

**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**

|  |  |  |
| --- | --- | --- |
| S.NO | TITLE | PAGE NO. |
| 1. | ABSTRACT | 3 |
| 2. | PROBLEM STATEMENT | 3 |
| 3. | ER DIAGRAM | 4 |
| 4. | RELATIONAL TABLES | 5 |
| 5. | DDL COMMANDS | 5-8 |
| 6. | SQL QUERIES | 8-9 |
| 7. | PL/SQL | 10-11 |
| 8. | UI DESIGN | 13-16 |
| 9. | REFERENCES | 17 |

**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.

α →β **,** where α ⊆ R andβ ⊆ R , α is a superkey for R.

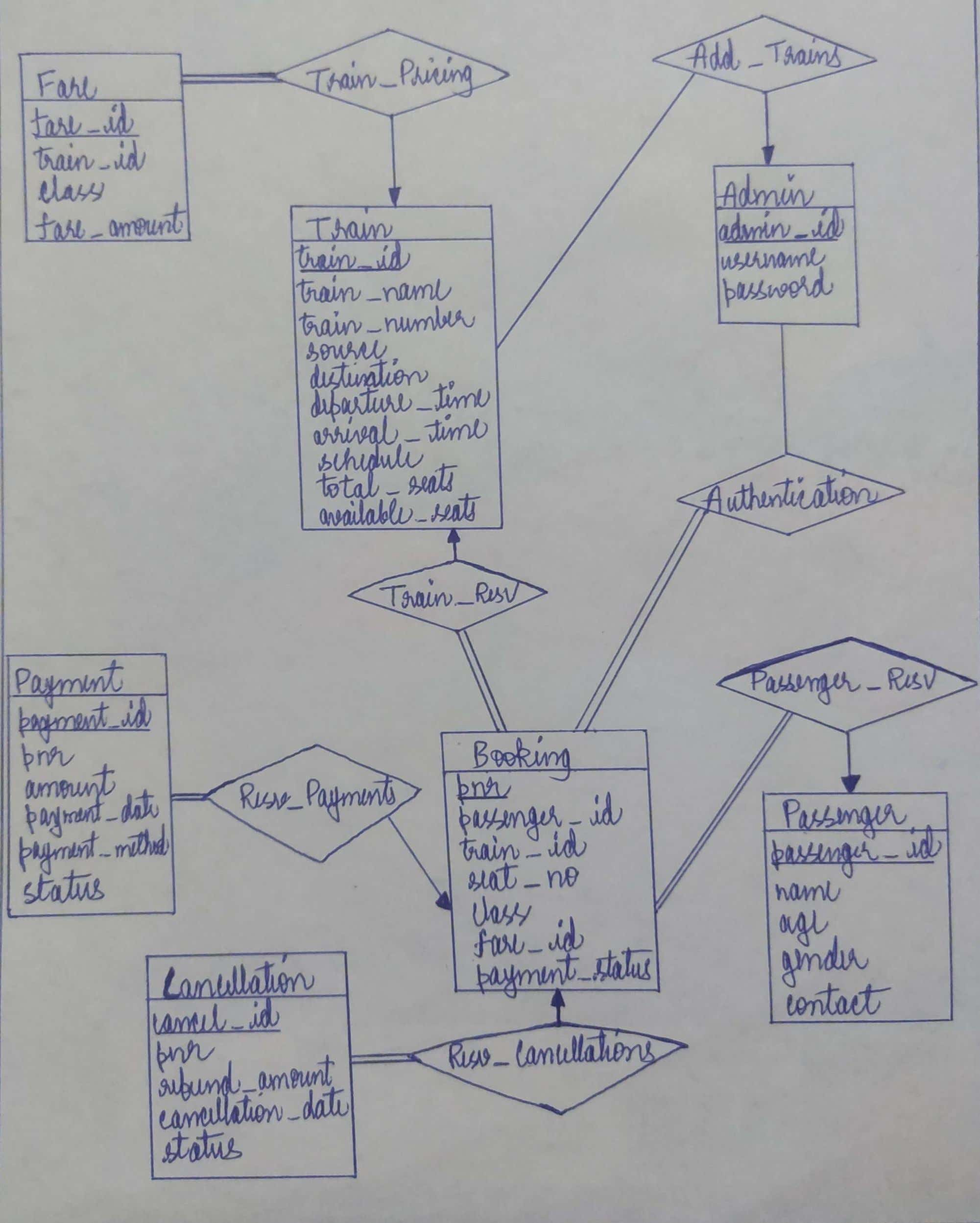
Here ‘α’ is the primary key of the table or a superset of it.

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**

****

**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

);

-- 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' );

1. 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, arrival\_time, schedule, total\_seats, available\_seats)

    VALUES (p\_train\_name, p\_train\_number, p\_source, p\_destination, p\_departure\_time, p\_arrival\_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";

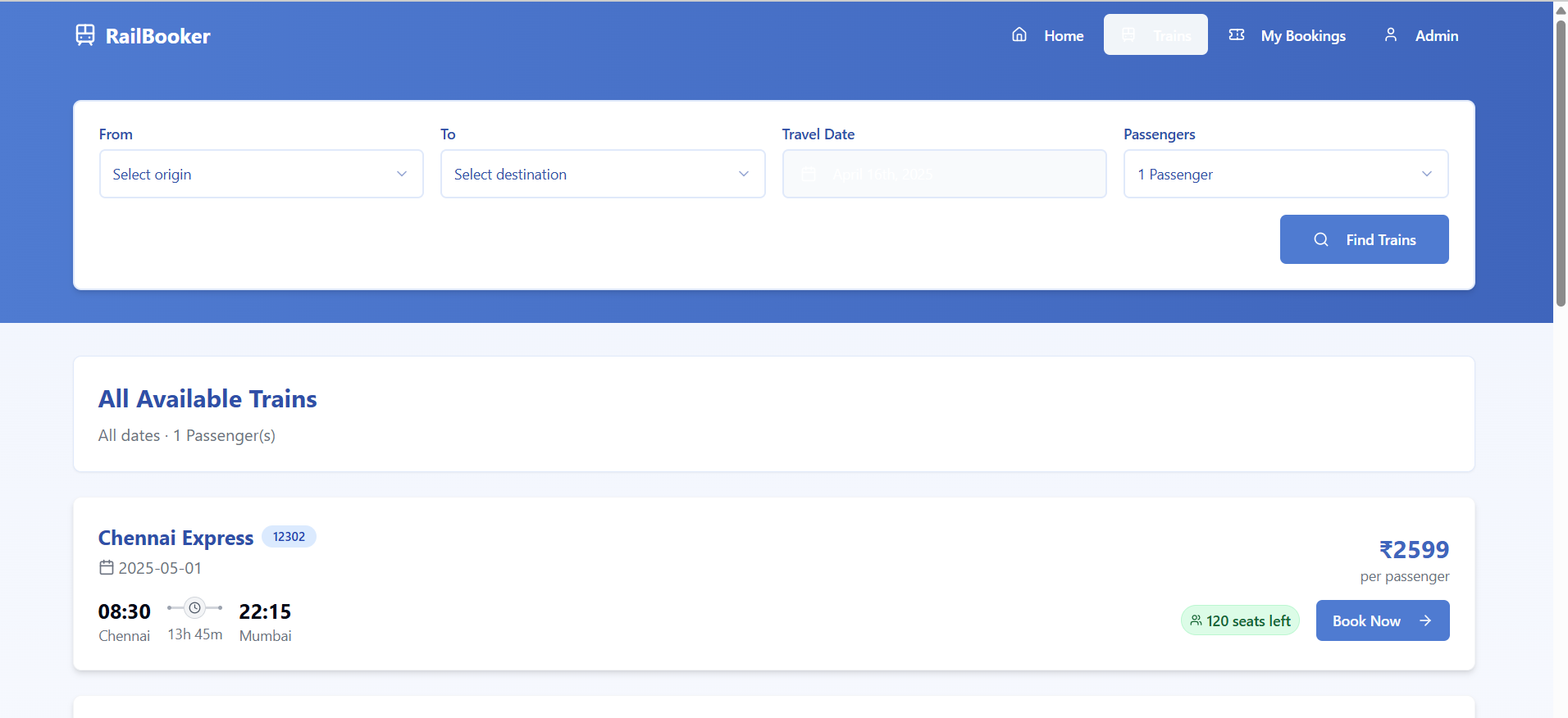
const SUPABASE\_PUBLISHABLE\_KEY = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6ImVzdXhsZmFlY2l2dWpnZGNxem1iIiwicm9sZSI6ImFub24iLCJpYXQiOjE3NDQ3MzMwMzMsImV4cCI6MjA2MDMwOTAzM30.PEOsAHjGxEBKSWCuRSEZC6frpAy9pK69G86GjGKLec4";

// Import the supabase client like this:

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

export const supabase = createClient<Database>(SUPABASE\_URL, SUPABASE\_PUBLISHABLE\_KEY);

**UI DESIGN**



A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a calendar

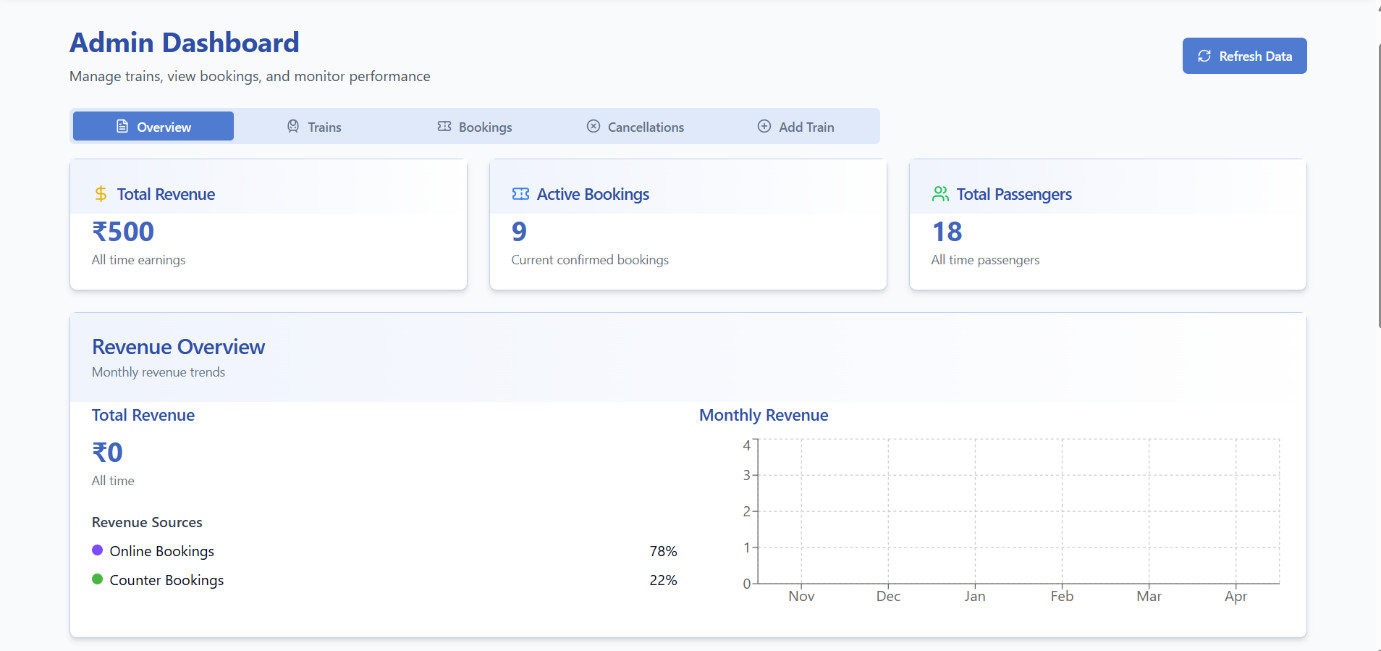
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

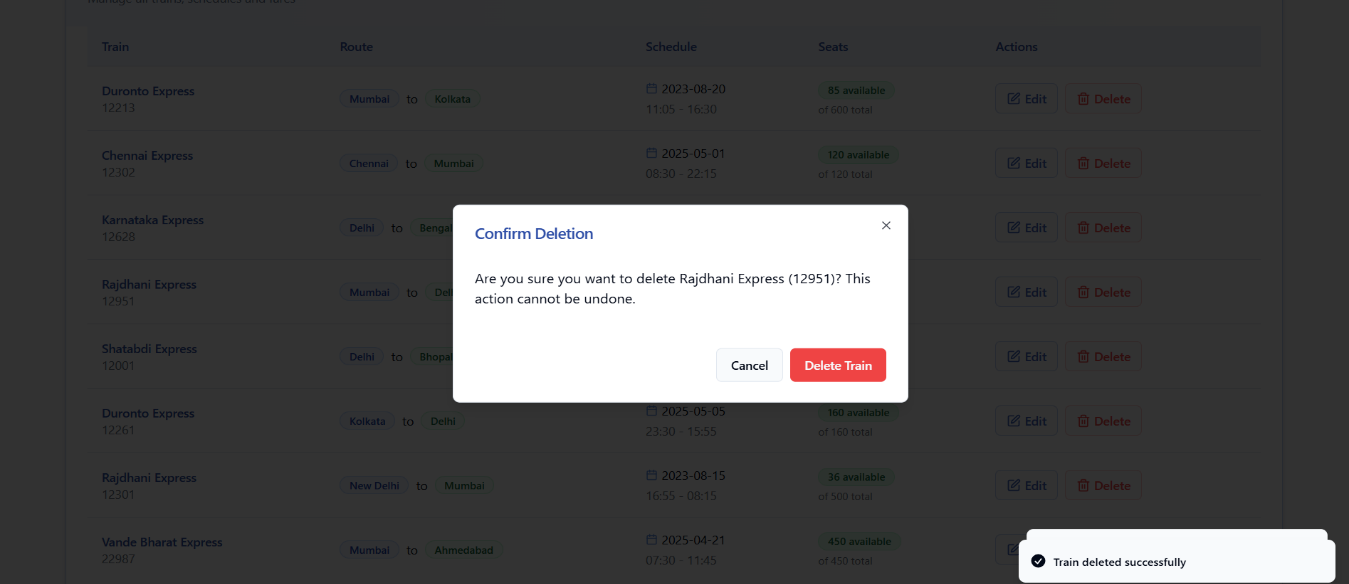


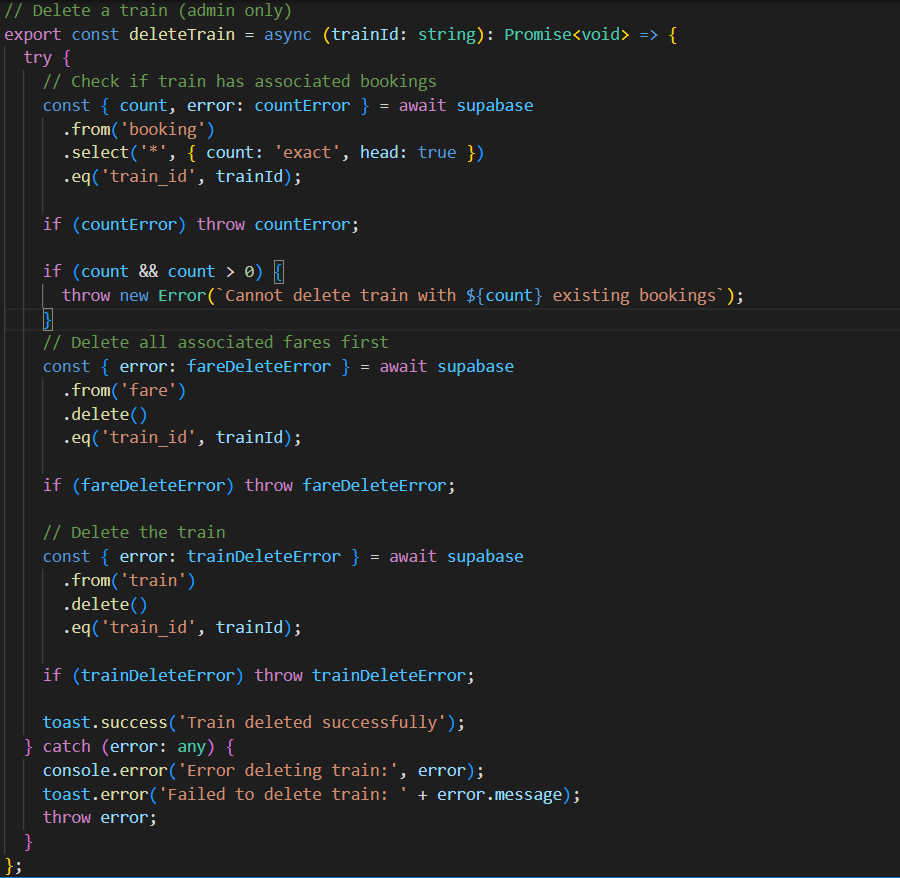
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

****

****

**REFERENCES**

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

React Official Documentation:

<https://reactjs.org/docs/getting>[-started.html](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](https://nodejs.org/en/docs%20%20)

Supabase: [https://supabase.com/](https://supabase.com/%20)

* [https://nodejs.org/en/docs](https://nodejs.org/en/docs%20)
* [https://expressjs.com/](https://expressjs.com/%20)
* [https://www.postgresql.org/docs/](https://www.postgresql.org/docs/%20)
* [https://www.npmjs.com/package/dotenv](https://www.npmjs.com/package/dotenv%20)
* [https://tailwindcss.com/docs](https://tailwindcss.com/docs%20)