MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DBS PROJECT

RAILWAY TICKET BOOKING AND CANCELLATION SYSTEM

SUBMITTED

BY

230905006 – Vignesh Upadhyaya (02)

230905152 - Hiten Raj Singh (23)

Department of Computer Science and Engineering

Manipal Institute of Technology, Manipal

April 2025

TABLE OF CONTENTS

ABSTRACT

The Railway Ticket Booking and Cancellation System is a database management project aimed at automating the reservation process of train tickets. The system replaces traditional manual ticketing with a digital platform that manages real-time train schedules, seat availability, and user bookings. By utilizing PostgreSQL and SQL procedures, the system enhances user experience through faster, more accurate, and reliable ticket booking and cancellation. The primary focus is on ensuring a smooth flow of operations between the user interface and the backend database system.

PROBLEM STATEMENT

Manual railway reservation systems suffer from inefficiencies like long queues, booking overlaps, data redundancy, and lack of real-time seat availability. Moreover, cancellations and refunds in such systems are slow and disorganized. The problem lies in the absence of an integrated, database-driven system that can handle bookings, seat allocation, cancellations, and payments seamlessly. The goal of this project is to develop a reliable and scalable ticket booking and cancellation system using PostgreSQL, which will allow users to interact with the system securely and efficiently.

NORMALIZATION-

All tables in the database have been normalized up to BCNF. This ensures that:

- Each table is efficient and free from redundancy.
- All non-key attributes are fully functionally dependent on the primary key.
- Every non-trivial functional dependency has a superkey as its determinant.

This is because every table has a primary key which acts as a superkey for the relation R and satisfies one of the conditions for BCNF form.

$\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, α is a superkey for R.

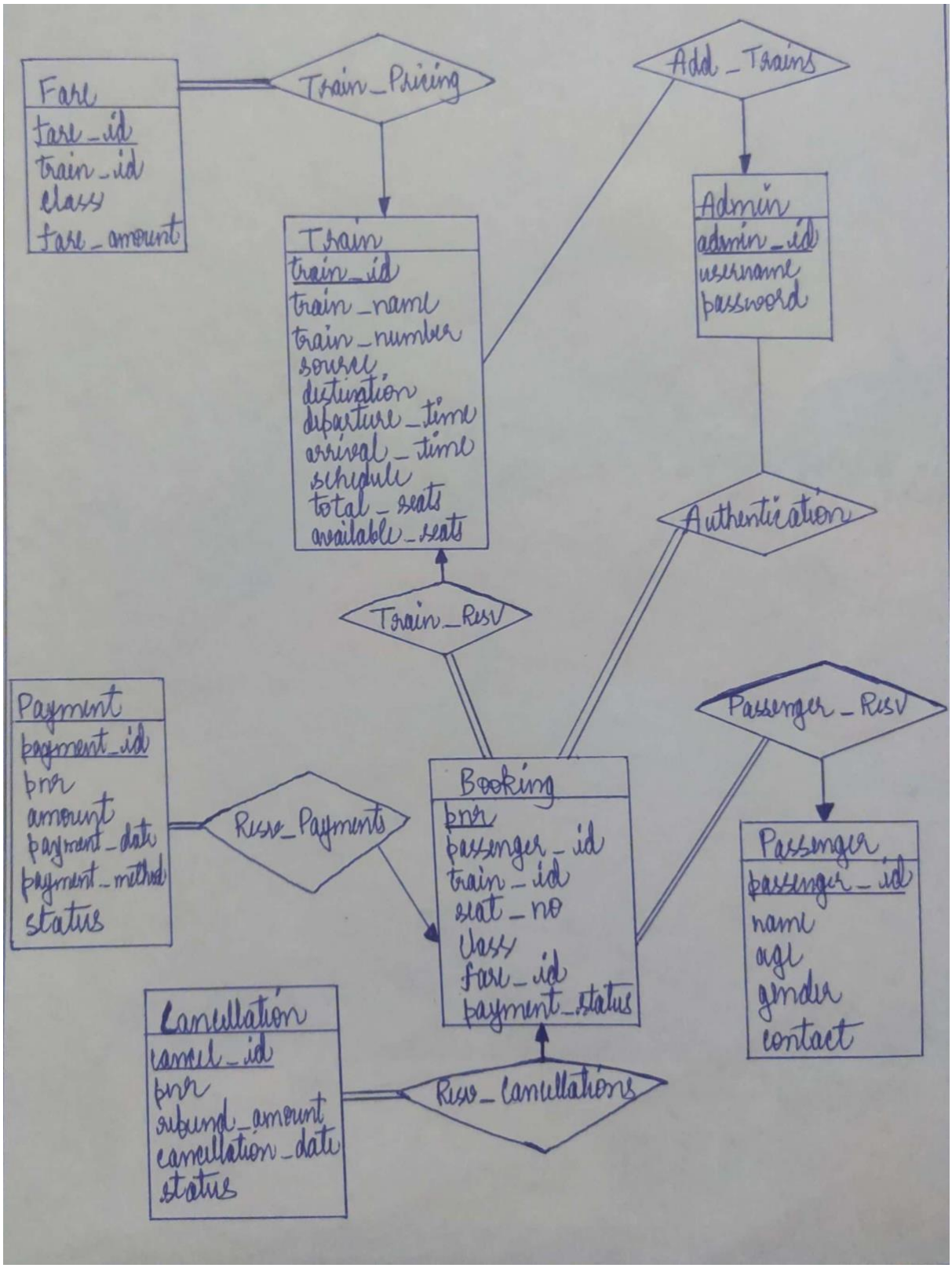
Here 'α' is the primary key of the table or a superset of it.

1NF: All attributes contain atomic values.

2NF: All non-key attributes are fully functionally dependent on the primary key.

3NF: There are no transitive dependencies.

ER DIAGRAM


Fare
fare_id
train_id
class
fare_amount
Train_Pricing
Add_Trains
Train
train_id
train_name
train_number
source
destination
departure_time
arrival_time
schedule
total_seats
available_seats
Admin
admin_id
username
password
Authentication
Train_Resv
Payment
payment_id
pnr
amount
payment_date
payment_method
status
Resv_Payments
Passenger_Resv
Booking
pnr
passenger_id
train_id
seat_no
class
fare_id
payment_status
Passenger
passenger_id
name
age
gender
contact
Cancellation
cancel_id
pnr
refund_amount
cancellation_date
status
Resv_Cancellations

RELATIONAL TABLES

1)Passenger(PassengerID (PK), Name, Age, Gender, Contact)

2)Train(TrainID (PK), TrainName, Source, Destination, Date, Time, TotalSeats, AvailableSeats)

3) Fare(FareID (PK), TrainID (FK), Class, FareAmount)

4) Booking(PNR (PK), PassengerID (FK), TrainID (FK), SeatNo, Class, FareID (FK), PaymentStatus)

5) Payment(PaymentID (PK), PNR (FK), Amount, Status, TransactionDate)

6) Cancellation(CancelID (PK), PNR (FK), RefundAmount, Status, CancellationDate)

7) Admin(AdminID (PK), Username, PasswordHash)

DDL COMMANDS

```
-- Create the Admin table
CREATE TABLE public.admin (
    admin_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    username TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL
);
```

```sql
-- Create the Passenger table
CREATE TABLE public.passenger (
    passenger_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name TEXT NOT NULL,
    age INTEGER NOT NULL CHECK (age > 0),
    gender TEXT NOT NULL,
    contact TEXT NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL
);

-- Create the Train table
CREATE TABLE public.train (
    train_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    train_name TEXT NOT NULL,
    train_number TEXT NOT NULL UNIQUE,
    source TEXT NOT NULL,
    destination TEXT NOT NULL,
    departure_time TIME NOT NULL,
    arrival_time TIME NOT NULL,
    schedule DATE NOT NULL,
    total_seats INTEGER NOT NULL CHECK (total_seats > 0),
    available_seats INTEGER NOT NULL CHECK (available_seats >= 0),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    CHECK (available_seats <= total_seats)
);

-- Create the Fare table
CREATE TABLE public.fare (
    fare_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
```

```
    train_id UUID NOT NULL REFERENCES public.train(train_id) ON DELETE
CASCADE,
    class TEXT NOT NULL,
    fare_amount DECIMAL(10, 2) NOT NULL CHECK (fare_amount > 0),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    UNIQUE(train_id, class)
);

-- Create the Booking table
CREATE TABLE public.booking (
    pnr TEXT PRIMARY KEY DEFAULT 'PNR' || floor(random() * 10000000)::text,
    passenger_id UUID NOT NULL REFERENCES public.passenger(passenger_id) ON
DELETE CASCADE,
    train_id UUID NOT NULL REFERENCES public.train(train_id) ON DELETE
CASCADE,
    seat_no TEXT NOT NULL,
    class TEXT NOT NULL,
    fare_id UUID NOT NULL REFERENCES public.fare(fare_id) ON DELETE CASCADE,
    booking_date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    payment_status TEXT NOT NULL DEFAULT 'Pending' CHECK (payment_status IN
('Pending', 'Completed', 'Failed')),
    booking_status TEXT NOT NULL DEFAULT 'Confirmed' CHECK (booking_status IN
('Confirmed', 'Cancelled')),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL
);

-- Create the Payment table
CREATE TABLE public.payment (
    payment_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    pnr TEXT NOT NULL REFERENCES public.booking(pnr) ON DELETE CASCADE,
    amount DECIMAL(10, 2) NOT NULL CHECK (amount > 0),
    payment_date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
```

```
    payment_method TEXT NOT NULL,
    status TEXT NOT NULL DEFAULT 'Pending' CHECK (status IN ('Pending', 'Completed',
'Failed')),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL
);

-- Create the Cancellation table
CREATE TABLE public.cancellation (
    cancel_id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    pnr TEXT NOT NULL REFERENCES public.booking(pnr) ON DELETE CASCADE,
    refund_amount DECIMAL(10, 2) NOT NULL,
    cancellation_date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    status TEXT NOT NULL DEFAULT 'Pending' CHECK (status IN ('Pending', 'Processed')),
    created_at TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL
);
```

SQL QUERIES

BASIC-

1) SELECT train_name, train_number FROM train WHERE available_seats > 0;
2) SELECT payment_id, amount, payment_date FROM public.payment WHERE status= 'Completed';
3) SELECT pnr, passenger_id FROM public.booking WHERE pnr IN (SELECT pnr FROM public.payment WHERE status = 'Failed');
4) SELECT c.pnr, c.refund_amount, c.status FROM public.cancellation c JOIN public.booking b ON c.pnr = b.pnrWHERE c.status = 'Processed';

COMPLEX-

1) SELECT p.name AS passenger_name, t.train_name ,b.class, f.fare_amount, b.payment_status FROM public.booking b

JOIN public.passenger p ON b.passenger_id = p.passenger_id

JOIN public.train t ON b.train_id = t.train_id

JOIN public.fare f ON b.fare_id = f.fare_id;

2) select t.train_name, COUNT(b.pnr) as total_bookings,

SUM(f.fare_amount) as total_revenue frompublic.train t

join public.booking b on t.train_id = b.train_id

join public.fare f on b.fare_id = f.fare_id group by t.train_name;

3) SELECT t.train_name, COUNT(b.pnr) AS booked_seats, t.total_seats, ROUND(COUNT(b.pnr)::decimal / t.total_seats * 100, 2) AS booking_percentage

FROM public.train t

LEFT JOIN public.booking b ON t.train_id = b.train_id AND b.booking_status = 'Confirmed' GROUP BY t.train_name, t.total_seats;

PL/SQL

PROCEDURES-

```
CREATE OR REPLACE PROCEDURE add_train(
    p_train_name TEXT,
    p_train_number TEXT,
    p_source TEXT,
    p_destination TEXT,
    p_departure_time TIME,
    p_arrival_time TIME,
    p_schedule DATE,
    p_total_seats INTEGER,
    p_available_seats INTEGER
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO train (train_name, train_number, source, destination, departure_time, arriva
l_time, schedule, total_seats, available_seats)
    VALUES (p_train_name, p_train_number, p_source, p_destination, p_departure_time, p_ar
rival_time, p_schedule, p_total_seats, p_available_seats);
END;
$$;
```

FUNCTIONS

```
CREATE OR REPLACE FUNCTION calculate_total_revenue()
RETURNS NUMERIC AS $$
DECLARE
    total_revenue NUMERIC;
BEGIN
```

```
    SELECT SUM(amount_paid) INTO total_revenue FROM payment WHERE status =
'Completed';
    RETURN total_revenue;
END;
$$ LANGUAGE plpgsql;
```

TRIGGERS

```
CREATE OR REPLACE FUNCTION update_total_revenue()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE admin SET total_revenue = calculate_total_revenue();
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_update_total_revenue
AFTER INSERT OR UPDATE ON payment
FOR EACH ROW
EXECUTE FUNCTION update_total_revenue();


CREATE OR REPLACE FUNCTION update_available_seats()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE train SET available_seats = available_seats - 1 WHERE train_id = NEW.train_id
;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
CREATE TRIGGER trg_update_available_seats
AFTER INSERT ON booking
FOR EACH ROW
EXECUTE FUNCTION update_available_seats()
```

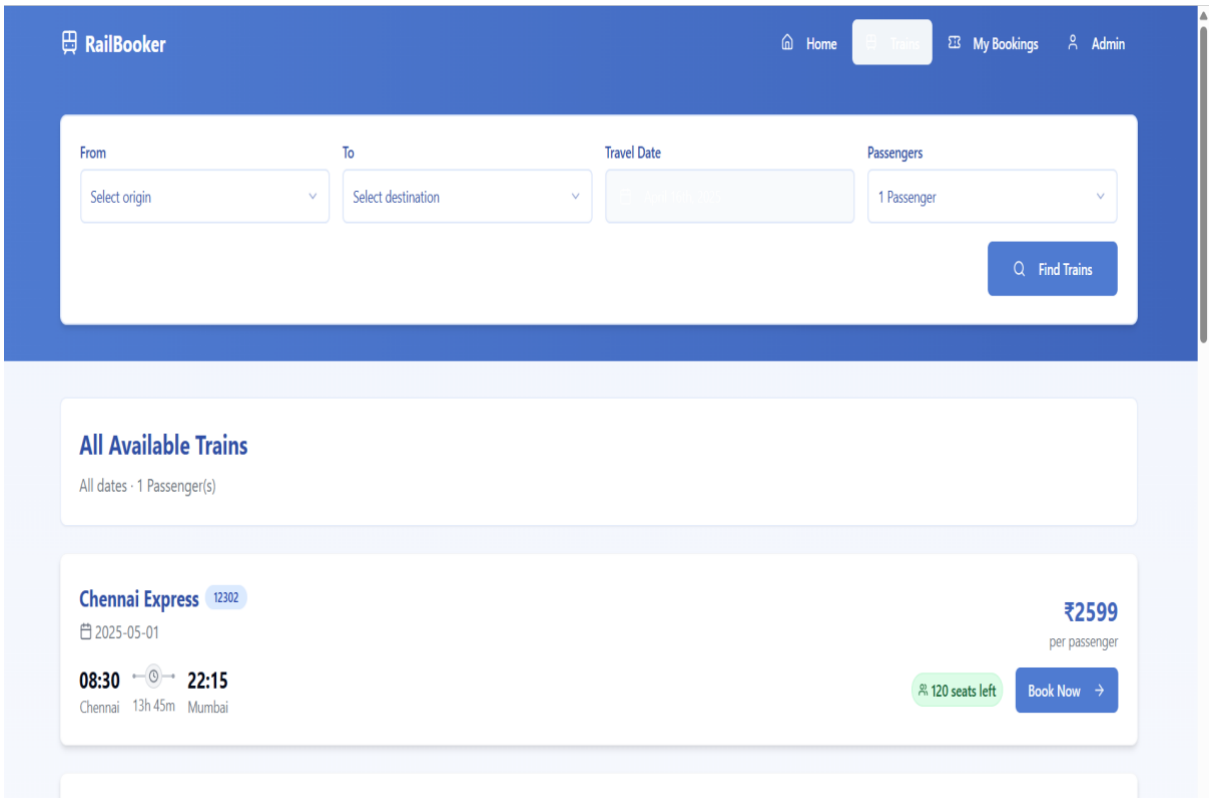
DB CONNECTIVITY

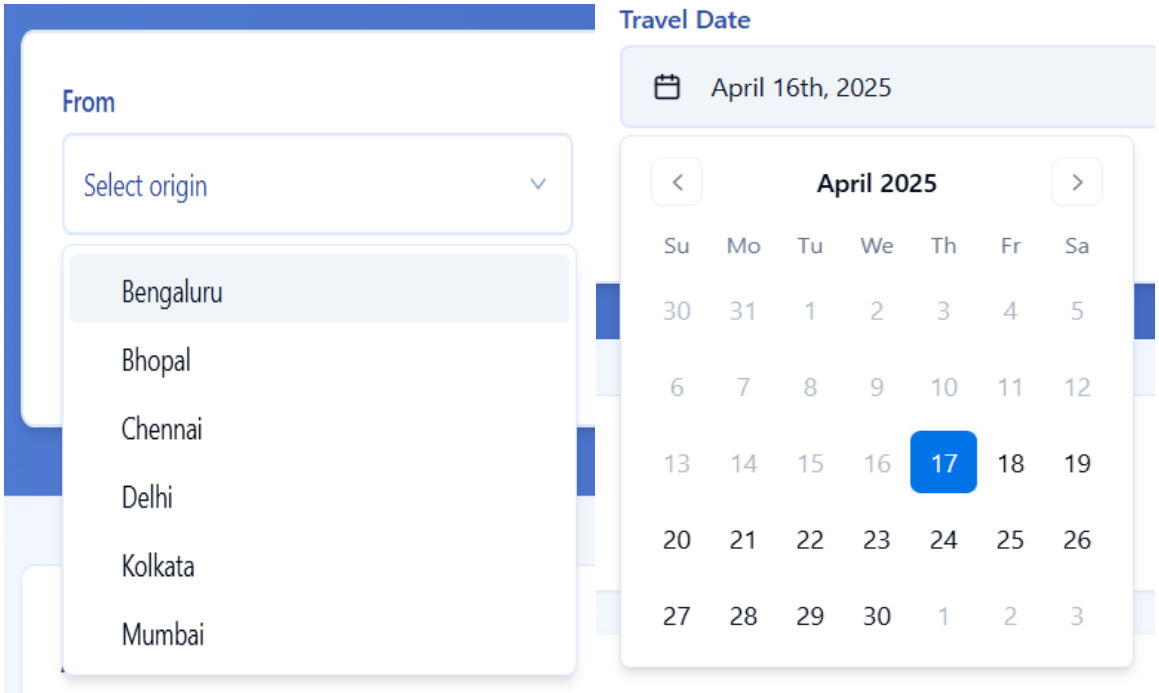
Supabse config.toml-

project_id = "esuxlfaecivujgdcqzmb"

```
import { createClient } from '@supabase/supabase-js';

import type { Database } from './types';


const SUPABASE_URL = "https://esuxlfaecivujgdcqzmb.supabase.co";

const SUPABASE_PUBLISHABLE_KEY =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImVzdX
hsZmFlY2l2dWpnZGNxem1iIiwicm9sZSI6ImFub24iLCJpYXQiOjE3NDQ3MzMwMzMsI
mV4cCI6MjA2MDMwOTAzM30.PEOsAHjGxEBKSWCuRSEZC6frpAy9pK69G86GjGKL
ec4";


// Import the supabase client like this:

// import { supabase } from "@/integrations/supabase/client";


export const supabase = createClient<Database>(SUPABASE_URL,
SUPABASE_PUBLISHABLE_KEY);
```

UI DESIGN


RailBooker
Home
Trains
My Bookings
Admin
From
Select origin
To
Select destination
Travel Date
Passengers
1 Passenger
Find Trains
All Available Trains
All dates · 1 Passenger(s)
Chennai Express
12302
2025-05-01
08:30
Chennai
13h 45m
22:15
Mumbai
₹2599
per passenger
120 seats left
Book Now


From
Select origin
Bengaluru
Bhopal
Chennai
Delhi
Kolkata
Mumbai
Travel Date
April 16th, 2025
April 2025
Su Mo Tu We Th Fr Sa
30 31 1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 1 2 3

Duronto Express 12261
2025-05-05
23:30 Kolkata — 16h 25m — 15:55 Delhi
₹2627 per passenger
160 seats left
Book Now

Karnataka Express 12628
2025-05-02
10:15 Delhi — 5h 30m — 15:45 Bengaluru
₹2330 per passenger
100 seats left
Book Now

RailBooker
Your trusted partner for hassle-free train travel across the country. Book tickets, check schedules, and enjoy a seamless journey.

Quick Links
Home
Find Trains
My Bookings
Cancellation Policy
FAQs

Popular Routes
Delhi to Mumbai
Chennai to Bangalore
Kolkata to Delhi
Mumbai to Goa
Hyderabad to Chennai

Contact Us
123 Railway Plaza, Central District
New Delhi, 110001
+91 1800-123-4567
support@railbooker.com

Passenger 1

Full Name

Age

Gender
Male Female Other

Contact Information

Email Address

Phone Number

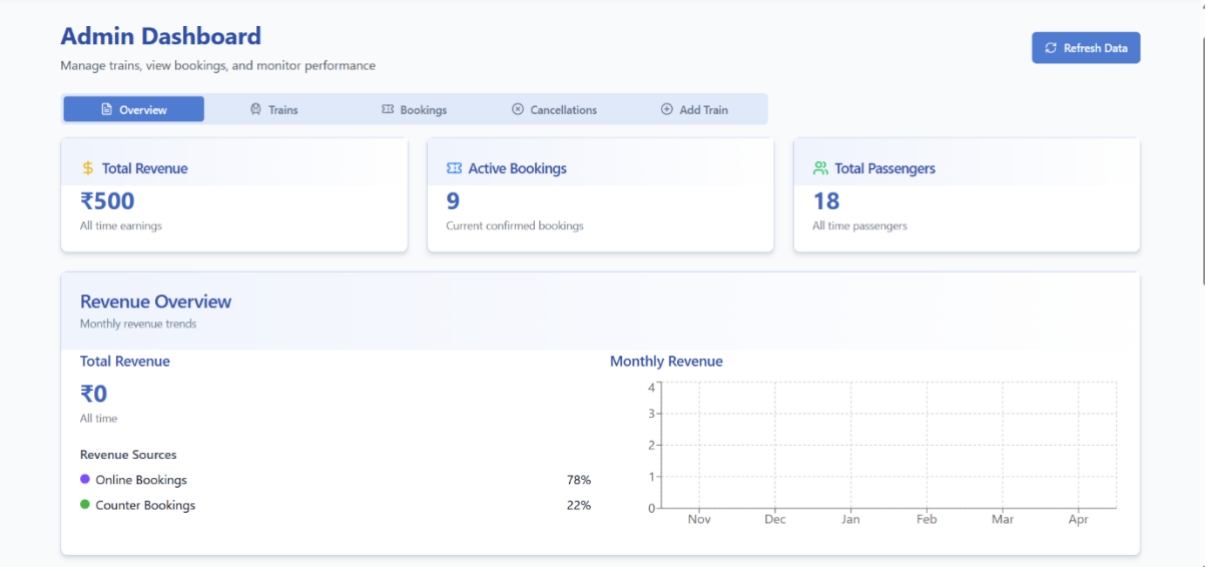
Payment Method

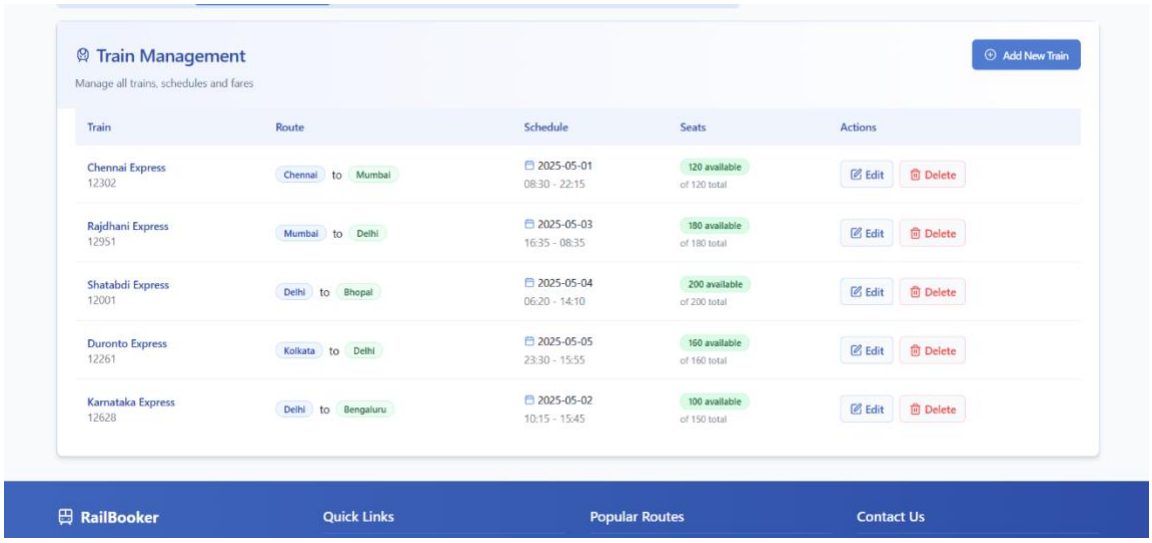
Credit Card
UPI
Net Banking

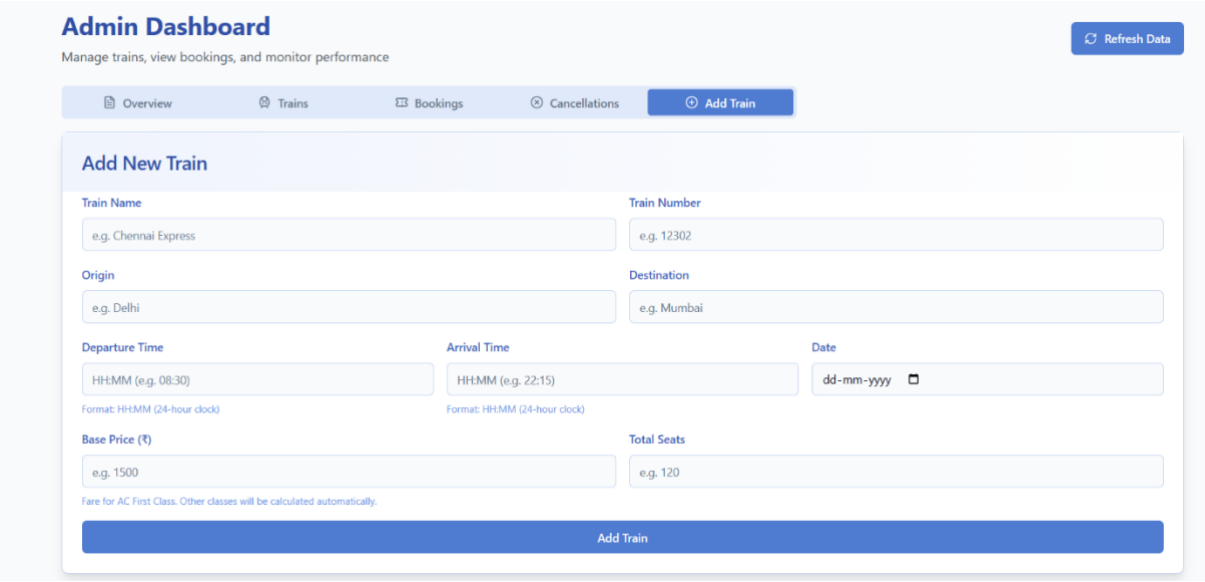
Fare Summary

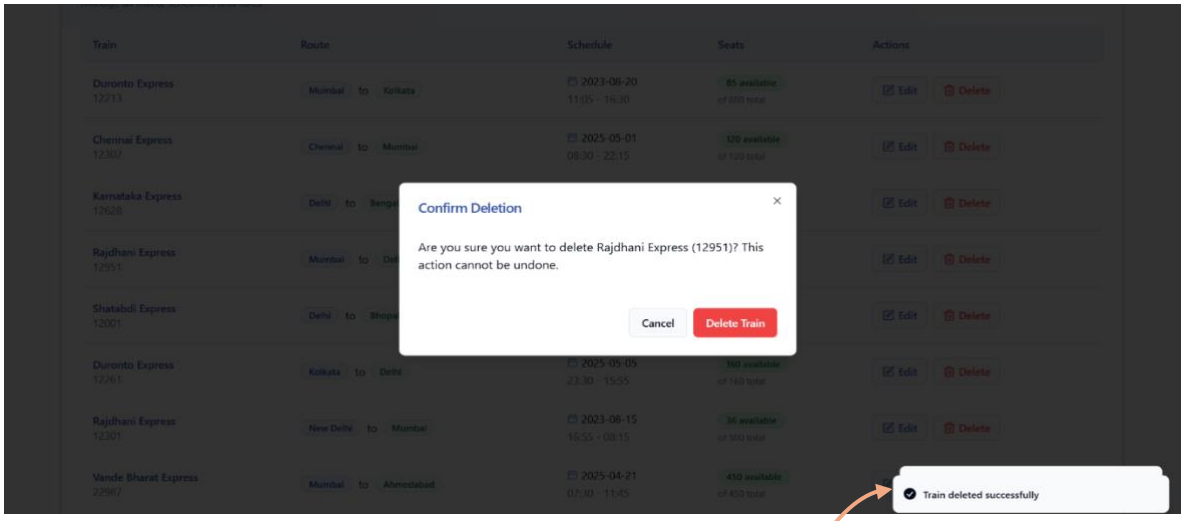
Base Fare per passenger	₹2330
Total Base Fare (1 passengers)	₹2330
Service Fee	₹50
Total Amount	**₹2380**

Confirm & Pay ₹2380


Admin Dashboard
Manage trains, view bookings, and monitor performance
Refresh Data
Overview
Trains
Bookings
Cancellations
Add Train
Total Revenue
₹500
All time earnings
Active Bookings
9
Current confirmed bookings
Total Passengers
18
All time passengers
Revenue Overview
Monthly revenue trends
Total Revenue
₹0
All time
Revenue Sources
Online Bookings 78%
Counter Bookings 22%
Monthly Revenue
4
3
2
1
0
Nov
Dec
Jan
Feb
Mar
Apr


Train Management
Manage all trains, schedules and fares
Add New Train
Train
Route
Schedule
Seats
Actions
Chennai Express 12302
Chennai to Mumbai
2025-05-01 08:30 - 22:15
120 available of 120 total
Edit Delete
Rajdhani Express 12951
Mumbai to Delhi
2025-05-03 16:35 - 08:35
180 available of 180 total
Edit Delete
Shatabdi Express 12001
Delhi to Bhopal
2025-05-04 06:20 - 14:10
200 available of 200 total
Edit Delete
Duronto Express 12261
Kolkata to Delhi
2025-05-05 23:30 - 15:55
160 available of 160 total
Edit Delete
Karnataka Express 12628
Delhi to Bengaluru
2025-05-02 10:15 - 15:45
100 available of 150 total
Edit Delete
RailBooker
Quick Links
Popular Routes
Contact Us


Admin Dashboard
Manage trains, view bookings, and monitor performance
Refresh Data
Overview
Trains
Bookings
Cancellations
Add Train
Add New Train
Train Name
e.g. Chennai Express
Train Number
e.g. 12302
Origin
e.g. Delhi
Destination
e.g. Mumbai
Departure Time
HH:MM (e.g. 08:30)
Format: HH:MM (24-hour clock)
Arrival Time
HH:MM (e.g. 22:15)
Format: HH:MM (24-hour clock)
Date
dd-mm-yyyy
Base Price (₹)
e.g. 1500
Total Seats
e.g. 120
Fare for AC First Class. Other classes will be calculated automatically.
Add Train




```python
// Delete a train (admin only)
export const deleteTrain = async (trainId: string): Promise<void> => {
  try {
    // Check if train has associated bookings
    const { count, error: countError } = await supabase
      .from('booking')
      .select('*', { count: 'exact', head: true })
      .eq('train_id', trainId);

    if (countError) throw countError;

    if (count && count > 0) {
      throw new Error(`Cannot delete train with ${count} existing bookings`);
    }
    // Delete all associated fares first
    const { error: fareDeleteError } = await supabase
      .from('fare')
      .delete()
      .eq('train_id', trainId);

    if (fareDeleteError) throw fareDeleteError;

    // Delete the train
    const { error: trainDeleteError } = await supabase
      .from('train')
      .delete()
      .eq('train_id', trainId);

    if (trainDeleteError) throw trainDeleteError;

    toast.success('Train deleted successfully');
  } catch (error: any) {
    console.error('Error deleting train:', error);
    toast.error('Failed to delete train: ' + error.message);
    throw error;
  }
};
```

REFERENCES

- React.js + TypeScript+ Vite+ shadcn-ui+ Tailwind CSS (Frontend)

React Official Documentation:
https://reactjs.org/docs/getting-started.html
TypeScript Official Documentation:
https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

- Node.js + Supabase (Backend)
- Node.js Official Docs: https://nodejs.org/en/docs

  Supabase: https://supabase.com/

- https://nodejs.org/en/docs
- https://expressjs.com/
- https://www.postgresql.org/docs/
- https://www.npmjs.com/package/dotenv
- https://tailwindcss.com/docs