RAILWAY ANALYTICS SYSTEM USING MERN STACK

MINI PROJECT REPORT

Submitted by

DHARSHINI R (9517202109014)
SANDHIYA DEVI B (9517202109046)
SHRI SUBIKSHA S T (9517202109048)

in

19AD581 –NO-SQL DATABASES LABORATORY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

OCTOBER 2023

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **DHARSHINI R (9517202109014), SANDHIYA DEVI B (9517202109046), SHRI SUBIKSHA S T(9517202109048)** for the mini project titled **“RAILWAY ANALYTICS SYSTEM USING MERN STACK”** in 19AD581 –**NO-SQL DATABASES LABORATORY** during the fifth semester July 2023 – October2023 under my supervision.

SIGNATURE	**SIGNATURE**
Dr.P.Thendral,	**Dr. J. Angela Jennifa Sujana,**
Asst. Professor (Senior Grade),	**Professor & Head,**
AI&DS Department,	AI&DS Department,
Mepco Schlenk Engg. College, Sivakasi	Mepco Schlenk Engg. College, Sivakasi.

ABSTRACT

The Railway Analytics System with MongoDB Integration is a specialized project aimed at optimizing and enhancing railway operations through the efficient utilization of MongoDB, a NoSQL database management system. This project focuses on leveraging MongoDB queries and data modeling techniques to enhance data management, retrieval, and analysis within the railway network. The primary objectives of this system are to streamline data storage, retrieval, and reporting for improved operational decision-making and resource utilization. The Key components of the Railway Analytics System include the design of a robust MongoDB database schema tailored to railway data, the creation of efficient MongoDB queries for data extraction, and the development of user-friendly interfaces for railway personnel to interact with the system. By structuring data within MongoDB and using optimized queries, the system aims to provide railway operators with rapid access to critical information, such as scheduling, maintenance records, and passenger statistics. The project focuses on the efficient storage and retrieval of data relevant to railway operations, including train schedules, maintenance logs, ticket sales, and passenger information. The MongoDB database is designed to accommodate the high volume and variety of data generated within the railway network. While not explicitly focused on data analytics, the Railway Analytics System with MongoDB Integration indirectly supports informed decision-making by providing quick and easy access to essential operational data. This streamlined access can help operators respond to real-time challenges, optimize schedules, and manage resources effectively.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

The Railway Analytics System, a cutting-edge web application built upon the powerful foundations of MongoDB and React.js, redefines the management and analysis of railway operations. With MongoDB's NoSQL database at its core, this system efficiently handles and scales to the vast and intricate datasets intrinsic to railway operations, including train schedules, passenger information, and maintenance records. Coupled with the responsive and intuitive user interface powered by React.js, it empowers railway staff to access real-time data, monitor train movements, optimize passenger loads, and generate insightful reports, all contributing to enhanced decision-making, operational efficiency, and passenger satisfaction within the railway industry.

1.2 TECHNOLOGY DESCRIPTION

MERN STACK

The MERN Stack is a popular and powerful web development stack that consists of four key technologies: MongoDB, Express.js, React, and Node.js. Each component plays a unique role in building full-stack web applications.

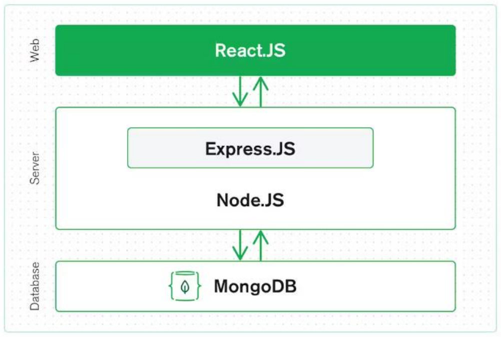


Fig 1.1 MERN Stack Architecture

Here's a brief description of each component in the MERN Stack,

MongoDB

Database - MongoDB is a NoSQL, document-oriented database that stores data in a flexible, JSON-like format called BSON. It is well-suited for handling unstructured or semi-structured data and provides scalability and high performance. MongoDB is used to store and manage the application's data.

Express.js

Server Framework - Express.js is a minimal and flexible Node.js web application framework that simplifies the process of building robust and scalable server-side applications. It provides a set of tools and middleware for creating RESTful APIs and handling HTTP requests and responses. Express.js is the back-end framework that handles the server-side logic and interacts with the database.

React

Front-end Library - React is a JavaScript library developed by Facebook for building user interfaces. It enables developers to create interactive and dynamic user interfaces with a component-based architecture. React is used on the client side to create the user interface, handle user interactions, and display data fetched from the server.

Node.js

Runtime Environment - Node.js is a server-side runtime environment that allows JavaScript to be executed outside the web browser. It is built on Chrome's V8 JavaScript engine and provides an event-driven, non-blocking I/O model, making it ideal for building scalable and performant server-side applications. In the MERN Stack, Node.js serves as the server runtime environment and runs the Express.js application.

The MERN Stack's architecture promotes the development of isomorphic applications, where JavaScript is used on both the server and client sides, allowing for seamless data flow and improved performance. This stack is known for its flexibility, speed, and ability to handle real-time updates, making it a popular choice for building a wide range of web applications, from single-page applications (SPAs) to e-commerce platforms, social networks, and more. It is especially well-suited for developers who want to work with a unified technology stack for both the front-end and back-end, promoting code reusability and rapid development.

CHAPTER 2

PROPOSED SYSTEM

2.1 FRONTEND DESIGN

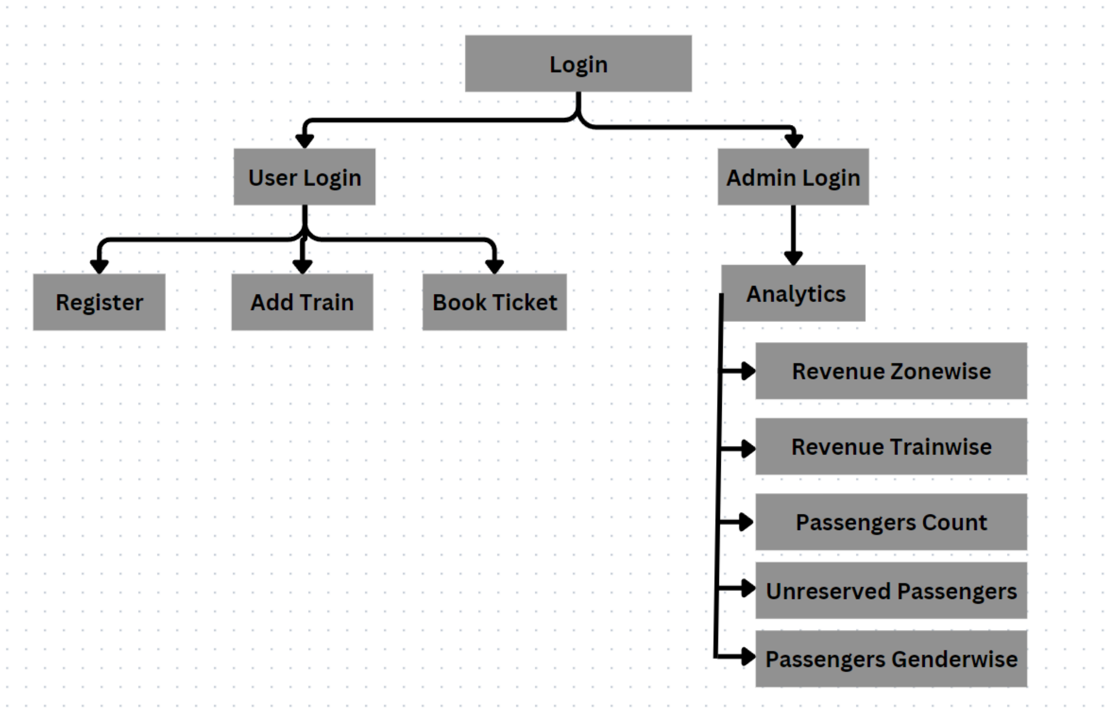
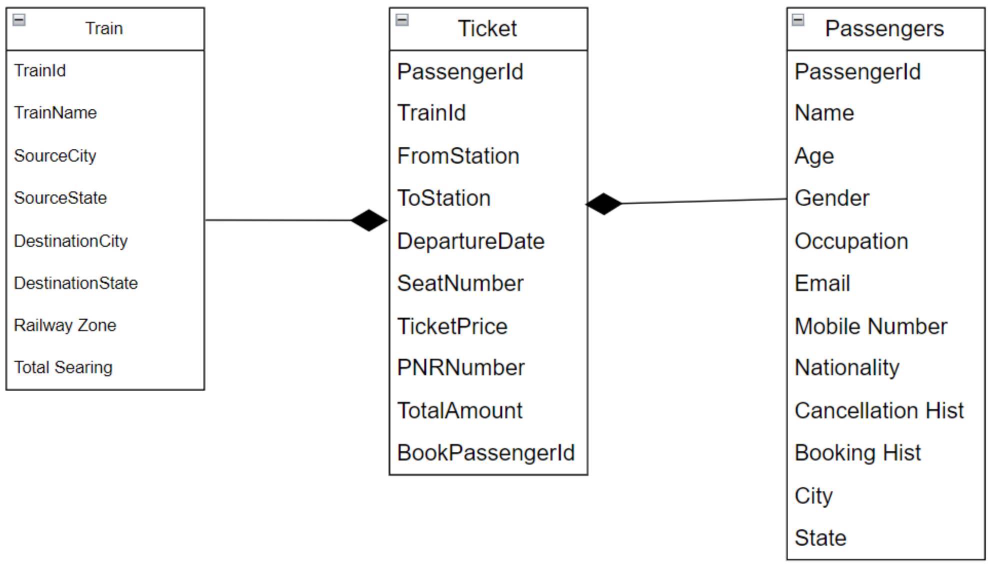


Fig 2.1 Proposed System

The proposed system is designed to enhance the user experience for managing passenger train bookings. It consists of two main roles namely user and admin, each with specific functionalities and access levels. Users will be provided with secure login credentials to access the system. User authentication ensures data privacy and access control. Users can add new passenger trains to the system. Train details, such as name, route, and departure times, can be provided. Users can search for available trains based on their preferences (e.g., date, destination). Once a suitable train is found, users can book tickets for themselves or others. The system will record passenger information and generate booking details. Admins have access to the database, allowing them to retrieve and analyze data. They can generate reports on various aspects, such as passenger statistics, popular routes, or revenue analysis. The user interface is designed to be intuitive and user-friendly, ensuring a seamless booking experience for users. Admin interfaces will be logically organized for easy access to data and analysis.

2.2 DATABASE DESIGN

2.2.1.AGGREGATE MODEL



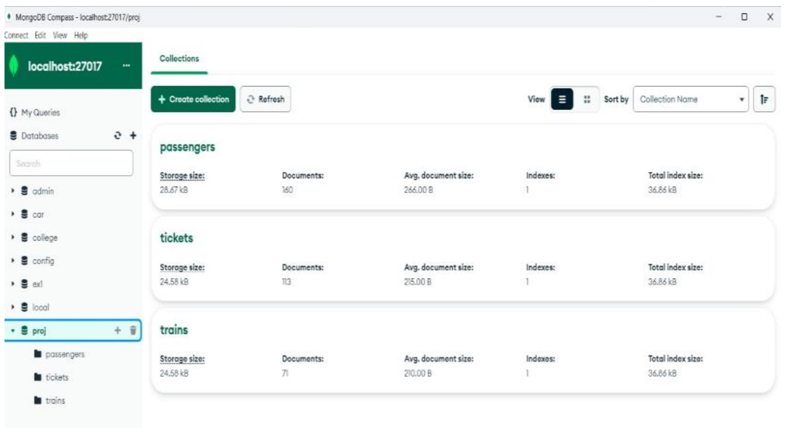
•

Fig 2.2 Aggregate Model of Railways Database

The aggregate data model for the railway system comprises three key collections: Train, Ticket, and Passenger. These collections are interrelated to manage and organize data efficiently for seamless train booking and passenger management. The Train collection stores information about the trains available, including their routes, departure times, and the number of available seats. The Ticket collection manages individual tickets, linking each ticket to a specific train and passenger. It also tracks booking dates, seat assignments, and ticket status. The Passenger collection stores passenger details and references the tickets associated with each passenger. Tickets are associated with specific trains via the train_id attribute in the Ticket collection. Passengers are connected to their respective tickets through an array of ticket IDs in the Passenger collection. This aggregate data model effectively organizes information about trains, passenger bookings, and passenger details. It enables efficient data retrieval and maintenance, facilitating seamless railway operations and passenger management.

2.2.2.DATABASE:

Railways



passengers

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
28.67 kB	160	266.00 B	1	36.86 kB

tickets

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
24.58 kB	113	215.00 B	1	36.86 kB

trains

Storage size:	Documents:	Avg. document size:	Indexes:	Total index size:
24.58 kB	71	210.00 B	1	36.86 kB

Fig 2.3 Railways Database

COLLECTIONS:

1.Passenger

This Collection stores information about passengers.

Data Types:

_id: ObjectId

passengerId: Integer

name: String

age: Integer

gender: String

occupation: String

email: String

mobileNum: Number

nationality: String

cancellationHistory: Object

bookingHistory: Object

addressCity: String

addressState: String

Sample Data:

_id:653949dd058e77c90397e8f9

passengerId:102

name:"Emily Johnson"

age:30

gender:"Female"

occupation:"Government"

email:"emily.johnson@gmail.com"

mobileNum:9894903888

nationality:"Indian"

cancellationHistory:Array (2)

bookingHistory:Array (1)

addressCity:"Kolkata"

addressState:"West Bengal"

2.Train

This Collection stores information about Trains.

Data Types:

_id: ObjectId

trainId: Integer

trainName: String

srccity: String

srcstate: String

descity: String

desstate: String

railwayZone: String

totalSeatingCapacity: Number

Sample Data:

_id: 65395058058e77c90397e992

trainId: 34567

trainName: "Duronto Express"

srccity: "Kolkata"

srcstate: "West Bengal"

descity: "Mumbai"

desstate: "Maharashtra"

railwayZone: "Central Railway"

totalSeatingCapacity: 1500

3.Ticket

This Collection stores information about Ticket.

Data Types:

_id: ObjectId

passengerId: Integer

trainId: Integer

fromStation: String

toStation: String

departureDate: Date

seatNumber: Integer

ticketPrice: Integer

pnrNumber: Integer

travelStatus: String

totalAmount: Integer

Sample Data:

_id: 65395b3d058e77c90397e9fa

passengerId: 131

trainId: 12348

fromStation: "Delhi"

toStation: "Kolkata"

departureDate: 2023-05-10T00:00:00.000+00:00

ticketPrice: 400

pnrNumber: 1005

travelStatus: "cancelled"

totalAmount: 400

2.2.3.USE CASE DIAGRAM

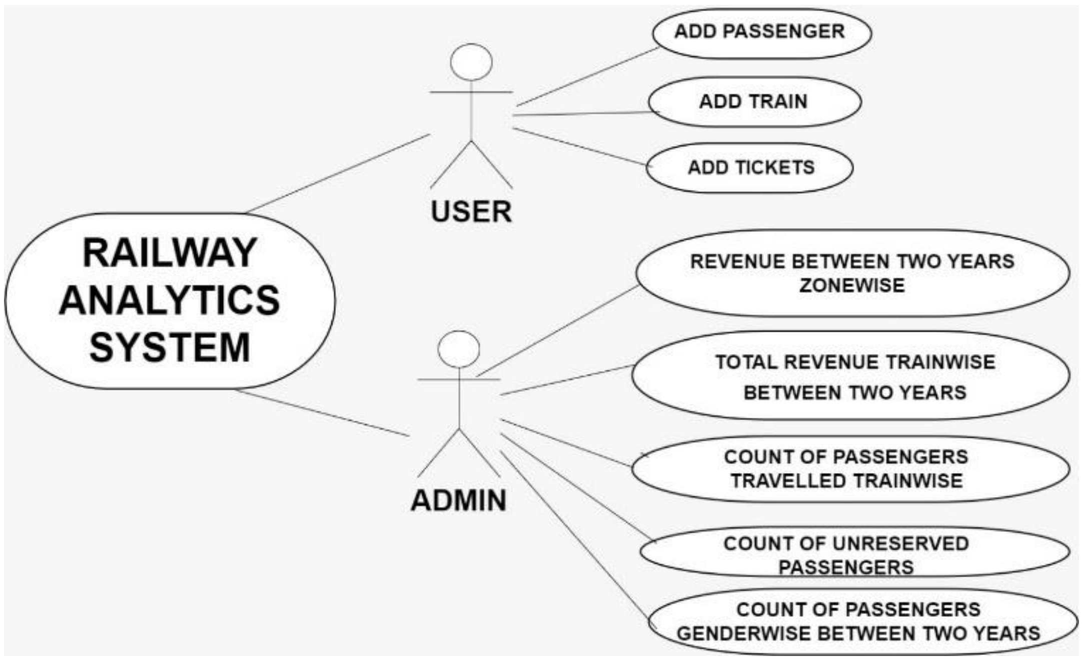

RAILWAY ANALYTICS SYSTEM
USER
ADMIN
ADD PASSENGER
ADD TRAIN
ADD TICKETS
REVENUE BETWEEN TWO YEARS ZONEWISE
TOTAL REVENUE TRAINWISE BETWEEN TWO YEARS
COUNT OF PASSENGERS TRAVELLED TRAINWISE
COUNT OF UNRESERVED PASSENGERS
COUNT OF PASSENGERS GENDERWISE BETWEEN TWO YEARS

Fig 2.4 Use case Diagram

2.3 MODULE DESCRIPTION

2.3.1 Passenger Registration

The AddPassenger module serves the purpose of registering passengers in the railway system. It offers a user-friendly interface for users to input various passenger details, including personal information, contact information, and travel history. This module enables user registration by collecting and storing passenger details, including name, age, date of birth, gender, and occupation. Users can provide contact details such as email address and mobile number, ensuring accurate data entry with validation. Users can select the passenger's city and state from dropdown menus, ensuring precise location data. Users have the option to input the passenger's travel history, including booking and cancellation history, using Passenger Name Record (PNR) numbers. As users interact with the form, the formData object updates in real-time, ensuring data accuracy and user convenience. When users submit the registration form, the module sends the provided data to a server via an HTTP POST request to the 'api/v1/user/AddPassenger' endpoint using the axios library. The module employs the 'message' component from the "antd" library to provide immediate user feedback, displaying success messages for successful passenger registration or error messages for any encountered issues. The module's user interface is designed for ease of use, with well-structured sections for different types of passenger information, simplifying the data entry process.

2.3.2 Train Details Insertion

The primary use case of this module is to allow administrators or authorized personnel to register new trains into the system. This includes providing essential information about the train, such as its ID, name, source and destination stations, railway zone, and seating capacity. When a new train route is established, the system can use this module to add the train responsible for servicing that route. It ensures that the route and train information are kept up to date. Train operators or organizations expanding their train fleet can use this module to add new trains to their existing operations. This can be crucial when a company acquires new trains or introduces new services. In cases where an existing train needs to be re-registered due to updates or changes in the system, this module allows for making modifications to the train's details.

2.3.3 Ticket Booking

Passengers can use this module to reserve train tickets. They provide details such as their Aadhar number (identification), the train they want to board, departure date, and the number of seats they want to book. The module calculates the total ticket fare based on the provided details, including the number of seats and ticket price. It generates a unique PNR (Passenger Name Record) number for each ticket booked. The PNR can be used for tracking and managing reservations. Passengers can indicate their travel status, whether they have traveled or canceled their reservation. This information is crucial for tracking the occupancy of the train. The module calculates the total fare for the booked tickets, which can be used for billing and payment processing. Travel status information can be used for obtaining feedback from passengers. For example, if a passenger cancels a ticket, they may be asked to provide a reason for cancellation. In cases where passengers need to modify their ticket details, such as changing the number of seats or the travel date, this module can be used for making adjustments.

2.3.4 Analytics Module

Analytics Module is used to perform analytics on five topics mainly calculating total revenue zone-wise, calculating total revenue train-wise, count of unreserved passengers, count of passengers train-wise and count of passengers gender-wise. The following are the queries used to perform the analytics.

Calculating Revenue Zone wise:

```
db.tickets.aggregate([
        {
            $lookup: {
              from: "passengers",
              localField: "passengerId",
              foreignField: "passengerId",
              as: "passengerData"
            }
          },
          {
            $lookup: {
              from: "trains",
              localField: "trainId",
              foreignField: "trainId",
              as: "trainData"
            }
          },
          {
            $match: {
              "departureDate": {
                $gte:st,//start date from user
                $lt: ey// end date from user
              }
            }
          },
          {
            $unwind: "$passengerData"
          },
          {
            $unwind: "$trainData"
          },
          {
            $group: {
              _id: "$trainData.railwayZone",
              totalRevenue: {$sum: "$totalAmount"}
            }
          },
          {
            $project: {
              _id: 0,
              railwayZone: "$_id",
              totalRevenue: 1
            }
          }
    ]);
```

This aggregation pipeline retrieves ticket data, including passenger and train information, filters it by departure date, groups it by railway zone, and calculates the total revenue for each zone during a specified date range. The final output is a document with "railwayZone" and "totalRevenue" fields for analysis and reporting.

Calculating Revenue Train wise:

```
db.tickets.aggregate([

        {
            $lookup: {
              from: "passengers",
              localField: "passengerId",
              foreignField: "passengerId",
              as: "passengerData"
            }
          },
          {
            $lookup: {
              from: "trains",
              localField: "trainId",
              foreignField: "trainId",
              as: "trainData"
            }
          },
          {
            $match: {
              "departureDate": {
                $gte:st,
                $lt: ey
              }
            }
          },
          {
            $unwind: "$passengerData"
          },
          {
            $unwind: "$trainData"
          },
          {
            $group: {
              _id: "$trainData.trainId",
              totalRevenue: { $sum: "$totalAmount" }
            }
          },
          {
            $project: {
              _id: 0,
              railwayZone: "$_id",
              totalRevenue: 1
            }
          }
    ]);
```

This aggregation pipeline retrieves ticket data, including passenger and train information, filters it by departure date, groups it by Train id, and calculates the total revenue for each train during a specified date range. The final output is a document with "trainId" and "totalRevenue" fields for analysis and reporting.

Count of passengers Train wise:

```
db.tickets.aggregate([

        {
            $lookup: {
              from: "passengers",
              localField: "passengerId",
              foreignField: "passengerId",
              as: "passengerData"
            }
          },
          {
            $lookup: {
              from: "trains",
              localField: "trainId",
              foreignField: "trainId",
              as: "trainData"
            }
          },
          {
            $match: {
              "departureDate": {
                $gte:st,
                $lt: ey
              }
            }
          },
          {
            $unwind: "$passengerData"
          },
          {
            $unwind: "$trainData"
          },
          {
            $group: {
              _id: "$trainData.trainId",
              totalcount: { $sum: 1 }
            }
          },
          {
            $project: {
              _id: 0,
              railwayZone: "$_id",
              totalcount: 1
            }
          }
    ]);
```

The pipeline is to return a list of train identifiers and the corresponding counts of tickets for each train in the specified date range giving the count of passengers.

Count of unreserved passengers:

```
db.passengers.aggregate([


      {
        $match: {


          "bookingHistory": { $size: 1}


        }
      },
      {
        $group: {
          _id: null,
          passengerCount: { $sum: 1 }
        }
      },
      {
        $project: {
          _id: 0,
          passengerCount: 1
        }
      }
]);
```

The pipeline is to return a count of passengers travelling unreserved.

Count of passengers gender wise:

```
db.tickets.aggregate([


        {
            $lookup: {
              from: "passengers",
              localField: "passengerId",
              foreignField: "passengerId",
              as: "passengerData"
            }
          },
          {
            $lookup: {
              from: "trains",
              localField: "trainId",
              foreignField: "trainId",
              as: "trainData"
            }
          },
          {
            $match: {
              "departureDate": {
                $gte:st,
                $lt: ey
              }
            }
          },
          {
            $unwind: "$passengerData"
          },
          {
            $unwind: "$trainData"
          },
          {
            $group: {
              _id: "$passengerData.gender",
              totalcount: { $sum: 1 }
            }
          },
          {
            $project: {
              _id: 0,
              gender: "$_id",
              totalcount: 1
            }
          }
    ]);
```

The pipeline is to return a list of genders and the corresponding counts of tickets for each gender in the specified date range.

CHAPTER 3

IMPLEMENTATION

3.1 RESULTS

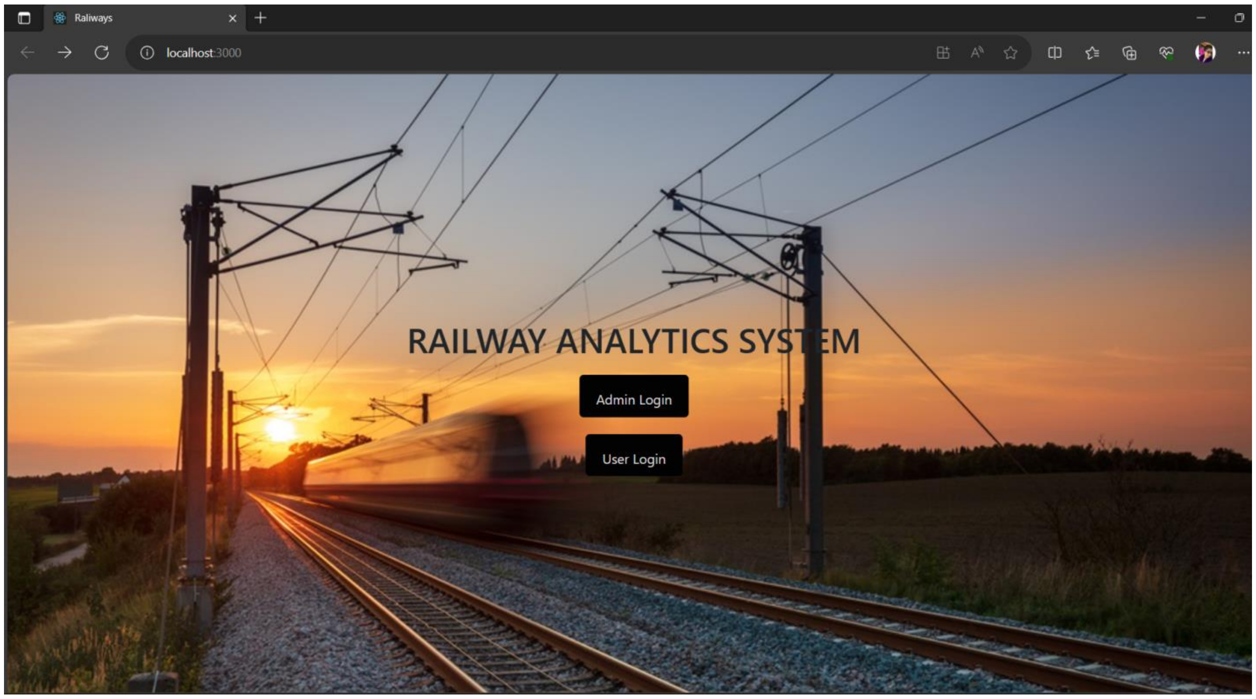
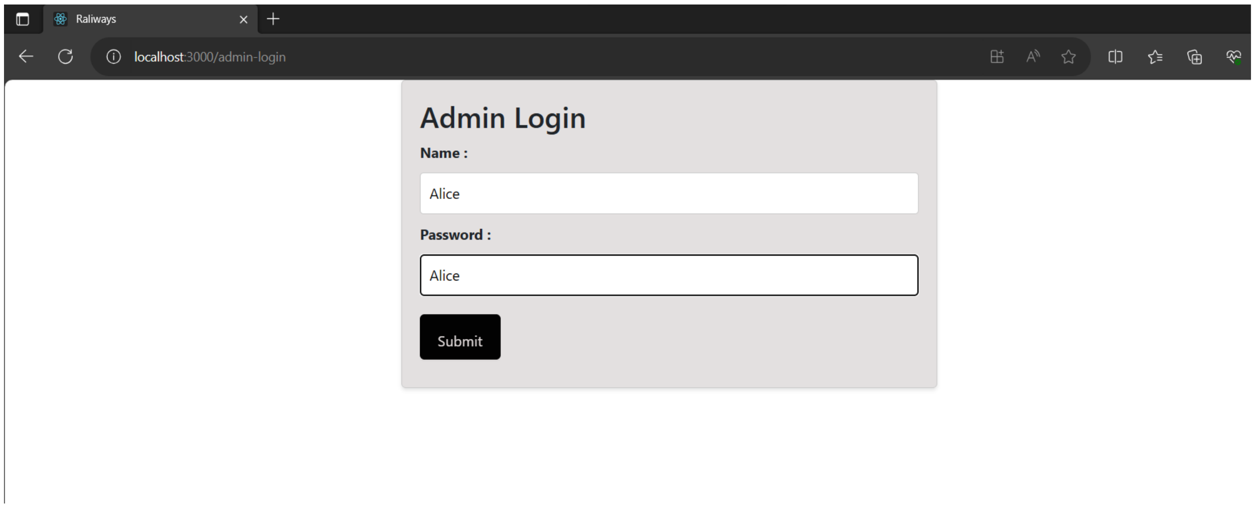


Fig:3.1

Home Page which includes the login for admin and user



Admin Login

Name : Alice

Password : Alice

Fig:3.2

Admin Login where the admin can login by using their name and password

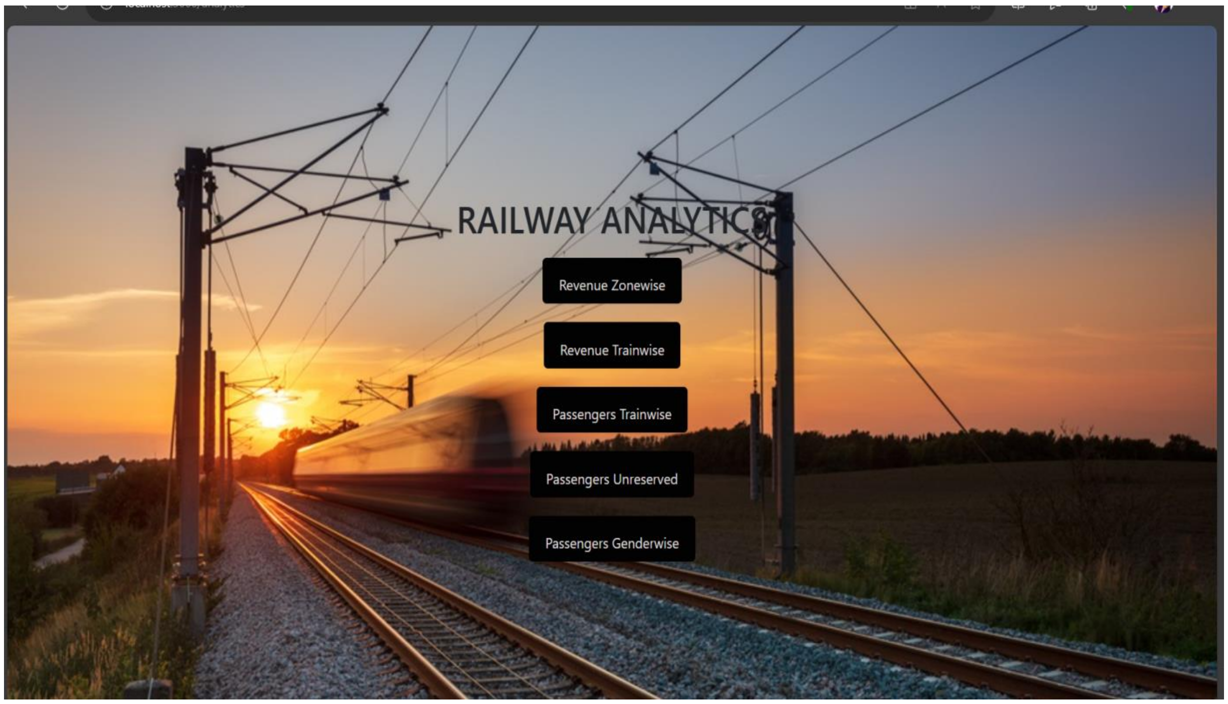


Fig:3.3
The admin login leads to a page where the admin alone can view the analytic results

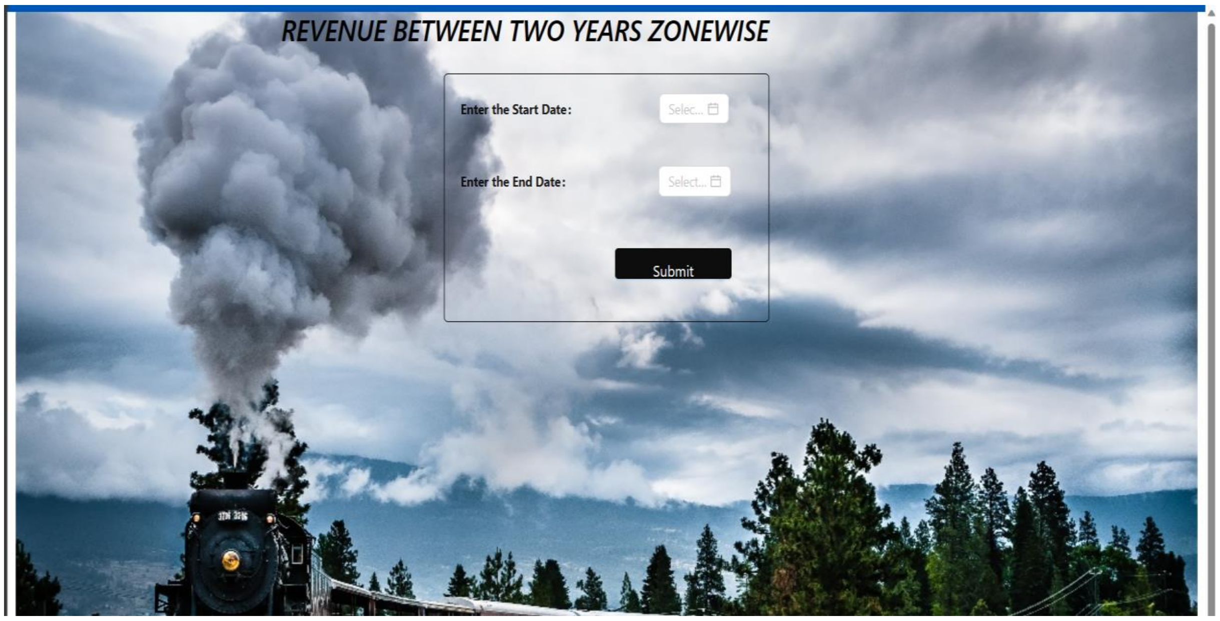
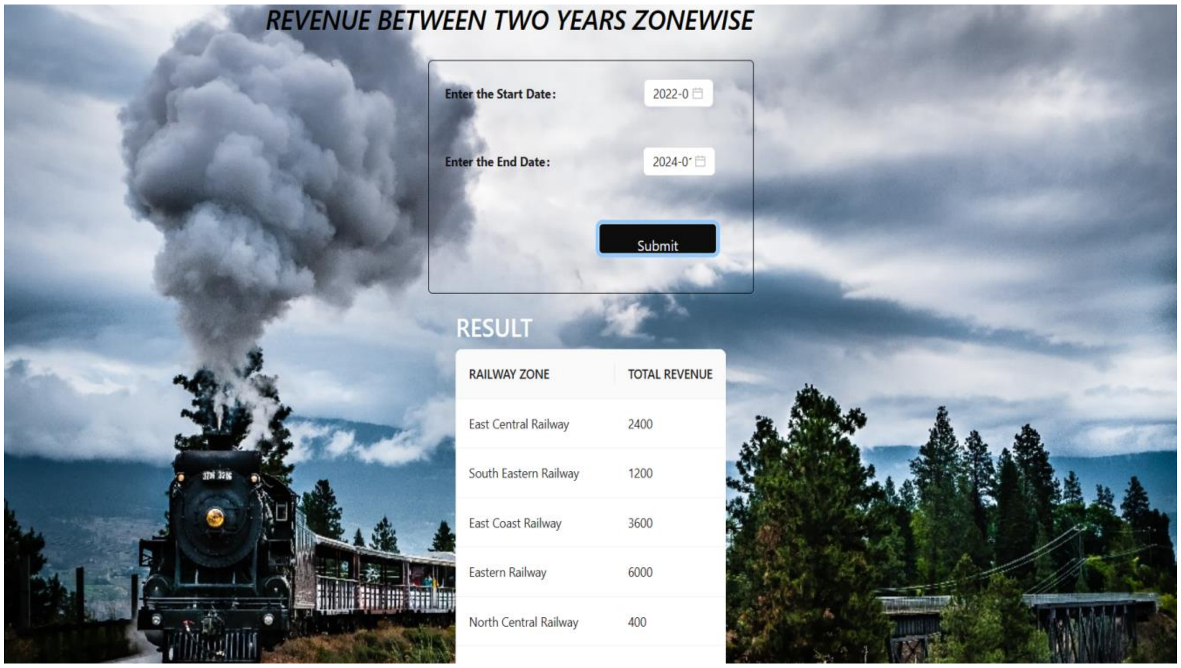


Fig:3.4
This is the query which displays the revenue between two years zonewise



RAILWAY ZONE	TOTAL REVENUE
East Central Railway	2400
South Eastern Railway	1200
East Coast Railway	3600
Eastern Railway	6000
North Central Railway	400

Fig:3.5
This figure shows the result of revenue between two years zonewise

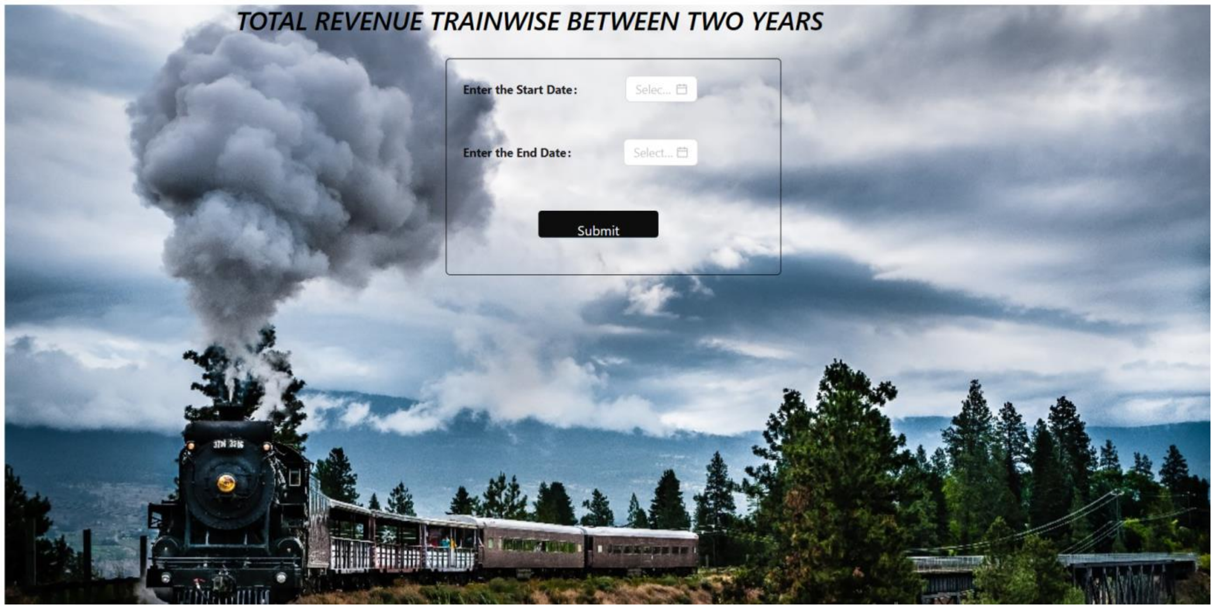
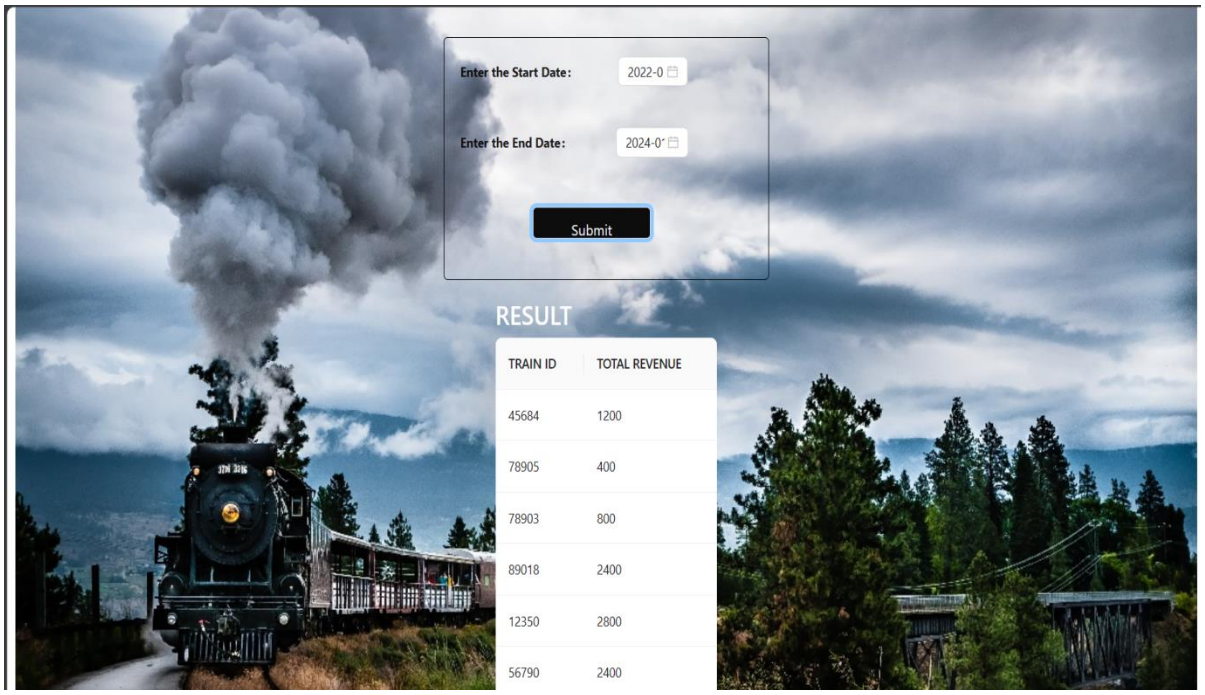


Fig:3.6
This is the query which displays the total revenue trainwise between two years



TRAIN ID	TOTAL REVENUE
45684	1200
78905	400
78903	800
89018	2400
12350	2800
56790	2400

Fig:3.7

This figure shows the result of total revenue trainwise between two years

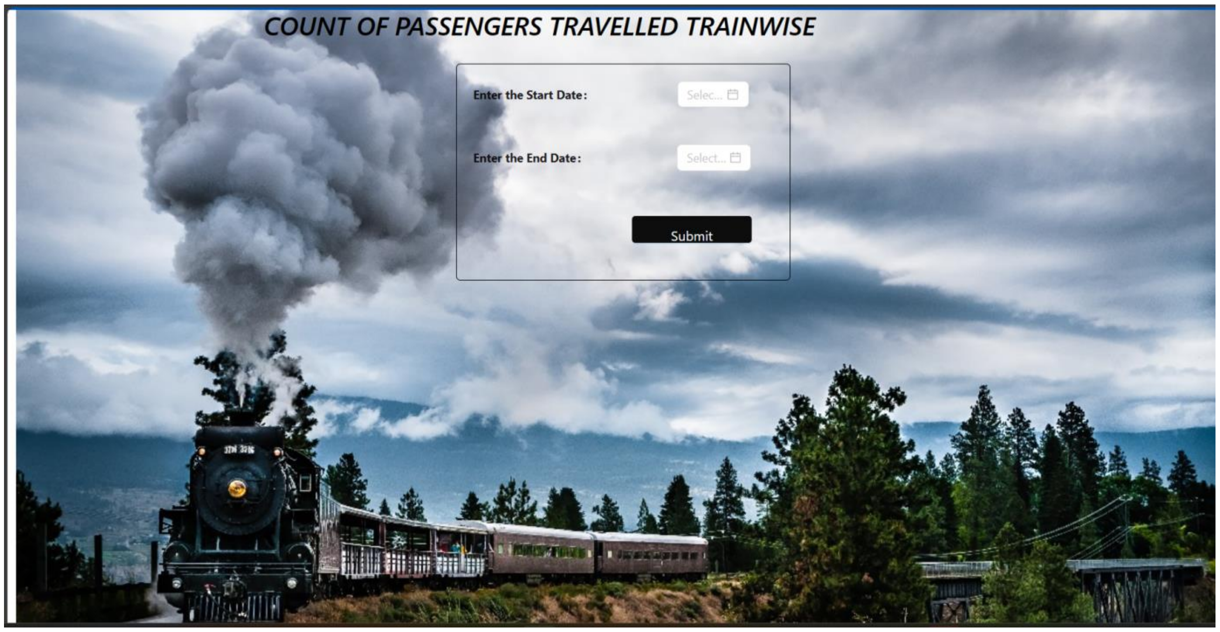
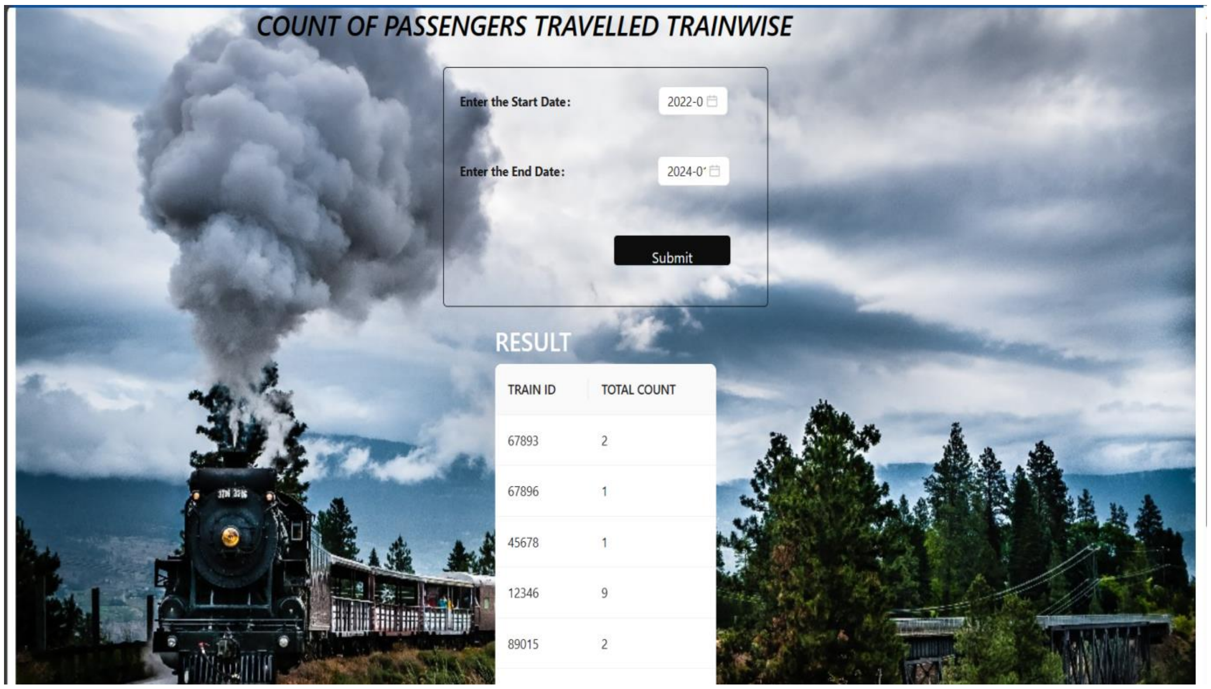


Fig:3.8

This is the query which displays the count of passengers travelled trainwise


COUNT OF PASSENGERS TRAVELLED TRAINWISE

Enter the Start Date: 2022-0

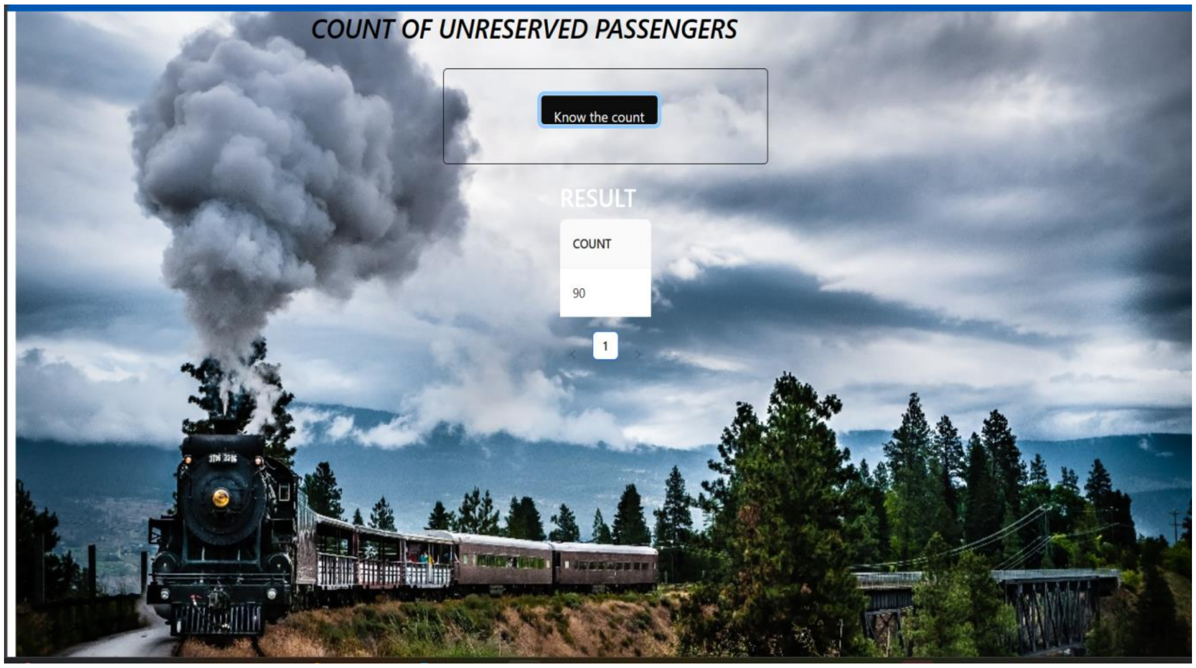
Enter the End Date: 2024-0'

RESULT

TRAIN ID	TOTAL COUNT
67893	2
67896	1
45678	1
12346	9
89015	2

Fig:3.9

This figure shows the result of count of passengers travelled trainwise


COUNT OF UNRESERVED PASSENGERS

RESULT

COUNT
90

Fig:3.10

This figure shows the result of count unreserved passengers

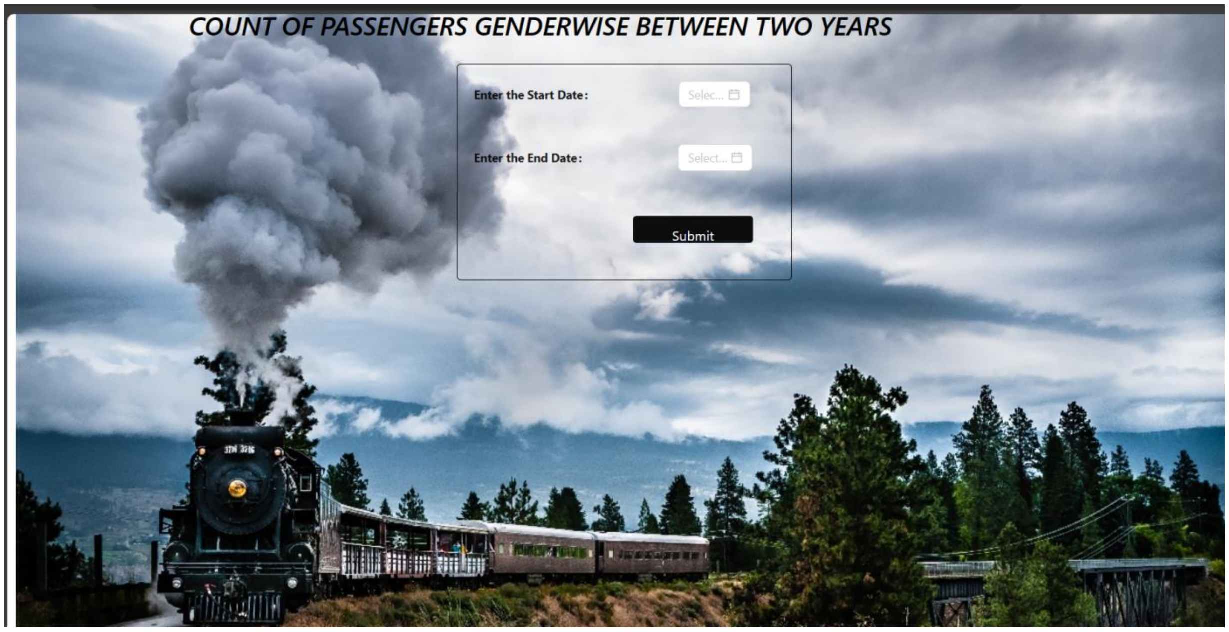
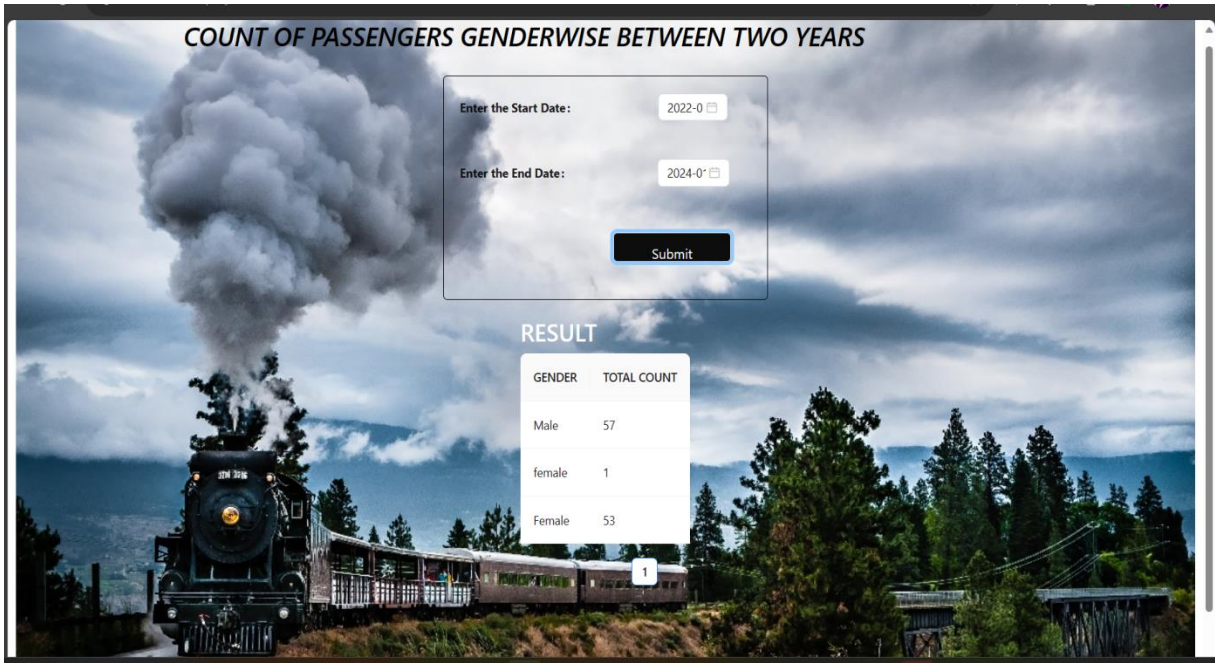


Fig:3.11

This is the query which displays the count of passengers gender –wise between two years



GENDER	TOTAL COUNT
Male	57
female	1
Female	53

Fig:3.12

This figure shows the result of count of passengers genderwise between two years

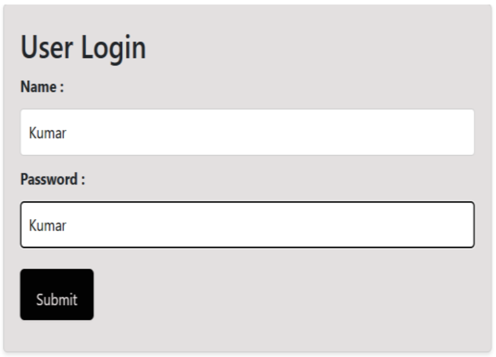


Fig:3.13
User Login where the user can login by using their name and password

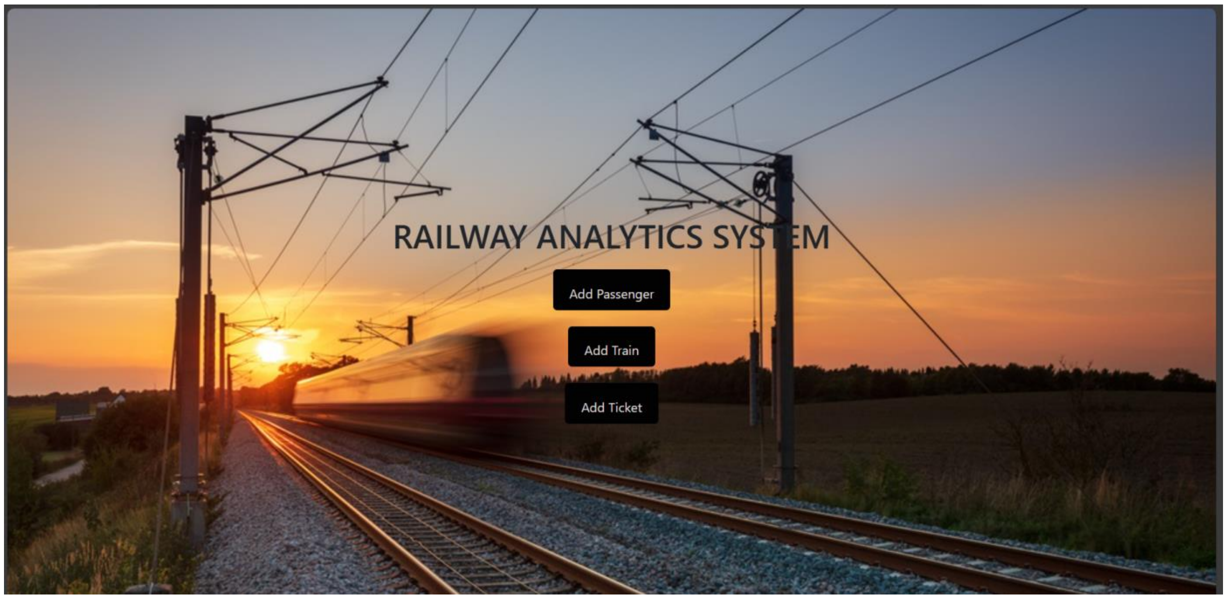
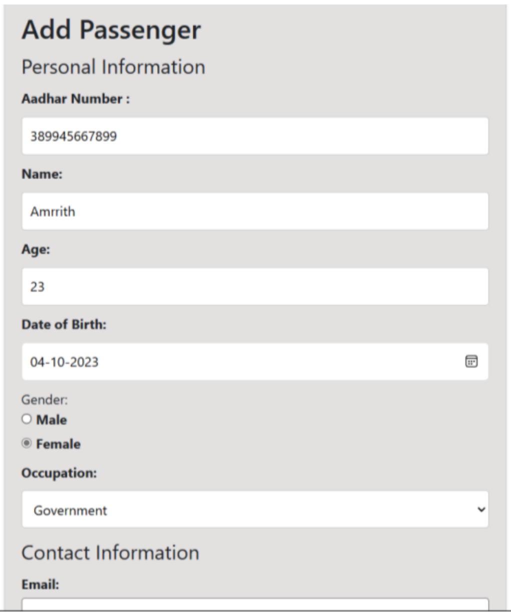


Fig:3.14
The user login leads to a page where data can be inserted in passenger, train, tickets collection



Add Passenger

Personal Information

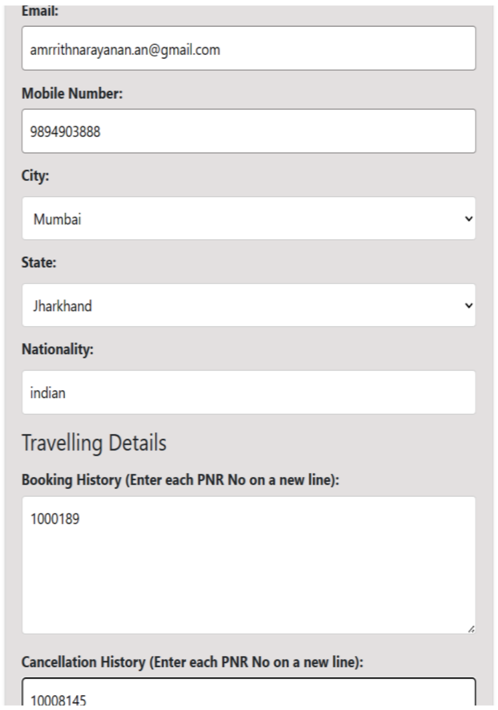
Aadhar Number :	389945667899
Name:	Amrrith
Age:	23
Date of Birth:	04-10-2023
Gender:	Male [✓ Female]
Occupation:	Government

Contact Information

Email:

Fig:3.15

The user can add passenger details which is then inserted into MongoDB



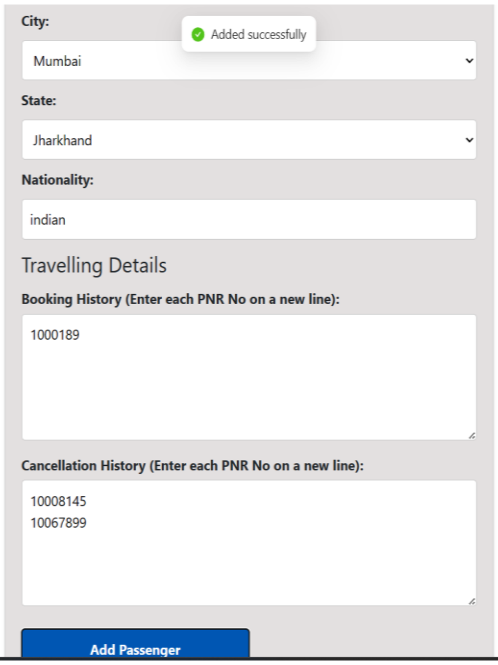
Email:	amrrithnarayanan.an@gmail.com
Mobile Number:	9894903888
City:	Mumbai
State:	Jharkhand
Nationality:	indian

Travelling Details

Booking History (Enter each PNR No on a new line):	1000189
Cancellation History (Enter each PNR No on a new line):	10008145

Fig:3.16

The passenger can add their personal travelling details



City:

Added successfully

Mumbai

State:

Jharkhand

Nationality:

indian

Travelling Details

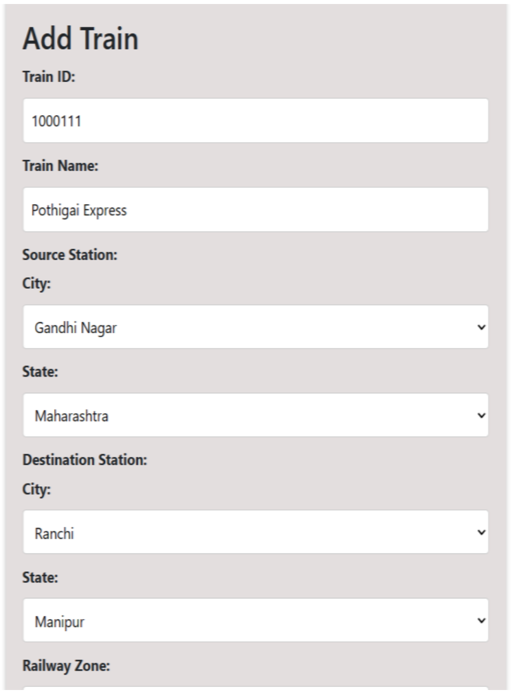
Booking History (Enter each PNR No on a new line):

1000189

Cancellation History (Enter each PNR No on a new line):

10008145
10067899

Fig:3.17
The passenger details once inserted notifies that the data is added successfully



Add Train

Train ID:

1000111

Train Name:

Pothigai Express

Source Station:

City:

Gandhi Nagar

State:

Maharashtra

Destination Station:

City:

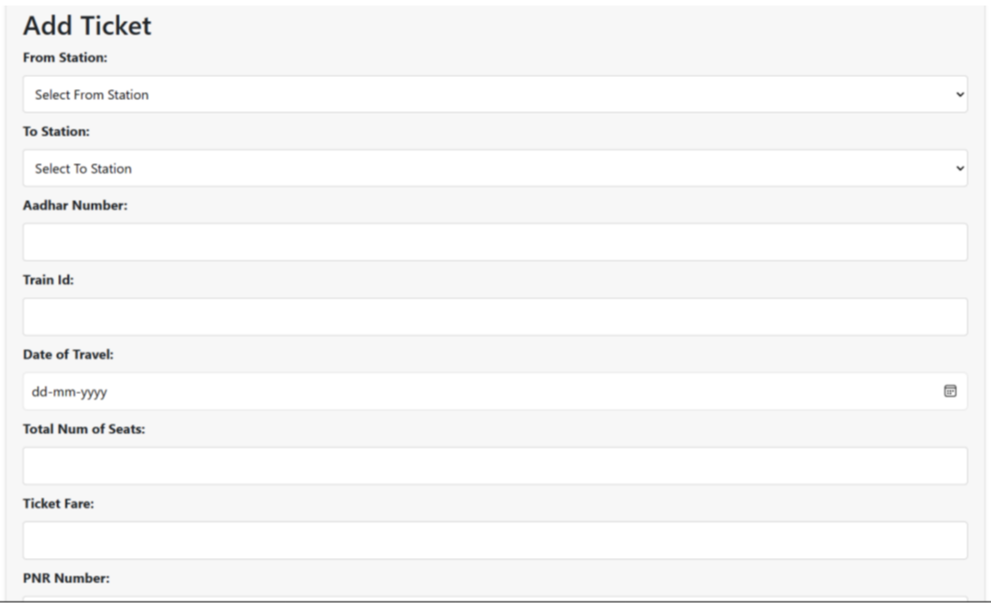
Ranchi

State:

Manipur

Railway Zone:

Fig:3.18
The user can add train details which is then inserted into MongoDB



Add Ticket

From Station:

Select From Station

To Station:

Select To Station

Aadhar Number:

Train Id:

Date of Travel:

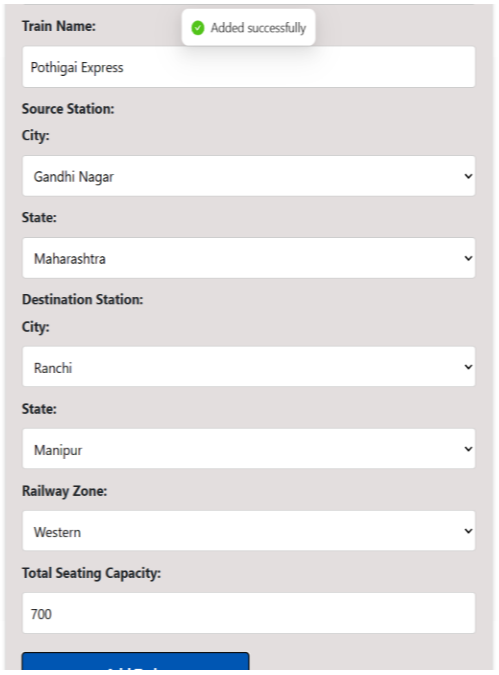
dd-mm-yyyy

Total Num of Seats:

Ticket Fare:

PNR Number:

Fig:3.19
The user can add ticket details which is then inserted into MongoDB



Added successfully

Train Name: Pothigai Express

Source Station:

City: Gandhi Nagar

State: Maharashtra

Destination Station:

City: Ranchi

State: Manipur

Railway Zone: Western

Total Seating Capacity: 700

Fig:3.20
The train details once inserted notifies that the data is added successfully

3.2 APPENDIX

//Database Connection

```
const mongoose = require("mongoose");
const colors = require("colors");
const connectDB = async () => {
  try {
    await mongoose.connect(process.env.MONGO_URL);
    console.log(`Mongodb connected ${mongoose.connection.host}`.bgGreen.white);
  } catch (error) {
    console.log(`Mongodb Server Issue ${error}`.bgRed.white);
  }
};
module.exports = connectDB;
```

//Server

```
const express = require("express");
const colors = require("colors");
const morgan = require("morgan");
const dotenv = require("dotenv");
const connectDB = require("./config/db.js");

dotenv.config();

//mongodb connection
connectDB();

//rest obejct
const app = express();

//middlewares
app.use(express.json());
app.use(morgan("dev"));
//routescd
app.use("/api/v1/user",require("./routes/userRoutes"));
app.use("/api/v1/analytics",require("./routes/query1Routes.js"));
//port
const port = process.env.PORT || 8080;
//listen port
app.listen(port, () => {
  console.log(
    `Server Running in ${process.env.NODE_MODE} Mode on port ${process.env.PORT}`
      .bgCyan.white
  );
});
```

//Schema Definitions

```
const mongoose = require("mongoose");
const ticketSchema = new mongoose.Schema({
    passengerId: {
```

```
    type: Number,
    required: [true, "passengerId is require"],
  },
  trainId: {
    type: Number,
    required: [true, "trainId is require"],
  },
  fromStation: {
    type: String,
    required: [true, "fromStation is require"],
  },
  toStation: {
    type: String,
    required: [true, "toStation is require"],
  },
  departureDate: {
    type: Date,
    required: [true, "departureDate is require"],
  },
  seatNumber: {
    type: Number,
    required: [true, "seatNumber is require"],
  },
  ticketPrice: {
    type: Number,
    required: [true, "ticketPrice is require"],
  },
  pnrNumber: {
    type: Number,
    required: [true, "pnrNumber is require"],
  },
  travelStatus: {
    type: String,
    required: [true, "travelStatus is require"],
  },
  totalAmount: {
    type: Number,
    required: [true, "totalAmount is require"],
  },
  bookedByPassengerId: {
    type: Number,
    required: [true, "bookedByPassengerId is require"],
  },


});

const ticketModel = mongoose.model("Tickets", ticketSchema);

module.exports = ticketModel;

const mongoose = require("mongoose");


const trainSchema = new mongoose.Schema({
    trainId: {
            type: Number,
            required: false,
          },
          trainName: {
            type: String,
            required: false,
          },

          srccity: {
            type: String,
            required: false,
          },
          srcstate: {
            type: String,
            required: false,
          },
          descity: {
            type: String,
            required: false,
          },
          desstate: {
            type: String,
            required: false,
          },
          railwayZone: {
            type: String,
            required: false,
          },
          totalSeatingCapacity: {
            type: Number,
            required: false,
          },
        });

const trainModel = mongoose.model("Trains", trainSchema);

module.exports = trainModel;

const mongoose = require("mongoose");

const userSchema = new mongoose.Schema({
    passengerId: {
    type: Number,
    required: [true, "passengerId is require"],
  },
  name: {
    type: String,
    required: [true, "name is require"],
  },
  age: {
    type: Number,
    required: [true, "age is require"],
  },
  dateOfBirth: {
    type: Date,
    required: [true, "dateOfBirth is require"],
  },
  gender: {
    type: String,
    required: [true, "gender is require"],
  },
  occupation: {
    type: String,
    required: [true, "occupation is require"],
  },
  email: {
    type: String,
    required: [true, "email is require"],
  },
  mobileNum: {
    type: Number,
    required: [true, "mobileNum is require"],
  },
  nationality: {
    type: String,
    required: [true, "nationality is require"],
  },
  cancellationHistory: {
    type: Array,
    required: [true, "cancellationHistory is require"],
  },
  bookingHistory: {
    type: Array,
    required: [false, "bookingHistory is require"],
  },

  addressCity: {
    type: String,
    required: [true, "addressCity is require"],
  },
  addressState: {
    type: String,
    required: [true, "addressState is require"],
  },

});

const userModel = mongoose.model("Passengers", userSchema);

module.exports = userModel;
```

//Queries

```
const express = require('express');
const router = express.Router();
const tickets1 = require('../models/ticketModels'); // Replace 'YourModel' with your
actual model
const passengers1 = require('../models/userModels');
const trains1 = require('../models/trainModels');

router.get('/calculateRevenue', async (req, res) => {
  try {
    const st=new Date(req.query.startDate);
    const ey=new Date(req.query.endDate);
    const z=req.query.zone;
    const result = await tickets1.aggregate([

        {
            $lookup: {
              from: "passengers",
              localField: "passengerId",
              foreignField: "passengerId",
              as: "passengerData"
            }
          },
          {
            $lookup: {
              from: "trains",
              localField: "trainId",
              foreignField: "trainId",
              as: "trainData"
            }
          },
          {
            $match: {
                "departureDate": {
                  $gte:st,
                  $lt: ey
                }
              }
            },
            {
              $unwind: "$passengerData"
            },
            {
              $unwind: "$trainData"
            },
            {
              $group: {
                _id: "$trainData.railwayZone",
                totalRevenue: { $sum: "$totalAmount" }
              }
            },
            {
              $project: {
                _id: 0,
                railwayZone: "$_id",
                totalRevenue: 1
              }
            }
      ]);

      console.log("Query1 API Response:", result);
          if (result.length === 0) {
              res.status(404).send({
                  message: "No data found for the specified criteria",
                  success: false
              });
          } else {
              res.status(200).json(result);
          }
      } catch (error) {
          console.error(error);
          res.status(500).json({
              message: "Error querying data",
              success: false,
          });
      }
  });
```

```
router.get('/query2', async (req, res) => {
  try {
    const st=new Date(req.query.startDate);
    const ey=new Date(req.query.endDate);
    const z=req.query.trainid;
    const result = await tickets1.aggregate([

        {
            $lookup: {
              from: "passengers",
              localField: "passengerId",
              foreignField: "passengerId",
              as: "passengerData"
            }
          },
          {
            $lookup: {
              from: "trains",
              localField: "trainId",
              foreignField: "trainId",
              as: "trainData"
            }
          },
          {
            $match: {
              "departureDate": {
                $gte:st,
                $lt: ey
              }
            }
          },
          {
            $unwind: "$passengerData"
          },
          {
            $unwind: "$trainData"
          },
          {
            $group: {
              _id: "$trainData.trainId",
              totalRevenue: { $sum: "$totalAmount" }
            }
          },
          {
              $project: {
                _id: 0,
                railwayZone: "$_id",
                totalRevenue: 1
              }
            }
        ]);

        console.log("Query2 API Response:", result);
            if (result.length === 0) {
                res.status(404).send({
                    message: "No data found for the specified criteria",
                    success: false
                });
            } else {
                res.status(200).json(result);
            }
        } catch (error) {
            console.error(error);
            res.status(500).json({
                message: "Error querying data",
                success: false,
            });
        }
    });
    router.get('/query3', async (req, res) => {
      try {

        const st=new Date(req.query.startDate);
        const ey=new Date(req.query.endDate);
        const result = await tickets1.aggregate([

            {
                $lookup: {
                  from: "passengers",
                  localField: "passengerId",
                  foreignField: "passengerId",
                  as: "passengerData"
                }
              },
              {
                $lookup: {
                  from: "trains",
                  localField: "trainId",
                  foreignField: "trainId",
                  as: "trainData"
                }
              },
              {
                $match: {
                  "departureDate": {
                    $gte:st,
                    $lt: ey
                  }
                }
              },
              {
                $unwind: "$passengerData"
              },
              {
                $unwind: "$trainData"
              },
              {
                $group: {
                  _id: "$trainData.trainId",
                  totalcount: { $sum: 1 }
                }
              },
              {
                $project: {
                  _id: 0,
                  railwayZone: "$_id",
                  totalcount: 1
                }
              }
        ]);

        console.log("Query3 API Response:", result);
            if (result.length === 0) {
                res.status(404).send({
                    message: "No data found for the specified criteria",
                    success: false
                });
            } else {
                res.status(200).json(result);
            }
        } catch (error) {
            console.error(error);
```

```
        res.status(500).json({
            message: "Error querying data",
            success: false,
        });
    }
});
router.get('/query4', async (req, res) => {
  try {

    const st=new Date(req.query.startDate);
    const ey=new Date(req.query.endDate);
    const z=req.query.trainid;
    const result = await tickets1.aggregate([

        {
            $lookup: {
              from: "passengers",
              localField: "passengerId",
              foreignField: "passengerId",
              as: "passengerData"
            }
          },
          {
            $lookup: {
              from: "trains",
              localField: "trainId",
              foreignField: "trainId",
              as: "trainData"
            }
          },
          {
            $match: {
              "departureDate": {
                $gte:st,
                $lt: ey
              }
            }
          },
          {
            $unwind: "$passengerData"
          },
          {
            $unwind: "$trainData"
          },
          {
            $group: {
              _id: "$passengerData.gender",
              totalcount: { $sum: 1 }
            }
          },
          {
            $project: {
              _id: 0,
              gender: "$_id",
              totalcount: 1
            }
          }
    ]);

    console.log("Query4 API Response:", result);
        if (result.length === 0) {
            res.status(404).send({
                message: "No data found for the specified criteria",
                success: false
            });
        } else {
            res.status(200).json(result);
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({
            message: "Error querying data",
            success: false,
        });
    }
});
router.get('/query5', async (req, res) => {
  try {

    const st=new Date(req.query.startDate);
    const ey=new Date(req.query.endDate);

    const result = await passengers1.aggregate([

      {
        $match: {

          "bookingHistory": { $size: 1}
        }
      },
      {
        $group: {
          _id: null,
          passengerCount: { $sum: 1 }
        }
      },
      {
        $project: {
          _id: 0,
          passengerCount: 1
        }
      }
]);
    console.log("Query5 API Response:", result);
        if (result.length === 0) {
            res.status(404).send({
                message: "No data found for the specified criteria",
                success: false
            });
        } else {
            res.status(200).json(result);
        }
    } catch (error) {
        console.error(error);
        res.status(500).json({
            message: "Error querying data",
            success: false,
        });
    }
});

module.exports = router;
```

CHAPTER 4
CONCLUSION

4.1 CONCLUSION

In conclusion, the Railway Analytics System, leveraging the strengths of MongoDB and React.js, represents a transformative solution for the railway industry. By efficiently managing and analyzing complex railway data, this system not only streamlines operations but also enhances the overall passenger experience and safety. MongoDB's NoSQL database provides the flexibility and scalability required to handle the dynamic nature of railway information, while React.js delivers a responsive and user-friendly interface. Together, they empower railway professionals to make data-driven decisions, optimize resources, and drive operational excellence. This project sets the stage for a more efficient and modern railway management system, benefiting both the industry and its passengers by embracing the power of data analytics and user-centric design.

REFERENCES

- https://stackoverflow.com/
- https://www.youtube.com/watch?v=7CqJlxBYj-M
- https://www.mongodb.com/languages/mern-stack-tutorial
- https://www.geeksforgeeks.org/how-to-connect-mongodb-with-reactjs/