}")]
[Authorize(Roles = nameof(UserRole.Backoffice))]
public async Task<ActionResult> DeleteUser(string id)
{
    _logger.LogInformation("Deleting user: {UserId}", id);

    var result = await _userService.DeleteUserAsync(id);

    if (!result)
    {
        return NotFound("User not found");
    }

    return Ok(new { message = "User deleted successfully" });
}

```

```

        }

    }

}

/*
 * File: Booking.cs
 *
 * Project: EV Charging Station Booking System
 *
 * Description: Booking model with QR code generation and business rules validation
 *
 * Author: EV Charging System
 *
 * Date: September 27, 2025
 */


```

```

using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace EVChargingBackend.Models

{
    /// <summary>
    /// Represents a charging station booking with QR code and time restrictions
    /// </summary>
    public class Booking
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string? Id { get; set; }

        [BsonElement("ownerNIC")]
        public string OwnerNIC { get; set; } = string.Empty;
    }
}
```

```
[BsonElement("stationId")]

public string StationId { get; set; } = string.Empty;

[BsonElement("startTime")]

public DateTime StartTime { get; set; }

[BsonElement("endTime")]

public DateTime EndTime { get; set; }

[BsonElement("status")]

public BookingStatus Status { get; set; } = BookingStatus.Active;

[BsonElement("qrCode")]

public string QRCode { get; set; } = string.Empty;

[BsonElement("totalAmount")]

public decimal TotalAmount { get; set; }

[BsonElement("createdAt")]

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

[BsonElement("updatedAt")]

public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;

[BsonElement("confirmedAt")]

public DateTime? ConfirmedAt { get; set; }

[BsonElement("cancelledAt")]

public DateTime? CancelledAt { get; set; }
```

```

    }

/// <summary>
/// Status enumeration for bookings
/// </summary>

public enum BookingStatus
{
    Active = 0,
    Confirmed = 1,
    Completed = 2,
    Cancelled = 3,
    NoShow = 4
}

/*
 * File: ChargingStation.cs
 * Project: EV Charging Station Booking System
 * Description: Charging station model with AC/DC types and slot management
 * Author: EV Charging System
 * Date: September 27, 2025
 */

using MongoDB.Bson;

using MongoDB.Bson.Serialization.Attributes;

namespace EVChargingBackend.Models
{
    /// <summary>
    /// Represents a charging station with its properties and availability

```

```
/// </summary>

public class ChargingStation
{
    [BsonId]
    [BsonRepresentation(BsonType.ObjectId)]
    public string? Id { get; set; }

    [BsonElement("name")]
    public string Name { get; set; } = string.Empty;

    [BsonElement("location")]
    public string Location { get; set; } = string.Empty;

    [BsonElement("type")]
    public StationType Type { get; set; }

    [BsonElement("totalSlots")]
    public int TotalSlots { get; set; }

    [BsonElement("availableSlots")]
    public int AvailableSlots { get; set; }

    [BsonElement("status")]
    public StationStatus Status { get; set; } = StationStatus.Active;

    [BsonElement("pricePerHour")]
    public decimal PricePerHour { get; set; }

    [BsonElement("createdAt")]
}
```

```

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

[BsonElement("updatedAt")]

public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;

}

/// <summary>

/// Types of charging stations (AC or DC)

/// </summary>

public enum StationType

{

    AC = 0,

    DC = 1

}

/// <summary>

/// Status of charging stations

/// </summary>

public enum StationStatus

{

    Active = 0,

    Inactive = 1,

    Maintenance = 2

}

}

/* 

* File: Notification.cs

* Project: EV Charging Station Booking System

* Description: Notification model for system notifications

* Author: EV Charging System

```

\* Date: October 7, 2025

\*/

```
using MongoDB.Bson;
```

```
using MongoDB.Bson.Serialization.Attributes;
```

```
namespace EVChargingBackend.Models
```

```
{
```

```
/// <summary>
```

```
/// Represents a notification in the system
```

```
/// </summary>
```

```
public class Notification
```

```
{
```

```
[BsonId]
```

```
[BsonRepresentation(BsonType.ObjectId)]
```

```
public string? Id { get; set; }
```

```
[BsonElement("recipientNIC")]
```

```
public string RecipientNIC { get; set; } = string.Empty;
```

```
[BsonElement("title")]
```

```
public string Title { get; set; } = string.Empty;
```

```
[BsonElement("message")]
```

```
public string Message { get; set; } = string.Empty;
```

```
[BsonElement("type")]
```

```
public NotificationType Type { get; set; }
```

```
[BsonElement("relatedEntityId")]

public string? RelatedEntityId { get; set; }

[BsonElement("relatedEntityType")]

public string? RelatedEntityType { get; set; }

[BsonElement("isRead")]

public bool IsRead { get; set; } = false;

[BsonElement("isDelivered")]

public bool IsDelivered { get; set; } = false;

[BsonElement("priority")]

public NotificationPriority Priority { get; set; } = NotificationPriority.Normal;

[BsonElement("createdAt")]

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

[BsonElement("readAt")]

public DateTime? ReadAt { get; set; }

[BsonElement("deliveredAt")]

public DateTime? DeliveredAt { get; set; }

[BsonElement("expiresAt")]

public DateTime? ExpiresAt { get; set; }

[BsonElement("metadata")]

public Dictionary<string, object>? Metadata { get; set; }
```

```
}

/// <summary>

/// Types of notifications in the system

/// </summary>

public enum NotificationType

{

    BookingConfirmation = 0,

    BookingCancellation = 1,

    BookingReminder = 2,

    StationUpdate = 3,

    SystemAlert = 4,

    PaymentConfirmation = 5,

    BookingExpired = 6

}

/// <summary>

/// Priority levels for notifications

/// </summary>

public enum NotificationPriority

{

    Low = 0,

    Normal = 1,

    High = 2,

    Critical = 3

}

/*
 * File: User.cs
 *
 * Project: EV Charging Station Booking System

```

```
* Description: User model representing different user roles (Backoffice, Station Operator, EV Owner)
* Author: EV Charging System
* Date: September 27, 2025
*/

```

```
using MongoDB.Bson;
using MongoDB.Bson.Serialization.Attributes;

namespace EVChargingBackend.Models

{
    /// <summary>
    /// Represents a user in the EV Charging system with role-based access
    /// </summary>
    public class User
    {
        [BsonId]
        [BsonRepresentation(BsonType.ObjectId)]
        public string? Id { get; set; }

        [BsonElement("nic")]
        public string NIC { get; set; } = string.Empty;

        [BsonElement("firstName")]
        public string FirstName { get; set; } = string.Empty;

        [BsonElement("lastName")]
        public string LastName { get; set; } = string.Empty;

        [BsonElement("email")]
        public string Email { get; set; } = string.Empty;
    }
}
```

```

[BsonElement("password")]

public string Password { get; set; } = string.Empty;

[BsonElement("role")]

public UserRole Role { get; set; }

[BsonElement("stationId")]

[BsonRepresentation(BsonType.ObjectId)]

public string? StationId { get; set; }

[BsonElement("isActive")]

public bool IsActive { get; set; } = true;

[BsonElement("phoneNumber")]

public string? PhoneNumber { get; set; }

[BsonElement("createdAt")]

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

[BsonElement("updatedAt")]

public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;

}

/// <summary>

/// Enumeration of user roles in the system

/// </summary>

public enum UserRole

{

    Backoffice = 0,

```

```

        StationOperator = 1,
        EVOwner = 2
    }

}

/*
 * File: BookingRepository.cs
 *
 * Project: EV Charging Station Booking System
 *
 * Description: Repository implementation for Booking operations
 *
 * Author: EV Charging System
 *
 * Date: September 27, 2025
 */

```

```

using EVChargingBackend.Config;
using EVChargingBackend.Models;
using MongoDB.Driver;

namespace EVChargingBackend.Repositories
{
    /// <summary>
    /// Repository implementation for Booking data operations
    /// </summary>
    public class BookingRepository : IBookingRepository
    {
        private readonly IMongoCollection<Booking> _bookings;

        /// <summary>
        /// Initializes booking repository with MongoDB context
        /// </summary>

```

```
/// <param name="context">MongoDB database context</param>
public BookingRepository(MongoDbContext context)
{
    _bookings = context.Bookings;
}

/// <summary>
/// Creates a new booking
/// </summary>
/// <param name="booking">Booking to create</param>
/// <returns>Created booking</returns>
public async Task<Booking> CreateAsync(Booking booking)
{
    booking.CreatedAt = DateTime.UtcNow;
    booking.UpdatedAt = DateTime.UtcNow;
    await _bookings.InsertOneAsync(booking);
    return booking;
}

/// <summary>
/// Gets booking by ID
/// </summary>
/// <param name="id">Booking ID</param>
/// <returns>Booking if found, null otherwise</returns>
public async Task<Booking?> GetByIdAsync(string id)
{
    return await _bookings.Find(b => b.Id == id).FirstOrDefaultAsync();
}

/// <summary>
/// Gets all bookings for a specific user

```

```

/// </summary>

/// <param name="ownerNic">Owner NIC</param>

/// <returns>List of user bookings</returns>

public async Task<List<Booking>> GetByOwnerAsync(string ownerNic)

{

    return await _bookings.Find(b => b.OwnerNIC == ownerNic)

        .SortByDescending(b => b.CreatedAt)

        .ToListAsync();

}

/// <summary>

/// Gets all bookings for a specific station

/// </summary>

/// <param name="stationId">Station ID</param>

/// <returns>List of station bookings</returns>

public async Task<List<Booking>> GetByStationAsync(string stationId)

{

    return await _bookings.Find(b => b.StationId == stationId)

        .SortByDescending(b => b.StartTime)

        .ToListAsync();

}

/// <summary>

/// Gets active bookings for a station within a time range

/// </summary>

/// <param name="stationId">Station ID</param>

/// <param name="startTime">Start time</param>

/// <param name="endTime">End time</param>

/// <returns>List of overlapping bookings</returns>

public async Task<List<Booking>> GetOverlappingBookingsAsync(string stationId, DateTime

startTime, DateTime endTime)

```

```

{
    var filter = Builders<Booking>.Filter.And(
        Builders<Booking>.Filter.Eq(b => b.StationId, stationId),
        Builders<Booking>.Filter.In(b => b.Status, new[] { BookingStatus.Active,
BookingStatus.Confirmed })),
    Builders<Booking>.Filter.Or(
        // New booking starts during existing booking
        Builders<Booking>.Filter.And(
            Builders<Booking>.Filter.Lte(b => b.StartTime, startTime),
            Builders<Booking>.Filter.Gt(b => b.EndTime, startTime)
        ),
        // New booking ends during existing booking
        Builders<Booking>.Filter.And(
            Builders<Booking>.Filter.Lt(b => b.StartTime, endTime),
            Builders<Booking>.Filter.Gte(b => b.EndTime, endTime)
        ),
        // New booking completely contains existing booking
        Builders<Booking>.Filter.And(
            Builders<Booking>.Filter.Gte(b => b.StartTime, startTime),
            Builders<Booking>.Filter.Lte(b => b.EndTime, endTime)
        )
    );
}

return await _bookings.Find(filter).ToListAsync();
}

/// <summary>
/// Gets all bookings

```

```

/// </summary>

/// <returns>List of all bookings</returns>

public async Task<List<Booking>> GetAllAsync()

{

    return await _bookings.Find(_ => true)

        .SortByDescending(b => b.CreatedAt)

        .ToListAsync();

}

/// <summary>

/// Updates booking information

/// </summary>

/// <param name="booking">Booking with updated information</param>

/// <returns>Updated booking</returns>

public async Task<Booking> UpdateAsync(Booking booking)

{

    booking.UpdatedAt = DateTime.UtcNow;

    await _bookings.ReplaceOneAsync(b => b.Id == booking.Id, booking);

    return booking;

}

/// <summary>

/// Confirms a booking (operator action)

/// </summary>

/// <param name="bookingId">Booking ID</param>

/// <returns>Success status</returns>

public async Task<bool> ConfirmBookingAsync(string bookingId)

{

    var update = Builders<Booking>.Update

        .Set(b => b.Status, BookingStatus.Confirmed)

```

```

    .Set(b => b.ConfirmedAt, DateTime.UtcNow)
    .Set(b => b.UpdatedAt, DateTime.UtcNow);

    var result = await _bookings.UpdateOneAsync(b => b.Id == bookingId, update);
    return result.ModifiedCount > 0;
}

/// <summary>
/// Cancels a booking
/// </summary>
/// <param name="bookingId">Booking ID</param>
/// <returns>Success status</returns>

public async Task<bool> CancelBookingAsync(string bookingId)
{
    var update = Builders<Booking>.Update
        .Set(b => b.Status, BookingStatus.Cancelled)
        .Set(b => b.CancelledAt, DateTime.UtcNow)
        .Set(b => b.UpdatedAt, DateTime.UtcNow);

    var result = await _bookings.UpdateOneAsync(b => b.Id == bookingId, update);
    return result.ModifiedCount > 0;
}

/// <summary>
/// Gets active bookings for a station (for slot management)
/// </summary>
/// <param name="stationId">Station ID</param>
/// <returns>Count of active bookings</returns>

public async Task<long> GetActiveBookingCountAsync(string stationId)
{

```

```

var filter = Builders<Booking>.Filter.And(
    Builders<Booking>.Filter.Eq(b => b.StationId, stationId),
    Builders<Booking>.Filter.In(b => b.Status, new[] { BookingStatus.Active,
    BookingStatus.Confirmed })
);

return await _bookings.CountDocumentsAsync(filter);
}

/// <summary>
/// Gets upcoming confirmed bookings within a time range (for reminders)
/// </summary>
/// <param name="fromTime">Start time range</param>
/// <param name="toTime">End time range</param>
/// <returns>List of upcoming confirmed bookings</returns>

public async Task<List<Booking>> GetUpcomingConfirmedBookingsAsync(DateTime fromTime,
DateTime toTime)
{
    var filter = Builders<Booking>.Filter.And(
        Builders<Booking>.Filter.Eq(b => b.Status, BookingStatus.Confirmed),
        Builders<Booking>.Filter.Gte(b => b.StartTime, fromTime),
        Builders<Booking>.Filter.Lte(b => b.StartTime, toTime)
    );

    return await _bookings.Find(filter).ToListAsync();
}
}

/*

```

```
* File: ChargingStationRepository.cs  
* Project: EV Charging Station Booking System  
* Description: Repository implementation for Charging Station operations  
* Author: EV Charging System  
* Date: September 27, 2025  
*/
```

```
using EVChargingBackend.Config;  
using EVChargingBackend.Models;  
using MongoDB.Driver;  
  
namespace EVChargingBackend.Repositories  
{  
    /// <summary>  
    /// Repository implementation for Charging Station data operations  
    /// </summary>  
    public class ChargingStationRepository : IChargingStationRepository  
    {  
        private readonly IMongoCollection<ChargingStation> _stations;  
  
        /// <summary>  
        /// Initializes charging station repository with MongoDB context  
        /// </summary>  
        /// <param name="context">MongoDB database context</param>  
        public ChargingStationRepository(MongoDbContext context)  
        {  
            _stations = context.ChargingStations;  
        }  
    }
```

```

/// <summary>
/// Creates a new charging station
/// </summary>
/// <param name="station">Charging station to create</param>
/// <returns>Created charging station</returns>
public async Task<ChargingStation> CreateAsync(ChargingStation station)

{
    station.CreatedAt = DateTime.UtcNow;
    station.UpdatedAt = DateTime.UtcNow;
    station.AvailableSlots = station.TotalSlots; // Initially all slots are available
    await _stations.InsertOneAsync(station);
    return station;
}

/// <summary>
/// Gets charging station by ID
/// </summary>
/// <param name="id">Station ID</param>
/// <returns>Charging station if found, null otherwise</returns>
public async Task<ChargingStation?> GetByIdAsync(string id)

{
    return await _stations.Find(s => s.Id == id).FirstOrDefaultAsync();
}

/// <summary>
/// Gets all charging stations
/// </summary>
/// <returns>List of all charging stations</returns>
public async Task<List<ChargingStation>> GetAllAsync()

```

```

{
    return await _stations.Find(_ => true).ToListAsync();
}

/// <summary>
/// Gets active charging stations only
/// </summary>
/// <returns>List of active charging stations</returns>
public async Task<List<ChargingStation>> GetActiveStationsAsync()

{
    return await _stations.Find(s => s.Status == StationStatus.Active).ToListAsync();
}

/// <summary>
/// Updates charging station information
/// </summary>
/// <param name="station">Station with updated information</param>
/// <returns>Updated charging station</returns>
public async Task<ChargingStation> UpdateAsync(ChargingStation station)

{
    station.UpdatedAt = DateTime.UtcNow;

    await _stations.ReplaceOneAsync(s => s.Id == station.Id, station);

    return station;
}

/// <summary>
/// Updates available slots for a station
/// </summary>
/// <param name="stationId">Station ID</param>

```

```

/// <param name="slotsChange">Change in available slots (positive or negative)</param>
/// <returns>Success status</returns>

public async Task<bool> UpdateAvailableSlotsAsync(string stationId, int slotsChange)
{
    var update = Builders<ChargingStation>.Update
        .Inc(s => s.AvailableSlots, slotsChange)
        .Set(s => s.UpdatedAt, DateTime.UtcNow);

    var result = await _stations.UpdateOneAsync(s => s.Id == stationId, update);
    return result.ModifiedCount > 0;
}

/// <summary>
/// Deletes a charging station
/// </summary>

/// <param name="id">Station ID</param>
/// <returns>Success status</returns>

public async Task<bool> DeleteAsync(string id)
{
    var result = await _stations.DeleteOneAsync(s => s.Id == id);
    return result.DeletedCount > 0;
}

}

/*
 * File: IBookingRepository.cs
 * Project: EV Charging Station Booking System
 * Description: Interface for Booking repository operations

```

\* Author: EV Charging System

\* Date: September 27, 2025

\*/

```
using EVChargingBackend.Models;
```

```
namespace EVChargingBackend.Repositories
```

```
{
```

```
/// <summary>
```

```
/// Interface defining booking repository operations
```

```
/// </summary>
```

```
public interface IBookingRepository
```

```
{
```

```
/// <summary>
```

```
/// Creates a new booking
```

```
/// </summary>
```

```
/// <param name="booking">Booking to create</param>
```

```
/// <returns>Created booking</returns>
```

```
Task<Booking> CreateAsync(Booking booking);
```

```
/// <summary>
```

```
/// Gets booking by ID
```

```
/// </summary>
```

```
/// <param name="id">Booking ID</param>
```

```
/// <returns>Booking if found, null otherwise</returns>
```

```
Task<Booking?> GetByIdAsync(string id);
```

```
/// <summary>
```

```
/// Gets all bookings for a specific user
```

```
/// </summary>

/// <param name="ownerNic">Owner NIC</param>
/// <returns>List of user bookings</returns>
Task<List<Booking>> GetByOwnerAsync(string ownerNic);

/// <summary>
/// Gets all bookings for a specific station
/// </summary>
/// <param name="stationId">Station ID</param>
/// <returns>List of station bookings</returns>
Task<List<Booking>> GetByStationAsync(string stationId);

/// <summary>
/// Gets active bookings for a station within a time range
/// </summary>
/// <param name="stationId">Station ID</param>
/// <param name="startTime">Start time</param>
/// <param name="endTime">End time</param>
/// <returns>List of overlapping bookings</returns>
Task<List<Booking>> GetOverlappingBookingsAsync(string stationId, DateTime startTime, DateTime endTime);

/// <summary>
/// Gets all bookings
/// </summary>
/// <returns>List of all bookings</returns>
Task<List<Booking>> GetAllAsync();

/// <summary>
```

```
/// Updates booking information
/// </summary>
/// <param name="booking">Booking with updated information</param>
/// <returns>Updated booking</returns>
Task<Booking> UpdateAsync(Booking booking);

/// <summary>
/// Confirms a booking (operator action)
/// </summary>
/// <param name="bookingId">Booking ID</param>
/// <returns>Success status</returns>
Task<bool> ConfirmBookingAsync(string bookingId);

/// <summary>
/// Cancels a booking
/// </summary>
/// <param name="bookingId">Booking ID</param>
/// <returns>Success status</returns>
Task<bool> CancelBookingAsync(string bookingId);

/// <summary>
/// Gets active bookings for a station (for slot management)
/// </summary>
/// <param name="stationId">Station ID</param>
/// <returns>Count of active bookings</returns>
Task<long> GetActiveBookingCountAsync(string stationId);

/// <summary>
/// Gets upcoming confirmed bookings within a time range (for reminders)
```

```

/// </summary>

/// <param name="fromTime">Start time range</param>
/// <param name="toTime">End time range</param>
/// <returns>List of upcoming confirmed bookings</returns>
Task<List<Booking>> GetUpcomingConfirmedBookingsAsync(DateTime fromTime, DateTime toTime);

}

}

/*
* File: IChargingStationRepository.cs
* Project: EV Charging Station Booking System
* Description: Interface for Charging Station repository operations
* Author: EV Charging System
* Date: September 27, 2025
*/
using EVChargingBackend.Models;

namespace EVChargingBackend.Repositories
{
    /// <summary>
    /// Interface defining charging station repository operations
    /// </summary>
    public interface IChargingStationRepository
    {
        /// <summary>
        /// Creates a new charging station
        /// </summary>
        /// <param name="station">Charging station to create</param>

```

```
/// <returns>Created charging station</returns>
Task<ChargingStation> CreateAsync(ChargingStation station);

/// <summary>
/// Gets charging station by ID
/// </summary>
/// <param name="id">Station ID</param>
/// <returns>Charging station if found, null otherwise</returns>
Task<ChargingStation?> GetByIdAsync(string id);

/// <summary>
/// Gets all charging stations
/// </summary>
/// <returns>List of all charging stations</returns>
Task<List<ChargingStation>> GetAllAsync();

/// <summary>
/// Gets active charging stations only
/// </summary>
/// <returns>List of active charging stations</returns>
Task<List<ChargingStation>> GetActiveStationsAsync();

/// <summary>
/// Updates charging station information
/// </summary>
/// <param name="station">Station with updated information</param>
/// <returns>Updated charging station</returns>
Task<ChargingStation> UpdateAsync(ChargingStation station);
```

```

/// <summary>
/// Updates available slots for a station
/// </summary>
/// <param name="stationId">Station ID</param>
/// <param name="slotsChange">Change in available slots (positive or negative)</param>
/// <returns>Success status</returns>
Task<bool> UpdateAvailableSlotsAsync(string stationId, int slotsChange);

/// <summary>
/// Deletes a charging station
/// </summary>
/// <param name="id">Station ID</param>
/// <returns>Success status</returns>
Task<bool> DeleteAsync(string id);

}

}

/*
* File: INotificationRepository.cs
* Project: EV Charging Station Booking System
* Description: Interface for notification repository operations
* Author: EV Charging System
* Date: October 7, 2025
*/
using EVChargingBackend.Models;

namespace EVChargingBackend.Repositories
{

```

```
/// <summary>
/// Interface for notification repository operations
/// </summary>

public interface INotificationRepository
{
    /// <summary>
    /// Creates a new notification
    /// </summary>
    /// <param name="notification">Notification to create</param>
    /// <returns>Created notification</returns>
    Task<Notification> CreateAsync(Notification notification);

    /// <summary>
    /// Creates multiple notifications at once
    /// </summary>
    /// <param name="notifications">List of notifications to create</param>
    /// <returns>List of created notifications</returns>
    Task<List<Notification>> CreateManyAsync(List<Notification> notifications);

    /// <summary>
    /// Gets notification by ID
    /// </summary>
    /// <param name="id">Notification ID</param>
    /// <returns>Notification if found</returns>
    Task<Notification?> GetByIdAsync(string id);

    /// <summary>
    /// Gets all notifications for a specific user
    /// </summary>
```

```

/// <param name="recipientNIC">User NIC</param>
/// <param name="includeRead">Include read notifications</param>
/// <param name="limit">Maximum number of notifications</param>
/// <param name="offset">Number of notifications to skip</param>
/// <returns>List of notifications</returns>

Task<List<Notification>> GetByRecipientAsync(string recipientNIC, bool includeRead = true, int limit
= 50, int offset = 0);

/// <summary>
/// Gets unread notifications for a specific user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <returns>List of unread notifications</returns>

Task<List<Notification>> GetUnreadByRecipientAsync(string recipientNIC);

/// <summary>
/// Gets notifications by type for a specific user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <param name="type">Notification type</param>
/// <param name="limit">Maximum number of notifications</param>
/// <returns>List of notifications</returns>

Task<List<Notification>> GetByRecipientAndTypeAsync(string recipientNIC, NotificationType type, int
limit = 20);

/// <summary>
/// Gets notifications related to a specific entity
/// </summary>
/// <param name="entityId">Related entity ID</param>
/// <param name="entityType">Related entity type</param>

```

```
/// <returns>List of notifications</returns>

Task<List<Notification>> GetByRelatedEntityAsync(string entityId, string entityType);

/// <summary>
/// Marks a notification as read
/// </summary>

/// <param name="id">Notification ID</param>
/// <returns>Success status</returns>

Task<bool> MarkAsReadAsync(string id);

/// <summary>
/// Marks multiple notifications as read
/// </summary>

/// <param name="ids">List of notification IDs</param>
/// <returns>Number of notifications marked as read</returns>

Task<int> MarkMultipleAsReadAsync(List<string> ids);

/// <summary>
/// Marks all notifications for a user as read
/// </summary>

/// <param name="recipientNIC">User NIC</param>
/// <returns>Number of notifications marked as read</returns>

Task<int> MarkAllAsReadAsync(string recipientNIC);

/// <summary>
/// Marks a notification as delivered
/// </summary>

/// <param name="id">Notification ID</param>
/// <returns>Success status</returns>
```

```
Task<bool> MarkAsDeliveredAsync(string id);

/// <summary>
/// Deletes a notification
/// </summary>
/// <param name="id">Notification ID</param>
/// <returns>Success status</returns>

Task<bool> DeleteAsync(string id);

/// <summary>
/// Deletes expired notifications
/// </summary>
/// <returns>Number of deleted notifications</returns>

Task<int> DeleteExpiredNotificationsAsync();

/// <summary>
/// Gets notification count for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <param name="includeRead">Include read notifications in count</param>
/// <returns>Notification count</returns>

Task<int> GetNotificationCountAsync(string recipientNIC, bool includeRead = true);

/// <summary>
/// Gets unread notification count for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <returns>Unread notification count</returns>

Task<int> GetUnreadCountAsync(string recipientNIC);
```

```
    }

}

/*
 * File: IUserRepository.cs
 * Project: EV Charging Station Booking System
 * Description: Interface for User repository operations
 * Author: EV Charging System
 * Date: September 27, 2025
 */
```

```
using EVChargingBackend.Models;
```

```
namespace EVChargingBackend.Repositories

{
    /// <summary>
    /// Interface defining user repository operations
    /// </summary>

    public interface IUserRepository
    {
        /// <summary>
        /// Creates a new user in the database
        /// </summary>
        /// <param name="user">User entity to create</param>
        /// <returns>Created user</returns>
        Task<User> CreateAsync(User user);

        /// <summary>
        /// Gets user by email address
    }
}
```

```
/// </summary>

/// <param name="email">Email address</param>
/// <returns>User if found, null otherwise</returns>
Task<User?> GetByEmailAsync(string email);

/// <summary>
/// Gets user by NIC (National Identity Card)
/// </summary>
/// <param name="nic">NIC number</param>
/// <returns>User if found, null otherwise</returns>
Task<User?> GetByNicAsync(string nic);

/// <summary>
/// Gets user by ID
/// </summary>
/// <param name="id">User ID</param>
/// <returns>User if found, null otherwise</returns>
Task<User?> GetByIdAsync(string id);

/// <summary>
/// Gets all users
/// </summary>
/// <returns>List of all users</returns>
Task<List<User>> GetAllAsync();

/// <summary>
/// Updates user information
/// </summary>
/// <param name="user">User entity with updated information</param>
```

```

/// <returns>Updated user</returns>
Task<User> UpdateAsync(User user);

/// <summary>
/// Activates or deactivates a user
/// </summary>
/// <param name="id">User ID</param>
/// <param name="isActive">Active status</param>
/// <returns>Success status</returns>
Task<bool> SetActiveStatusAsync(string id, bool isActive);

/// <summary>
/// Deletes a user from database
/// </summary>
/// <param name="id">User ID</param>
/// <returns>Success status</returns>
Task<bool> DeleteAsync(string id);

}

}

/*
* File: NotificationRepository.cs
* Project: EV Charging Station Booking System
* Description: Repository implementation for notification operations
* Author: EV Charging System
* Date: October 7, 2025
*/
using EVChargingBackend.Config;

```

```

using EVChargingBackend.Models;
using Microsoft.Extensions.Options;
using MongoDB.Driver;

namespace EVChargingBackend.Repositories
{
    /// <summary>
    /// Repository implementation for notification operations
    /// </summary>
    public class NotificationRepository : INotificationRepository
    {
        private readonly IMongoCollection<Notification> _notifications;
        private readonly ILogger<NotificationRepository> _logger;

        /// <summary>
        /// Initializes notification repository
        /// </summary>
        /// <param name="context">MongoDB context</param>
        /// <param name="logger">Logger</param>
        public NotificationRepository(MongoDbContext context, ILogger<NotificationRepository> logger)
        {
            _notifications = context.Notifications;
            _logger = logger;
        }

        /// <summary>
        /// Creates a new notification
        /// </summary>
        /// <param name="notification">Notification to create</param>

```

```

/// <returns>Created notification</returns>

public async Task<Notification> CreateAsync(Notification notification)

{
    await _notifications.InsertOneAsync(notification);

    _logger.LogInformation("Created notification {NotificationId} for user {RecipientNIC}",
notification.Id, notification.RecipientNIC);

    return notification;
}

/// <summary>

/// Creates multiple notifications at once

/// </summary>

/// <param name="notifications">List of notifications to create</param>

/// <returns>List of created notifications</returns>

public async Task<List<Notification>> CreateManyAsync(List<Notification> notifications)

{
    if (notifications.Any())

    {
        await _notifications.InsertManyAsync(notifications);

        _logger.LogInformation("Created {Count} notifications", notifications.Count);
    }

    return notifications;
}

/// <summary>

/// Gets notification by ID

/// </summary>

/// <param name="id">Notification ID</param>

/// <returns>Notification if found</returns>

```

```

public async Task<Notification?> GetByIdAsync(string id)
{
    return await _notifications.Find(n => n.Id == id).FirstOrDefaultAsync();
}

/// <summary>
/// Gets all notifications for a specific user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <param name="includeRead">Include read notifications</param>
/// <param name="limit">Maximum number of notifications</param>
/// <param name="offset">Number of notifications to skip</param>
/// <returns>List of notifications</returns>

public async Task<List<Notification>> GetByRecipientAsync(string recipientNIC, bool includeRead =
true, int limit = 50, int offset = 0)
{
    var filter = Builders<Notification>.Filter.Eq(n => n.RecipientNIC, recipientNIC);

    if (!includeRead)
    {
        filter = filter & Builders<Notification>.Filter.Eq(n => n.IsRead, false);
    }

    return await _notifications
        .Find(filter)
        .SortByDescending(n => n.CreatedAt)
        .Skip(offset)
        .Limit(limit)
        .ToListAsync();
}

```

```

    }

    /// <summary>
    /// Gets unread notifications for a specific user
    /// </summary>
    /// <param name="recipientNIC">User NIC</param>
    /// <returns>List of unread notifications</returns>
    public async Task<List<Notification>> GetUnreadByRecipientAsync(string recipientNIC)
    {
        var filter = Builders<Notification>.Filter.Eq(n => n.RecipientNIC, recipientNIC) &
            Builders<Notification>.Filter.Eq(n => n.IsRead, false);

        return await _notifications
            .Find(filter)
            .SortByDescending(n => n.CreatedAt)
            .ToListAsync();
    }

    /// <summary>
    /// Gets notifications by type for a specific user
    /// </summary>
    /// <param name="recipientNIC">User NIC</param>
    /// <param name="type">Notification type</param>
    /// <param name="limit">Maximum number of notifications</param>
    /// <returns>List of notifications</returns>
    public async Task<List<Notification>> GetByRecipientAndTypeAsync(string recipientNIC,
        NotificationType type, int limit = 20)
    {
        var filter = Builders<Notification>.Filter.Eq(n => n.RecipientNIC, recipientNIC) &

```

```

        Builders<Notification>.Filter.Eq(n => n.Type, type);

    return await _notifications
        .Find(filter)
        .SortByDescending(n => n.CreatedAt)
        .Limit(limit)
        .ToListAsync();
    }

/// <summary>
/// Gets notifications related to a specific entity
/// </summary>
/// <param name="entityId">Related entity ID</param>
/// <param name="entityType">Related entity type</param>
/// <returns>List of notifications</returns>
public async Task<List<Notification>> GetByRelatedEntityAsync(string entityId, string entityType)
{
    var filter = Builders<Notification>.Filter.Eq(n => n.RelatedEntityId, entityId) &
        Builders<Notification>.Filter.Eq(n => n.RelatedEntityType, entityType);

    return await _notifications
        .Find(filter)
        .SortByDescending(n => n.CreatedAt)
        .ToListAsync();
}

/// <summary>
/// Marks a notification as read
/// </summary>

```

```

/// <param name="id">Notification ID</param>
/// <returns>Success status</returns>

public async Task<bool> MarkAsReadAsync(string id)
{
    var update = Builders<Notification>.Update
        .Set(n => n.IsRead, true)
        .Set(n => n.ReadAt, DateTime.UtcNow);

    var result = await _notifications.UpdateOneAsync(n => n.Id == id, update);

    if (result.ModifiedCount > 0)
    {
        _logger.LogInformation("Marked notification {NotificationId} as read", id);
    }

    return result.ModifiedCount > 0;
}

/// <summary>
/// Marks multiple notifications as read
/// </summary>

/// <param name="ids">List of notification IDs</param>
/// <returns>Number of notifications marked as read</returns>

public async Task<int> MarkMultipleAsReadAsync(List<string> ids)
{
    var filter = Builders<Notification>.Filter.In(n => n.Id, ids);
    var update = Builders<Notification>.Update
        .Set(n => n.IsRead, true)
        .Set(n => n.ReadAt, DateTime.UtcNow);
}

```

```

var result = await _notifications.UpdateManyAsync(filter, update);

_logger.LogInformation("Marked {Count} notifications as read", result.ModifiedCount);

return (int)result.ModifiedCount;
}

/// <summary>
/// Marks all notifications for a user as read
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <returns>Number of notifications marked as read</returns>
public async Task<int> MarkAllAsReadAsync(string recipientNIC)
{
    var filter = Builders<Notification>.Filter.Eq(n => n.RecipientNIC, recipientNIC) &
        Builders<Notification>.Filter.Eq(n => n.IsRead, false);

    var update = Builders<Notification>.Update
        .Set(n => n.IsRead, true)
        .Set(n => n.ReadAt, DateTime.UtcNow);

    var result = await _notifications.UpdateManyAsync(filter, update);

    _logger.LogInformation("Marked all {Count} notifications as read for user {RecipientNIC}",
        result.ModifiedCount, recipientNIC);

    return (int)result.ModifiedCount;
}

```

```

/// <summary>
/// Marks a notification as delivered
/// </summary>
/// <param name="id">Notification ID</param>
/// <returns>Success status</returns>
public async Task<bool> MarkAsDeliveredAsync(string id)
{
    var update = Builders<Notification>.Update
        .Set(n => n.IsDelivered, true)
        .Set(n => n.DeliveredAt, DateTime.UtcNow);

    var result = await _notifications.UpdateOneAsync(n => n.Id == id, update);

    return result.ModifiedCount > 0;
}

/// <summary>
/// Deletes a notification
/// </summary>
/// <param name="id">Notification ID</param>
/// <returns>Success status</returns>
public async Task<bool> DeleteAsync(string id)
{
    var result = await _notifications.DeleteOneAsync(n => n.Id == id);

    if (result.DeletedCount > 0)
    {
        _logger.LogInformation("Deleted notification {NotificationId}", id);
    }
}

```

```

        }

        return result.DeletedCount > 0;
    }

    /// <summary>
    /// Deletes expired notifications
    /// </summary>
    /// <returns>Number of deleted notifications</returns>
    public async Task<int> DeleteExpiredNotificationsAsync()
    {
        var filter = Builders<Notification>.Filter.Lt(n => n.ExpiresAt, DateTime.UtcNow);

        var result = await _notifications.DeleteManyAsync(filter);

        _logger.LogInformation("Deleted {Count} expired notifications", result.DeletedCount);

        return (int)result.DeletedCount;
    }

    /// <summary>
    /// Gets notification count for a user
    /// </summary>
    /// <param name="recipientNIC">User NIC</param>
    /// <param name="includeRead">Include read notifications in count</param>
    /// <returns>Notification count</returns>
    public async Task<int> GetNotificationCountAsync(string recipientNIC, bool includeRead = true)
    {
        var filter = Builders<Notification>.Filter.Eq(n => n.RecipientNIC, recipientNIC);

```

```

        if (!includeRead)
    {

        filter = filter & Builders<Notification>.Filter.Eq(n => n.IsRead, false);

    }

    return (int)await _notifications.CountDocumentsAsync(filter);
}

/// <summary>
/// Gets unread notification count for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <returns>Unread notification count</returns>

public async Task<int> GetUnreadCountAsync(string recipientNIC)
{
    var filter = Builders<Notification>.Filter.Eq(n => n.RecipientNIC, recipientNIC) &
        Builders<Notification>.Filter.Eq(n => n.IsRead, false);

    return (int)await _notifications.CountDocumentsAsync(filter);
}
}

/*
* File: UserRepository.cs
* Project: EV Charging Station Booking System
* Description: Repository implementation for User operations
* Author: EV Charging System
* Date: September 27, 2025

```

```
*/  
  
using EVChargingBackend.Config;  
using EVChargingBackend.Models;  
using MongoDB.Driver;  
  
namespace EVChargingBackend.Repositories  
{  
    /// <summary>  
    /// Repository implementation for User data operations  
    /// </summary>  
    public class UserRepository : IUserRepository  
    {  
        private readonly IMongoCollection<User> _users;  
  
        /// <summary>  
        /// Initializes user repository with MongoDB context  
        /// </summary>  
        /// <param name="context">MongoDB database context</param>  
        public UserRepository(MongoDbContext context)  
        {  
            _users = context.Users;  
        }  
  
        /// <summary>  
        /// Creates a new user in the database  
        /// </summary>  
        /// <param name="user">User entity to create</param>  
        /// <returns>Created user</returns>
```

```

public async Task<User> CreateAsync(User user)
{
    user.CreatedAt = DateTime.UtcNow;
    user.UpdatedAt = DateTime.UtcNow;
    await _users.InsertOneAsync(user);
    return user;
}

/// <summary>
/// Gets user by email address
/// </summary>
/// <param name="email">Email address</param>
/// <returns>User if found, null otherwise</returns>
public async Task<User?> GetByEmailAsync(string email)
{
    return await _users.Find(u => u.Email == email).FirstOrDefaultAsync();
}

/// <summary>
/// Gets user by NIC (National Identity Card)
/// </summary>
/// <param name="nic">NIC number</param>
/// <returns>User if found, null otherwise</returns>
public async Task<User?> GetByNicAsync(string nic)
{
    return await _users.Find(u => u.NIC == nic).FirstOrDefaultAsync();
}

/// <summary>

```

```
/// Gets user by ID
/// </summary>
/// <param name="id">User ID</param>
/// <returns>User if found, null otherwise</returns>
public async Task<User?> GetByIdAsync(string id)
{
    return await _users.Find(u => u.Id == id).FirstOrDefaultAsync();
}

/// <summary>
/// Gets all users
/// </summary>
/// <returns>List of all users</returns>
public async Task<List<User>> GetAllAsync()
{
    return await _users.Find(_ => true).ToListAsync();
}

/// <summary>
/// Updates user information
/// </summary>
/// <param name="user">User entity with updated information</param>
/// <returns>Updated user</returns>
public async Task<User> UpdateAsync(User user)
{
    user.UpdatedAt = DateTime.UtcNow;
    await _users.ReplaceOneAsync(u => u.Id == user.Id, user);
    return user;
}
```

```

/// <summary>
/// Activates or deactivates a user
/// </summary>
/// <param name="id">User ID</param>
/// <param name="isActive">Active status</param>
/// <returns>Success status</returns>
public async Task<bool> SetActiveStatusAsync(string id, bool isActive)
{
    var update = Builders<User>.Update
        .Set(u => u.IsActive, isActive)
        .Set(u => u.UpdatedAt, DateTime.UtcNow);

    var result = await _users.UpdateOneAsync(u => u.Id == id, update);
    return result.ModifiedCount > 0;
}

/// <summary>
/// Deletes a user from database
/// </summary>
/// <param name="id">User ID</param>
/// <returns>Success status</returns>
public async Task<bool> DeleteAsync(string id)
{
    var result = await _users.DeleteOneAsync(u => u.Id == id);
    return result.DeletedCount > 0;
}

```

```
/*
 * File: AuthService.cs
 * Project: EV Charging Station Booking System
 * Description: Authentication service implementation with JWT token generation
 * Author: EV Charging System
 * Date: September 27, 2025
 */
```

```
using EVChargingBackend.Config;
```

```
using EVChargingBackend.DTOs;
```

```
using EVChargingBackend.Models;
```

```
using EVChargingBackend.Repositories;
```

```
using Microsoft.Extensions.Options;
```

```
using Microsoft.IdentityModel.Tokens;
```

```
using System.IdentityModel.Tokens.Jwt;
```

```
using System.Security.Claims;
```

```
using System.Text;
```

```
namespace EVChargingBackend.Services
```

```
{
```

```
/// <summary>
```

```
/// Service implementation for user authentication and JWT token management
```

```
/// </summary>
```

```
public class AuthService : IAuthService
```

```
{
```

```
    private readonly IUserRepository _userRepository;
```

```
    private readonly JwtSettings _jwtSettings;
```

```
    private readonly ILogger<AuthService> _logger;
```

```

/// <summary>
/// Initializes authentication service with dependencies
/// </summary>

/// <param name="userRepository">User repository for data operations</param>
/// <param name="jwtSettings">JWT configuration settings</param>
/// <param name="logger">Logger for service operations</param>

public AuthService(
    IUserRepository userRepository,
    IOptions<JwtSettings> jwtSettings,
    ILogger<AuthService> logger)
{
    _userRepository = userRepository;
    _jwtSettings = jwtSettings.Value;
    _logger = logger;
}

/// <summary>
/// Registers a new user in the system
/// </summary>

/// <param name="registerDto">User registration data</param>
/// <returns>Created user response</returns>

public async Task<UserResponseDto> RegisterAsync(RegisterUserDto registerDto)
{
    // Check if user already exists by email
    var existingUserByEmail = await _userRepository.GetByEmailAsync(registerDto.Email);
    if (existingUserByEmail != null)
    {
        throw new ArgumentException("User with this email already exists");
    }
}

```

```

    }

    // Check if user already exists by NIC

    var existingUserByNic = await _userRepository.GetByNicAsync(registerDto.NIC);

    if (existingUserByNic != null)

    {

        throw new ArgumentException("User with this NIC already exists");

    }

    // Hash password (in production, use proper password hashing like BCrypt)

    var hashedPassword = BCrypt.Net.BCrypt.HashPassword(registerDto.Password);

    // Create user entity

    var user = new User

    {

        NIC = registerDto.NIC,

        FirstName = registerDto.FirstName,

        LastName = registerDto.LastName,

        Email = registerDto.Email,

        Password = hashedPassword,

        Role = registerDto.Role,

        StationId = (int)registerDto.Role == 1 ? registerDto.StationId : null,

        PhoneNumber = registerDto.PhoneNumber,

        IsActive = true

    };

    // Save user to database

    var createdUser = await _userRepository.CreateAsync(user);

```

```

_logger.LogInformation("User registered successfully with email: {Email}", registerDto.Email);

// Return user response DTO

return MapToUserResponseDto(createdUser);

}

/// <summary>
/// Authenticates user login credentials
/// </summary>

/// <param name="loginDto">Login credentials</param>
/// <returns>Login response with JWT token</returns>

public async Task<LoginResponseDto> LoginAsync(LoginDto loginDto)

{
    // Get user by email

    var user = await _userRepository.GetByEmailAsync(loginDto.Email);

    if (user == null)
    {
        throw new UnauthorizedAccessException("Invalid email or password");
    }

    // Check if user is active

    if (!user.IsActive)
    {
        throw new UnauthorizedAccessException("User account is deactivated");
    }

    // Verify password

    if (!BCrypt.Net.BCrypt.Verify(loginDto.Password, user.Password))
    {

```

```

        throw new UnauthorizedAccessException("Invalid email or password");
    }

    var userResponse = MapToUserResponseDto(user);

    var token = GenerateJwtToken(userResponse);

    _logger.LogInformation("User logged in successfully: {Email}", loginDto.Email);

    if (user.Role == UserRole.StationOperator)
    {
        userResponse.StationId = user.StationId;
    }

    return new LoginResponseDto
    {
        Token = token,
        User = userResponse
    };
}

/// <summary>
/// Generates JWT token for authenticated user
/// </summary>
/// <param name="user">User data for token generation</param>
/// <returns>JWT token string</returns>

public string GenerateJwtToken(UserResponseDto user)
{
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtSettings.Key));
    var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
}

```

```

var claims = new[]
{
    new Claim(ClaimTypes.NameIdentifier, user.Id),
    new Claim(ClaimTypes.Name, user.Email),
    new Claim(ClaimTypes.Role, user.Role.ToString()),
    new Claim("nic", user.NIC),
    new Claim("firstName", user.FirstName),
    new Claim("lastName", user.LastName)
};

var token = new JwtSecurityToken(
    issuer: _jwtSettings.Issuer,
    audience: _jwtSettings.Audience,
    claims: claims,
    expires: DateTime.UtcNow.AddMinutes(_jwtSettings.ExpiryInMinutes),
    signingCredentials: credentials
);

return new JwtSecurityTokenHandler().WriteToken(token);
}

/// <summary>
/// Maps User entity to UserResponseDto
/// </summary>
/// <param name="user">User entity</param>
/// <returns>User response DTO</returns>

private static UserResponseDto MapToUserResponseDto(User user)
{

```

```

        return new UserResponseDto
    {
        Id = user.Id ?? string.Empty,
        NIC = user.NIC,
        FirstName = user.FirstName,
        LastName = user.LastName,
        Email = user.Email,
        Role = user.Role,
        StationId = user.StationId,
        IsActive = user.IsActive,
        PhoneNumber = user.PhoneNumber,
        CreatedAt = user.CreatedAt
    };
}

}

}

/*
 * File: BookingService.cs
 *
 * Project: EV Charging Station Booking System
 *
 * Description: Booking management service implementation with business rules
 *
 * Author: EV Charging System
 *
 * Date: September 27, 2025
 */

using EVChargingBackend.DTOs;
using EVChargingBackend.Models;
using EVChargingBackend.Repositories;
namespace EVChargingBackend.Services

```

```

{
    /// <summary>
    /// Service implementation for booking management operations
    /// </summary>
    public class BookingService : IBookingService
    {
        private readonly IBookingRepository _bookingRepository;
        private readonly IChargingStationRepository _stationRepository;
        private readonly INotificationService _notificationService;
        private readonly ILogger<BookingService> _logger;

        /// <summary>
        /// Initializes booking service with dependencies
        /// </summary>
        /// <param name="bookingRepository">Booking repository for data operations</param>
        /// <param name="stationRepository">Station repository for validation</param>
        /// <param name="notificationService">Notification service for sending notifications</param>
        /// <param name="logger">Logger for service operations</param>
        public BookingService(
            IBookingRepository bookingRepository,
            IChargingStationRepository stationRepository,
            INotificationService notificationService,
            ILogger<BookingService> logger)
        {
            _bookingRepository = bookingRepository;
            _stationRepository = stationRepository;
            _notificationService = notificationService;
            _logger = logger;
        }
    }
}

```

```

/// <summary>
/// Creates a new booking with business rule validation
/// </summary>

/// <param name="createDto">Booking creation data</param>
/// <param name="ownerNic">Owner NIC from JWT token</param>
/// <returns>Created booking</returns>

public async Task<BookingResponseDto> CreateBookingAsync(CreateBookingDto createDto, string
ownerNic)

{
    // Validate time constraints
    ValidateBookingTimes(createDto.StartTime, createDto.EndTime);

    // Get and validate station
    var station = await _stationRepository.GetByIdAsync(createDto.StationId);
    if (station == null)
    {
        throw new KeyNotFoundException("Charging station not found");
    }

    if (station.Status != StationStatus.Active)
    {
        throw new ArgumentException("Charging station is not active");
    }

    if (station.AvailableSlots <= 0)
    {
        throw new ArgumentException("No available slots at this charging station");
    }
}

```

```

// Check for overlapping bookings

var overlappingBookings = await _bookingRepository.GetOverlappingBookingsAsync(
    createDto.StationId, createDto.StartTime, createDto.EndTime);

if (overlappingBookings.Count >= station.TotalSlots)
{
    throw new ArgumentException("No available slots for the requested time period");
}

// Calculate total amount

var duration = createDto.EndTime - createDto.StartTime;
var totalAmount = (decimal)duration.TotalHours * station.PricePerHour;

// Create booking

var booking = new Booking
{
    OwnerNIC = ownerNic,
    StationId = createDto.StationId,
    StartTime = createDto.StartTime,
    EndTime = createDto.EndTime,
    Status = BookingStatus.Active,
    QRCode = GenerateQRCode(),
    TotalAmount = totalAmount
};

var createdBooking = await _bookingRepository.CreateAsync(booking);

// Update available slots

```

```

        await _stationRepository.UpdateAvailableSlotsAsync(createDto.StationId, -1);

        _logger.LogInformation("Booking created: {BookingId} for station {StationId}", createdBooking.Id,
createDto.StationId);

    return await MapToBookingResponseDto(createdBooking);
}

/// <summary>
/// Gets all bookings (backoffice and operators)
/// </summary>
/// <returns>List of all bookings</returns>

public async Task<List<BookingResponseDto>> GetAllBookingsAsync()
{
    var bookings = await _bookingRepository.GetAllAsync();

    var bookingDtos = new List<BookingResponseDto>();

    foreach (var booking in bookings)
    {
        bookingDtos.Add(await MapToBookingResponseDto(booking));
    }

    return bookingDtos;
}

/// <summary>
/// Gets bookings for specific user
/// </summary>
/// <param name="ownerNic">Owner NIC</param>

```

```

/// <returns>List of user bookings</returns>

public async Task<List<BookingResponseDto>> GetUserBookingsAsync(string ownerNic)
{
    var bookings = await _bookingRepository.GetByOwnerAsync(ownerNic);

    var bookingDtos = new List<BookingResponseDto>();

    foreach (var booking in bookings)
    {
        bookingDtos.Add(await MapToBookingResponseDto(booking));
    }

    return bookingDtos;
}

/// <summary>
/// Gets bookings for specific station (operators)
/// </summary>

/// <param name="stationId">Station ID</param>

/// <returns>List of station bookings</returns>

public async Task<List<BookingResponseDto>> GetStationBookingsAsync(string stationId)
{
    var bookings = await _bookingRepository.GetByStationAsync(stationId);

    var bookingDtos = new List<BookingResponseDto>();

    foreach (var booking in bookings)
    {
        bookingDtos.Add(await MapToBookingResponseDto(booking));
    }
}

```

```

        return bookingDtos;
    }

    /// <summary>
    /// Gets booking by ID
    /// </summary>
    /// <param name="id">Booking ID</param>
    /// <returns>Booking if found</returns>
    public async Task<BookingResponseDto> GetBookingByIdAsync(string id)
    {
        var booking = await _bookingRepository.GetByIdAsync(id);
        if (booking == null)
        {
            throw new KeyNotFoundException("Booking not found");
        }

        return await MapToBookingResponseDto(booking);
    }

    /// <summary>
    /// Updates booking with business rule validation
    /// </summary>
    /// <param name="id">Booking ID</param>
    /// <param name="updateDto">Updated booking data</param>
    /// <param name="ownerNic">Owner NIC from JWT token</param>
    /// <returns>Updated booking</returns>
    public async Task<BookingResponseDto> UpdateBookingAsync(string id, UpdateBookingDto
updateDto, string ownerNic)
    {

```

```

var booking = await _bookingRepository.GetByIdAsync(id);

if (booking == null)

{
    throw new KeyNotFoundException("Booking not found");
}

// Only the booking owner can update their booking

if (booking.OwnerNIC != ownerNic)

{
    throw new UnauthorizedAccessException("You can only update your own bookings");
}

// Check if booking can be modified (12 hours before start time)

if (DateTime.UtcNow.AddHours(12) > booking.StartTime)

{
    throw new ArgumentException("Cannot modify booking within 12 hours of start time");
}

// Only active bookings can be updated

if (booking.Status != BookingStatus.Active)

{
    throw new ArgumentException("Only active bookings can be updated");
}

var hasTimeChanges = false;

// Update time if provided

if (updateDto.StartTime.HasValue || updateDto.EndTime.HasValue)

{

```

```

var newStartTime = updateDto.StartTime ?? booking.StartTime;

var newEndTime = updateDto.EndTime ?? booking.EndTime;

ValidateBookingTimes(newStartTime, newEndTime);

// Check for overlapping bookings (exclude current booking)

var overlappingBookings = await _bookingRepository.GetOverlappingBookingsAsync(
    booking.StationId, newStartTime, newEndTime);

var conflictingBookings = overlappingBookings.Where(b => b.Id != id).ToList();

var station = await _stationRepository.GetByIdAsync(booking.StationId);

if (conflictingBookings.Count >= station!.TotalSlots)
{
    throw new ArgumentException("No available slots for the requested time period");
}

booking.StartTime = newStartTime;
booking.EndTime = newEndTime;
hasTimeChanges = true;
}

// Update status if provided

if (updateDto.Status.HasValue)
{
    booking.Status = updateDto.Status.Value;
}

// Recalculate total amount if time changed

```

```

        if (hasTimeChanges)
    {

        var station = await _stationRepository.GetByIdAsync(booking.StationId);

        var duration = booking.EndTime - booking.StartTime;

        booking.TotalAmount = (decimal)duration.TotalHours * station!.PricePerHour;

    }

    var updatedBooking = await _bookingRepository.UpdateAsync(booking);

    _logger.LogInformation("Booking updated: {BookingId}", id);

    return await MapToBookingResponseDto(updatedBooking);

}

/// <summary>
/// Confirms booking (station operator action)
/// </summary>
/// <param name="bookingId">Booking ID</param>
/// <returns>Success status</returns>

public async Task<bool> ConfirmBookingAsync(string bookingId)

{
    var booking = await _bookingRepository.GetByIdAsync(bookingId);

    if (booking == null)

    {
        throw new KeyNotFoundException("Booking not found");
    }

    if (booking.Status != BookingStatus.Active)

    {

```

```

        throw new ArgumentException("Only active bookings can be confirmed");

    }

    var result = await _bookingRepository.ConfirmBookingAsync(bookingId);

    if (result)

    {
        _logger.LogInformation("Booking confirmed: {BookingId}", bookingId);

        // Send confirmation notification to the EV owner
        try
        {
            var station = await _stationRepository.GetByIdAsync(booking.StationId);
            var stationName = station?.Name ?? "Charging Station";

            await _notificationService.CreateBookingConfirmationNotificationAsync(
                booking.OwnerNIC,
                bookingId,
                stationName,
                booking.StartTime,
                booking.EndTime
            );
        }

        _logger.LogInformation("Confirmation notification sent for booking: {BookingId}", bookingId);
    }

    catch (Exception ex)
    {
        _logger.LogError(ex, "Failed to send confirmation notification for booking: {BookingId}",
            bookingId);
    }
}

```

```

        // Don't fail the booking confirmation if notification fails
    }

}

return result;
}

/// <summary>
/// Cancels booking with time validation
/// </summary>

/// <param name="id">Booking ID</param>
/// <param name="ownerNic">Owner NIC from JWT token</param>
/// <returns>Success status</returns>

public async Task<bool> CancelBookingAsync(string id, string ownerNic)
{
    var booking = await _bookingRepository.GetByldAsync(id);
    if (booking == null)
    {
        throw new KeyNotFoundException("Booking not found");
    }

    // Only the booking owner can cancel their booking
    if (booking.OwnerNIC != ownerNic)
    {
        throw new UnauthorizedAccessException("You can only cancel your own bookings");
    }

    // Check if booking can be cancelled (12 hours before start time)
    if (DateTime.UtcNow.AddHours(12) > booking.StartTime)

```

```

    {

        throw new ArgumentException("Cannot cancel booking within 12 hours of start time");

    }

    // Only active or confirmed bookings can be cancelled

    if (booking.Status != BookingStatus.Active && booking.Status != BookingStatus.Confirmed)

    {

        throw new ArgumentException("Only active or confirmed bookings can be cancelled");

    }

    var result = await _bookingRepository.CancelBookingAsync(id);

    if (result)

    {

        // Return the slot to available pool

        await _stationRepository.UpdateAvailableSlotsAsync(booking.StationId, 1);

        _logger.LogInformation("Booking cancelled: {BookingId}", id);

        // Send cancellation notification to the EV owner

        try

        {

            var station = await _stationRepository.GetByIdAsync(booking.StationId);

            var stationName = station?.Name ?? "Charging Station";



            await _notificationService.CreateBookingCancellationNotificationAsync(

                booking.OwnerNIC,

                id,

                stationName,

                "Cancelled by user"

```

```

    );
}

_logger.LogInformation("Cancellation notification sent for booking: {BookingId}", id);

}

catch (Exception ex)

{

    _logger.LogError(ex, "Failed to send cancellation notification for booking: {BookingId}", id);

    // Don't fail the booking cancellation if notification fails

}

}

return result;
}

/// <summary>

/// Cancels booking by operator (station operator or backoffice)

/// </summary>

/// <param name="bookingId">Booking ID</param>

/// <param name="reason">Cancellation reason</param>

/// <returns>Success status</returns>

public async Task<bool> CancelBookingByOperatorAsync(string bookingId, string reason = "")

{

    var booking = await _bookingRepository.GetByIdAsync(bookingId);

    if (booking == null)

    {

        throw new KeyNotFoundException("Booking not found");

    }

    // Only active or confirmed bookings can be cancelled
}

```

```

if (booking.Status != BookingStatus.Active && booking.Status != BookingStatus.Confirmed)
{
    throw new ArgumentException("Only active or confirmed bookings can be cancelled");
}

var result = await _bookingRepository.CancelBookingAsync(bookingId);

if (result)
{
    // Return the slot to available pool
    await _stationRepository.UpdateAvailableSlotsAsync(booking.StationId, 1);
    _logger.LogInformation("Booking cancelled by operator: {BookingId}, Reason: {Reason}",
        bookingId, reason);

    // Send cancellation notification to the EV owner
    try
    {
        var station = await _stationRepository.GetByIdAsync(booking.StationId);
        var stationName = station?.Name ?? "Charging Station";

        var fullReason = string.IsNullOrEmpty(reason) ? "Cancelled by station operator" : $"Cancelled
by station operator - {reason}";

        await _notificationService.CreateBookingCancellationNotificationAsync(
            booking.OwnerNIC,
            bookingId,
            stationName,
            fullReason
        );
    }
}

```

```

        _logger.LogInformation("Cancellation notification sent for booking: {BookingId}", bookingId);

    }

    catch (Exception ex)

    {

        _logger.LogError(ex, "Failed to send cancellation notification for booking: {BookingId}",
bookingId);

        // Don't fail the booking cancellation if notification fails

    }

}

return result;

}

/// <summary>

/// Validates booking time constraints

/// </summary>

/// <param name="startTime">Booking start time</param>

/// <param name="endTime">Booking end time</param>

private static void ValidateBookingTimes(DateTime startTime, DateTime endTime)

{

    // Ensure start time is in the future

    if (startTime <= DateTime.UtcNow)

    {

        throw new ArgumentException("Booking start time must be in the future");

    }

    // Ensure end time is after start time

    if (endTime <= startTime)

    {

```

```
        throw new ArgumentException("Booking end time must be after start time");

    }

    // Maximum 7 days in advance

    if (startTime > DateTime.UtcNow.AddDays(7))

    {

        throw new ArgumentException("Bookings can only be made up to 7 days in advance");

    }

    // Minimum booking duration (1 hour)

    if ((endTime - startTime).TotalHours < 1)

    {

        throw new ArgumentException("Minimum booking duration is 1 hour");

    }

    // Maximum booking duration (24 hours)

    if ((endTime - startTime).TotalHours > 24)

    {

        throw new ArgumentException("Maximum booking duration is 24 hours");

    }

}

/// <summary>

/// Generates a unique QR code for the booking

/// </summary>

/// <returns>QR code string</returns>

private static string GenerateQRCode()

{

    return Guid.NewGuid().ToString("N").ToUpper();
```

```

    }

    /// <summary>
    /// Maps Booking entity to BookingResponseDto with station details
    /// </summary>
    /// <param name="booking">Booking entity</param>
    /// <returns>Booking response DTO</returns>
    private async Task<BookingResponseDto> MapToBookingResponseDto(Booking booking)
    {
        var station = await _stationRepository.GetByIdAsync(booking.StationId);

        return new BookingResponseDto
        {
            Id = booking.Id ?? string.Empty,
            OwnerNIC = booking.OwnerNIC,
            StationId = booking.StationId,
            StationName = station?.Name ?? "Unknown Station",
            StationLocation = station?.Location ?? "Unknown Location",
            StartTime = booking.StartTime,
            EndTime = booking.EndTime,
            Status = booking.Status,
            QRCode = booking.QRCode,
            TotalAmount = booking.TotalAmount,
            CreatedAt = booking.CreatedAt,
            ConfirmedAt = booking.ConfirmedAt,
            CancelledAt = booking.CancelledAt
        };
    }
}

```

```

}

/*
 * File: ChargingStationService.cs
 * Project: EV Charging Station Booking System
 * Description: Charging station management service implementation
 * Author: EV Charging System
 * Date: September 27, 2025
 */

using EVChargingBackend.DTOs;
using EVChargingBackend.Models;
using EVChargingBackend.Repositories;

namespace EVChargingBackend.Services
{
    /// <summary>
    /// Service implementation for charging station management operations
    /// </summary>
    public class ChargingStationService : IChargingStationService
    {
        private readonly IChargingStationRepository _stationRepository;
        private readonly IBookingRepository _bookingRepository;
        private readonly ILogger<ChargingStationService> _logger;

        /// <summary>
        /// Initializes charging station service with dependencies
        /// </summary>
        /// <param name="stationRepository">Station repository for data operations</param>

```

```

/// <param name="bookingRepository">Booking repository for validation</param>
/// <param name="logger">Logger for service operations</param>

public ChargingStationService(
    IChargingStationRepository stationRepository,
    IBookingRepository bookingRepository,
    ILogger<ChargingStationService> logger)
{
    _stationRepository = stationRepository;
    _bookingRepository = bookingRepository;
    _logger = logger;
}

/// <summary>
/// Creates a new charging station
/// </summary>
/// <param name="createDto">Station creation data</param>
/// <returns>Created station</returns>

public async Task<StationResponseDto> CreateStationAsync(CreateStationDto createDto)
{
    var station = new ChargingStation
    {
        Name = createDto.Name,
        Location = createDto.Location,
        Type = createDto.Type,
        TotalSlots = createDto.TotalSlots,
        AvailableSlots = createDto.TotalSlots,
        PricePerHour = createDto.PricePerHour,
        Status = StationStatus.Active
    };
}

```

```

        var createdStation = await _stationRepository.CreateAsync(station);

        _logger.LogInformation("Charging station created: {StationId}", createdStation.Id);

        return MapToStationResponseDto(createdStation);
    }

/// <summary>
/// Gets all charging stations
/// </summary>
/// <returns>List of all stations</returns>
public async Task<List<StationResponseDto>> GetAllStationsAsync()

{
    var stations = await _stationRepository.GetAllAsync();

    return stations.Select(MapToStationResponseDto).ToList();
}

/// <summary>
/// Gets active charging stations only
/// </summary>
/// <returns>List of active stations</returns>
public async Task<List<StationResponseDto>> GetActiveStationsAsync()

{
    var stations = await _stationRepository.GetActiveStationsAsync();

    return stations.Select(MapToStationResponseDto).ToList();
}

/// <summary>

```

```

/// Gets charging station by ID

/// </summary>

/// <param name="id">Station ID</param>

/// <returns>Station if found</returns>

public async Task<StationResponseDto> GetStationByIdAsync(string id)

{

    var station = await _stationRepository.GetByIdAsync(id);

    if (station == null)

    {

        throw new KeyNotFoundException("Charging station not found");

    }

    return MapToStationResponseDto(station);

}

/// <summary>

/// Updates charging station information

/// </summary>

/// <param name="id">Station ID</param>

/// <param name="updateDto">Updated station data</param>

/// <returns>Updated station</returns>

public async Task<StationResponseDto> UpdateStationAsync(string id, UpdateStationDto updateDto)

{

    var station = await _stationRepository.GetByIdAsync(id);

    if (station == null)

    {

        throw new KeyNotFoundException("Charging station not found");

    }

}

```

```

// Update only provided fields

if (!string.IsNullOrEmpty(updateDto.Name))
    station.Name = updateDto.Name;

if (!string.IsNullOrEmpty(updateDto.Location))
    station.Location = updateDto.Location;

if (updateDto.Type.HasValue)
    station.Type = updateDto.Type.Value;

if (updateDto.TotalSlots.HasValue)
{
    var slotsDifference = updateDto.TotalSlots.Value - station.TotalSlots;
    station.TotalSlots = updateDto.TotalSlots.Value;
    station.AvailableSlots += slotsDifference; // Adjust available slots

    // Ensure available slots don't go below 0
    if (station.AvailableSlots < 0)
        station.AvailableSlots = 0;
}

if (updateDto.PricePerHour.HasValue)
    station.PricePerHour = updateDto.PricePerHour.Value;

if (updateDto.Status.HasValue)
    station.Status = updateDto.Status.Value;

var updatedStation = await _stationRepository.UpdateAsync(station);

```

```

    _logger.LogInformation("Charging station updated: {StationId}", id);

    return MapToStationResponseDto(updatedStation);
}

/// <summary>
/// Deactivates a charging station (with validation)
/// </summary>
/// <param name="id">Station ID</param>
/// <returns>Success status</returns>

public async Task<bool> DeactivateStationAsync(string id)
{
    var station = await _stationRepository.GetByIdAsync(id);

    if (station == null)
    {
        throw new KeyNotFoundException("Charging station not found");
    }

    // Check for active bookings

    var activeBookings = await _bookingRepository.GetActiveBookingCountAsync(id);

    if (activeBookings > 0)
    {
        throw new ArgumentException("Cannot deactivate station with active bookings");
    }

    station.Status = StationStatus.Inactive;

    await _stationRepository.UpdateAsync(station);

    _logger.LogInformation("Charging station deactivated: {StationId}", id);
}

```

```

        return true;
    }

/// <summary>
/// Deletes a charging station
/// </summary>

/// <param name="id">Station ID</param>
/// <returns>Success status</returns>

public async Task<bool> DeleteStationAsync(string id)
{
    var station = await _stationRepository.GetByIdAsync(id);
    if (station == null)
    {
        throw new KeyNotFoundException("Charging station not found");
    }

    // Check for any bookings (not just active ones)
    var allBookings = await _bookingRepository.GetByStationAsync(id);
    if (allBookings.Any())
    {
        throw new ArgumentException("Cannot delete station with existing bookings");
    }

    var result = await _stationRepository.DeleteAsync(id);

    if (result)
    {
        _logger.LogInformation("Charging station deleted: {StationId}", id);
    }
}

```

```

    }

    return result;
}

/// <summary>
/// Maps ChargingStation entity to StationResponseDto
/// </summary>
/// <param name="station">Charging station entity</param>
/// <returns>Station response DTO</returns>

private static StationResponseDto MapToStationResponseDto(ChargingStation station)
{
    return new StationResponseDto
    {
        Id = station.Id ?? string.Empty,
        Name = station.Name,
        Location = station.Location,
        Type = station.Type,
        TotalSlots = station.TotalSlots,
        AvailableSlots = station.AvailableSlots,
        Status = station.Status,
        PricePerHour = station.PricePerHour,
        CreatedAt = station.CreatedAt,
        UpdatedAt = station.UpdatedAt
    };
}
}
}

```

```

/*
 * File: IAuthService.cs
 * Project: EV Charging Station Booking System
 * Description: Interface for authentication services
 * Author: EV Charging System
 * Date: September 27, 2025
 */

using EVChargingBackend.DTOs;

namespace EVChargingBackend.Services
{
    /// <summary>
    /// Interface defining authentication service operations
    /// </summary>
    public interface IAuthService
    {
        /// <summary>
        /// Registers a new user in the system
        /// </summary>
        /// <param name="registerDto">User registration data</param>
        /// <returns>Created user response</returns>
        Task<UserResponseDto> RegisterAsync(RegisterUserDto registerDto);

        /// <summary>
        /// Authenticates user login credentials
        /// </summary>
    }
}

```

```

    /// <param name="loginDto">Login credentials</param>
    /// <returns>Login response with JWT token</returns>
    Task<LoginResponseDto> LoginAsync(LoginDto loginDto);

    /// <summary>
    /// Generates JWT token for authenticated user
    /// </summary>
    /// <param name="user">User data for token generation</param>
    /// <returns>JWT token string</returns>
    string GenerateJwtToken(UserResponseDto user);

}

}

/*
* File: IBookingService.cs
* Project: EV Charging Station Booking System
* Description: Interface for booking management services
* Author: EV Charging System
* Date: September 27, 2025
*/
using EVChargingBackend.DTOs;

namespace EVChargingBackend.Services
{
    /// <summary>
    /// Interface defining booking management service operations
    /// </summary>
    public interface IBookingService

```

```

{

    /// <summary>
    /// Creates a new booking with business rule validation
    /// </summary>

    /// <param name="createDto">Booking creation data</param>
    /// <param name="ownerNic">Owner NIC from JWT token</param>
    /// <returns>Created booking</returns>

    Task<BookingResponseDto> CreateBookingAsync(CreateBookingDto createDto, string ownerNic);

    /// <summary>
    /// Gets all bookings (backoffice and operators)
    /// </summary>

    /// <returns>List of all bookings</returns>

    Task<List<BookingResponseDto>> GetAllBookingsAsync();

    /// <summary>
    /// Gets bookings for specific user
    /// </summary>

    /// <param name="ownerNic">Owner NIC</param>
    /// <returns>List of user bookings</returns>

    Task<List<BookingResponseDto>> GetUserBookingsAsync(string ownerNic);

    /// <summary>
    /// Gets bookings for specific station (operators)
    /// </summary>

    /// <param name="stationId">Station ID</param>
    /// <returns>List of station bookings</returns>

    Task<List<BookingResponseDto>> GetStationBookingsAsync(string stationId);
}

```

```

/// <summary>
/// Gets booking by ID
/// </summary>
/// <param name="id">Booking ID</param>
/// <returns>Booking if found</returns>
Task<BookingResponseDto> GetBookingByIdAsync(string id);

/// <summary>
/// Updates booking with business rule validation
/// </summary>
/// <param name="id">Booking ID</param>
/// <param name="updateDto">Updated booking data</param>
/// <param name="ownerNic">Owner NIC from JWT token</param>
/// <returns>Updated booking</returns>
Task<BookingResponseDto> UpdateBookingAsync(string id, UpdateBookingDto updateDto, string
ownerNic);

/// <summary>
/// Confirms booking (station operator action)
/// </summary>
/// <param name="bookingId">Booking ID</param>
/// <returns>Success status</returns>
Task<bool> ConfirmBookingAsync(string bookingId);

/// <summary>
/// Cancels booking with time validation
/// </summary>
/// <param name="id">Booking ID</param>
/// <param name="ownerNic">Owner NIC from JWT token</param>

```

```

/// <returns>Success status</returns>

Task<bool> CancelBookingAsync(string id, string ownerNic);

/// <summary>
/// Cancels booking by operator (station operator or backoffice)
/// </summary>

/// <param name="bookingId">Booking ID</param>
/// <param name="reason">Cancellation reason</param>
/// <returns>Success status</returns>

Task<bool> CancelBookingByOperatorAsync(string bookingId, string reason = "");

}

}

/*
* File: IChargingStationService.cs
* Project: EV Charging Station Booking System
* Description: Interface for charging station management services
* Author: EV Charging System
* Date: September 27, 2025
*/

```

using EVChargingBackend.DTOs;

```

namespace EVChargingBackend.Services
{
    /// <summary>
    /// Interface defining charging station management service operations
    /// </summary>
    public interface IChargingStationService

```

```
{  
    /// <summary>  
    /// Creates a new charging station  
    /// </summary>  
    /// <param name="createDto">Station creation data</param>  
    /// <returns>Created station</returns>  
    Task<StationResponseDto> CreateStationAsync(CreateStationDto createDto);  
  
    /// <summary>  
    /// Gets all charging stations  
    /// </summary>  
    /// <returns>List of all stations</returns>  
    Task<List<StationResponseDto>> GetAllStationsAsync();  
  
    /// <summary>  
    /// Gets active charging stations only  
    /// </summary>  
    /// <returns>List of active stations</returns>  
    Task<List<StationResponseDto>> GetActiveStationsAsync();  
  
    /// <summary>  
    /// Gets charging station by ID  
    /// </summary>  
    /// <param name="id">Station ID</param>  
    /// <returns>Station if found</returns>  
    Task<StationResponseDto> GetStationByIdAsync(string id);  
  
    /// <summary>  
    /// Updates charging station information  
    /// </summary>
```

```

    /// </summary>
    /// <param name="id">Station ID</param>
    /// <param name="updateDto">Updated station data</param>
    /// <returns>Updated station</returns>
    Task<StationResponseDto> UpdateStationAsync(string id, UpdateStationDto updateDto);

    /// <summary>
    /// Deactivates a charging station (with validation)
    /// </summary>
    /// <param name="id">Station ID</param>
    /// <returns>Success status</returns>
    Task<bool> DeactivateStationAsync(string id);

    /// <summary>
    /// Deletes a charging station
    /// </summary>
    /// <param name="id">Station ID</param>
    /// <returns>Success status</returns>
    Task<bool> DeleteStationAsync(string id);
}

}

/*
* File: INotificationService.cs
* Project: EV Charging Station Booking System
* Description: Interface for notification service operations
* Author: EV Charging System
* Date: October 7, 2025
*/

```

```
using EVChargingBackend.DTOs;
using EVChargingBackend.Models;

namespace EVChargingBackend.Services
{
    /// <summary>
    /// Interface for notification service operations
    /// </summary>
    public interface INotificationService
    {
        /// <summary>
        /// Creates a notification
        /// </summary>
        /// <param name="createDto">Notification creation data</param>
        /// <returns>Created notification</returns>
        Task<NotificationResponseDto> CreateNotificationAsync(CreateNotificationDto createDto);

        /// <summary>
        /// Creates bulk notifications for multiple recipients
        /// </summary>
        /// <param name="bulkDto">Bulk notification data</param>
        /// <returns>List of created notifications</returns>
        Task<List<NotificationResponseDto>> CreateBulkNotificationAsync(BulkNotificationDto bulkDto);

        /// <summary>
        /// Creates a booking confirmation notification
        /// </summary>
        /// <param name="recipientNIC">Recipient's NIC</param>
    }
}
```

```

/// <param name="bookingId">Booking ID</param>
/// <param name="stationName">Charging station name</param>
/// <param name="startTime">Booking start time</param>
/// <param name="endTime">Booking end time</param>
/// <returns>Created notification</returns>

Task<NotificationResponseDto> CreateBookingConfirmationNotificationAsync(
    string recipientNIC, string bookingId, string stationName, DateTime startTime, DateTime endTime);

/// <summary>
/// Creates a booking cancellation notification
/// </summary>

/// <param name="recipientNIC">Recipient's NIC</param>
/// <param name="bookingId">Booking ID</param>
/// <param name="stationName">Charging station name</param>
/// <param name="reason">Cancellation reason</param>
/// <returns>Created notification</returns>

Task<NotificationResponseDto> CreateBookingCancellationNotificationAsync(
    string recipientNIC, string bookingId, string stationName, string reason = "");

/// <summary>
/// Creates a booking reminder notification
/// </summary>

/// <param name="recipientNIC">Recipient's NIC</param>
/// <param name="bookingId">Booking ID</param>
/// <param name="stationName">Charging station name</param>
/// <param name="startTime">Booking start time</param>
/// <returns>Created notification</returns>

Task<NotificationResponseDto> CreateBookingReminderNotificationAsync(
    string recipientNIC, string bookingId, string stationName, DateTime startTime);

```

```

/// <summary>
/// Gets all notifications for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <param name="includeRead">Include read notifications</param>
/// <param name="limit">Maximum number of notifications</param>
/// <param name="offset">Number of notifications to skip</param>
/// <returns>List of notifications</returns>

Task<List<NotificationResponseDto>> GetUserNotificationsAsync(
    string recipientNIC, bool includeRead = true, int limit = 50, int offset = 0);

/// <summary>
/// Gets unread notifications for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <returns>List of unread notifications</returns>

Task<List<NotificationResponseDto>> GetUnreadNotificationsAsync(string recipientNIC);

/// <summary>
/// Gets notifications by type for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <param name="type">Notification type</param>
/// <param name="limit">Maximum number of notifications</param>
/// <returns>List of notifications</returns>

Task<List<NotificationResponseDto>> GetNotificationsByTypeAsync(
    string recipientNIC, NotificationType type, int limit = 20);

```

```

/// <summary>
/// Gets notification by ID
/// </summary>
/// <param name="id">Notification ID</param>
/// <returns>Notification if found</returns>
Task<NotificationResponseDto> GetNotificationByIdAsync(string id);

/// <summary>
/// Marks notifications as read
/// </summary>
/// <param name="notificationIds">List of notification IDs</param>
/// <param name="userNIC">User NIC for validation</param>
/// <returns>Number of notifications marked as read</returns>
Task<int> MarkNotificationsAsReadAsync(List<string> notificationIds, string userNIC);

/// <summary>
/// Marks all notifications for a user as read
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <returns>Number of notifications marked as read</returns>
Task<int> MarkAllAsReadAsync(string recipientNIC);

/// <summary>
/// Deletes a notification (soft delete by setting expiry)
/// </summary>
/// <param name="id">Notification ID</param>
/// <param name="userNIC">User NIC for validation</param>
/// <returns>Success status</returns>
Task<bool> DeleteNotificationAsync(string id, string userNIC);

```

```

/// <summary>
/// Gets notification summary for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <returns>Notification summary</returns>
Task<NotificationSummaryDto> GetNotificationSummaryAsync(string recipientNIC);

/// <summary>
/// Cleans up expired notifications
/// </summary>
/// <returns>Number of deleted notifications</returns>
Task<int> CleanupExpiredNotificationsAsync();

}

}

/*
* File: ISeedDataService.cs
* Project: EV Charging Station Booking System
* Description: Interface for seed data service
* Author: EV Charging System
* Date: September 27, 2025
*/

```

```

namespace EVChargingBackend.Services
{
    /// <summary>
    /// Interface for seeding initial data into the database
    /// </summary>

```

```

public interface ISeedDataService
{
    /// <summary>
    /// Seeds initial data including users, stations, and sample bookings
    /// </summary>
    /// <returns>Task representing the async operation</returns>
    Task SeedDataAsync();
}

/*
 * File: IUserService.cs
 * Project: EV Charging Station Booking System
 * Description: Interface for user management services
 * Author: EV Charging System
 * Date: September 27, 2025
 */

```

```

using EVChargingBackend.DTOs;

namespace EVChargingBackend.Services
{
    /// <summary>
    /// Interface defining user management service operations
    /// </summary>
    public interface IUserService
    {
        /// <summary>
        /// Gets all users (backoffice only)
        /// </summary>

```

```
/// </summary>

/// <returns>List of all users</returns>

Task<List<UserResponseDto>> GetAllUsersAsync();

/// <summary>
/// Gets user by ID
/// </summary>

/// <param name="id">User ID</param>
/// <returns>User if found</returns>

Task<UserResponseDto> GetUserByIdAsync(string id);

/// <summary>
/// Updates user information
/// </summary>

/// <param name="id">User ID</param>
/// <param name="updateDto">Updated user data</param>
/// <returns>Updated user</returns>

Task<UserResponseDto> UpdateUserAsync(string id, UpdateUserDto updateDto);

/// <summary>
/// Activates or deactivates a user (backoffice only)
/// </summary>

/// <param name="id">User ID</param>
/// <param name="isActive">Active status</param>
/// <returns>Success status</returns>

Task<bool> SetUserActiveStatusAsync(string id, bool isActive);

/// <summary>
/// Deactivates current user (EV Owner self-deactivation)
/// </summary>
```

```

    /// </summary>

    /// <param name="id">User ID</param>
    /// <returns>Success status</returns>

    Task<bool> DeactivateUserAsync(string id);

    /// <summary>
    /// Deletes a user (backoffice only)
    /// </summary>

    /// <param name="id">User ID</param>
    /// <returns>Success status</returns>

    Task<bool> DeleteUserAsync(string id);

}

}

/*
 * File: NotificationBackgroundService.cs
 *
 * Project: EV Charging Station Booking System
 *
 * Description: Background service for handling notification reminders and cleanup
 *
 * Author: EV Charging System
 *
 * Date: October 7, 2025
 */

using EVChargingBackend.Models;
using EVChargingBackend.Repositories;
using EVChargingBackend.Services;

namespace EVChargingBackend.Services
{
    /// <summary>

```

```

/// Background service for notification reminders and cleanup

/// </summary>

public class NotificationBackgroundService : BackgroundService
{
    private readonly IServiceProvider _serviceProvider;
    private readonly ILogger<NotificationBackgroundService> _logger;
    private readonly TimeSpan _reminderCheckInterval = TimeSpan.FromMinutes(30); // Check every 30 minutes
    private readonly TimeSpan _cleanupInterval = TimeSpan.FromHours(6); // Cleanup every 6 hours
    private DateTime _lastCleanup = DateTime.UtcNow;

    /// <summary>
    /// Initializes notification background service
    /// </summary>

    /// <param name="serviceProvider">Service provider for dependency injection</param>
    /// <param name="logger">Logger</param>

    public NotificationBackgroundService(
        IServiceProvider serviceProvider,
        ILogger<NotificationBackgroundService> logger)
    {
        _serviceProvider = serviceProvider;
        _logger = logger;
    }

    /// <summary>
    /// Main execution loop for background service
    /// </summary>

    /// <param name="stoppingToken">Cancellation token</param>

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)

```

```
{  
    _logger.LogInformation("Notification background service started");  
  
    while (!stoppingToken.IsCancellationRequested)  
    {  
        try  
        {  
            using var scope = _serviceProvider.CreateScope();  
  
            // Send booking reminders  
  
            await SendBookingRemindersAsync(scope.ServiceProvider);  
  
            // Cleanup expired notifications if it's time  
  
            if (DateTime.UtcNow - _lastCleanup >= _cleanupInterval)  
            {  
                await CleanupExpiredNotificationsAsync(scope.ServiceProvider);  
  
                _lastCleanup = DateTime.UtcNow;  
            }  
        }  
        catch (Exception ex)  
        {  
            _logger.LogError(ex, "Error in notification background service");  
        }  
  
        await Task.Delay(_reminderCheckInterval, stoppingToken);  
    }  
  
    _logger.LogInformation("Notification background service stopped");  
}
```

```

/// <summary>
/// Sends booking reminder notifications
/// </summary>
/// <param name="serviceProvider">Service provider</param>
private async Task SendBookingRemindersAsync(IServiceProvider serviceProvider)

{
    try
    {
        var bookingRepository = serviceProvider.GetRequiredService<IBookingRepository>();
        var stationRepository = serviceProvider.GetRequiredService<IChargingStationRepository>();
        var notificationService = serviceProvider.GetRequiredService<INotificationService>();
        var notificationRepository = serviceProvider.GetRequiredService<INotificationRepository>();

        var currentTime = DateTime.UtcNow;
        var reminderWindow = currentTime.AddHours(2); // Send reminders 2 hours before

        // Get confirmed bookings starting within the next 2 hours
        var upcomingBookings = await bookingRepository.GetUpcomingConfirmedBookingsAsync(
            currentTime, reminderWindow);

        _logger.LogDebug("Found {Count} upcoming bookings for reminders",
            upcomingBookings.Count);

        foreach (var booking in upcomingBookings)
        {
            try
            {
                // Check if reminder notification already sent for this booking

```

```

var existingReminders = await notificationRepository.GetByRelatedEntityAsync(
    booking.Id!, nameof(Booking));

var reminderExists = existingReminders.Any(n =>
    n.Type == NotificationType.BookingReminder &&
    n.RecipientNIC == booking.OwnerNIC);

if (reminderExists)
{
    continue; // Reminder already sent
}

// Get station details

var station = await stationRepository.GetByIdAsync(booking.StationId);

var stationName = station?.Name ?? "Charging Station";

// Send reminder notification

await notificationService.CreateBookingReminderNotificationAsync(
    booking.OwnerNIC,
    booking.Id!,
    stationName,
    booking.StartTime
);

_logger.LogInformation("Sent booking reminder for booking {BookingId} to user
{OwnerNIC}",
    booking.Id, booking.OwnerNIC);
}
catch (Exception ex)

```

```

    {
        _logger.LogError(ex, "Failed to send reminder for booking {BookingId}", booking.Id);
    }
}

}

catch (Exception ex)
{
    _logger.LogError(ex, "Error in SendBookingRemindersAsync");
}
}

/// <summary>
/// Cleans up expired notifications
/// </summary>
/// <param name="serviceProvider">Service provider</param>

private async Task CleanupExpiredNotificationsAsync(IServiceProvider serviceProvider)
{
    try
    {
        var notificationService = serviceProvider.GetRequiredService<INotificationService>();

        var deletedCount = await notificationService.CleanupExpiredNotificationsAsync();

        if (deletedCount > 0)
        {
            _logger.LogInformation("Cleaned up {Count} expired notifications", deletedCount);
        }
    }
}

catch (Exception ex)

```

```

    {
        _logger.LogError(ex, "Error in CleanupExpiredNotificationsAsync");
    }
}

}

/*
 * File: NotificationService.cs
 * Project: EV Charging Station Booking System
 * Description: Service implementation for notification operations
 * Author: EV Charging System
 * Date: October 7, 2025
 */

```

```

using EVChargingBackend.DTOs;
using EVChargingBackend.Models;
using EVChargingBackend.Repositories;

namespace EVChargingBackend.Services
{
    /// <summary>
    /// Service implementation for notification operations
    /// </summary>
    public class NotificationService : INotificationService
    {
        private readonly INotificationRepository _notificationRepository;
        private readonly ILogger<NotificationService> _logger;

```

```

/// <summary>
/// Initializes notification service with dependencies
/// </summary>

/// <param name="notificationRepository">Notification repository</param>
/// <param name="logger">Logger</param>

public NotificationService(
    INotificationRepository notificationRepository,
    ILogger<NotificationService> logger)
{
    _notificationRepository = notificationRepository;
    _logger = logger;
}

/// <summary>
/// Creates a notification
/// </summary>

/// <param name="createDto">Notification creation data</param>
/// <returns>Created notification</returns>

public async Task<NotificationResponseDto> CreateNotificationAsync(CreateNotificationDto
createDto)
{
    var notification = new Notification
    {
        RecipientNIC = createDto.RecipientNIC,
        Title = createDto.Title,
        Message = createDto.Message,
        Type = createDto.Type,
        RelatedEntityId = createDto.RelatedEntityId,
        RelatedEntityType = createDto.RelatedEntityType,
    };
}

```

```

        Priority = createDto.Priority,
        ExpiresAt = createDto.ExpiresAt,
        Metadata = createDto.Metadata
    };

    var createdNotification = await _notificationRepository.CreateAsync(notification);

    _logger.LogInformation("Created notification {NotificationId} for user {RecipientNIC}",
        createdNotification.Id, createDto.RecipientNIC);

    return MapToResponseDto(createdNotification);
}

/// <summary>
/// Creates bulk notifications for multiple recipients
/// </summary>
/// <param name="bulkDto">Bulk notification data</param>
/// <returns>List of created notifications</returns>

public async Task<List<NotificationResponseDto>> CreateBulkNotificationAsync(BulkNotificationDto
bulkDto)
{
    var notifications = bulkDto.RecipientNICs.Select(nic => new Notification
    {
        RecipientNIC = nic,
        Title = bulkDto.Title,
        Message = bulkDto.Message,
        Type = bulkDto.Type,
        Priority = bulkDto.Priority,
        ExpiresAt = bulkDto.ExpiresAt,
    });
}

```

```

        Metadata = bulkDto.Metadata
    }).ToList();

    var createdNotifications = await _notificationRepository.CreateManyAsync(notifications);

    _logger.LogInformation("Created {Count} bulk notifications", createdNotifications.Count);

    return createdNotifications.Select(MapToResponseDto).ToList();
}

/// <summary>
/// Creates a booking confirmation notification
/// </summary>
/// <param name="recipientNIC">Recipient's NIC</param>
/// <param name="bookingId">Booking ID</param>
/// <param name="stationName">Charging station name</param>
/// <param name="startTime">Booking start time</param>
/// <param name="endTime">Booking end time</param>
/// <returns>Created notification</returns>

public async Task<NotificationResponseDto> CreateBookingConfirmationNotificationAsync(
    string recipientNIC, string bookingId, string stationName, DateTime startTime, DateTime endTime)
{
    var title = "Booking Confirmed";
    var message = $"Your booking at {stationName} has been confirmed for {startTime:yyyy-MM-dd HH:mm} - {endTime:yyyy-MM-dd HH:mm}. Your charging session is ready!";
}

var createDto = new CreateNotificationDto
{
    RecipientNIC = recipientNIC,

```

```

        Title = title,
        Message = message,
        Type = NotificationType.BookingConfirmation,
        RelatedEntityId = bookingId,
        RelatedEntityType = nameof(Booking),
        Priority = NotificationPriority.High,
        Metadata = new Dictionary<string, object>
    {
        ["stationName"] = stationName,
        ["startTime"] = startTime,
        ["endTime"] = endTime,
        ["bookingId"] = bookingId
    }
};

return await CreateNotificationAsync(createDto);
}

/// <summary>
/// Creates a booking cancellation notification
/// </summary>
/// <param name="recipientNIC">Recipient's NIC</param>
/// <param name="bookingId">Booking ID</param>
/// <param name="stationName">Charging station name</param>
/// <param name="reason">Cancellation reason</param>
/// <returns>Created notification</returns>
public async Task<NotificationResponseDto> CreateBookingCancellationNotificationAsync(
    string recipientNIC, string bookingId, string stationName, string reason = "") {
}

```

```

var title = "Booking Cancelled";

var message = $"Your booking at {stationName} has been cancelled";

if (!string.IsNullOrEmpty(reason))
{
    message += $. Reason: {reason}";

}

message += ". You can make a new booking anytime./";

var createDto = new CreateNotificationDto
{
    RecipientNIC = recipientNIC,
    Title = title,
    Message = message,
    Type = NotificationType.BookingCancellation,
    RelatedEntityId = bookingId,
    RelatedEntityType = nameof(Booking),
    Priority = NotificationPriority.High,
    Metadata = new Dictionary<string, object>
    {
        ["stationName"] = stationName,
        ["bookingId"] = bookingId,
        ["reason"] = reason ?? ""
    }
};

return await CreateNotificationAsync(createDto);
}

```

```

/// <summary>
/// Creates a booking reminder notification
/// </summary>

/// <param name="recipientNIC">Recipient's NIC</param>
/// <param name="bookingId">Booking ID</param>
/// <param name="stationName">Charging station name</param>
/// <param name="startTime">Booking start time</param>
/// <returns>Created notification</returns>

public async Task<NotificationResponseDto> CreateBookingReminderNotificationAsync(
    string recipientNIC, string bookingId, string stationName, DateTime startTime)
{
    var title = "Booking Reminder";
    var message = $"Reminder: Your charging session at {stationName} starts at {startTime:yyyy-MM-dd HH:mm}. Don't forget to arrive on time!";

    var createDto = new CreateNotificationDto
    {
        RecipientNIC = recipientNIC,
        Title = title,
        Message = message,
        Type = NotificationType.BookingReminder,
        RelatedEntityId = bookingId,
        RelatedEntityType = nameof(Booking),
        Priority = NotificationPriority.Normal,
        ExpiresAt = startTime.AddHours(2), // Reminder expires 2 hours after start time
        Metadata = new Dictionary<string, object>
        {
            ["stationName"] = stationName,

```

```

        ["startTime"] = startTime,
        ["bookingId"] = bookingId
    }

};

return await CreateNotificationAsync(createDto);
}

/// <summary>
/// Gets all notifications for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <param name="includeRead">Include read notifications</param>
/// <param name="limit">Maximum number of notifications</param>
/// <param name="offset">Number of notifications to skip</param>
/// <returns>List of notifications</returns>

public async Task<List<NotificationResponseDto>> GetUserNotificationsAsync(
    string recipientNIC, bool includeRead = true, int limit = 50, int offset = 0)
{
    var notifications = await _notificationRepository.GetByRecipientAsync(
        recipientNIC, includeRead, limit, offset);

    return notifications.Select(MapToResponseDto).ToList();
}

/// <summary>
/// Gets unread notifications for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>

```

```

/// <returns>List of unread notifications</returns>

public async Task<List<NotificationResponseDto>> GetUnreadNotificationsAsync(string recipientNIC)
{
    var notifications = await _notificationRepository.GetUnreadByRecipientAsync(recipientNIC);
    return notifications.Select(MapToResponseDto).ToList();
}

/// <summary>
/// Gets notifications by type for a user
/// </summary>

/// <param name="recipientNIC">User NIC</param>
/// <param name="type">Notification type</param>
/// <param name="limit">Maximum number of notifications</param>
/// <returns>List of notifications</returns>

public async Task<List<NotificationResponseDto>> GetNotificationsByTypeAsync(
    string recipientNIC, NotificationType type, int limit = 20)
{
    var notifications = await _notificationRepository.GetByRecipientAndTypeAsync(
        recipientNIC, type, limit);

    return notifications.Select(MapToResponseDto).ToList();
}

/// <summary>
/// Gets notification by ID
/// </summary>

/// <param name="id">Notification ID</param>
/// <returns>Notification if found</returns>

public async Task<NotificationResponseDto> GetNotificationByIdAsync(string id)

```

```

{
    var notification = await _notificationRepository.GetByIdAsync(id);

    if (notification == null)
    {
        throw new KeyNotFoundException("Notification not found");
    }

    return MapToResponseDto(notification);
}

/// <summary>
/// Marks notifications as read
/// </summary>
/// <param name="notificationIds">List of notification IDs</param>
/// <param name="userNIC">User NIC for validation</param>
/// <returns>Number of notifications marked as read</returns>
public async Task<int> MarkNotificationsAsReadAsync(List<string> notificationIds, string userNIC)
{
    // Validate that all notifications belong to the user
    foreach (var id in notificationIds)
    {
        var notification = await _notificationRepository.GetByIdAsync(id);

        if (notification == null || notification.RecipientNIC != userNIC)
        {
            throw new UnauthorizedAccessException($"Notification {id} not found or access denied");
        }
    }
}

```

```

var markedCount = await _notificationRepository.MarkMultipleAsReadAsync(notificationIds);

_logger.LogInformation("Marked {Count} notifications as read for user {UserNIC}",
    markedCount, userNIC);

return markedCount;
}

/// <summary>
/// Marks all notifications for a user as read
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <returns>Number of notifications marked as read</returns>
public async Task<int> MarkAllAsReadAsync(string recipientNIC)
{
    var markedCount = await _notificationRepository.MarkAllAsReadAsync(recipientNIC);

    _logger.LogInformation("Marked all {Count} notifications as read for user {RecipientNIC}",
        markedCount, recipientNIC);

    return markedCount;
}

/// <summary>
/// Deletes a notification (soft delete by setting expiry)
/// </summary>
/// <param name="id">Notification ID</param>
/// <param name="userNIC">User NIC for validation</param>
/// <returns>Success status</returns>

```

```

public async Task<bool> DeleteNotificationAsync(string id, string userNIC)
{
    var notification = await _notificationRepository.GetByIdAsync(id);

    if (notification == null || notification.RecipientNIC != userNIC)
    {
        throw new UnauthorizedAccessException("Notification not found or access denied");
    }

    var result = await _notificationRepository.DeleteAsync(id);

    if (result)
    {
        _logger.LogInformation("Deleted notification {NotificationId} for user {UserNIC}", id, userNIC);
    }
}

return result;
}

/// <summary>
/// Gets notification summary for a user
/// </summary>
/// <param name="recipientNIC">User NIC</param>
/// <returns>Notification summary</returns>

public async Task<NotificationSummaryDto> GetNotificationSummaryAsync(string recipientNIC)
{
    var totalCount = await _notificationRepository.GetNotificationCountAsync(recipientNIC, true);
    var unreadCount = await _notificationRepository.GetUnreadCountAsync(recipientNIC);
}

```

```

        var unreadNotifications = await
    _notificationRepository.GetUnreadByRecipientAsync(recipientNIC);

        var highPriorityCount = unreadNotifications.Count(n => n.Priority == NotificationPriority.High);
        var criticalCount = unreadNotifications.Count(n => n.Priority == NotificationPriority.Critical);

        var latestNotification = unreadNotifications.OrderByDescending(n =>
n.CreatedAt).FirstOrDefault();

        return new NotificationSummaryDto
    {
        TotalNotifications = totalCount,
        UnreadNotifications = unreadCount,
        HighPriorityNotifications = highPriorityCount,
        CriticalNotifications = criticalCount,
        LastNotificationTime = latestNotification?.CreatedAt
    };
}

/// <summary>
/// Cleans up expired notifications
/// </summary>
/// <returns>Number of deleted notifications</returns>
public async Task<int> CleanupExpiredNotificationsAsync()
{
    var deletedCount = await _notificationRepository.DeleteExpiredNotificationsAsync();

    _logger.LogInformation("Cleaned up {Count} expired notifications", deletedCount);

    return deletedCount;
}

```

```

/// <summary>
/// Maps notification entity to response DTO
/// </summary>

/// <param name="notification">Notification entity</param>
/// <returns>Notification response DTO</returns>

private static NotificationResponseDto MapToResponseDto(Notification notification)
{
    return new NotificationResponseDto
    {
        Id = notification.Id ?? "",
        RecipientNIC = notification.RecipientNIC,
        Title = notification.Title,
        Message = notification.Message,
        Type = notification.Type,
        RelatedEntityId = notification.RelatedEntityId,
        RelatedEntityType = notification.RelatedEntityType,
        IsRead = notification.IsRead,
        IsDelivered = notification.IsDelivered,
        Priority = notification.Priority,
        CreatedAt = notification.CreatedAt,
        ReadAt = notification.ReadAt,
        DeliveredAt = notification.DeliveredAt,
        ExpiresAt = notification.ExpiresAt,
        Metadata = notification.Metadata
    };
}
}

```

```
/*
 * File: SeedDataService.cs
 * Project: EV Charging Station Booking System
 * Description: Service for seeding initial data into the database
 * Author: EV Charging System
 * Date: September 27, 2025
 */
```

```
using EVChargingBackend.Models;
using EVChargingBackend.Repositories;
using EVChargingBackend.Config;
using MongoDB.Driver;

namespace EVChargingBackend.Services
{
    /// <summary>
    /// Service implementation for seeding initial data into the database
    /// </summary>
    public class SeedDataService : ISeedDataService
    {
        private readonly IUserRepository _userRepository;
        private readonly IChargingStationRepository _stationRepository;
        private readonly IBookingRepository _bookingRepository;
        private readonly DbContext _context;
        private readonly ILogger<SeedDataService> _logger;

        /// <summary>
        /// Initializes seed data service with repositories
        /// </summary>
```

```

/// </summary>

/// <param name="userRepository">User repository</param>
/// <param name="stationRepository">Station repository</param>
/// <param name="bookingRepository">Booking repository</param>
/// <param name="context">MongoDB context</param>
/// <param name="logger">Logger for service operations</param>

public SeedDataService(
    I UserRepository userRepository,
    I ChargingStationRepository stationRepository,
    I BookingRepository bookingRepository,
    MongoDbContext context,
    ILogger<SeedDataService> logger)
{
    _userRepository = userRepository;
    _stationRepository = stationRepository;
    _bookingRepository = bookingRepository;
    _context = context;
    _logger = logger;
}

/// <summary>
/// Seeds initial data including users, stations, and sample bookings
/// </summary>

/// <returns>Task representing the async operation</returns>

public async Task SeedDataAsync()
{
    try
    {
        _logger.LogInformation("Starting database seeding...");
    }

```

```

// Handle potential schema mismatch by clearing corrupted collections

try
{
    var existingUsers = await _userRepository.GetAllAsync();
    if (existingUsers.Any())
    {
        _logger.LogInformation("Valid data already exists, skipping seed");
        return;
    }
}

catch (Exception ex)
{
    _logger.LogWarning(ex, "Schema mismatch detected. Clearing collections and reseeding...");
}

// Clear potentially corrupted collections
await ClearCorruptedCollectionsAsync();
_logger.LogInformation("Cleared corrupted collections. Proceeding with fresh seed...");
}

// Seed Users
await SeedUsersAsync();

// Seed Charging Stations
await SeedChargingStationsAsync();

// Seed Sample Bookings
await SeedBookingsAsync();

```

```

        _logger.LogInformation("Database seeding completed successfully");

    }

    catch (Exception ex)
    {

        _logger.LogError(ex, "Error occurred during database seeding");

        throw new InvalidOperationException("An error occurred in SeedDataService.SeedDataAsync.
See inner exception for details.", ex);
    }
}

/// <summary>
/// Seeds initial user accounts
/// </summary>

private async Task SeedUsersAsync()
{
    var users = new[]
    {
        new User
        {
            NIC = "123456789V",
            FirstName = "Admin",
            LastName = "User",
            Email = "admin@evcharging.com",
            Password = BCrypt.Net.BCrypt.HashPassword("Admin123!"),
            Role = UserRole.Backoffice,
            PhoneNumber = "+94771234567",
            IsActive = true
        },
        new User
    };
}

```

```
{  
    NIC = "987654321V",  
    FirstName = "Station",  
    LastName = "Operator",  
    Email = "operator@evcharging.com",  
    Password = BCrypt.Net.BCrypt.HashPassword("Operator123!"),  
    Role = UserRole.StationOperator,  
    PhoneNumber = "+94777654321",  
    IsActive = true  
},  
new User  
{  
    NIC = "456789123V",  
    FirstName = "John",  
    LastName = "Doe",  
    Email = "john.doe@example.com",  
    Password = BCrypt.Net.BCrypt.HashPassword("John123!"),  
    Role = UserRole.EVOwner,  
    PhoneNumber = "+94774567891",  
    IsActive = true  
},  
new User  
{  
    NIC = "789123456V",  
    FirstName = "Jane",  
    LastName = "Smith",  
    Email = "jane.smith@example.com",  
    Password = BCrypt.Net.BCrypt.HashPassword("Jane123!"),  
    Role = UserRole.EVOwner,
```

```

        PhoneNumber = "+94777891234",
        IsActive = true
    },
    new User
    {
        NIC = "321654987V",
        FirstName = "Bob",
        LastName = "Wilson",
        Email = "bob.wilson@example.com",
        Password = BCrypt.Net.BCrypt.HashPassword("Bob123!"),
        Role = UserRole.EVOwner,
        PhoneNumber = "+94773216549",
        IsActive = false // Deactivated user
    }
};

foreach (var user in users)
{
    await _userRepository.CreateAsync(user);
    _logger.LogInformation("Seeded user: {Email}", user.Email);
}

/// <summary>
/// Seeds initial charging stations
/// </summary>
private async Task SeedChargingStationsAsync()
{
    var stations = new[]

```

```
{  
    new ChargingStation  
    {  
        Name = "City Center DC Fast Charger",  
        Location = "Colombo City Center, Main Street",  
        Type = StationType.DC,  
        TotalSlots = 4,  
        AvailableSlots = 3,  
        Status = StationStatus.Active,  
        PricePerHour = 500.00m  
    },  
    new ChargingStation  
    {  
        Name = "Shopping Mall AC Charger",  
        Location = "Kandy Shopping Complex, Level B1",  
        Type = StationType.AC,  
        TotalSlots = 8,  
        AvailableSlots = 6,  
        Status = StationStatus.Active,  
        PricePerHour = 300.00m  
    },  
    new ChargingStation  
    {  
        Name = "Highway Rest Stop Charger",  
        Location = "Southern Expressway, Rest Area 1",  
        Type = StationType.DC,  
        TotalSlots = 6,  
        AvailableSlots = 6,  
        Status = StationStatus.Active,
```

```

        PricePerHour = 600.00m
    },
    new ChargingStation
    {
        Name = "University Campus Charger",
        Location = "University of Colombo, Parking Area C",
        Type = StationType.AC,
        TotalSlots = 12,
        AvailableSlots = 10,
        Status = StationStatus.Active,
        PricePerHour = 250.00m
    },
    new ChargingStation
    {
        Name = "Business District Charger",
        Location = "World Trade Center, Underground Parking",
        Type = StationType.DC,
        TotalSlots = 3,
        AvailableSlots = 3,
        Status = StationStatus.Maintenance,
        PricePerHour = 550.00m
    }
};

foreach (var station in stations)
{
    await _stationRepository.CreateAsync(station);
    _logger.LogInformation("Seeded charging station: {Name}", station.Name);
}

```

```

}

/// <summary>
/// Seeds sample bookings
/// </summary>

private async Task SeedBookingsAsync()
{
    // Get seeded data for references
    var users = await _userRepository.GetAllAsync();
    var stations = await _stationRepository.GetAllAsync();

    var evOwners = users.Where(u => u.Role == UserRole.EVOwner && u.IsActive).ToList();
    var activeStations = stations.Where(s => s.Status == StationStatus.Active).ToList();

    if (!evOwners.Any() || !activeStations.Any())
    {
        _logger.LogWarning("No EV owners or active stations found for booking seeding");
        return;
    }

    var bookings = new[]
    {
        new Booking
        {
            OwnerNIC = evOwners[0].NIC, // John Doe
            StationId = activeStations[0].Id!, // City Center DC Fast Charger
            StartTime = DateTime.UtcNow.AddHours(2),
            EndTime = DateTime.UtcNow.AddHours(4),
            Status = BookingStatus.Active,
        }
    };
}

```

```

QRCode = Guid.NewGuid().ToString("N").ToUpper(),

TotalAmount = 1000.00m

},

new Booking

{

    OwnerNIC = evOwners[1].NIC, // Jane Smith

    StationId = activeStations[1].Id!, // Shopping Mall AC Charger

    StartTime = DateTime.UtcNow.AddDays(1),

    EndTime = DateTime.UtcNow.AddDays(1).AddHours(3),

    Status = BookingStatus.Confirmed,

    QRCode = Guid.NewGuid().ToString("N").ToUpper(),

    TotalAmount = 900.00m,

    ConfirmedAt = DateTime.UtcNow.AddMinutes(-30)

},

new Booking

{

    OwnerNIC = evOwners[0].NIC, // John Doe

    StationId = activeStations[2].Id!, // Highway Rest Stop Charger

    StartTime = DateTime.UtcNow.AddDays(-1),

    EndTime = DateTime.UtcNow.AddDays(-1).AddHours(2),

    Status = BookingStatus.Completed,

    QRCode = Guid.NewGuid().ToString("N").ToUpper(),

    TotalAmount = 1200.00m,

    ConfirmedAt = DateTime.UtcNow.AddDays(-1).AddMinutes(-30)

},

new Booking

{

    OwnerNIC = evOwners[1].NIC, // Jane Smith

    StationId = activeStations[3].Id!, // University Campus Charger

```

```

        StartTime = DateTime.UtcNow.AddHours(6),
        EndTime = DateTime.UtcNow.AddHours(8),
        Status = BookingStatus.Cancelled,
        QRCode = Guid.NewGuid().ToString("N").ToUpper(),
        TotalAmount = 500.00m,
        CancelledAt = DateTime.UtcNow.AddMinutes(-15)
    }

};

foreach (var booking in bookings)
{
    await _bookingRepository.CreateAsync(booking);
    _logger.LogInformation("Seeded booking: {BookingId} for station {StationId}", booking.Id,
    booking.StationId);
}

/// <summary>
/// Clears corrupted collections that have schema mismatches
/// </summary>
private async Task ClearCorruptedCollectionsAsync()
{
    try
    {
        _logger.LogWarning("Clearing potentially corrupted MongoDB collections...");

        // Drop collections that might have schema issues
        await _context.Database.DropCollectionAsync("Users");
        await _context.Database.DropCollectionAsync("ChargingStations");
    }
}

```

```

        await _context.Database.DropCollectionAsync("Bookings");

        await _context.Database.DropCollectionAsync("Notifications");

        _logger.LogInformation("Successfully cleared corrupted collections");

    }

    catch (Exception ex)

    {

        _logger.LogError(ex, "Error clearing corrupted collections");

        // Continue anyway - collections might not exist

    }

}

}

}

}

/*



* File: UserService.cs

* Project: EV Charging Station Booking System

* Description: User management service implementation

* Author: EV Charging System

* Date: September 27, 2025

*/



using EVChargingBackend.DTOs;

using EVChargingBackend.Models;

using EVChargingBackend.Repositories;

namespace EVChargingBackend.Services

{
    /// <summary>

```

```

/// Service implementation for user management operations

/// </summary>

public class UserService : IUserService
{
    private readonly IUserRepository _userRepository;
    private readonly ILogger<UserService> _logger;

    /// <summary>
    /// Initializes user service with dependencies
    /// </summary>

    /// <param name="userRepository">User repository for data operations</param>
    /// <param name="logger">Logger for service operations</param>

    public UserService(IUserRepository userRepository, ILogger<UserService> logger)
    {
        _userRepository = userRepository;
        _logger = logger;
    }

    /// <summary>
    /// Gets all users (backoffice only)
    /// </summary>

    /// <returns>List of all users</returns>

    public async Task<List<UserResponseDto>> GetAllUsersAsync()
    {
        var users = await _userRepository.GetAllAsync();
        return users.Select(MapToUserResponseDto).ToList();
    }

    /// <summary>

```

```

/// Gets user by ID

/// </summary>

/// <param name="id">User ID</param>

/// <returns>User if found</returns>

public async Task<UserResponseDto> GetUserByIdAsync(string id)

{

    var user = await _userRepository.GetByldAsync(id);

    if (user == null)

    {

        throw new KeyNotFoundException("User not found");

    }

    return MapToUserResponseDto(user);

}

/// <summary>

/// Updates user information

/// </summary>

/// <param name="id">User ID</param>

/// <param name="updateDto">Updated user data</param>

/// <returns>Updated user</returns>

public async Task<UserResponseDto> UpdateUserAsync(string id, UpdateUserDto updateDto)

{

    var user = await _userRepository.GetByldAsync(id);

    if (user == null)

    {

        throw new KeyNotFoundException("User not found");

    }

}

```

```

// Update only provided fields

if (!string.IsNullOrEmpty(updateDto.FirstName))

    user.FirstName = updateDto.FirstName;

if (!string.IsNullOrEmpty(updateDto.LastName))

    user.LastName = updateDto.LastName;

if (!string.IsNullOrEmpty(updateDto.Email))

{

    // Check if new email is already in use

    var existingUser = await _userRepository.GetByEmailAsync(updateDto.Email);

    if (existingUser != null && existingUser.Id != id)

    {

        throw new ArgumentException("Email is already in use");

    }

    user.Email = updateDto.Email;

}

if (updateDto.PhoneNumber != null)

    user.PhoneNumber = updateDto.PhoneNumber;

var updatedUser = await _userRepository.UpdateAsync(user);

_logger.LogInformation("User updated successfully: {UserId}", id);

return MapToUserResponseDto(updatedUser);

}

/// <summary>

```

```

/// Activates or deactivates a user (backoffice only)

/// </summary>

/// <param name="id">User ID</param>

/// <param name="isActive">Active status</param>

/// <returns>Success status</returns>

public async Task<bool> SetUserActiveStatusAsync(string id, bool isActive)

{

    var user = await _userRepository.GetByIdAsync(id);

    if (user == null)

    {

        throw new KeyNotFoundException("User not found");

    }

    var result = await _userRepository.SetActiveStatusAsync(id, isActive);

    if (result)

    {

        _logger.LogInformation("User {UserId} status changed to {Status}", id, isActive ? "Active" : "Inactive");

    }

    return result;

}

/// <summary>

/// Deactivates current user (EV Owner self-deactivation)

/// </summary>

/// <param name="id">User ID</param>

/// <returns>Success status</returns>

```

```
public async Task<bool> DeactivateUserAsync(string id)
{
    var user = await _userRepository.GetByAsync(id);
    if (user == null)
    {
        throw new KeyNotFoundException("User not found");
    }

    // Only EV Owners can deactivate themselves
    if (user.Role != UserRole.EVOwner)
    {
        throw new UnauthorizedAccessException("Only EV Owners can deactivate themselves");
    }

    var result = await _userRepository.SetActiveStatusAsync(id, false);

    if (result)
    {
        _logger.LogInformation("EV Owner {UserId} self-deactivated", id);
    }
}

return result;
}

/// <summary>
/// Deletes a user (backoffice only)
/// </summary>
/// <param name="id">User ID</param>
/// <returns>Success status</returns>
```

```

public async Task<bool> DeleteUserAsync(string id)
{
    var user = await _userRepository.GetByAsync(id);
    if (user == null)
    {
        throw new KeyNotFoundException("User not found");
    }

    var result = await _userRepository.DeleteAsync(id);

    if (result)
    {
        _logger.LogInformation("User deleted: {UserId}", id);
    }

    return result;
}

/// <summary>
/// Maps User entity to UserResponseDto
/// </summary>
/// <param name="user">User entity</param>
/// <returns>User response DTO</returns>

private static UserResponseDto MapToUserResponseDto(User user)
{
    return new UserResponseDto
    {
        Id = user.Id ?? string.Empty,
        NIC = user.NIC,
    };
}

```

```

        FirstName = user.FirstName,
        LastName = user.LastName,
        Email = user.Email,
        Role = user.Role,
        IsActive = user.IsActive,
        PhoneNumber = user.PhoneNumber,
        CreatedAt = user.CreatedAt
    };
}

}

}

/*
 * File: JwtSettings.cs
 *
 * Project: EV Charging Station Booking System
 *
 * Description: JWT authentication configuration settings
 *
 * Author: EV Charging System
 *
 * Date: September 27, 2025
 */

```

```

namespace EVChargingBackend.Config
{
    /// <summary>
    /// JWT token configuration for authentication
    /// </summary>
    public class JwtSettings
    {
        public string Key { get; set; } = string.Empty;
        public string Issuer { get; set; } = string.Empty;
    }
}

```

```

        public string Audience { get; set; } = string.Empty;
        public int ExpiryInMinutes { get; set; } = 60;
    }

}

/*
 * File: MongoDBContext.cs
 * Project: EV Charging Station Booking System
 * Description: MongoDB context for database operations
 * Author: EV Charging System
 * Date: September 27, 2025
 */

```

```

using EVChargingBackend.Models;
using Microsoft.Extensions.Options;
using MongoDB.Driver;

namespace EVChargingBackend.Config
{
    /// <summary>
    /// MongoDB database context providing access to collections
    /// </summary>
    public class MongoDBContext
    {
        private readonly IMongoDatabase _database;

        /// <summary>
        /// Initializes MongoDB context with connection settings
        /// </summary>

```

```

/// <param name="settings">MongoDB configuration settings</param>
public MongoDBContext(IOptions<MongoDbSettings> settings)
{
    var client = new MongoClient(settings.Value.ConnectionString);
    _database = client.GetDatabase(settings.Value.DatabaseName);
}

/// <summary>
/// Gets the Users collection
/// </summary>
public IMongoCollection<User> Users => _database.GetCollection<User>("Users");

/// <summary>
/// Gets the ChargingStations collection
/// </summary>
public IMongoCollection<ChargingStation> ChargingStations =>
    _database.GetCollection<ChargingStation>("ChargingStations");

/// <summary>
/// Gets the Bookings collection
/// </summary>
public IMongoCollection<Booking> Bookings => _database.GetCollection<Booking>("Bookings");

/// <summary>
/// Gets the Notifications collection
/// </summary>
public IMongoCollection<Notification> Notifications =>
    _database.GetCollection<Notification>("Notifications");

/// <summary>

```

```

    /// Gets the MongoDB database instance
    /// </summary>
    public IMongoDatabase Database => _database;
}

}

/*
* File: MongoDBSettings.cs
* Project: EV Charging Station Booking System
* Description: MongoDB configuration settings
* Author: EV Charging System
* Date: September 27, 2025
*/

```

```

namespace EVChargingBackend.Config
{
    /// <summary>
    /// MongoDB connection and database configuration
    /// </summary>
    public class MongoDBSettings
    {
        public string ConnectionString { get; set; } = string.Empty;
        public string DatabaseName { get; set; } = string.Empty;
    }
}

# EV Charging API - IIS Deployment Script (PowerShell)
# Run this script as Administrator for best results

param(

```

```

[string]$PublishPath = "C:\inetpub\wwwroot\EVChargingAPI",
[string]$SiteName = "EV Charging API",
[string]$AppPoolName = "EVChargingAPI",
[int]$Port = 5000
)

Write-Host "===== -ForegroundColor Green
Write-Host "EV Charging API - IIS Deployment Script" -ForegroundColor Green
Write-Host "===== -ForegroundColor Green
Write-Host ""

$ProjectPath = $PSScriptRoot
Write-Host "Project Path: $ProjectPath" -ForegroundColor Yellow
Write-Host "Publish Path: $PublishPath" -ForegroundColor Yellow
Write-Host ""

try {
    # Step 1: Clean and Build
    Write-Host "[1/6] Cleaning previous build..." -ForegroundColor Cyan
    & dotnet clean -c Release
    if ($LASTEXITCODE -ne 0) { throw "Failed to clean project" }

    Write-Host "[2/6] Restoring packages..." -ForegroundColor Cyan
    & dotnet restore
    if ($LASTEXITCODE -ne 0) { throw "Failed to restore packages" }

    # Step 2: Publish
    Write-Host "[3/6] Publishing application..." -ForegroundColor Cyan
    & dotnet publish -c Release -o $PublishPath --no-restore
    if ($LASTEXITCODE -ne 0) { throw "Failed to publish application" }

```

```

# Step 3: Copy environment file

Write-Host "[4/6] Copying environment file..." -ForegroundColor Cyan

if (Test-Path ".env") {

    Copy-Item ".env" "$PublishPath\.env" -Force

    Write-Host "Environment file copied successfully" -ForegroundColor Green

} else {

    Write-Host "WARNING: .env file not found" -ForegroundColor Yellow

}

# Step 4: Configure IIS (if running as admin)

$isAdmin = ([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]
"Administrator")

if ($isAdmin) {

    Write-Host "[5/6] Configuring IIS..." -ForegroundColor Cyan

    # Import WebAdministration module

    Import-Module WebAdministration -ErrorAction SilentlyContinue

    if (Get-Module -Name WebAdministration) {

        # Create Application Pool

        if (Get-WebAppPool -Name $AppPoolName -ErrorAction SilentlyContinue) {

            Write-Host "Application Pool '$AppPoolName' already exists" -ForegroundColor Yellow

        } else {

            New-WebAppPool -Name $AppPoolName

            Set-WebAppPool -Name $AppPoolName -processModel.identityType ApplicationPoolIdentity

            Set-WebAppPool -Name $AppPoolName -managedRuntimeVersion ""

            Write-Host "Application Pool '$AppPoolName' created" -ForegroundColor Green

        }

    }

}

```

```

}

# Create Website

if (Get-WebSite -Name $SiteName -ErrorAction SilentlyContinue) {

    Write-Host "Website '$SiteName' already exists - updating..." -ForegroundColor Yellow

    Set-WebSite -Name $SiteName -PhysicalPath $PublishPath

} else {

    New-WebSite -Name $SiteName -Port $Port -PhysicalPath $PublishPath -ApplicationPool
$AppPoolName

    Write-Host "Website '$SiteName' created" -ForegroundColor Green

}

# Start the site

Start-WebSite -Name $SiteName

Write-Host "Website started" -ForegroundColor Green

} else {

    Write-Host "IIS WebAdministration module not available - configure manually" -ForegroundColor
Yellow

}

} else {

    Write-Host "[5/6] Skipping IIS configuration (requires Administrator privileges)" -ForegroundColor
Yellow

}

# Step 5: Configure Firewall

Write-Host "[6/6] Configuring Windows Firewall..." -ForegroundColor Cyan

if ($isAdmin) {

    $ruleName = "EV Charging API Port $Port"

    $existingRule = Get-NetFirewallRule -DisplayName $ruleName -ErrorAction SilentlyContinue
}

```

```

if (-not $existingRule) {

    New-NetFirewallRule -DisplayName $ruleName -Direction Inbound -Protocol TCP -LocalPort $Port -
Action Allow

    Write-Host "Firewall rule created for port $Port" -ForegroundColor Green

} else {

    Write-Host "Firewall rule already exists" -ForegroundColor Yellow

}

} else {

    Write-Host "Run as Administrator to configure firewall automatically" -ForegroundColor Yellow

}

Write-Host ""

Write-Host "===== -ForegroundColor Green

Write-Host "Deployment completed successfully!" -ForegroundColor Green

Write-Host "===== -ForegroundColor Green

Write-Host ""

Write-Host "Application URL: http://localhost:$Port" -ForegroundColor Cyan

Write-Host "Swagger UI: http://localhost:$Port/" -ForegroundColor Cyan

Write-Host ""

Write-Host "Manual steps (if not running as Administrator):" -ForegroundColor Yellow

Write-Host "1. Open IIS Manager (inetmgr)" -ForegroundColor White

Write-Host "2. Create Application Pool: $AppPoolName (No Managed Code)" -ForegroundColor White

Write-Host "3. Create Website: $SiteName -> $PublishPath" -ForegroundColor White

Write-Host "4. Allow port $Port in Windows Firewall" -ForegroundColor White

}

} catch {

    Write-Host ""

    Write-Host "ERROR: $($_.Exception.Message)" -ForegroundColor Red

    Write-Host "Deployment failed!" -ForegroundColor Red
}

```

```

exit 1

}

Write-Host ""

Write-Host "Press any key to continue..." -ForegroundColor Gray

$null = $Host.UI.RawUI.ReadKey("NoEcho,IncludeKeyDown")

/*
 * File: BookingDTOs.cs
 *
 * Project: EV Charging Station Booking System
 *
 * Description: Data Transfer Objects for Booking operations
 *
 * Author: EV Charging System
 *
 * Date: September 27, 2025
 */

using EVChargingBackend.Models;

using System.ComponentModel.DataAnnotations;

namespace EVChargingBackend.DTOs
{
    /// <summary>
    /// DTO for creating a new booking
    /// </summary>
    public class CreateBookingDto
    {
        [Required]
        public string StationId { get; set; } = string.Empty;

        [Required]
    }
}

```

```

public DateTime StartTime { get; set; }

[Required]

public DateTime EndTime { get; set; }

}

/// <summary>
/// DTO for updating booking information
/// </summary>

public class UpdateBookingDto

{
    public DateTime? StartTime { get; set; }

    public DateTime? EndTime { get; set; }

    public BookingStatus? Status { get; set; }

}

/// <summary>
/// DTO for booking response with station details
/// </summary>

public class BookingResponseDto

{
    public string Id { get; set; } = string.Empty;

    public string OwnerNIC { get; set; } = string.Empty;

    public string StationId { get; set; } = string.Empty;

    public string StationName { get; set; } = string.Empty;

    public string StationLocation { get; set; } = string.Empty;

    public DateTime StartTime { get; set; }

    public DateTime EndTime { get; set; }

    public BookingStatus Status { get; set; }
}

```

```

        public string QRCode { get; set; } = string.Empty;
        public decimal TotalAmount { get; set; }
        public DateTime CreatedAt { get; set; }
        public DateTime? ConfirmedAt { get; set; }
        public DateTime? CancelledAt { get; set; }
    }

/// <summary>
/// DTO for booking confirmation by station operator
/// </summary>
public class ConfirmBookingDto
{
    [Required]
    public string BookingId { get; set; } = string.Empty;
}

/// <summary>
/// DTO for booking cancellation by operator
/// </summary>
public class CancelBookingByOperatorDto
{
    [StringLength(500)]
    public string Reason { get; set; } = string.Empty;
}

/*
 * File: ChargingStationDTOs.cs
 * Project: EV Charging Station Booking System

```

```
* Description: Data Transfer Objects for Charging Station operations  
* Author: EV Charging System  
* Date: September 27, 2025  
*/
```

```
using EVChargingBackend.Models;  
using System.ComponentModel.DataAnnotations;  
  
namespace EVChargingBackend.DTOs  
{  
    /// <summary>  
    /// DTO for creating a new charging station  
    /// </summary>  
    public class CreateStationDto  
    {  
        [Required]  
        public string Name { get; set; } = string.Empty;  
  
        [Required]  
        public string Location { get; set; } = string.Empty;  
  
        [Required]  
        public StationType Type { get; set; }  
  
        [Required]  
        [Range(1, 50)]  
        public int TotalSlots { get; set; }  
  
        [Required]
```

```

[Range(0.01, 1000.00)]

public decimal PricePerHour { get; set; }

}

/// <summary>
/// DTO for updating charging station information
/// </summary>

public class UpdateStationDto
{
    public string? Name { get; set; }

    public string? Location { get; set; }

    public StationType? Type { get; set; }

    public int? TotalSlots { get; set; }

    public decimal? PricePerHour { get; set; }

    public StationStatus? Status { get; set; }

}

/// <summary>
/// DTO for charging station response
/// </summary>

public class StationResponseDto
{
    public string Id { get; set; } = string.Empty;

    public string Name { get; set; } = string.Empty;

    public string Location { get; set; } = string.Empty;

    public StationType Type { get; set; }

    public int TotalSlots { get; set; }

    public int AvailableSlots { get; set; }

    public StationStatus Status { get; set; }

```

```

        public decimal PricePerHour { get; set; }

        public DateTime CreatedAt { get; set; }

        public DateTime UpdatedAt { get; set; }

    }

}

/*
 * File: NotificationDTOs.cs
 * Project: EV Charging Station Booking System
 * Description: Data Transfer Objects for notification operations
 * Author: EV Charging System
 * Date: October 7, 2025
 */

```

```

using EVChargingBackend.Models;

using System.ComponentModel.DataAnnotations;

namespace EVChargingBackend.DTOs
{
    /// <summary>
    /// DTO for creating notifications
    /// </summary>
    public class CreateNotificationDto
    {
        [Required]
        public string RecipientNIC { get; set; } = string.Empty;

        [Required]
        [StringLength(200)]

```

```
public string Title { get; set; } = string.Empty;

[Required]
[StringLength(1000)]

public string Message { get; set; } = string.Empty;

[Required]

public NotificationType Type { get; set; }

public string? RelatedEntityId { get; set; }

public string? RelatedEntityType { get; set; }

public NotificationPriority Priority { get; set; } = NotificationPriority.Normal;

public DateTime? ExpiresAt { get; set; }

public Dictionary<string, object>? Metadata { get; set; }

}

/// <summary>
/// DTO for notification responses
/// </summary>

public class NotificationResponseDto

{

    public string Id { get; set; } = string.Empty;

    public string RecipientNIC { get; set; } = string.Empty;

    public string Title { get; set; } = string.Empty;

    public string Message { get; set; } = string.Empty;
```

```

public NotificationType Type { get; set; }

public string? RelatedEntityId { get; set; }

public string? RelatedEntityType { get; set; }

public bool IsRead { get; set; }

public bool IsDelivered { get; set; }

public NotificationPriority Priority { get; set; }

public DateTime CreatedAt { get; set; }

public DateTime? ReadAt { get; set; }

public DateTime? DeliveredAt { get; set; }

public DateTime? ExpiresAt { get; set; }

public Dictionary<string, object>? Metadata { get; set; }

}

/// <summary>

/// DTO for updating notification read status

/// </summary>

public class MarkNotificationReadDto

{

    [Required]

    public List<string> NotificationIds { get; set; } = new List<string>();

}

/// <summary>

/// DTO for notification summary/statistics

/// </summary>

public class NotificationSummaryDto

{

    public int TotalNotifications { get; set; }

    public int UnreadNotifications { get; set; }

}

```

```
public int HighPriorityNotifications { get; set; }

public int CriticalNotifications { get; set; }

public DateTime? LastNotificationTime { get; set; }

}

/// <summary>
/// DTO for bulk notification creation
/// </summary>

public class BulkNotificationDto

{

    [Required]

    public List<string> RecipientNICs { get; set; } = new List<string>();

    [Required]

    [StringLength(200)]

    public string Title { get; set; } = string.Empty;

    [Required]

    [StringLength(1000)]

    public string Message { get; set; } = string.Empty;

    [Required]

    public NotificationType Type { get; set; }

    public NotificationPriority Priority { get; set; } = NotificationPriority.Normal;

    public DateTime? ExpiresAt { get; set; }

    public Dictionary<string, object>? Metadata { get; set; }
```

```
    }

}

/*
 * File: UserDTOs.cs
 * Project: EV Charging Station Booking System
 * Description: Data Transfer Objects for User operations
 * Author: EV Charging System
 * Date: September 27, 2025
 */

using EVChargingBackend.Models;
using System.ComponentModel.DataAnnotations;

namespace EVChargingBackend.DTOs
{
    /// <summary>
    /// DTO for user registration
    /// </summary>
    public class RegisterUserDto
    {
        [Required]
        public string NIC { get; set; } = string.Empty;

        [Required]
        public string FirstName { get; set; } = string.Empty;

        [Required]
        public string LastName { get; set; } = string.Empty;
    }
}
```

```
[Required]

[EmailAddress]

public string Email { get; set; } = string.Empty;

[Required]

[MinLength(6)]

public string Password { get; set; } = string.Empty;

[Required]

public UserRole Role { get; set; }

public string? StationId { get; set; }

public string? PhoneNumber { get; set; }

}

/// <summary>
/// DTO for user login
/// </summary>

public class LoginDto

{

    [Required]

    public string Email { get; set; } = string.Empty;

    [Required]

    public string Password { get; set; } = string.Empty;

}
```

```
/// <summary>
/// DTO for updating user information
/// </summary>

public class UpdateUserDto

{

    public string? FirstName { get; set; }

    public string? LastName { get; set; }

    public string? PhoneNumber { get; set; }

    public string? Email { get; set; }

}

/// <summary>
/// DTO for user response (excludes sensitive data)
/// </summary>

public class UserResponseDto

{

    public string Id { get; set; } = string.Empty;

    public string NIC { get; set; } = string.Empty;

    public string FirstName { get; set; } = string.Empty;

    public string LastName { get; set; } = string.Empty;

    public string Email { get; set; } = string.Empty;

    public UserRole Role { get; set; }

    public bool IsActive { get; set; }

    public string? PhoneNumber { get; set; }

    public DateTime CreatedAt { get; set; }

    public string? StationId { get; set; }

}

/// <summary>
```

```
/// DTO for login response with token
/// </summary>
public class LoginResponseDto
{
    public string Token { get; set; } = string.Empty;
    public UserResponseDto User { get; set; } = null!;
}
```

## 8 Challenges Faced

- Real-time Synchronization
  - Maintaining consistency between the mobile app's local SQLite database and the centralized NoSQL database was difficult, especially when users operated offline.
- Concurrency Issues
  - Handling multiple EV owners booking the same slot at the same time required careful design to prevent double-bookings.
- Scalability
  - Ensuring the system can handle a growing number of EV owners, charging stations, and bookings without performance degradation.
- Security Concerns
  - Implementing secure authentication, protecting sensitive user data (e.g., NIC numbers), and preventing unauthorized access.
- API Integration
  - Integration Google Maps API into the mobile app to show nearby charging stations while ensuring accurate data and minimal performance impact.
- Complex Role Management
  - Designing and enforcing role-based access (Backoffice, EV operator, EV owner) without overlaps or privilege escalation.
- Cross-Platform Consistency
  - Keeping the web app (Backoffice/Operator) and mobile app (EV owners/Operators) consistent in terms of data, design, and performance.
- Development Issues
  - Hosting the Web API on IIS and ensuring proper configuration with NoSQL database connectivity was challenging.
- Testing Scenarios
  - Simulating real-life workflows like last-minute booking cancellations, expired reservations, or deactivated accounts to ensure the system handled them properly.
- User Experience Design
  - Creating modern, responsive, and intuitive UIs different types of users (Admin, Operator, EV owner) using Bootstrap/Tailwind and Android UI components.
- Performance Optimization
  - Reducing server response times and ensuring quick retrieval of booking and station details from the NoSQL database.
- Error Handling and Notifications
  - Designing clear error message and alert for scenarios like slot unavailability, failed reservations, or invalid QR codes.