# Database Management System

**[MINI PROJECT REPORT]**


**Team Details:**

**Name:** Vinod N Melmari

**SRN:** PES1UG22CS699

**Name:** Vivekananda k

**SRN:** PES1UG22CS708


**Title:**

Railway Management System

## Description:

The Railway Management System is a comprehensive web-based application designed to streamline the management and operations of a railway network. The system facilitates seamless interactions between passengers and the railway administration, enhancing user experience and operational efficiency. It encompasses functionalities such as user registration, ticket booking, payment processing, schedule management, and real-time updates on train status. Built using JavaScript for the frontend and Express for the backend, the system employs a robust database structure to manage core entities like Employees, Users, Bookings, Payments, Schedules, Trains, History, and Stations. This project aims to modernize railway services by offering an intuitive interface, secure transactions, and efficient data management, ensuring convenience and reliability for both passengers and administrators.

## User Requirement Specification:

### 1. Functional Requirements

### 1.1 User Registration and Login

- Users should be able to create accounts by providing personal details such as name, email, contact number, and a secure password.
- Registered users should log in using their credentials.

  Implement a role-based authentication system (e.g., Passenger, Admin).

- 

- Password recovery options should be available via email.

**1.2 Train Information Management**

- The system should allow users to search for trains by parameters such as source, destination, and date of journey.
- Train schedules (arrival/departure time) should be displayed.
- Provide detailed information about available trains, including type (passenger, express), seat availability, and fare details.

**1.3 Ticket Booking**

- Users should be able to book tickets online by selecting the train, travel class, and number of passengers.
- The system should handle seat allotment dynamically based on availability.
- Generate unique booking IDs for each transaction.
- Provide a summary of the booking, including fare breakdown and train details.

**1.4 Payment Processing**

- Integrate secure online payment options (credit/debit cards, UPI, net banking).
- Provide real-time payment confirmation.
- Issue e-receipts with a unique payment ID for successful transactions.

**1.5 Booking History and Cancellation**

- Users should be able to view their booking history, including past and current bookings.
- Provide an option to cancel tickets within defined policies, displaying applicable refund details.

**1.6 Train Tracking and Updates**

- Offer real-time train tracking with updates on delays, cancellations, and platform changes.
- Notify users via email/SMS for significant changes in their booked train status.

**1.7 Admin Panel Features**

- Admins should have the ability to manage train schedules, routes, and fare structures.
- Provide access to monitor booking and payment statistics.

- Enable admins to update train status, including delays and cancellations.
- Manage user accounts and resolve disputes.

**1.8 Station Management**

- Maintain a database of stations with details such as location, amenities, and connecting trains.
- Allow users to search for trains based on station names.

**2. Non-Functional Requirements**

**2.1 Performance**

- The system should handle up to 1,000 simultaneous user sessions efficiently.
- All operations (e.g., booking, payment) should complete within 2–3 seconds under normal load.

**2.2 Scalability**

- The system should support future expansion, such as adding new stations, trains, and payment gateways.

**2.3 Security**

- Ensure secure user authentication and authorization.
- Encrypt sensitive user data, including payment details and passwords.
- Implement measures to prevent SQL injection, cross-site scripting (XSS), and other vulnerabilities.

**2.4 Availability**

- The system must maintain 99.9% uptime with robust error handling and recovery mechanisms. **2.5 Usability**
- The system should have an intuitive, user-friendly interface with mobile compatibility.
- Ensure multilingual support to cater to diverse user demographics.

**2.6 Maintainability**

- Use modular architecture for easy maintenance and updates.
- Provide comprehensive documentation for developers and administrators. **3. Constraints**

- The system must operate within budget constraints, using cost-effective hosting and third-party tools.
- Integration with government railway systems may require adherence to regulatory standards.

**4. User Roles and Access Levels**

**4.1 Passengers**

- Access train information, book tickets, make payments, and view booking history.

**4.2 Admins**

- Manage the database of trains, users, stations, and bookings.
- Handle exceptions such as payment failures and disputed bookings.

**4.3 Employees**

- Access limited admin features, such as managing schedules and monitoring platform activities.

## List of Software/Tools/Programming Languages Used:

**a) Frontend Development**

1. **React.js** ○ A JavaScript library for building dynamic and interactive user interfaces.
    - Used to develop reusable components and implement client-side routing.
2. **Tailwind CSS** ○ A utility-first CSS framework for styling the application.
    - Enabled efficient and responsive design with a focus on customization.

**b) Backend Development**

3. **Node.js** ○ A JavaScript runtime environment for executing server-side code.
    - Used to create a scalable and efficient backend.

•

4. **Express.js** ○ A lightweight web application framework for Node.js.

- Used for routing, handling API requests, and managing server logic.

**c) Database Management** 5. **SQL**

- A relational database management system (RDBMS) used to store and manage data.
- Enabled the efficient handling of entities like Users, Trains, Bookings, and Payments.

**d) Development Tools**

6. **Visual Studio Code (VS Code)** ○ A code editor for writing, debugging, and editing code.
   - Enhanced productivity through extensions for React, Tailwind, and Node.js.
7. **Postman**
   - API testing tool used for validating RESTful endpoints.
   - Assisted in debugging backend API requests and responses.

**e) Version Control and Collaboration** 8. **Git**

- A version control system for tracking code changes.

9. **GitHub**
   - A platform for repository hosting and team collaboration.

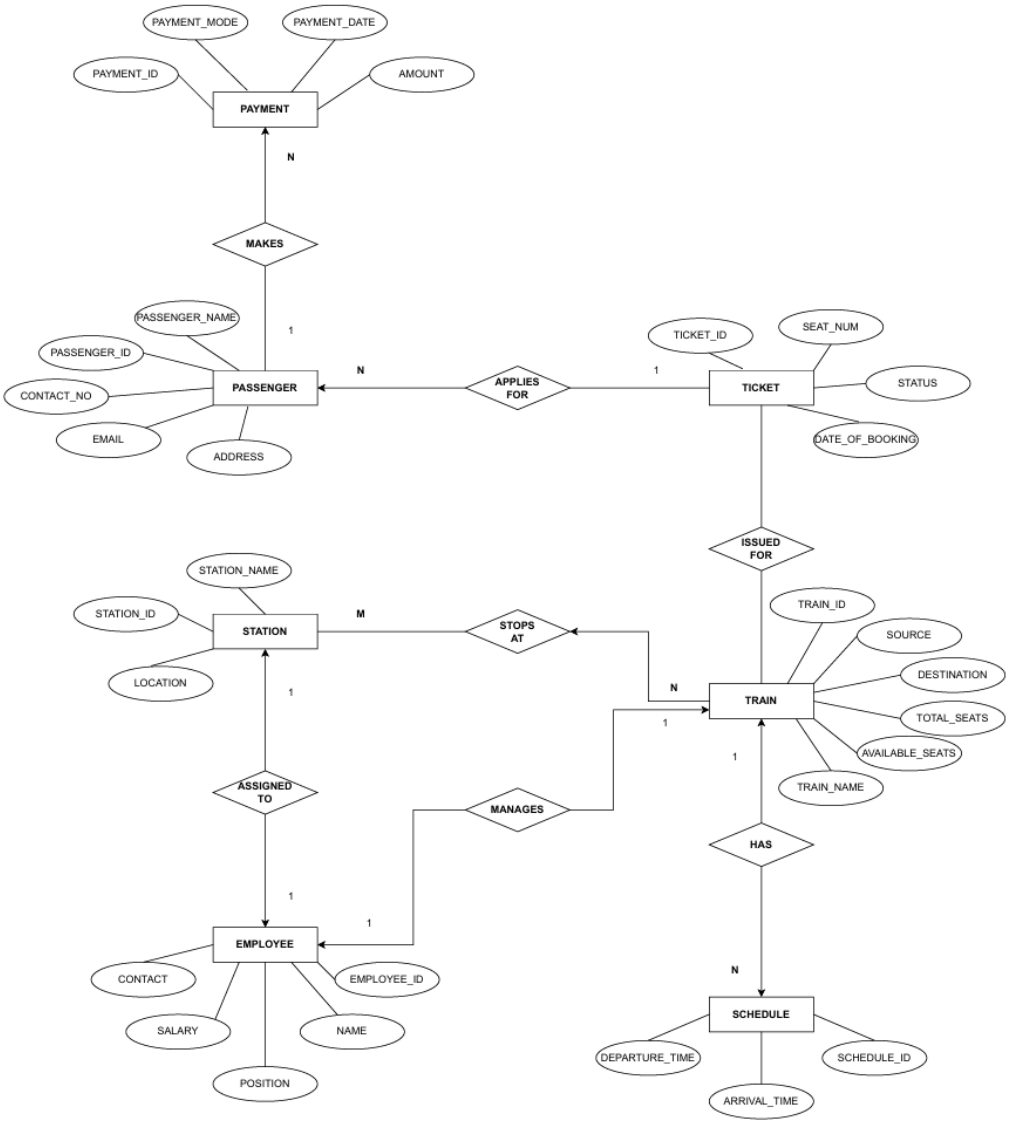
**npm (Node Package Manager)**

- Used to manage dependencies for Node.js and React.

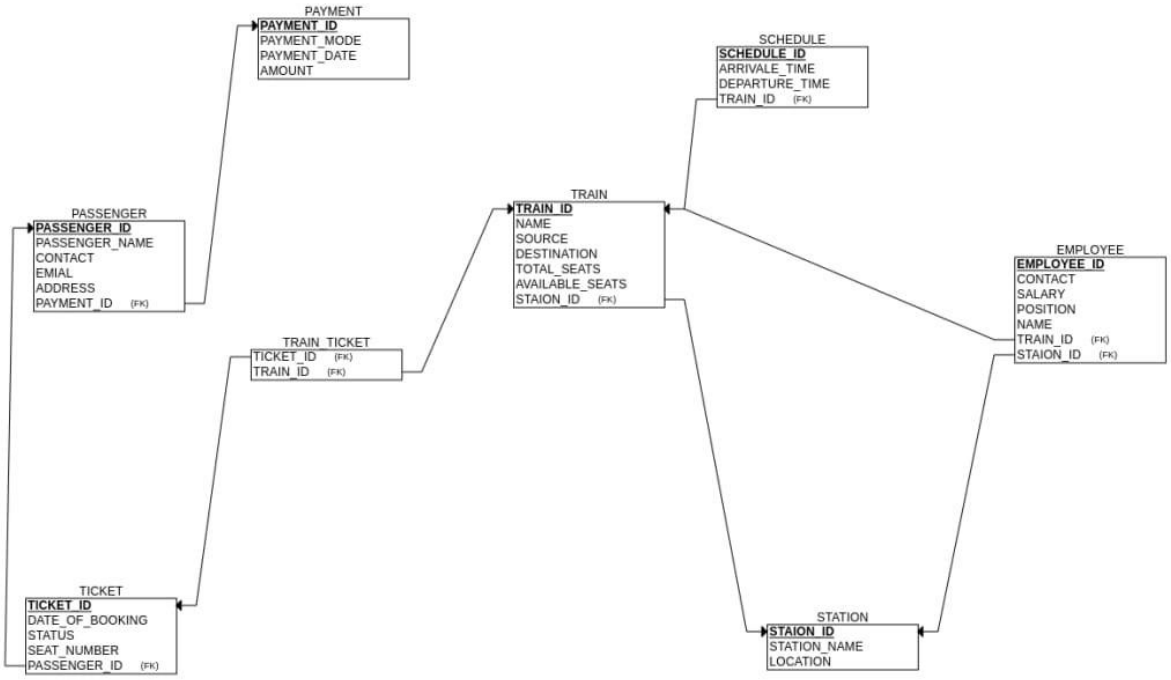
**Browser Developer Tools**

- Built-in browser tools used for debugging and testing UI responsiveness

## ER Diagram:




## Relational Schema:




## DDL commands:

CREATE TABLE users (...);

CREATE TABLE trains (...);

CREATE TABLE schedules (...);

CREATE TABLE bookings (...);

CREATE TABLE payments (...);

CREATE TABLE history (...);

CREATE TABLE Employee (...);

CREATE TABLE stations (...);

ALTER TABLE schedules ADD FOREIGN KEY (train_id) REFERENCES trains(id);

ALTER TABLE bookings ADD FOREIGN KEY (user_id) REFERENCES users(id);

ALTER TABLE bookings ADD FOREIGN KEY (schedule_id) REFERENCES schedules(id);

ALTER TABLE payments ADD FOREIGN KEY (booking_id) REFERENCES bookings(id);

ALTER TABLE history ADD FOREIGN KEY (user_id) REFERENCES users(id);

## CRUD operation:

### 1) FOR USER TABLE

#### A) CREATE

```
MariaDB [TrainWay]> INSERT INTO users (id, name, email, password,contact,address,created_at) VALUES ('1', 'sumeet', 'Sumeet@gmail.com', 'password123','8293739480','jamkhandi','2024-11-20 23:13:06');
Query OK, 1 row affected (0.006 sec)

MariaDB [TrainWay]> select * from users;
+--------------------------------------+-----------+----------------------------+-------------+------------+-------------+---------------------+
| id                                   | name      | email                      | password    | contact    | address     | created_at          |
+--------------------------------------+-----------+----------------------------+-------------+------------+-------------+---------------------+
| 1                                    | sumeet    | Sumeet@gmail.com           | password123 | 8293739480 | jamkhandi   | 2024-11-20 23:13:06 |
| 22df618b-a0d2-11ef-8960-c8dbf4321e00 | AKASH C N | akashnyamagoud45@gmail.com | akash@123   | 6360663654 | sutta gulli | 2024-11-12 14:13:06 |
| a0a52004-9d02-11ef-8baf-cef8714e310c | aditya    | armirji@gmail.com          | akash@123   | 7090123456 | nagarabhavi | 2024-11-07 17:50:45 |
| c5dccb54-a244-11ef-89b3-c9e0067e6684 | gouse     | gouse05.x@gmail.com        | 12345       | 8088818635 | gangavati   | 2024-11-14 10:26:13 |
+--------------------------------------+-----------+----------------------------+-------------+------------+-------------+---------------------+
4 rows in set (0.000 sec)
```

#### B) READ

```
MariaDB [TrainWay]> select * from users;
+--------------------------------------+-----------+----------------------------+-------------+------------+-------------+---------------------+
| id                                   | name      | email                      | password    | contact    | address     | created_at          |
+--------------------------------------+-----------+----------------------------+-------------+------------+-------------+---------------------+
| 1                                    | sumeet    | Sumeet@gmail.com           | password123 | 8293739480 | jamkhandi   | 2024-11-20 23:13:06 |
| 22df618b-a0d2-11ef-8960-c8dbf4321e00 | AKASH C N | akashnyamagoud45@gmail.com | akash@123   | 6360663654 | sutta gulli | 2024-11-12 14:13:06 |
| a0a52004-9d02-11ef-8baf-cef8714e310c | aditya    | armirji@gmail.com          | akash@123   | 7090123456 | nagarabhavi | 2024-11-07 17:50:45 |
| c5dccb54-a244-11ef-89b3-c9e0067e6684 | gouse     | gouse05.x@gmail.com        | 12345       | 8088818635 | gangavati   | 2024-11-14 10:26:13 |
+--------------------------------------+-----------+----------------------------+-------------+------------+-------------+---------------------+
4 rows in set (0.000 sec)
```

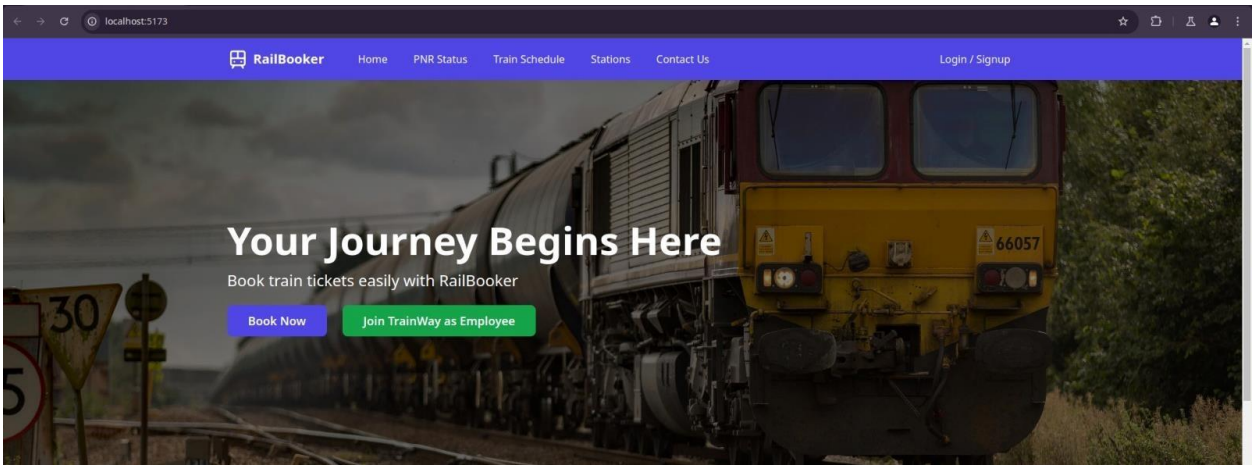
#### C) UPDATE

```
MariaDB [TrainWay]> select * from users where id='1';
+----+--------+------------------+-------------+------------+-----------+---------------------+
| id | name   | email            | password    | contact    | address   | created_at          |
+----+--------+------------------+-------------+------------+-----------+---------------------+
| 1  | sumeet | Sumeet@gmail.com | password123 | 8293739480 | jamkhandi | 2024-11-20 23:13:06 |
+----+--------+------------------+-------------+------------+-----------+---------------------+
1 row in set (0.000 sec)

MariaDB [TrainWay]> UPDATE users SET email = 'sumeet829@gmail.com' WHERE id = '1';
Query OK, 1 row affected (0.006 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [TrainWay]> select * from users;
+--------------------------------------+-----------+----------------------------+-------------+------------+-------------+---------------------+
| id                                   | name      | email                      | password    | contact    | address     | created_at          |
+--------------------------------------+-----------+----------------------------+-------------+------------+-------------+---------------------+
| 1                                    | sumeet    | sumeet829@gmail.com        | password123 | 8293739480 | jamkhandi   | 2024-11-20 23:13:06 |
| 22df618b-a0d2-11ef-8960-c8dbf4321e00 | AKASH C N | akashnyamagoud45@gmail.com | akash@123   | 6360663654 | sutta gulli | 2024-11-12 14:13:06 |
| a0a52004-9d02-11ef-8baf-cef8714e310c | aditya    | armirji@gmail.com          | akash@123   | 7090123456 | nagarabhavi | 2024-11-07 17:50:45 |
| c5dccb54-a244-11ef-89b3-c9e0067e6684 | gouse     | gouse05.x@gmail.com        | 12345       | 8088818635 | gangavati   | 2024-11-14 10:26:13 |
+--------------------------------------+-----------+----------------------------+-------------+------------+-------------+---------------------+
4 rows in set (0.000 sec)
```

#### D) DELETE

```
MariaDB [TrainWay]> DELETE FROM users WHERE id = '1';
Query OK, 1 row affected (0.006 sec)

MariaDB [TrainWay]> select * from users;
+--------------------------------------+-----------+----------------------------+-----------+------------+-------------+---------------------+
| id                                   | name      | email                      | password  | contact    | address     | created_at          |
+--------------------------------------+-----------+----------------------------+-----------+------------+-------------+---------------------+
| 22df618b-a0d2-11ef-8960-c8dbf4321e00 | AKASH C N | akashnyamagoud45@gmail.com | akash@123 | 6360663654 | sutta gulli | 2024-11-12 14:13:06 |
| a0a52004-9d02-11ef-8baf-cef8714e310c | aditya    | armirji@gmail.com          | akash@123 | 7090123456 | nagarabhavi | 2024-11-07 17:50:45 |
| c5dccb54-a244-11ef-89b3-c9e0067e6684 | gouse     | gouse05.x@gmail.com        | 12345     | 8088818635 | gangavati   | 2024-11-14 10:26:13 |
+--------------------------------------+-----------+----------------------------+-----------+------------+-------------+---------------------+
3 rows in set (0.000 sec)
```

## SCREENSHOTS:




## Why Choose Rail-Way?

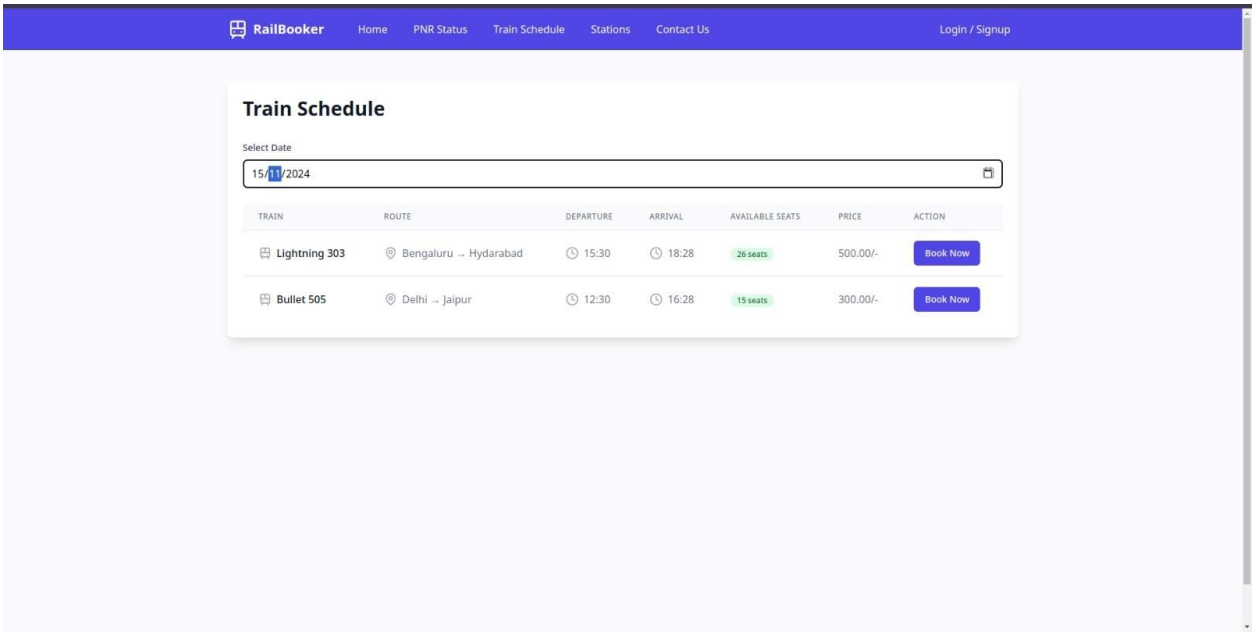
**Quick PNR Status**
Check your PNR status instantly with our real-time tracking system

**Live Train Schedule**
Access real-time train schedules and plan your journey efficiently

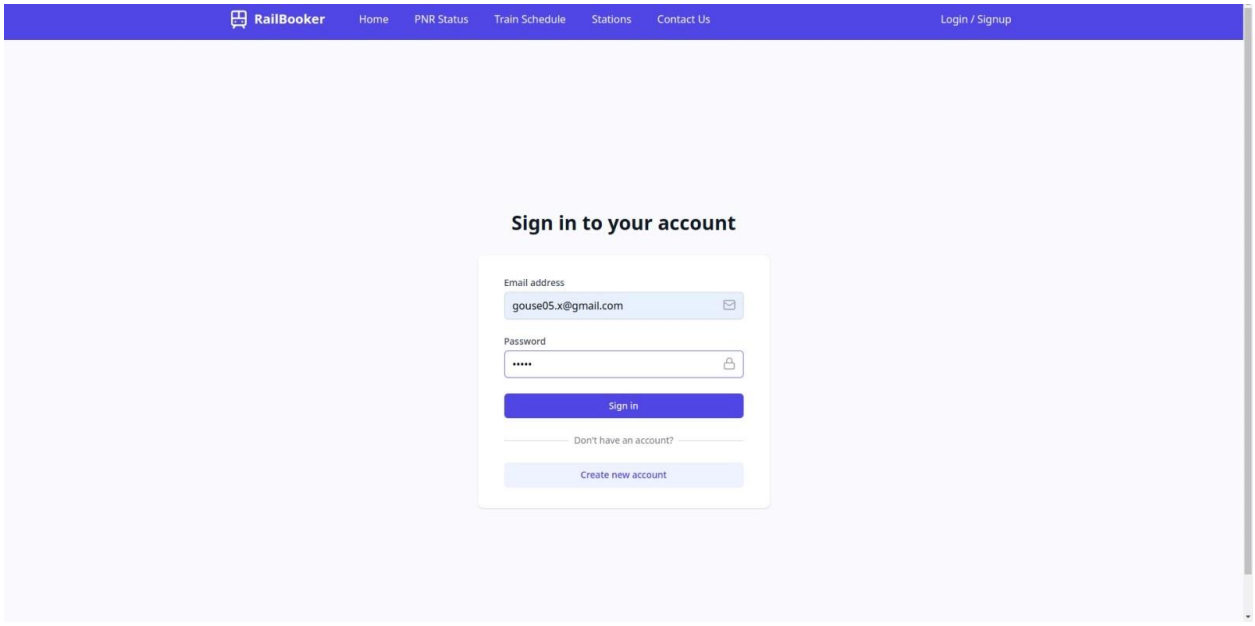
**Station Information**
Get detailed information about stations and available facilities

## Latest Updates

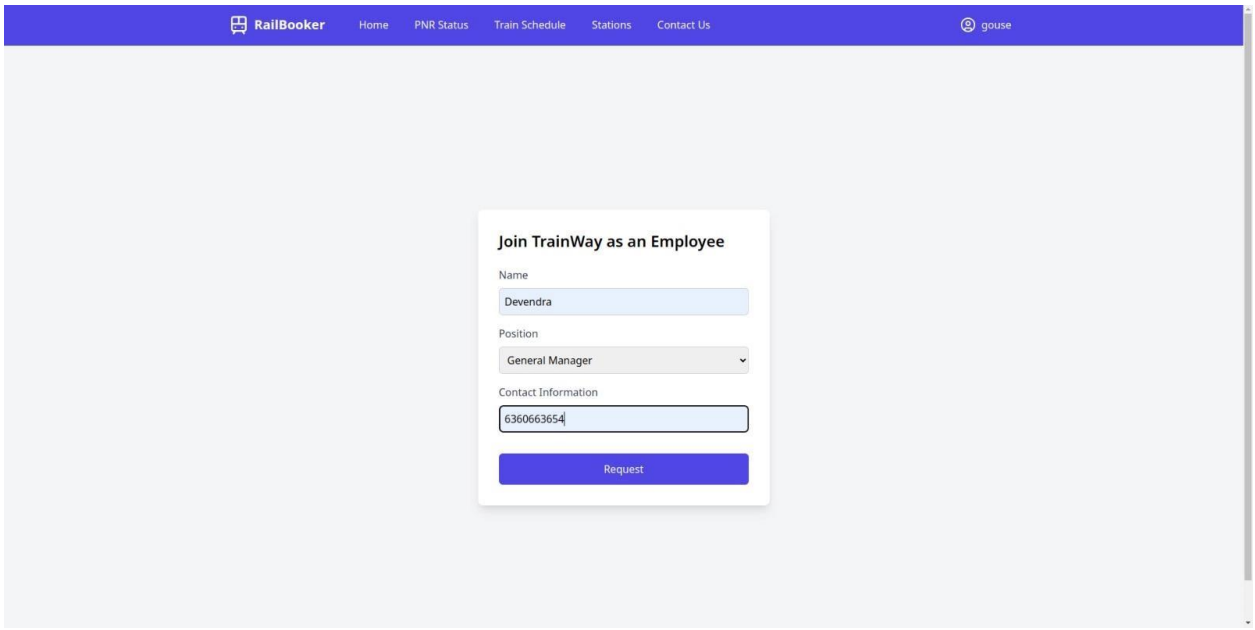
**New Express Service**
Introducing new express trains connecting major cities with faster travel times.
Learn More →



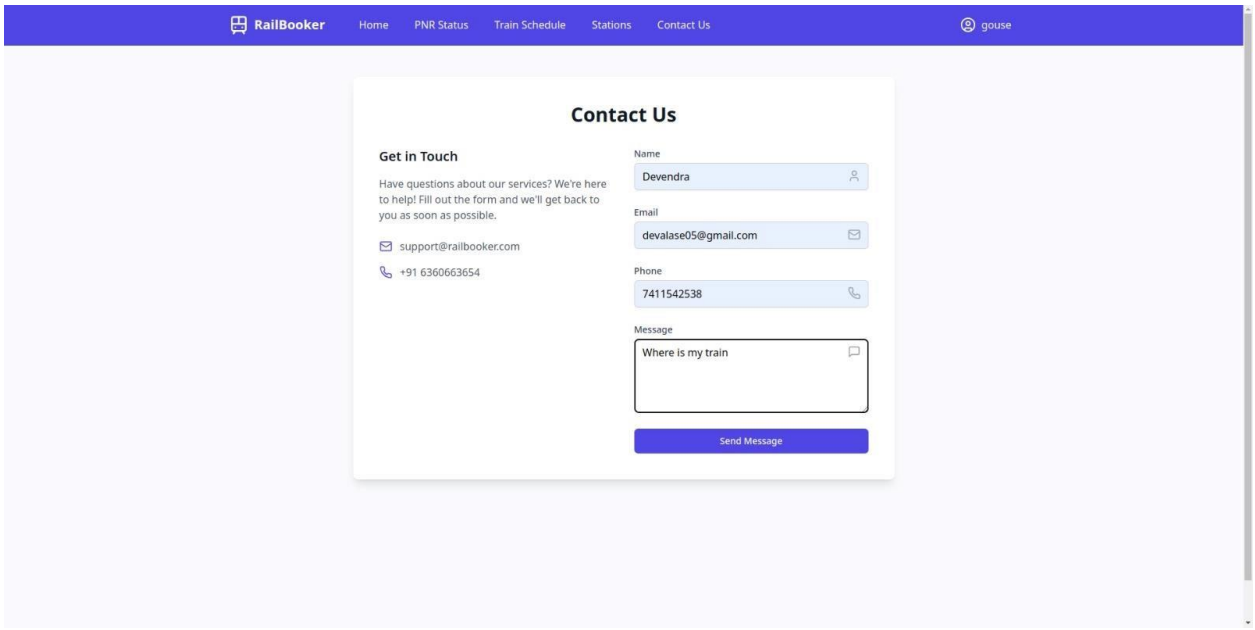

**REDIRECTS TO LOGIN PAGE**



## EMPLOYEE

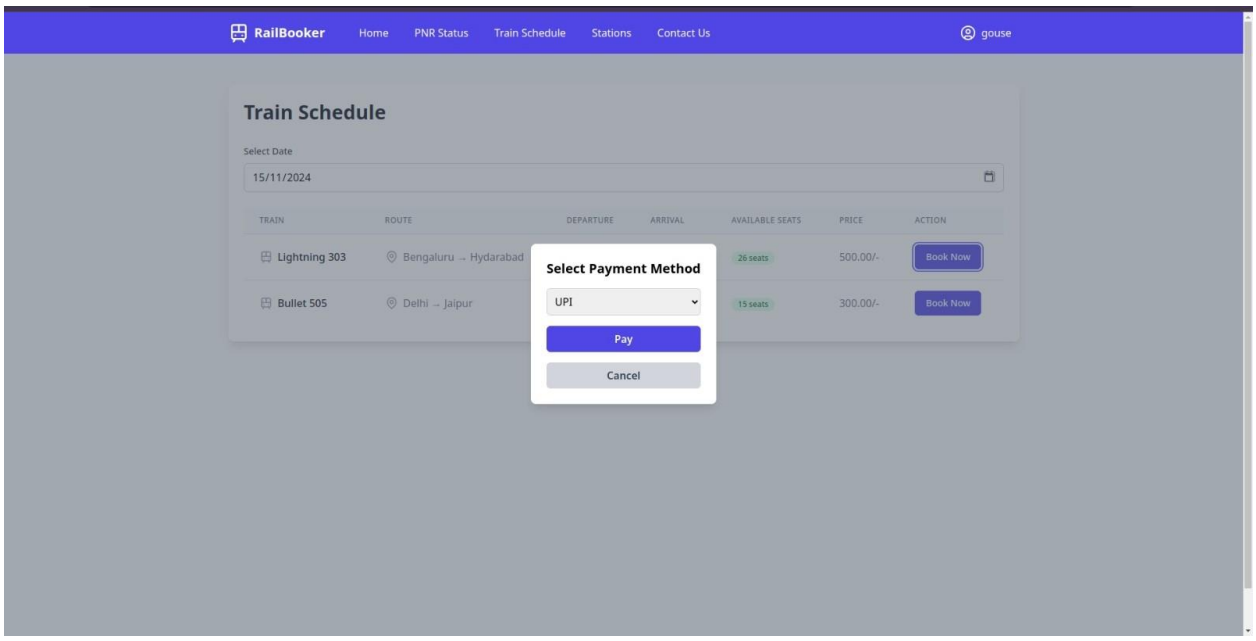



## CONTACTUS



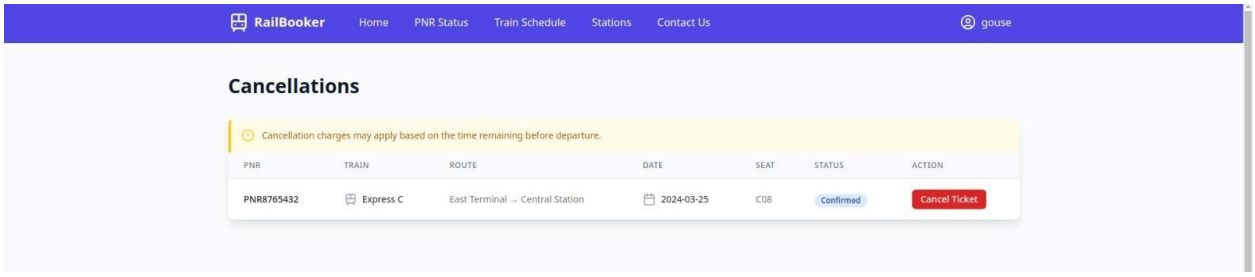
**BOOKING**

Seats get updated after booking ticket(USED TRIGGER)

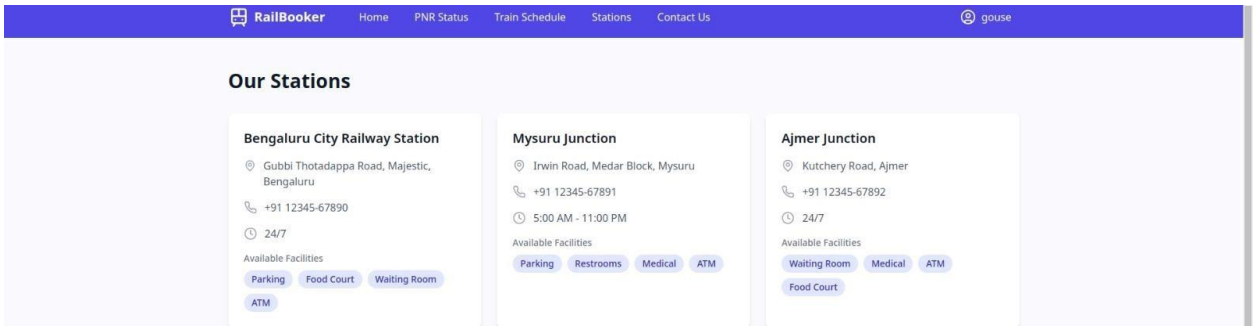


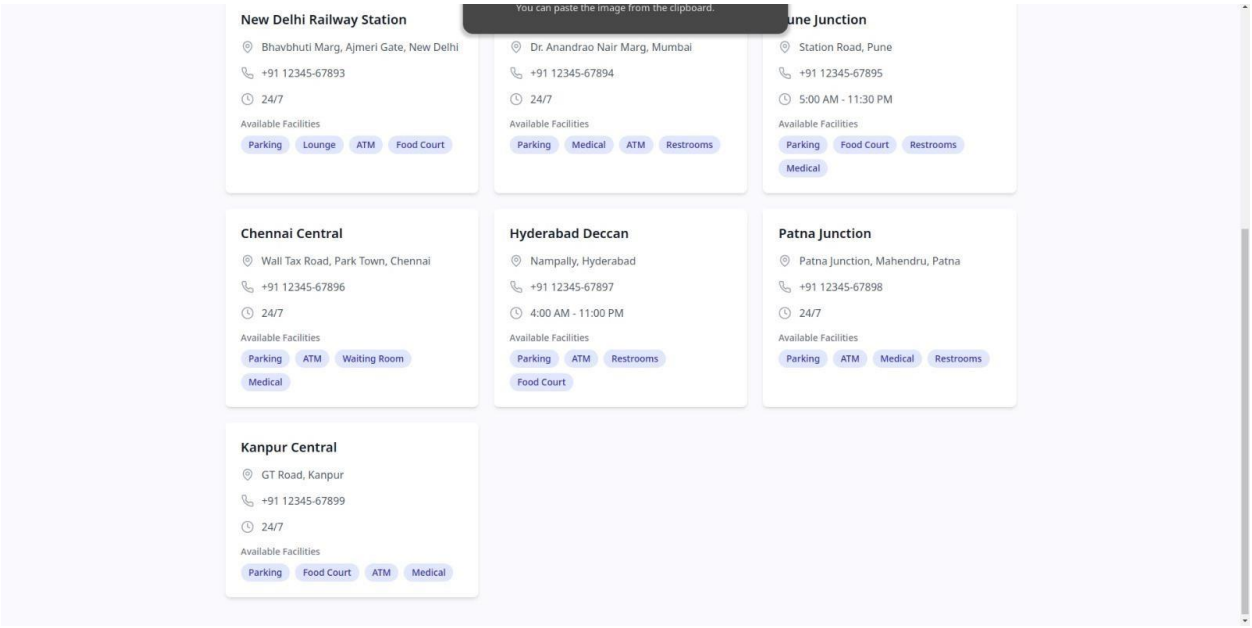

**CANCEL TICKETS**

After the tickets gets cancelled it will trigger the booking table and payment table



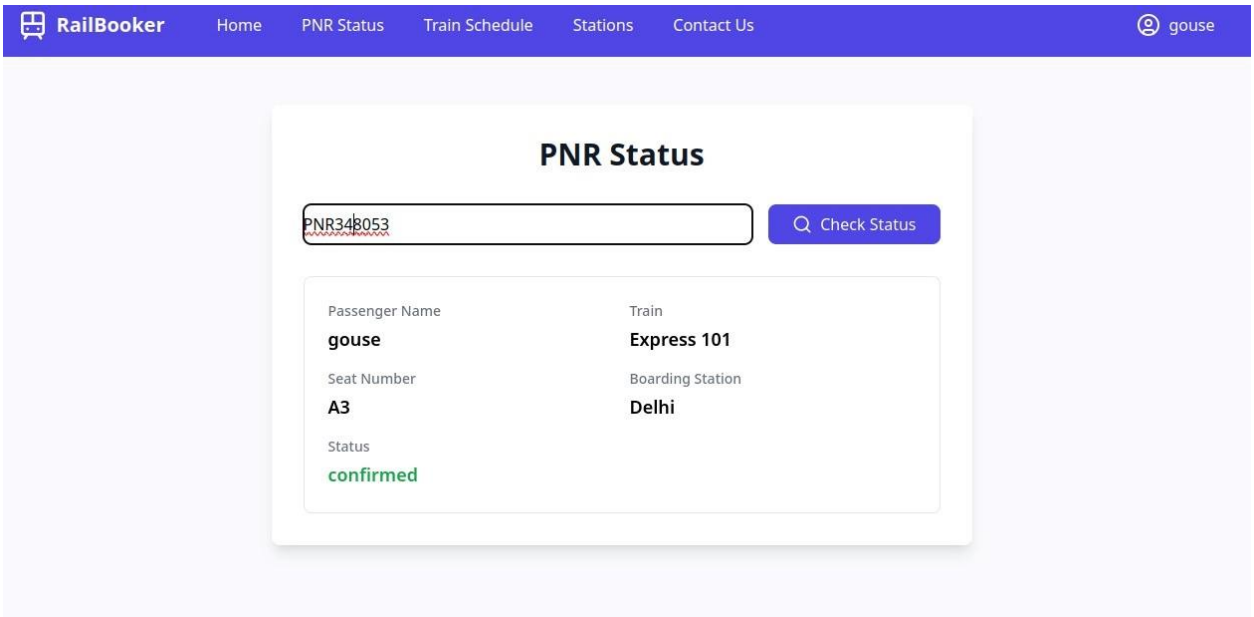

**STATIONS**






**PNR STATUS**

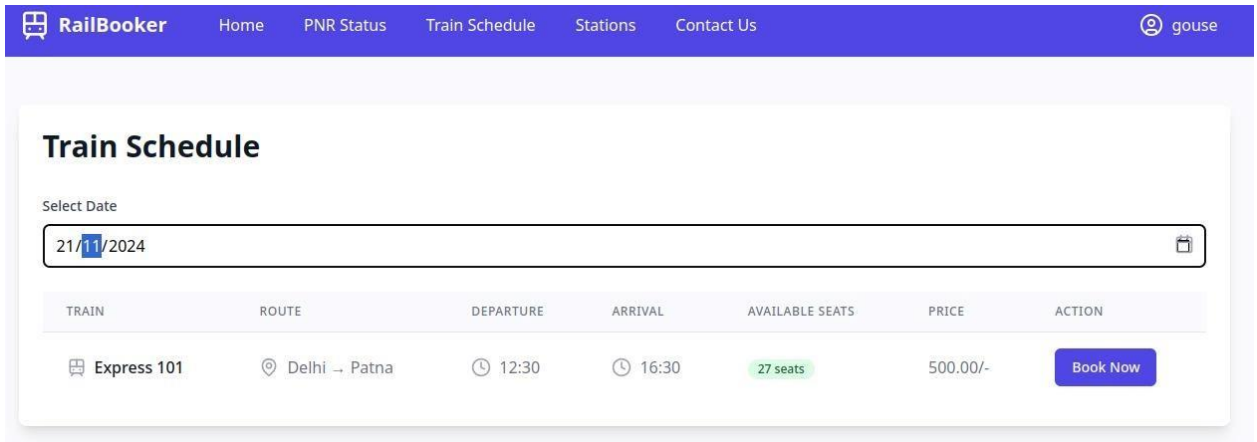
To fetch the pnr status we have used function and store procedure




## TRIGGER, FUNTIONS/STORE PROCEDURE, NESTED QUERY, JOIN, AGGREGATE QUERIES:

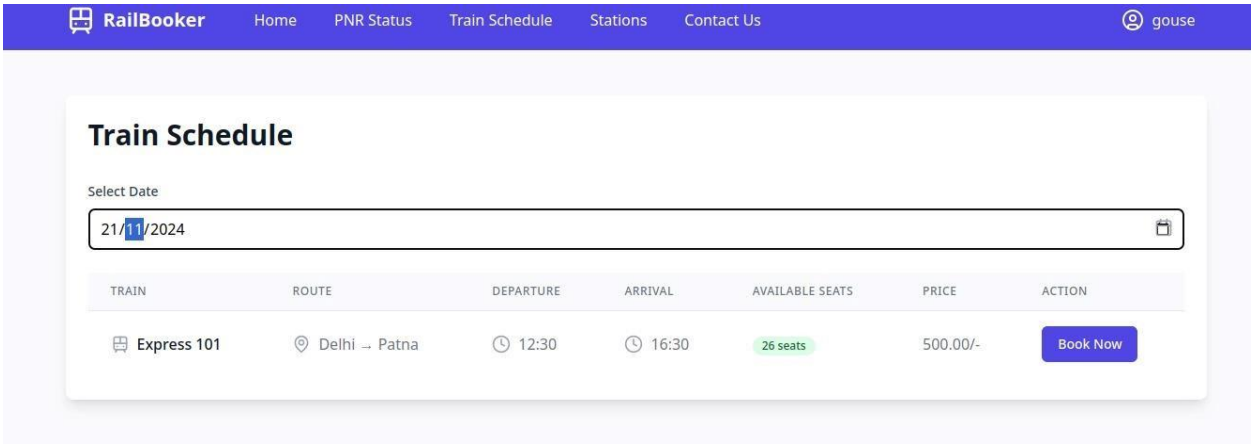
## A) TRIGGER

Before booking ticket, Look at the available seats (i.e. 27)




After Booking Ticket

| d26ba383-f0b3-4cf1-8a17-4baad3abe95b | PNR095560 | c5dccb54-a244-11ef-89b3-c9e0067e6684 | 131a2c8d-78e2-46f4-9ce1-bfe764e845bc | A4 | confirmed | 2024-11-21 00:55:15 |




Now available seats 26

Code for the above Trigger

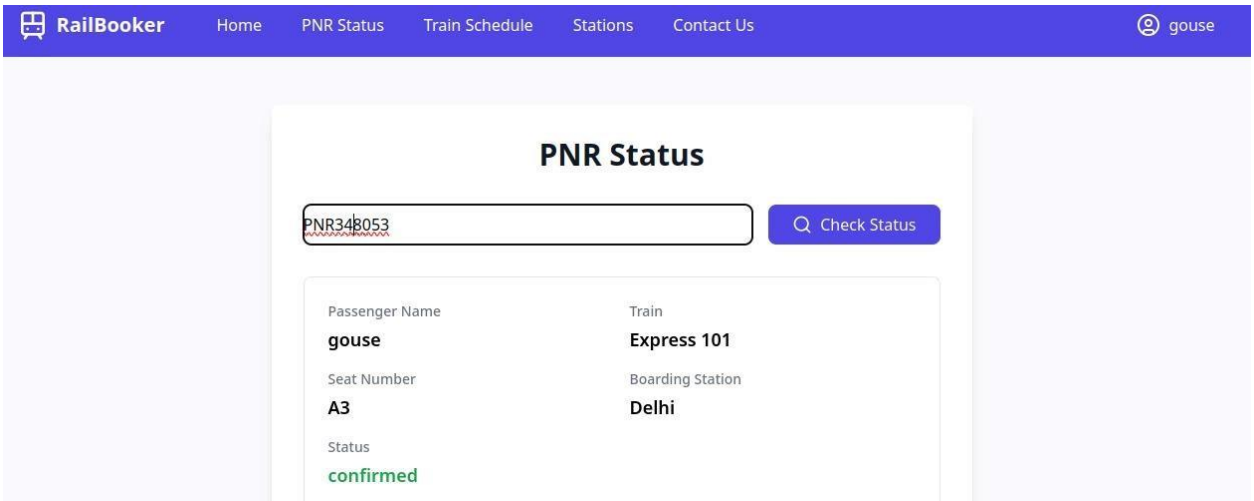
```sql
-- Trigger to update available seats after booking
CREATE TRIGGER after_booking_insert
AFTER INSERT ON bookings
FOR EACH ROW
BEGIN
  -- Update train seats
  UPDATE trains t
  JOIN schedules s ON s.train_id = t.id
  SET t.available_seats = t.available_seats - 1
  WHERE s.id = NEW.schedule_id
  AND NEW.status = 'confirmed';

  -- Add to history
  INSERT INTO history (id, user_id, action_type, reference_id, description, metadata)
  VALUES (
    UUID(),
    NEW.user_id,
    'booking',
    NEW.id,
    CONCAT('New booking created with PNR: ', NEW.pnr),
    JSON_OBJECT(
      'pnr', NEW.pnr,
      'seat_number', NEW.seat_number,
      'status', NEW.status
    )
  );
END//
```

```sql
-- Trigger to update available seats after cancellation
CREATE TRIGGER after_booking_update
AFTER UPDATE ON bookings
FOR EACH ROW
BEGIN
  IF NEW.status = 'cancelled' AND OLD.status = 'confirmed' THEN
    -- Update train seats
    UPDATE trains t
    JOIN schedules s ON s.train_id = t.id
    SET t.available_seats = t.available_seats + 1
    WHERE s.id = NEW.schedule_id;

    -- Add to history
    INSERT INTO history (id, user_id, action_type, reference_id, description, metadata)
    VALUES (
      UUID(),
      NEW.user_id,
      'cancellation',
      NEW.id,
      CONCAT('Booking cancelled for PNR: ', NEW.pnr),
      JSON_OBJECT(
        'pnr', NEW.pnr,
        'previous_status', OLD.status,
        'new_status', NEW.status
      )
    );
  END IF;
END//
```

## B) PROCEDURE/FUNCTION

FOR PNR STATUS

Let us take above booking pnr




```
-- Function to get PNR status
CREATE FUNCTION get_pnr_status(p_pnr VARCHAR(10))
RETURNS VARCHAR(50)
DETERMINISTIC
BEGIN
  DECLARE v_status VARCHAR(50);

  SELECT
    CASE
      WHEN b.status = 'cancelled' THEN 'Cancelled'
      WHEN b.status = 'confirmed' AND s.departure_time > NOW() THEN 'Confirmed'
      WHEN b.status = 'confirmed' AND s.departure_time <= NOW() THEN 'Completed'
      ELSE 'Not Found'
    END INTO v_status
  FROM bookings b
  JOIN schedules s ON s.id = b.schedule_id
  WHERE b.pnr = p_pnr;

  RETURN COALESCE(v_status, 'Not Found');
END//
```

```sql
-- Stored procedure for booking a ticket
CREATE PROCEDURE book_ticket(
  IN p_user_id VARCHAR(36),
  IN p_schedule_id VARCHAR(36),
  IN p_payment_method VARCHAR(50)
)
BEGIN
  DECLARE v_booking_id VARCHAR(36);
  DECLARE v_payment_id VARCHAR(36);
  DECLARE v_price DECIMAL(10, 2);
  DECLARE v_pnr VARCHAR(10);

  -- Start transaction
  START TRANSACTION;

  -- Get price from schedule
  SELECT price INTO v_price
  FROM schedules
  WHERE id = p_schedule_id;

  -- Generate PNR
  SET v_pnr = CONCAT('PNR', LPAD(FLOOR(RAND() * 1000000), 6, '0'));

  -- Generate UUIDs
  SET v_booking_id = UUID();
  SET v_payment_id = UUID();

  -- Create booking
  INSERT INTO bookings (id, pnr, user_id, schedule_id, seat_number)
  VALUES (v_booking_id, v_pnr, p_user_id, p_schedule_id, 'AUTO');

  -- Create payment
  INSERT INTO payments (id, booking_id, amount, payment_method, transaction_id)
  VALUES (
    v_payment_id,
    v_booking_id,
    v_price,
    p_payment_method,
    CONCAT('TXN', LPAD(FLOOR(RAND() * 1000000), 6, '0'))
  );

  -- Commit transaction
  COMMIT;

  -- Return booking details
  SELECT b.*, p.amount, p.payment_method, p.status as payment_status
  FROM bookings b
  JOIN payments p ON p.booking_id = b.id
  WHERE b.id = v_booking_id;
END//
```

GIT REPO LINK

https://github.com/Vivekananda-k/Railway-management-system.git