CSE471: System Analysis and Design

Project Report

Project Title: MetroHub—Smart Metro Rail Schedule & Ticketing System

Group No: 05, CSE471 Lab Section: 04, Summer 2025	
ID	**Name**
21301418	Md. Ratul Mushfique
21301411	Sadman Safiur Rahman
21301750	Kazi Amzad Abid

Submission Date: 08/09/2025

Table of Contents

System Request

Business need: The MetroHub Application helps users easily find metro schedules, book tickets, and manage their travel by providing tools for route finding, real-time updates, and payments in one simple platform.

Business requirements: Using the MetroHub Application, users will be able to better plan their metro trips and handle ticketing all in one place. The specific functionality that the system should have includes the following:

● Develop a secure and easy-to-use platform for user accounts, profiles, and metro pass management.

● Enable searching for schedules, viewing real-time updates, and finding routes with maps and multi-language support.

● Provide ticket booking with payments, notifications, and admin tools to manage schedules and user roles.

Business value: We expect that the MetroHub Application will improve daily commuting for users by making metro travel faster and more convenient through easy scheduling, booking, and updates. The app's simple design and helpful features will attract people who use public transport often, helping them save time and avoid hassles. Over time, we anticipate more users will rely on it for reliable metro info and seamless ticketing, leading to better user satisfaction and possible growth as new features are added.

Special issues or constraints:

● The development team sees this as a key tool for making metro travel smarter and more accessible in busy cities. Keeping it user-friendly is important to draw in more riders.

●Many users complain about confusing schedules or long lines for tickets, so the app must be quick and reliable to meet their needs. Not fixing these could mean users stick with old ways and ignore the app.

2. Functional Requirements

Module 1: User Account & Profile Management

1. Users can register and log in using email/mobile phone and password.
2. Users can update their profile including name, phone number, emails, passwords and preferred routes.
3. Admins can manage user roles and permissions (e.g., regular user, rapid pass user, remove user etc).
4. Users can set themselves as rapid pass users and can also deposit money in the pass.

Module 2: Metro Schedule, Route Finder and User Accessibility

1. Users can search for metro schedules by metro station, and time, and view real-time train updates done by Admin.
2. Google Maps API integration helps users visualize nearby metro stations and walking directions.
3. Admins can create, update, or delete metro train schedules, station lists and station info.
4. Users can view various options, sections and ticket fares in multiple languages (e.g., English and Bangla). Users can switch the app language with a toggle.

Module 3: Ticket Booking & Notifications

1. Users can book metro tickets with online payment methods (SSLCOMMERZ gateway or rapid pass option), download QR code for the ticket, scan QR code (simulation of entering and exiting metro station) and view booking history from their dashboard. Admins can also view tickets that are booked.
2. For rapid pass users, ticket costs are automatically deducted from their balance upon booking.
3. Users receive gmail confirmation for booked rides using Nodemail
4. Users receive fines when they are in station after a certain period of time, added to the profile. If the QR code scanned (indicating exit from station) 2nd time before that time limit then journey has ended, no fine. Admin can view all the users with fines.

3. Technology (Framework, Languages)

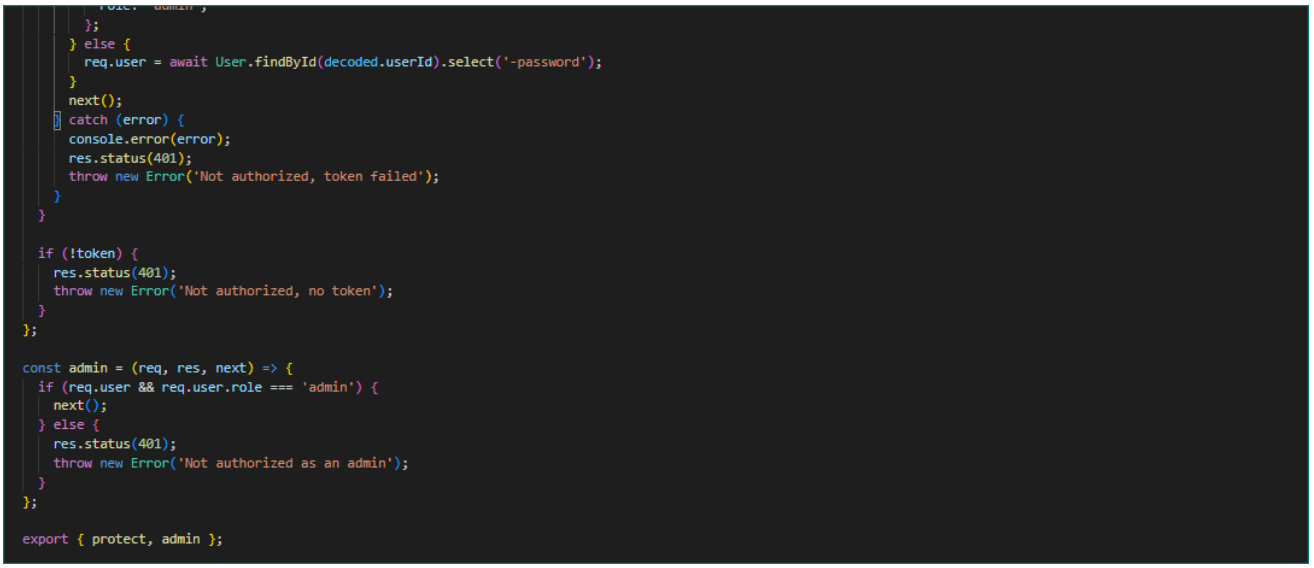
- Language: JavaScript
- Frontend Framework: React.js
- Backend Framework: Express.js with Node.js
- Styling: TailwindCSS

- Database: MongoDB
- ODM: Mongoose
- Deployment: Vercel (frontend) + Render (API backend)

4. Backend Development

1) Description of the Authentication Middleware

```
Backend > middleware > authMiddleware.js > protect
import jwt from 'jsonwebtoken';
import { User } from '../models/userModel.js';

const protect = async (req, res, next) => {
  let token;

  if (req.headers.authorization && req.headers.authorization.startsWith('Bearer')) {
    try {
      token = req.headers.authorization.split(' ')[1];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);

      if (decoded.userId === 'hardcoded_admin_user_id') {
        req.user = {
          _id: 'hardcoded_admin_user_id',
          name: 'Admin User',
          email: 'admin1@gmail.com',
          role: 'admin',
        };
      } else {
        req.user = await User.findById(decoded.userId).select('-password');
      }
      next();
    } catch (error) {
      console.error(error);
      res.status(401);
      throw new Error('Not authorized, token failed');
    }
  } else if (req.headers['x-auth-token']) {
    try {
      token = req.headers['x-auth-token'];
      const decoded = jwt.verify(token, process.env.JWT_SECRET);

      if (decoded.userId === 'hardcoded_admin_user_id') {
        req.user = {
          _id: 'hardcoded_admin_user_id',
          name: 'Admin User',
          email: 'admin1@gmail.com',
          role: 'admin',
        };
```



```
        };
      } else {
        req.user = await User.findById(decoded.userId).select('-password');
      }
      next();
    } catch (error) {
      console.error(error);
      res.status(401);
      throw new Error('Not authorized, token failed');
    }
  }

  if (!token) {
    res.status(401);
    throw new Error('Not authorized, no token');
  }
};

const admin = (req, res, next) => {
  if (req.user && req.user.role === 'admin') {
    next();
  } else {
    res.status(401);
    throw new Error('Not authorized as an admin');
  }
};

export { protect, admin };
```

Basic Token Verification Middleware: This snippet checks for the presence of an authorization header with a Bearer token, verifies it using JWT (JSON Web Token), and retrieves the user from a database. If the token is invalid or the user is not found, it

throws an unauthorized error.

Admin Role Check Middleware: Extends the basic verification by ensuring the authenticated user has an "admin" role. If the role doesn't match, it throws an unauthorized error.

Alternative Token Source Middleware: Checks for a custom "x-auth-token" header as an alternative to the Bearer token, verifying it with JWT and handling the admin user case similarly.

Hardcoded Admin User Middleware: Includes a fallback where a hardcoded admin user is returned if the decoded token matches a specific admin user ID, useful for testing or default admin access.

Error Handling Middleware: Incorporates error catching to handle cases where token verification or database queries fail, returning a 401 status with an appropriate error message.

2) Description of the AdminController.js

```
const updateUserRole = async (req, res) => {
  const user = await User.findById(req.params.id);

  if (user) {
    const oldRole = user.role;
    user.role = req.body.role || user.role;

    if (oldRole === 'rapidPassUser' && user.role !== 'rapidPassUser' && user.rapidPassId) {
      await RapidPass.findOneAndUpdate({ rapidPassId: user.rapidPassId }, { user: 'Not used in MetroHub' });
      user.rapidPassId = null;
    }

    const updatedUser = await user.save();
    res.json({
      _id: updatedUser._id,
      name: updatedUser.name,
      email: updatedUser.email,
      role: updatedUser.role,
    });
  } else {
    res.status(404);
    throw new Error('User not found');
  }
};

// @desc    Get recent users
// @route   GET /api/admin/recent-users
// @access  Private/Admin
const getRecentUsers = async (req, res) => {
  const users = await User.find({ email: { $ne: 'admin1@gmail.com' } })
    .sort({ createdAt: -1 })
    .limit(5)
    .select('-password -plainTextPassword'); // Exclude sensitive fields
  res.json(users);
};
```

```
// @access  Private/Admin
const getAllTickets = async (req, res) => {
  try {
    const tickets = await Ticket.find({}).populate('user', 'name email').populate('schedule');
    res.json(tickets);
  } catch (error) {
    console.error('Error fetching all tickets:', error);
    res.status(500).json({ message: 'Server Error' });
  }
};

export { getUsers, deleteUser, updateUserRole, getRecentUsers, getAllTickets, getActiveTrainsCount, getMetroStationsCount, getActiveFinesCount, getTotalOutstandingFines, getFinesPaidThisMonthCount


const getActiveTrainsCount = async (req, res) => {
  try {
    const distinctTrains = await MetroSchedule.distinct('trainName');
    res.json({ count: distinctTrains.length });
  } catch (error) {
    console.error('Error fetching active trains count:', error);
    res.status(500).json({ message: 'Server Error' });
  }
};

// @desc    Get metro stations count
// @route   GET /api/admin/statistics/metro-stations
// @access  Private/Admin
const getMetroStationsCount = async (req, res) => {
  try {
    const count = await MetroStation.countDocuments();
    res.json({ count });
  } catch (error) {
    console.error('Error fetching metro stations count:', error);
    res.status(500).json({ message: 'Server Error' });
  }
};
```

```
// @desc    Get active fines count
// @route   GET /api/admin/statistics/active-fines
// @access  Private/Admin
const getActiveFinesCount = async (req, res) => {
  try {
    const count = await User.countDocuments({ fine: { $gt: 0 } });
    res.json({ count });
  } catch (error) {
    console.error('Error fetching active fines count:', error);
    res.status(500).json({ message: 'Server Error' });
  }
};

// @desc    Get total outstanding fines
// @route   GET /api/admin/statistics/total-outstanding-fines
// @access  Private/Admin
const getTotalOutstandingFines = async (req, res) => {
  try {
    const result = await User.aggregate([
      {
        $group: {
          _id: null,
          totalFines: { $sum: '$fine' },
        },
      },
    ]);
    res.json({ total: result[0]?.totalFines || 0 });
  } catch (error) {
    console.error('Error fetching total outstanding fines:', error);
    res.status(500).json({ message: 'Server Error' });
  }
};
```

```
// @desc    Get recent fines
// @route   GET /api/admin/recent-fines
// @access  Private/Admin
const getRecentFines = async (req, res) => {
  try {
    const finedUsers = await User.find({ fine: { $gt: 0 } })
      .sort({ updatedAt: -1 })
      .limit(10)
      .select('name email fine updatedAt'); // Select relevant fields including updatedAt
    res.json(finedUsers);
  } catch (error) {
    console.error('Error fetching recent fines:', error);
    res.status(500).json({ message: 'Server Error' });
  }
};
```

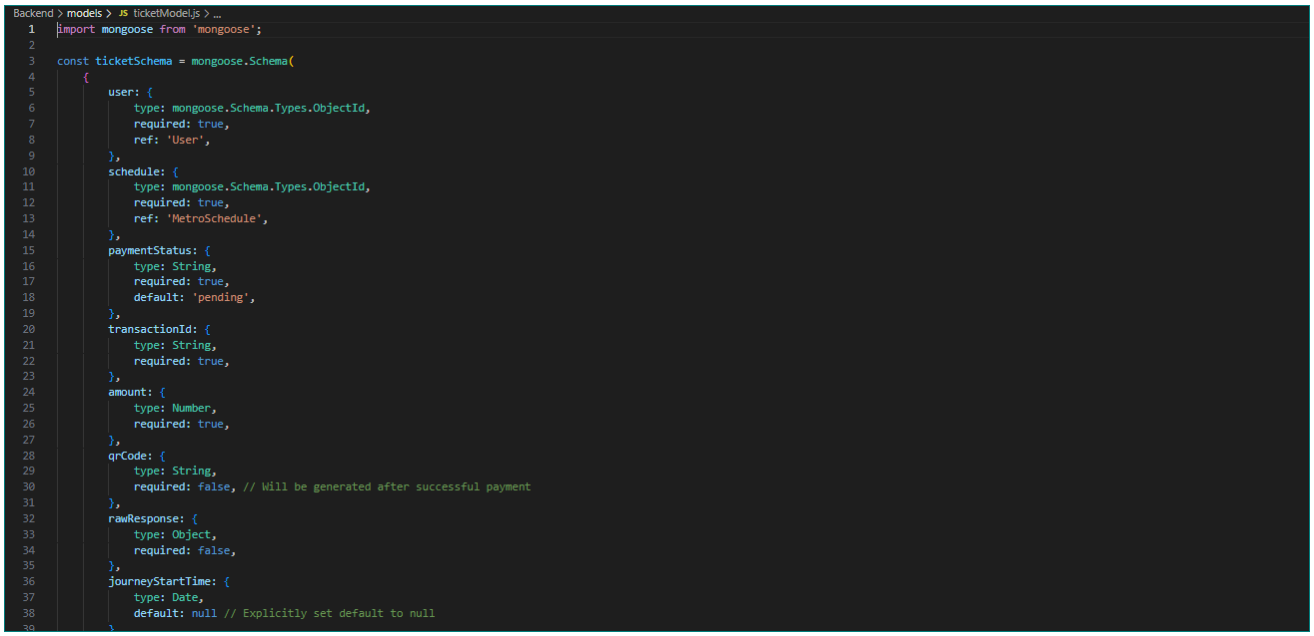
1. **Get Recent Fines**: This endpoint retrieves the most recent fines for users, limited to 10 records, sorted by the updatedAt field, and includes specific user fields like name, email, and fine details.
2. **Update User Data**: Allows updating a user's details, including role and password, with validation for rapidPass users and error handling for non-existent users.
3. **Delete User**: Provides functionality to delete a user by ID, with special handling for rapidPass users and appropriate success/error responses.
4. **Get All Tickets**: Fetches all tickets with populated user email and schedule details, handling errors with a server error response.
5. **Get Active Trains Count**: Retrieves the count of distinct train names from the MetroSchedule collection, serving as a statistic for administrative use.

3) Description of Ticket Model

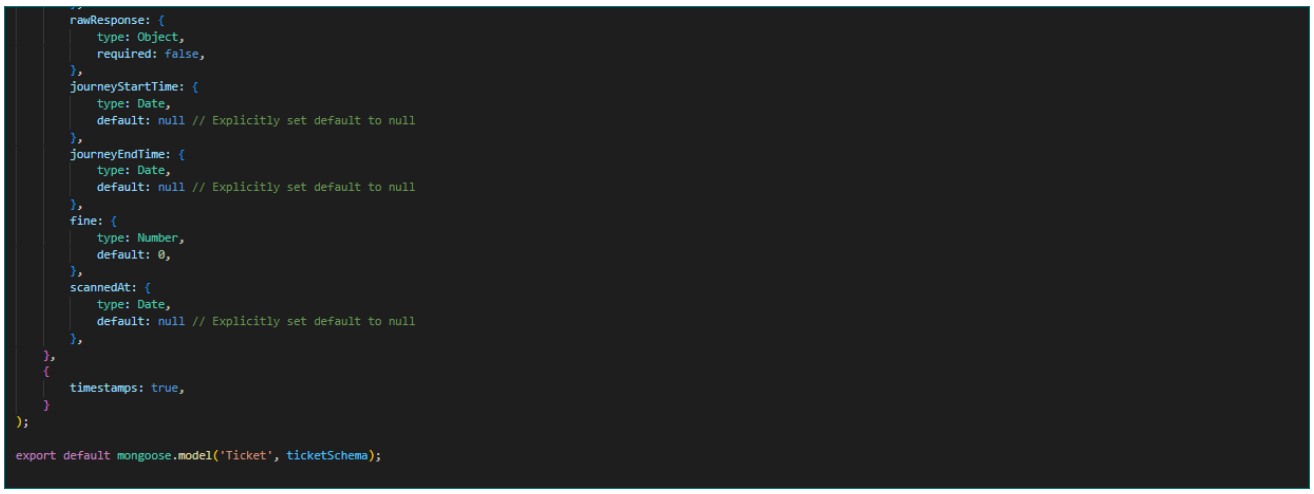
This schema represents a ticket entity with fields such as:

- user: A reference to a user object (required), linking to a User model.
- schedule: A reference to a schedule object (required), linking to a MetroSchedule model.
- paymentStatus: A string field with a default value of "pending" to track payment status.
- transactionId: A required string field for transaction identification.

- amount: A required number field for the ticket amount.
- qrCode: A string field (optional) that will be generated after a successful payment.
- rawResponse: An optional object field to store raw response data.
- journeyStartTime and journeyEndTime: Date fields with explicit null defaults for flexibility.
- fine: A number field with a default value of 0.
- scannedAt: A date field with an explicit null default.
- timestamps: Enabled to automatically manage createdAt and updatedAt fields.



```
Backend > models > JS ticketModel.js > ...
import mongoose from 'mongoose';

const ticketSchema = mongoose.Schema(
    {
        user: {
            type: mongoose.Schema.Types.ObjectId,
            required: true,
            ref: 'User',
        },
        schedule: {
            type: mongoose.Schema.Types.ObjectId,
            required: true,
            ref: 'MetroSchedule',
        },
        paymentStatus: {
            type: String,
            required: true,
            default: 'pending',
        },
        transactionId: {
            type: String,
            required: true,
        },
        amount: {
            type: Number,
            required: true,
        },
        qrCode: {
            type: String,
            required: false, // Will be generated after successful payment
        },
        rawResponse: {
            type: Object,
            required: false,
        },
        journeyStartTime: {
            type: Date,
            default: null // Explicitly set default to null
```



```
        rawResponse: {
            type: Object,
            required: false,
        },
        journeyStartTime: {
            type: Date,
            default: null // Explicitly set default to null
        },
        journeyEndTime: {
            type: Date,
            default: null // Explicitly set default to null
        },
        fine: {
            type: Number,
            default: 0,
        },
        scannedAt: {
            type: Date,
            default: null // Explicitly set default to null
        },
    },
    {
        timestamps: true,
    }
);

export default mongoose.model('Ticket', ticketSchema);
```

4) Description of PaymentRoute.js

This router defines the following payment-related endpoints:

- **POST /init**: Initiates a payment process, protected by authentication middleware.
- **ALL /success**: Handles successful payment callbacks from SSLCOMMERZ.
- **ALL /fail**: Handles failed payment callbacks from SSLCOMMERZ.
- **ALL /cancel**: Handles canceled payment callbacks from SSLCOMMERZ.
- **POST /ipn**: Processes Instant Payment Notifications from SSLCOMMERZ.
- **POST /pay-from-balance**: Allows payment using a balance, protected by authentication

middleware.

```
Backend > routes > JS paymentRoutes.js > ...
1  import express from 'express';
2  import { initiatePayment, sslcommerzSuccess, sslcommerzFail, sslcommerzCancel, sslcommerzIPN, payFromBalance } from '../controllers/paymentController.js';
3  import { protect } from '../middleware/authMiddleware.js';
4
5  const router = express.Router();
6
7  router.post('/init', protect, initiatePayment);
8  router.all('/success', sslcommerzSuccess);
9  router.all('/fail', sslcommerzFail);
10 router.all('/cancel', sslcommerzCancel);
11 router.post('/ipn', sslcommerzIPN); // For IPN (Instant Payment Notification)
12 router.post('/pay-from-balance', protect, payFromBalance);
13
14 export default router;
15
```

5) Description of server.js

This server configuration:

- Initializes an Express app with support for various HTTP methods and URL-encoded bodies.
- Mounts multiple route handlers for user, admin, station, schedule, payment, and QR-related endpoints.
- Configures a MongoDB connection using an environment variable (MONGODB_URL), with error handling and startup logging.
- Sets up CORS with a custom origin validation function to allow specific origins (e.g., localhost and Vercel preview URLs) and logs connection status.
- Includes environment variable checks for SSL store credentials and port configuration.

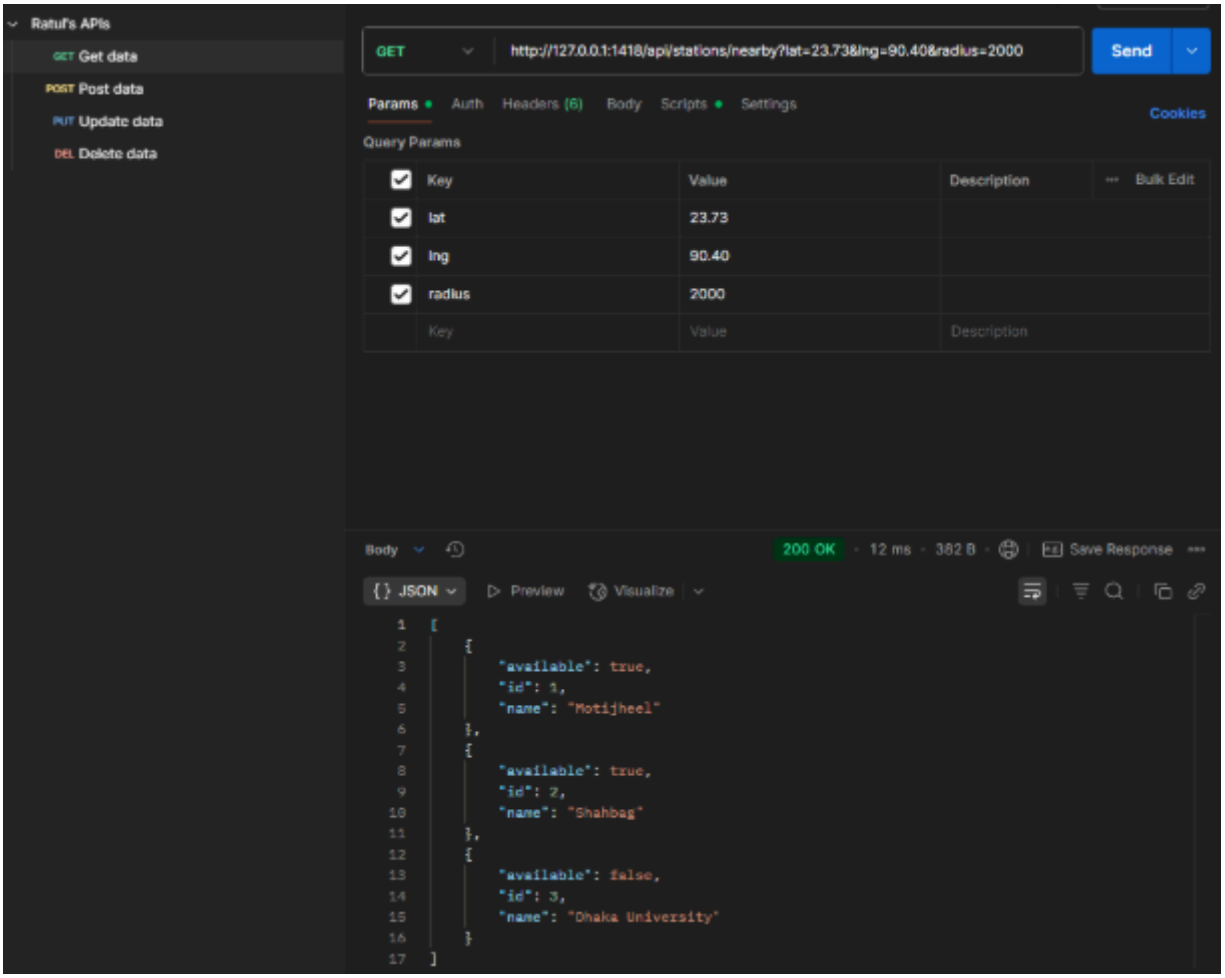
```
Backend > JS server.js > ...
1  import express from "express";
2  import cors from "cors";
3  import dotenv from 'dotenv';
4  import mongoose from "mongoose";
5  import authRoutes from './routes/authRoutes.js';
6  import userRoutes from './routes/userRoutes.js';
7  import adminRoutes from './routes/adminRoutes.js';
8  import stationRoutes from './routes/stationRoutes.js';
9  import scheduleRoutes from './routes/scheduleRoutes.js';
10 import paymentRoutes from './routes/paymentRoutes.js';
11 import qrRoutes from './routes/qrRoutes.cjs';
12
13 dotenv.config();
14
15 // quick debug (boolean only)
16 console.log('SSL_STORE_ID set?', !!process.env.SSL_STORE_ID);
17 console.log('SSL_STORE_PASSWORD set?', !!process.env.SSL_STORE_PASSWORD);
18 console.log('SSL_MODE:', process.env.SSL_MODE);
19
20 const app = express();
21
22 // controlled CORS and PORT fallback
23 const allowedOrigins = (process.env.ALLOWED_ORIGINS || 'http://localhost:5173,http://localhost:3000,https://metro-hub.vercel.app,https://metrohub-fivz.onrender.com')
24   .split(',')
25   .map(s => s.trim().replace(/\/$/, ''));
26
27 const vercelPreviewRegex = /https:\/\/.*-ratul-mushfiques-projects\.vercel\.app$/;
28
29 app.use(cors({
30   origin: (origin, callback) => {
31     if (!origin || origin === 'null') {
32       return callback(null, true);
33     }
34     const normalizedOrigin = origin.replace(/\/$/, '');
35     if (allowedOrigins.includes(normalizedOrigin) || vercelPreviewRegex.test(normalizedOrigin)) {
36       return callback(null, true);
37     }
38     return callback(new Error('CORS origin not allowed'));
39   },
```

```
Backend > JS server.js > ...
  },
  credentials: true,
  methods: 'GET,HEAD,PUT,PATCH,POST,DELETE'
}));
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

app.use('/api/auth', authRoutes);
app.use('/api/user', userRoutes);
app.use('/api/admin', adminRoutes);
app.use('/api/stations', stationRoutes);
app.use('/api/schedules', scheduleRoutes);
app.use('/api/payment', paymentRoutes);
app.use('/api/qr', qrRoutes);

// numeric PORT and startup logging
const PORT = Number(process.env.PORT) || 5000;
const MONGODB_URL = process.env.MONGODB_URL;
if (!MONGODB_URL) {
  console.error('FATAL: MONGODB_URL is not set');
  process.exit(1);
}

mongoose
  .connect(MONGODB_URL)
  .then(() => {
    console.log('App connected to database');
    app.listen(PORT, () => {
      console.log(`App is listening on port: ${PORT}`);
    });
  })
  .catch((error) => {
    console.error('DB connection error:', error);
    process.exit(1);
  });
```

POSTMAN Snippets

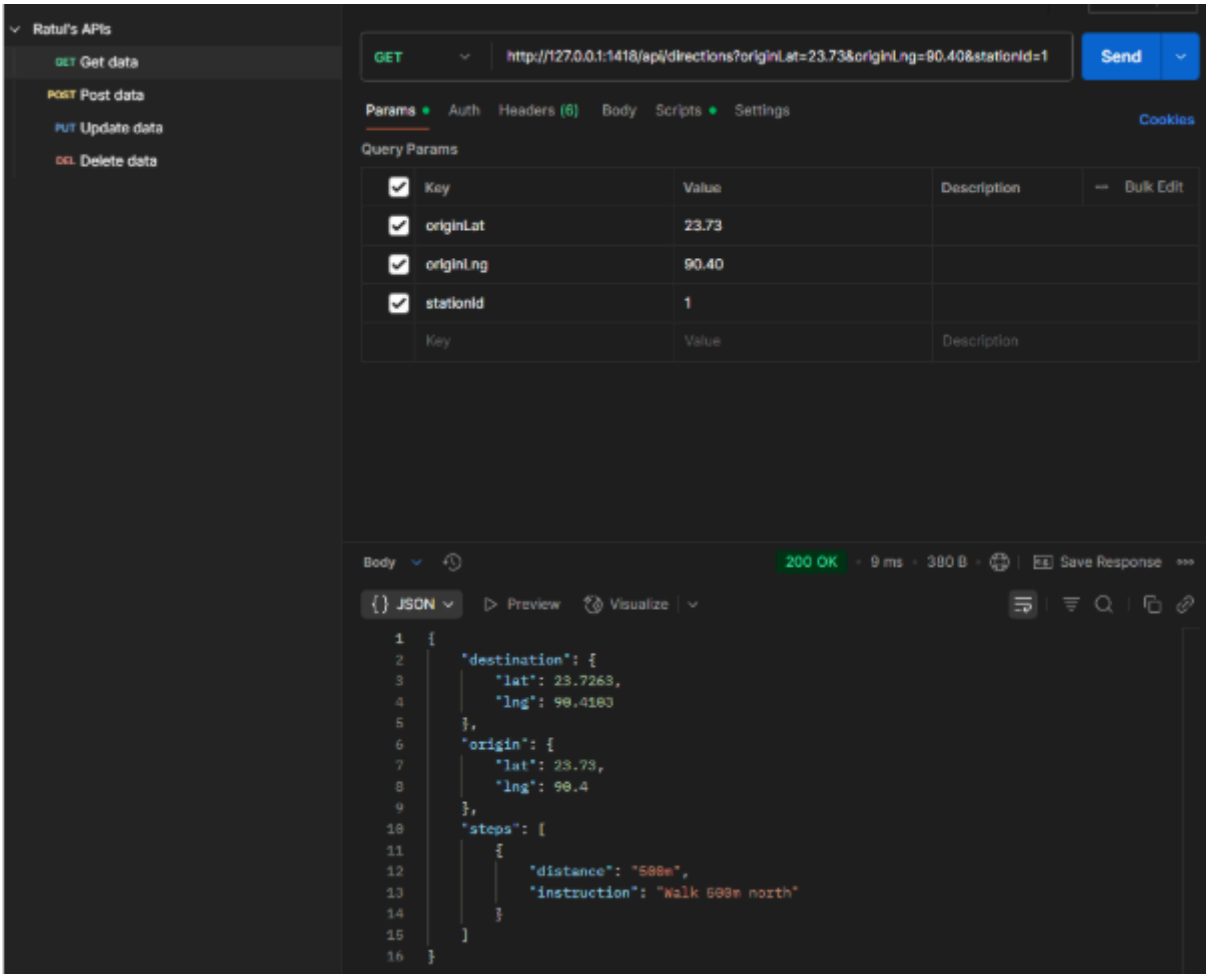
Get Nearby Metro stations:

```python
def calculate_distance(lat1, lon1, lat2, lon2):
    coords_1 = (lat1, lon1)
    coords_2 = (lat2, lon2)
    return geodesic(coords_1, coords_2).meters


@app.route('/api/stations/nearby', methods=['GET'])
def get_nearby_stations():
    lat = request.args.get('lat', type=float)
    lng = request.args.get('lng', type=float)
    radius = request.args.get('radius', default=5000, type=int)

    if lat is None or lng is None:
        return jsonify({"message": "lat and lng are required"}), 400

    stations = Station.query.all()
    nearby_stations = []
    for station in stations:
        distance = calculate_distance(lat, lng, station.latitude, station.longitude)
        if distance <= radius:
            nearby_stations.append({
                "id": station.id,
                "name": station.name,
                "available": station.available
            })
    return jsonify(nearby_stations)
```



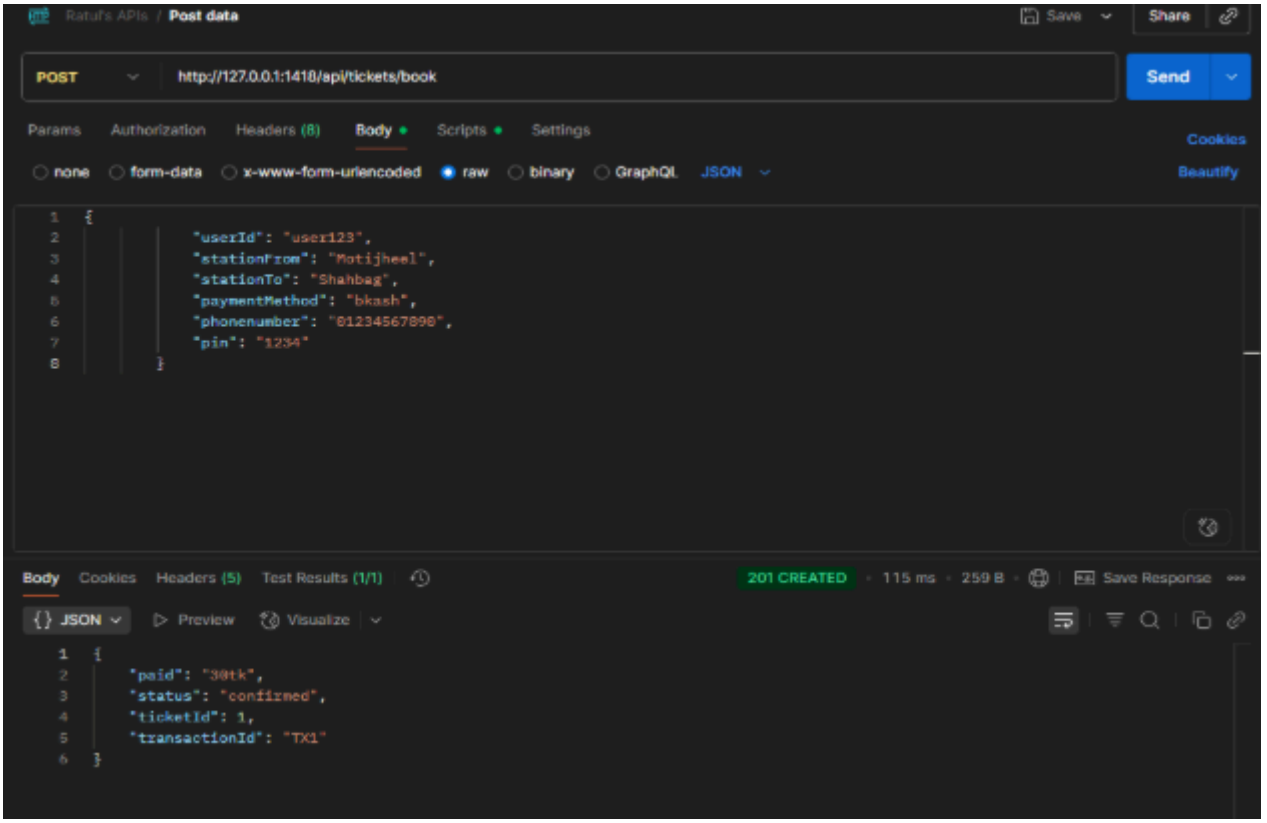
Get Walking direction:

```python
@app.route('/api/directions', methods=['GET'])
def get_directions():
    origin_lat = request.args.get('originLat', type=float)
    origin_lng = request.args.get('originLng', type=float)
    station_id = request.args.get('stationId', type=int)

    if not all([origin_lat, origin_lng, station_id]):
        return jsonify({"message": "originLat, originLng and stationId are required"}), 400

    station = Station.query.get(station_id)
    if not station:
        return jsonify({"message": "Station not found"}), 404

    directions = {
        "origin": {"lat": origin_lat, "lng": origin_lng},
        "destination": {"lat": station.latitude, "lng": station.longitude},
        "steps": [{"instruction": "Walk 500m north", "distance": "500m"}]
    }
    return jsonify(directions)
```



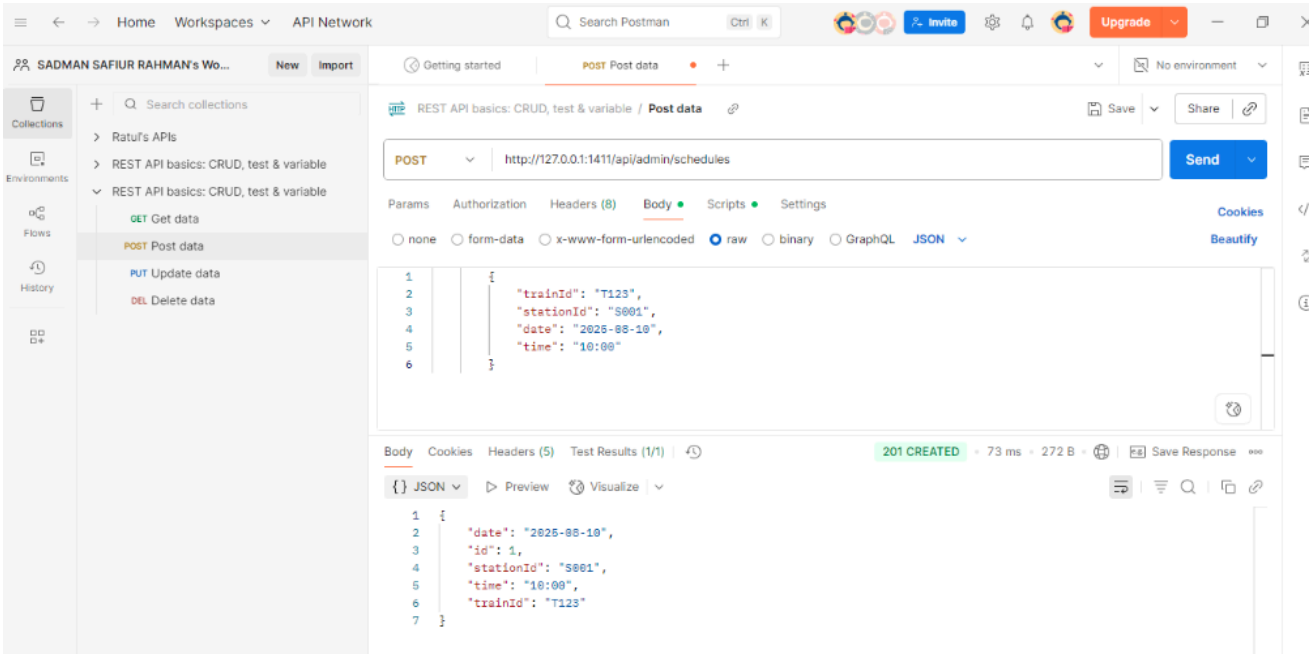
Payment Method:

```python
@app.route('/api/tickets/book', methods=['POST'])
def book_ticket():
    data = request.get_json()
    if not data or not all(k in data for k in ['userId', 'stationFrom', 'stationTo', 'paymentMethod']):
        return jsonify({"message": "Missing required fields"}), 400

    payment_method = data['paymentMethod']
    if payment_method == 'bkash':
        if not all(k in data for k in ['phonenumber', 'pin']):
            return jsonify({"message": "Missing required fields for bkash payment"}), 400
        new_ticket = Ticket(
            userId=data['userId'],
            stationFrom=data['stationFrom'],
            stationTo=data['stationTo'],
            paymentMethod=data['paymentMethod'],
            phonenumber=data['phonenumber'],
            pin=data['pin']
        )
    elif payment_method == 'card':
        if not all(k in data for k in ['cardid', 'card_holder', 'cvc_code']):
            return jsonify({"message": "Missing required fields for card payment"}), 400

        card_payment = CardPayment(
            cardid=data['cardid'],
            card_holder=data['card_holder'],
            cvc_code=data['cvc_code']
        )
        db.session.add(card_payment)

        new_ticket = Ticket(
            userId=data['userId'],
            stationFrom=data['stationFrom'],
            stationTo=data['stationTo'],
            paymentMethod=data['paymentMethod']
        )
    else:
        return jsonify({"message": "Invalid payment method"}), 400

    db.session.add(new_ticket)
    db.session.commit()

    return jsonify({
        "ticketId": new_ticket.id,
        "transactionId": f"IX{new_ticket.id}",
        "status": new_ticket.status,
        "paid": "30tk"
    }), 201
```



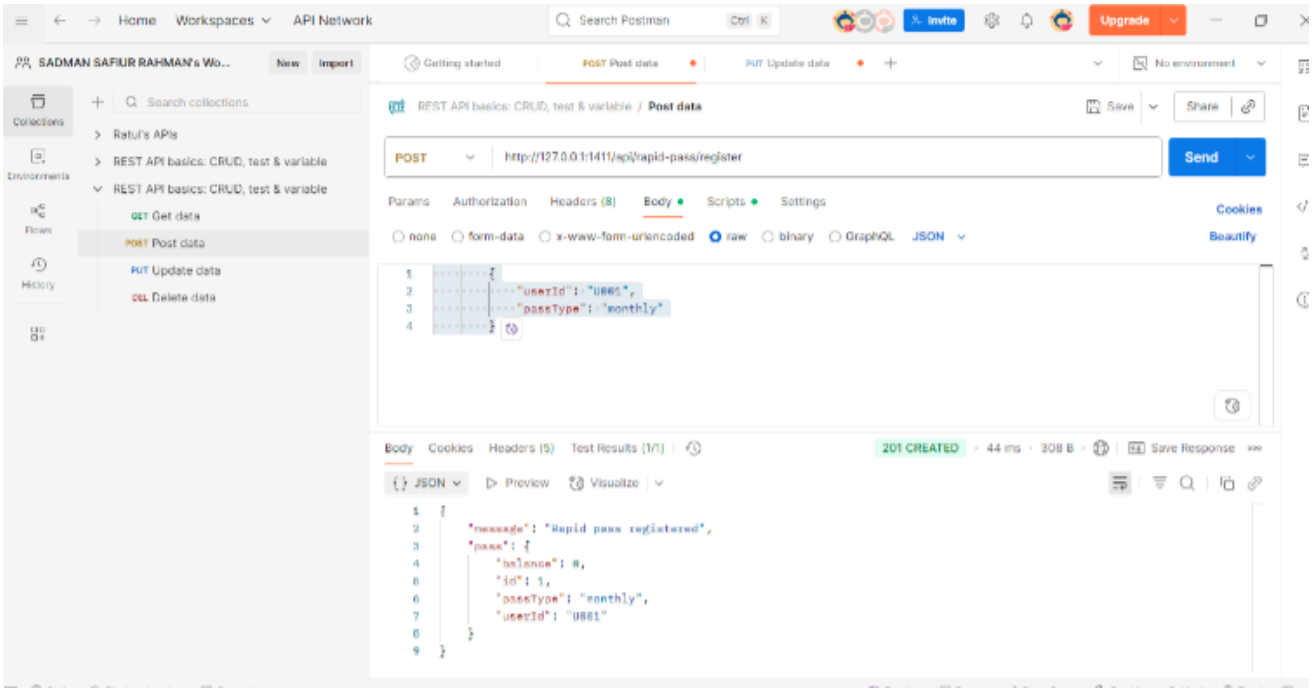
Create Metro Schedule:

```python
# API 1: Create Metro Schedule
@app.route('/api/admin/schedules', methods=['POST'])
def create_schedule():
    data = request.get_json()
    new_schedule = Schedule(
        trainId=data['trainId'],
        stationId=data['stationId'],
        date=data['date'],
        time=data['time']
    )
    db.session.add(new_schedule)
    db.session.commit()
    return jsonify({'id': new_schedule.id, 'trainId': new_schedule.trainId, 'stationId': new_schedule.stationId, 'date': new_schedule.date, 'time': new_schedule.t
```



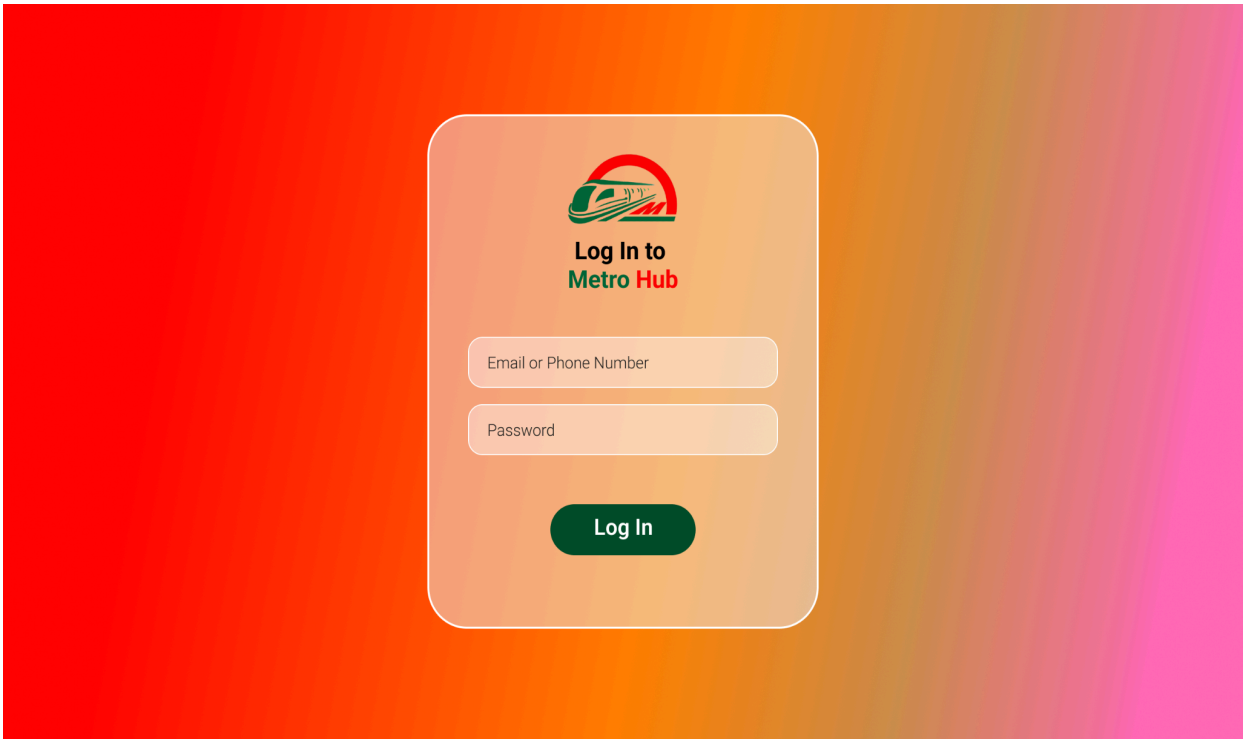
Rapid-Pass Register:

```python
@app.route('/api/rapid-pass/register', methods=['POST'])
def register_rapid_pass():
    data = request.get_json()
    new_pass = RapidPass(
        userId=data['userId'],
        passType=data['passType'],
        balance=0
    )
    db.session.add(new_pass)
    db.session.commit()
    return jsonify({'message': 'Rapid pass registered', 'pass': {'id': new_pass.id, 'userId': new_pass.userId, 'passType': new_pass.passType, 'balance': new_pass.
```

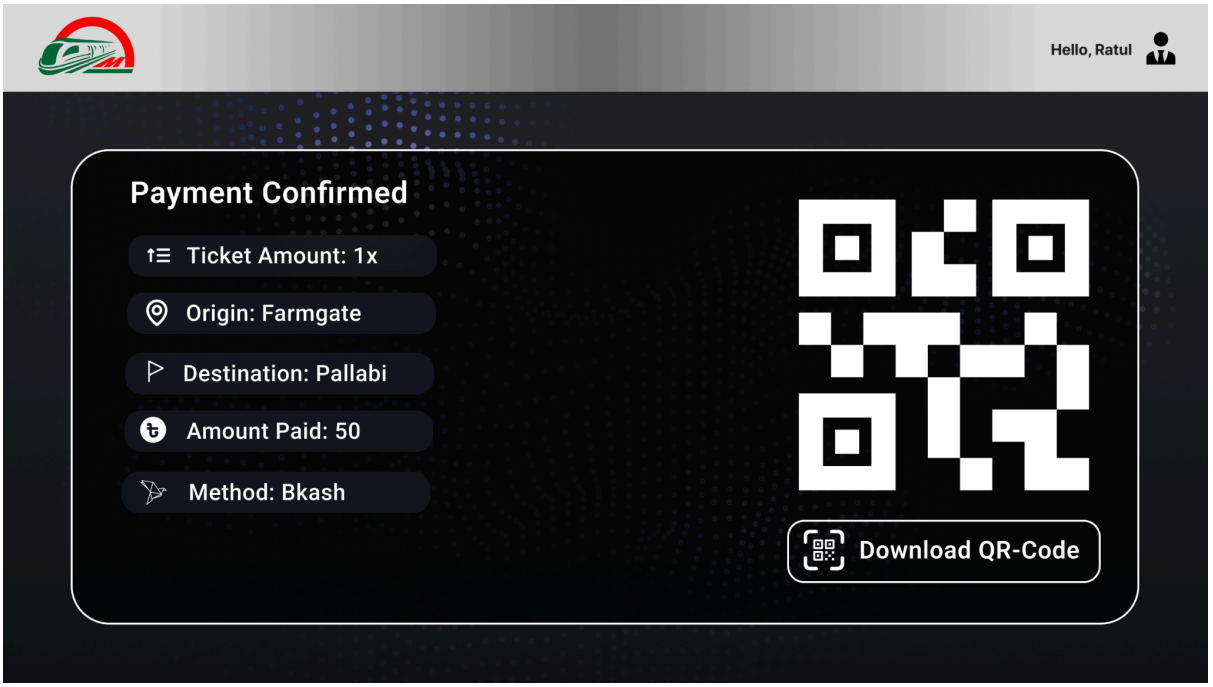


5. User Interface Design

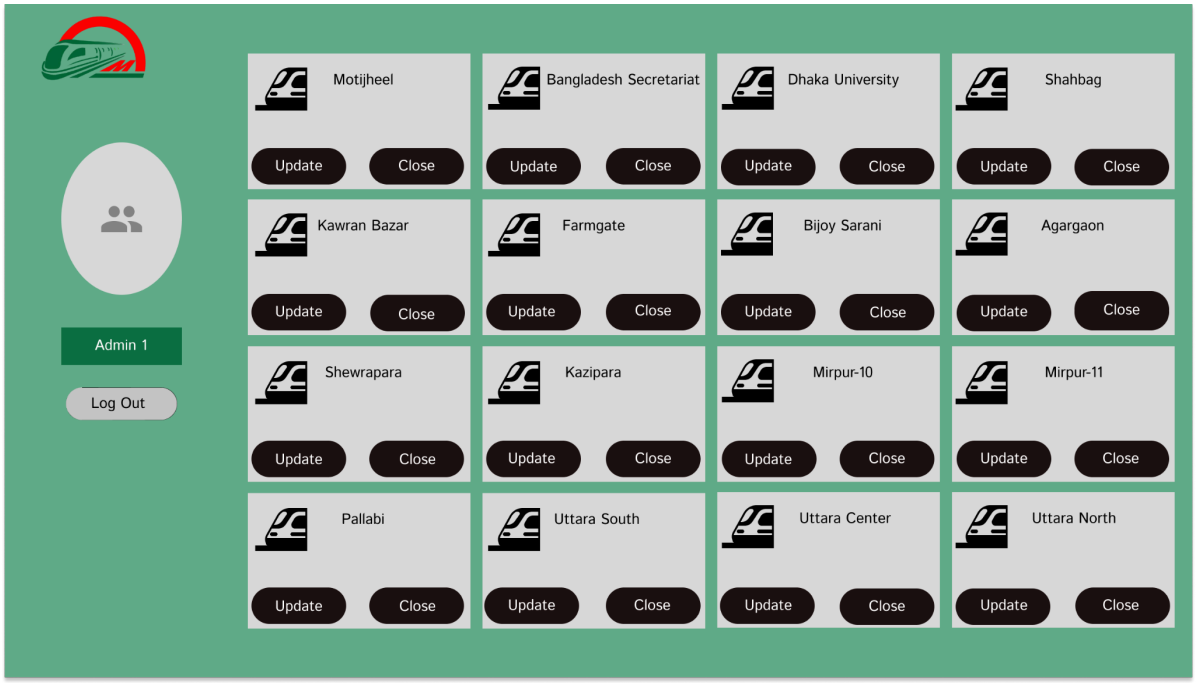
Login Page:



Payment Confirmation:

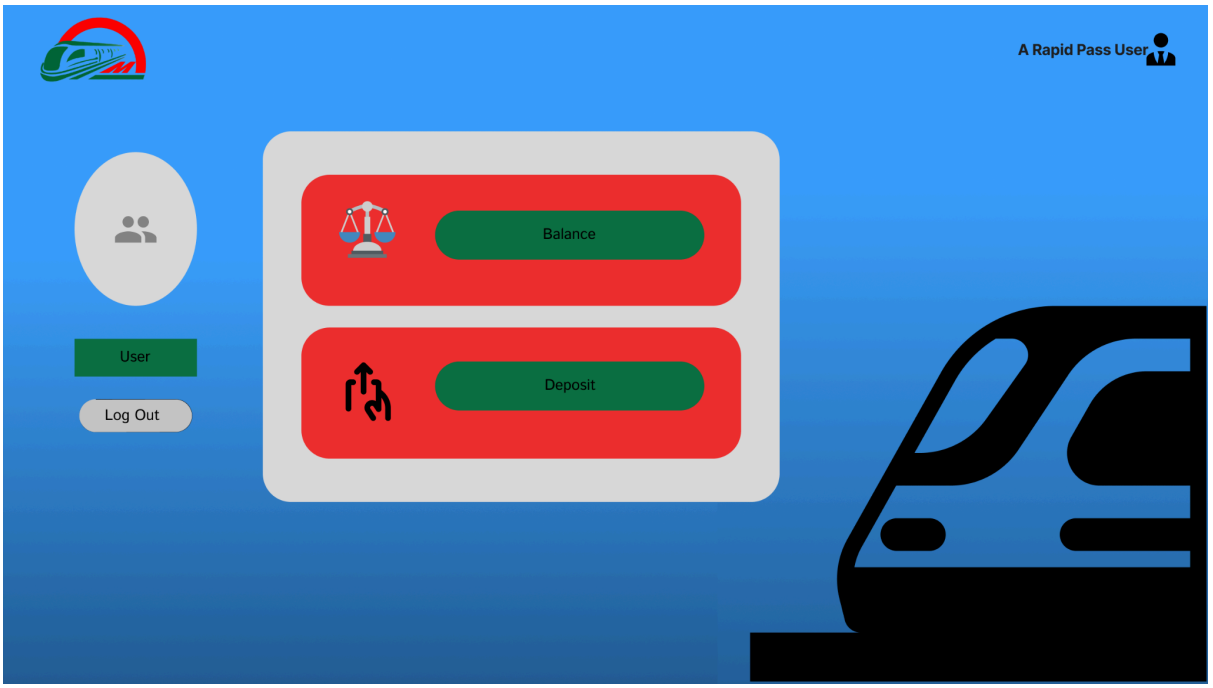


Admin updating and closing stations:



Available Stations:

User Dashboard for checking balance & depositing money:



Link: Figma

6. Frontend Development

1. BookTicket.jsx

```
import { useState, useEffect, useRef } from "react";

import { motion, AnimatePresence } from "framer-motion";

import Button from "../ui/Button";

import { Card, CardContent, CardHeader, CardTitle } from "../ui/Card";

import { Badge } from "../ui/badge.jsx";

import { Separator } from "../ui/separator.jsx";

import { Clock, MapPin, CreditCard, Wallet, Train, ArrowRight,
CheckCircle, ChevronDown } from "lucide-react";
```

```
import { Alert, AlertDescription } from "../ui/alert.jsx";

import axios from 'axios';

import { useTranslation } from 'react-i18next';

import { useAuth } from '../../context/AuthContext';



const API = import.meta.env.VITE_API_URL;



const parseTimeToMinutes = (t = '00:00') => {

  const [hh = '0', mm = '0'] = String(t).split(':');

  return Number(hh) * 60 + Number(mm);

};



const formatFareForLocale = (amount, i18n) => {

  try {

    const locale = i18n?.language === 'bn' ? 'bn-BD' : 'en-US';

    return new Intl.NumberFormat(locale, { maximumFractionDigits: 0
}).format(amount);

  } catch {

    return String(amount);

  }
```

```
};



export default function BookTicket() {

  const { t, i18n } = useTranslation();

  const { user, updateUser } = useAuth();

  const [stations, setStations] = useState([]);

  const [sourceStation, setSourceStation] = useState("");

  const [destinationStation, setDestinationStation] = useState("");

  const [schedules, setSchedules] = useState([]);

  const [selectedSchedule, setSelectedSchedule] = useState(null);

  const [loading, setLoading] = useState(false);

  const [schedulesLoading, setSchedulesLoading] = useState(false);

  const [error, setError] = useState("");

  const [paymentMethod, setPaymentMethod] = useState("gateway");

  const [isPaymentVisible, setIsPaymentVisible] = useState(false);



  const paymentSectionRef = useRef(null);



  useEffect(() => {
```

```
  const fetchStations = async () => {
    try {
      const res = await axios.get(`${API}/api/stations`);
      setStations(res.data || []);
    } catch (err) {
      setError(t('failed_to_fetch_stations'));
    }
  };
  fetchStations();
}, [t]);


useEffect(() => {
  setSelectedSchedule(null);
  setSchedules([]);
  setError("");
  if (!sourceStation || !destinationStation) return;


  const fetchSchedules = async () => {
    setSchedulesLoading(true);
```

```
      try {
        const res = await axios.get(
`${API}/api/schedules?sourceStation=${encodeURIComponent(sourceStation)}&destinationStation=${encodeURIComponent(destinationStation)}`
        );
        const data = res.data || [];
        setSchedules(data);
        if (data.length > 0) {
          setSelectedSchedule(data[0]);
        } else {
          setError(t('no_schedules_found'));
        }
      } catch (err) {
        console.error('fetch schedules error', err);
        setError(t('no_schedules_found'));
      } finally {
        setSchedulesLoading(false);
      }
    };
```

```
  fetchSchedules();
}, [sourceStation, destinationStation, t]);

useEffect(() => {
  const observer = new IntersectionObserver(
    ([entry]) => {
      setIsPaymentVisible(entry.isIntersecting);
    },
    {
      threshold: 0.2,
      rootMargin: "-100px 0px 0px 0px",
    }
  );

  if (paymentSectionRef.current) {
    observer.observe(paymentSectionRef.current);
  }
```

```
    return () => {
      if (paymentSectionRef.current) {
        observer.unobserve(paymentSectionRef.current);
      }
    };
  }, [selectedSchedule]);

  const scrollToPayment = () => {
    paymentSectionRef.current?.scrollIntoView({
      behavior: "smooth",
      block: "start",
    });
  };

  const handleBookTicket = async () => {
    if (!selectedSchedule) {
      setError(t('please_select_schedule'));
      return;
    }
```

```
    setLoading(true);
    setError("");

    try {
      const token = localStorage.getItem('token');
      const payload = {
        scheduleId: selectedSchedule.segmentIds ?
selectedSchedule.segmentIds[0] : selectedSchedule.scheduleId || null,
        segmentIds: selectedSchedule.segmentIds || null,
        amount: selectedSchedule.price || 0,
      };

      const res = await axios.post(`${API}/api/payment/init`, payload, {
        headers: {
          Authorization: token ? `Bearer ${token}` : undefined,
          'Content-Type': 'application/json'
        },
      });
```

```
      const gateway = res.data?.url || res.data?.GatewayPageURL ||
res.data?.redirect_url;

      if (gateway) window.location.replace(gateway);

      else setError(t('payment_initiation_failed'));

    } catch (err) {

      console.error('Payment initiation error', err.response?.data ||
err.message);

      setError(err.response?.data?.message || t('booking_failed'));

    } finally {

      setLoading(false);

    }

  };



  const handlePayFromBalance = async () => {

    if (!selectedSchedule) {

      setError(t('please_select_schedule'));

      return;

    }



    setLoading(true);
```

```
    setError("");



    try {

      const token = localStorage.getItem('token');

      const payload = {

        scheduleId: selectedSchedule.segmentIds ?
selectedSchedule.segmentIds[0] : selectedSchedule.scheduleId || null,

        amount: selectedSchedule.price || 0,

      };



      const res = await axios.post(`${API}/api/payment/pay-from-balance`,
payload, {

        headers: {

          Authorization: token ? `Bearer ${token}` : undefined,

          'Content-Type': 'application/json'

        },

      });



      const ticket = res.data;

      if (ticket && ticket._id) {
```

```
        // Fetch updated user data after successful payment
        const userRes = await axios.get(`${API}/api/user/profile`, {
          headers: {
            Authorization: token ? `Bearer ${token}` : undefined,
          },
        });
        console.log('User profile API response:', userRes.data);
        updateUser(userRes.data); // Update user context with new balance
        window.location.href = `/payment-success?ticket=${ticket._id}`;
      } else {
        setError(t('payment_failed'));
      }
    } catch (err) {
      console.error('Pay from balance error', err.response?.data || err.message);
      setError(err.response?.data?.message || t('payment_failed') || 'Payment failed.');
    } finally {
      setLoading(false);
    }
```

```
  };


  return (
    <div className="min-h-screen bg-background">
      <div className="container mx-auto px-4 py-8">
        <motion.div
          initial={{ opacity: 0, y: 20 }}
          animate={{ opacity: 1, y: 0 }}
          transition={{ duration: 0.5 }}
        >
          <div className="mb-8">
            <h1 className="text-4xl font-bold
mb-2">{t('book_your_ticket')}</h1>
            <p
className="text-muted-foreground">{t('select_route_description')}</p>
          </div>

          {error && (
            <Alert variant="destructive" className="mb-6">
              <AlertDescription>{error}</AlertDescription>
```

```
            </Alert>
          )}


          <div className="space-y-6">
            {/* Station Selection */}
            <Card>
              <CardHeader>
                <CardTitle className="flex items-center gap-2">
                  <MapPin className="h-5 w-5" />
                  {t('select_your_route')}
                </CardTitle>
              </CardHeader>
              <CardContent className="p-2 space-y-6">
                <div className="grid md:grid-cols-2 gap-4">
                  <div className="space-y-2">
                    <label className="text-m
font-semibold">{t('from')}</label>
                    <select
                      value={sourceStation}
                      onChange={(e) => setSourceStation(e.target.value)}
```

```
                    className={`w-full rounded-xl p-2 border-2
${sourceStation ? 'border-black' : 'border-gray-300'}`}>

                    <option
value="">{t('select_departure_station')}</option>

                    {stations.map((station) => (

                      <option key={station._id} value={station.name}>

                        {station.name}

                      </option>

                    ))}

                  </select>

                </div>



                <div className="space-y-2">

                  <label className="text-m
font-semibold">{t('to')}</label>

                  <select

                    value={destinationStation}

                    onChange={(e) =>
setDestinationStation(e.target.value)}

                    className={`w-full rounded-xl p-2 border-2
${destinationStation ? 'border-black' : 'border-gray-300'}`}>

                    <option
```

```
value="">{t('select_destination_station')}</option>

                    {stations.filter(s => s.name !==
sourceStation).map((station) => (

                      <option key={station._id} value={station.name}>

                        {station.name}

                      </option>

                    ))}

                  </select>

                </div>

              </div>



              {sourceStation && destinationStation && (

                <motion.div

                  initial={{ opacity: 0, height: 0 }}

                  animate={{ opacity: 1, height: "auto" }}

                  transition={{ duration: 0.3 }}

                >

                  <Separator className="my-6" />



                  <div className="space-y-4">
```

```
                    <h3 className="text-lg font-semibold flex
items-center gap-2">

                      <Clock className="h-5 w-5" />

                      {t('available_schedules')}

                    </h3>



                    {schedulesLoading ? (

                      <div className="space-y-3">

                        {[1, 2, 3].map((i) => (

                          <div key={i} className="animate-pulse">

                            <div className="h-20 bg-muted
rounded-lg"></div>

                          </div>

                        ))}

                      </div>

                    ) : schedules.length > 0 ? (

                      <div className="space-y-3">

                        {schedules.map((schedule) => {

                          const departureMinutes =
parseTimeToMinutes(schedule.departureTime);

                          const arrivalMinutes =
```

```
parseTimeToMinutes(schedule.arrivalTime);

                            const duration = arrivalMinutes -
departureMinutes;



                            return (

                              <motion.div

                                key={schedule._id}

                                whileHover={{ scale: 1.02 }}

                                transition={{ type: "spring", stiffness:
300 }}

                              >

                                <Card

                                  className={`cursor-pointer
transition-all ${

                                    selectedSchedule?._id === schedule._id

                                      ? 'ring-2 ring-primary bg-primary/5'

                                      : 'hover:shadow-md'

                                  }`}

                                  onClick={() =>
setSelectedSchedule(schedule)}

                                >

                                  <CardContent className="p-4">
```

```
                                    <div className="flex items-center
justify-between">

                                      <div className="flex-1">

                                        <div className="flex items-center
gap-2 mb-2">

                                          <Train className="h-4 w-4
text-primary" />

                                          <span
className="font-semibold">{schedule.trainName}</span>

                                          <Badge
variant="outline">{schedule.direction}</Badge>

                                        </div>



                                        <div className="flex items-center
gap-4 text-sm text-muted-foreground">

<span>{schedule.departureTime}</span>

                                          <ArrowRight className="h-4 w-4"
/>

<span>{schedule.arrivalTime}</span>

                                          <span>• {duration}
{t('min')}</span>

                                        </div>
```

```
                                        <div className="mt-2 text-sm
text-muted-foreground">
                                          {schedule.path}
                                        </div>
                                      </div>

                                      <div className="text-right">
                                        <div className="text-lg font-bold
text-primary">
৳{formatFareForLocale(schedule.price, i18n)}
                                        </div>
                                        {selectedSchedule?._id ===
schedule._id && (
                                          <CheckCircle className="h-5 w-5
text-green-600 ml-auto mt-1" />
                                        )}
                                      </div>
                                    </div>
                                  </CardContent>
                                </Card>
```

```
                                </motion.div>

                              );

                            })}

                          </div>

                        ) : (

                          <div className="text-center py-8
text-muted-foreground">

                            <Train className="h-12 w-12 mx-auto mb-4
opacity-50" />

                            <p>{t('no_schedules_found_for_this_route')}</p>

                          </div>

                        )}

                      </div>

                    </motion.div>

                  )}

                </CardContent>

              </Card>



              {/* Journey Summary */}

              {selectedSchedule && (
```

```
            <motion.div
              initial={{ opacity: 0, y: 20 }}
              animate={{ opacity: 1, y: 0 }}
              transition={{ duration: 0.3 }}
            >
              <Card className="rounded-xl border shadow-lg">
                <CardHeader className="p-4">
                  <CardTitle className="flex items-center gap-2 text-2xl
font-bold">
                    <CheckCircle className="h-5 w-5 text-green-600" />
                    {t('journey_summary')}
                  </CardTitle>
                </CardHeader>
                <CardContent className="p-6">
                  <div className="grid md:grid-cols-2 gap-6">
                    <div className="space-y-3">
                      <div className="flex justify-between">
                        <span className="text-m
text-muted-foreground">{t('route')}</span>
                        <span className="text-base
font-medium">{sourceStation} ---→ {destinationStation}</span>
```

```
                        </div>

                        <div className="flex justify-between">

                          <span className="text-m
text-muted-foreground">{t('train')}</span>

                          <span className="text-base
font-medium">{selectedSchedule.trainName}</span>

                        </div>

                      </div>

                      <div className="space-y-3">

                        <div className="flex justify-between">

                          <span className="text-m
text-muted-foreground">{t('departure')}</span>

                          <span className="text-base
font-medium">{selectedSchedule.departureTime}</span>

                        </div>

                        <div className="flex justify-between">

                          <span className="text-m
text-muted-foreground">{t('arrival')}</span>

                          <span className="text-base
font-medium">{selectedSchedule.arrivalTime}</span>

                        </div>

                      </div>
```

```
                    </div>

                    <Separator className="my-4" />

                    <div className="flex justify-between items-center">

                      <span className="text-2xl
font-bold">{t('total_fare')}</span>

                      <span className="text-2xl font-bold
text-primary">৳{formatFareForLocale(selectedSchedule.price, i18n)}</span>

                    </div>

                  </CardContent>

                </Card>

              </motion.div>

            )}



            {/* Payment Method Selection - More Prominent */}

            {selectedSchedule && (

              <motion.div

                ref={paymentSectionRef}

                initial={{ opacity: 0, y: 20 }}

                animate={{ opacity: 1, y: 0 }}

                transition={{ duration: 0.3, delay: 0.1 }}
```

```
            >

              <Card className="rounded-xl border shadow-lg">

                <CardHeader className="p-4">

                  <CardTitle className="flex items-center gap-2
text-xl">

                    <CreditCard className="h-6 w-6" />

                    {t('choose_payment_method')}

                  </CardTitle>

                  <p className="text-sm
text-muted-foreground">{t('select_payment_description')}</p>

                </CardHeader>

                <CardContent className="space-y-4 p-6">

                  {user.role === "rapidPassUser" ? (

                    <div className="grid md:grid-cols-2 gap-4">

                      <motion.div

                        className={`p-6 border-2 rounded-xl
cursor-pointer transition-all ${

                          paymentMethod === "gateway"

                            ? 'border-primary bg-primary/10 shadow-lg'

                            : 'border-border hover:border-primary/30
hover:shadow-md'
```

```
                        }`}
                        onClick={() => setPaymentMethod("gateway")}
                        whileHover={{ scale: 1.02 }}
                        whileTap={{ scale: 0.98 }}
                      >
                        <div className="text-center space-y-3">
                          <div className={`w-16 h-16 mx-auto
rounded-full flex items-center justify-center ${
                            paymentMethod === "gateway" ? 'bg-primary
text-primary-foreground' : 'bg-muted'
                          }`}>
                            <CreditCard className="h-8 w-8" />
                          </div>
                          <div>
                            <h3 className="font-semibold
text-lg">{t('payment_gateway')}</h3>
                            <p className="text-sm
text-muted-foreground">
                              {t('payment_gateway_description')}
                            </p>
                          </div>
```

```
                            {paymentMethod === "gateway" && (
                              <div className="flex items-center
justify-center gap-2 text-primary">
                                <CheckCircle className="h-5 w-5" />
                                <span className="text-sm
font-medium">{t('selected')}</span>
                              </div>
                            )}
                          </div>
                        </motion.div>

                        <motion.div
                          className={`p-6 border-2 rounded-xl
cursor-pointer transition-all ${
                            paymentMethod === "balance"
                              ? 'border-primary bg-primary/10 shadow-lg'
                              : 'border-border hover:border-primary/30
hover:shadow-md'
                          }`}
                          onClick={() => setPaymentMethod("balance")}
                          whileHover={{ scale: 1.02 }}
```

```
                            whileTap={{ scale: 0.98 }}

                          >

                            <div className="text-center space-y-3">

                              <div className={`w-16 h-16 mx-auto
rounded-full flex items-center justify-center ${

                                paymentMethod === "balance" ? 'bg-primary
text-primary-foreground' : 'bg-muted'

                              }`}>

                                <Wallet className="h-8 w-8" />

                              </div>

                              <div>

                                <h3 className="font-semibold
text-lg">{t('rapid_pass_balance')}</h3>

                                <p className="text-sm
text-muted-foreground">

                                  {t('pay_instantly_from_wallet')}

                                </p>

                                <div className="mt-2">

                                  <span className="text-lg font-bold
text-green-600">

                                    ৳{user && user.passBalance ?
user.passBalance.toFixed(2) : '0.00'} {t('available')}
```

```
                                </span>

                              </div>

                            </div>

                            {paymentMethod === "balance" && (

                              <div className="flex items-center
justify-center gap-2 text-primary">

                                <CheckCircle className="h-5 w-5" />

                                <span className="text-sm
font-medium">{t('selected')}</span>

                              </div>

                            )}

                          </div>

                        </motion.div>

                      </div>

                    ) : (

                      <div className="p-6 border-2 border-primary
rounded-xl bg-primary/5">

                        <div className="text-center space-y-3">

                          <div className="w-16 h-16 mx-auto rounded-full
bg-primary text-primary-foreground flex items-center justify-center">

                            <CreditCard className="h-8 w-8" />
```

```
                        </div>

                        <div>

                          <h3 className="font-semibold text-lg">Payment
Gateway</h3>

                          <p className="text-sm text-muted-foreground">

                            {t('secure_payment_description')}

                          </p>

                        </div>

                      </div>

                    </div>

                  )}



                  {/* Payment Button */}

                  <div className="pt-4 flex justify-center
items-center">

                    <Button

                      onClick={paymentMethod === 'gateway' ?
handleBookTicket : handlePayFromBalance}

                      disabled={loading || !selectedSchedule ||
(paymentMethod === "balance" && user.balance < selectedSchedule.price)}

                                                  className="text-lg
hover:bg-black hover:text-white rounded-md transition-all duration-300
```

```
ease-in-out px-4 py-2 border-black border-2"

                        size="lg"

                      >

                        {loading ? (

                          <div className="flex items-center gap-3">

                            <div className="animate-spin rounded-full h-5
w-5 border-b-2 border-white"></div>

                            <span>{t('processing_payment')}</span>

                          </div>

                        ) : paymentMethod === "balance" && user.balance <
selectedSchedule.price ? (

                          t('insufficient_balance')

                        ) : (

                          <div className="flex items-center gap-3 center
px-4">

                            {paymentMethod === "balance" ? <Wallet
className="h-5 w-5" /> : <CreditCard className="h-5 w-5" />}

                            <span>

                              {paymentMethod === "balance" ?
t('pay_from_balance') : t('proceed_to_payment')} -
৳{formatFareForLocale(selectedSchedule.price, i18n)}

                            </span>

                          </div>
```

```
                        )}

                      </Button>



                    {paymentMethod === "balance" && user.balance <
selectedSchedule.price && (

                      <div className="mt-3 p-3 bg-orange-50 border
border-orange-200 rounded-lg">

                        <p className="text-sm text-orange-700">

                          {t('insufficient_balance_top_up_message')}

                        </p>

                      </div>

                    )}

                  </div>

                </CardContent>

              </Card>

            </motion.div>

          )}

        </div>



        {/* Floating Scroll to Payment Button */}
```

```
          <AnimatePresence>
            {selectedSchedule && !isPaymentVisible && (
              <motion.div
                initial={{ opacity: 0, scale: 0, y: 20 }}
                animate={{
                  opacity: 1,
                  scale: 1,
                  y: 0,
                }}
                exit={{ opacity: 0, scale: 0, y: 20 }}
                transition={{ duration: 0.3, type: "spring", stiffness:
400 }}
                className="fixed bottom-6 right-6 z-50 group"
              >
                <motion.div
                  animate={{
                    y: [0, -4, 0],
                  }}
                  transition={{
                    duration: 2,
```

```
                    repeat: Infinity,
                    ease: "easeInOut"
                  }}
                >
                  <Button
                    onClick={scrollToPayment}
                    size="lg"
                    className="rounded-full h-16 w-16 shadow-lg
hover:shadow-xl transition-all duration-200 bg-primary hover:bg-primary/90
relative overflow-hidden"
                    aria-label="Scroll to payment section"
                  >
                    <motion.div
                      animate={{
                        y: [0, 3, 0],
                      }}
                      transition={{
                        duration: 1.5,
                        repeat: Infinity,
                        ease: "easeInOut"
```

```
                    }}
                  >
                    <ChevronDown className="h-6 w-6" />
                  </motion.div>

                  {/* Ripple effect */}
                  <motion.div
                    className="absolute inset-0 rounded-full
bg-white/20"
                    initial={{ scale: 0, opacity: 1 }}
                    animate={{ scale: 2, opacity: 0 }}
                    transition={{ duration: 2, repeat: Infinity }}
                  />
                </Button>
              </motion.div>

              {/* Tooltip */}
              <div className="absolute -top-12 left-1/2 transform
-translate-x-1/2 bg-black/80 text-white text-xs px-3 py-2 rounded-lg
opacity-0 group-hover:opacity-100 transition-opacity duration-200
whitespace-nowrap backdrop-blur-sm">
```

```
                {t('go_to_payment')}

                <div className="absolute top-full left-1/2 transform
-translate-x-1/2 w-0 h-0 border-l-4 border-r-4 border-t-4
border-transparent border-t-black/80"></div>

              </div>

            </motion.div>

          )}

        </AnimatePresence>

      </motion.div>

    </div>

  </div>

 );

}
```

- **Data Fetching:**
  - On component mount, it fetches a list of stations from the API endpoint /api/stations and stores them in state.
  - When both source and destination stations are selected, it queries /api/schedules with the route details to retrieve available schedules, updating the state accordingly.
- **User Interface:**
  - Features a dual-select dropdown for choosing source and destination stations, with dynamic filtering to exclude the selected source from destination options.
  - Displays a list of schedules with details such as train name, departure/arrival times, duration, and fare, presented in animated cards that users can select.
  - Shows a journey summary card with route, train, departure, arrival, and total fare details once a schedule is chosen.
  - Includes a payment section with options for payment gateway or balance (available for rapidPassUser role), featuring interactive cards with hover effects and balance visibility.
- **Payment Processing:**

    - Supports two payment methods:
        - Payment Gateway: Initiates a payment via POST /api/payment/init, redirecting to the gateway URL on success.
        - Balance Payment: Processes payment using POST /api/payment/pay-from-balance, checks balance sufficiency, updates user data via GET /api/user/profile, and redirects to a success page.
    - Includes loading states, error handling, and validation (e.g., requires a selected schedule and sufficient balance).
- **Enhancements:**
    - Uses Framer Motion for animations, such as fade-ins for cards, hover scaling for schedule cards, and a floating "Scroll to Payment" button with a ripple effect.
    - Implements an Intersection Observer to toggle the visibility of the floating button based on the payment section's position.
    - Supports internationalization with useTranslation, formatting fares according to the user's language (e.g., Bangla or English).
    - Integrates with AuthContext to access and update user data, using a token from localStorage for authenticated API requests.
- **Error and User Experience Management:**
    - Displays errors (e.g., failed API calls, insufficient balance) in a destructive alert.
    - Provides loading indicators (pulse animations) during data fetches and payment processing.
    - Ensures responsive design with grid layouts adjusting for different screen sizes.

Flow of Operation

1. **Initialization:** Loads stations on mount and sets up the UI.
2. **Route Selection:** User picks source and destination, triggering schedule data fetch.
3. **Schedule Selection:** User selects a schedule, updating the summary and enabling payment options.
4. **Payment Choice:** User selects a payment method, and the component handles the appropriate API call.
5. **Completion:** Processes payment, updates user context if successful, or shows errors, redirecting as needed.

2. Map.jsx

```
import React, { useState, useEffect } from 'react';

import { useAuth } from '../../context/AuthContext';

import axios from 'axios'; // for Nominatim API

import { useTranslation } from 'react-i18next';

import { motion } from 'framer-motion';
```

```
const Map = () => {
  const { t } = useTranslation();
  const { user } = useAuth();
  const [mapSrc, setMapSrc] = useState('');
  const [hoverMapSrc, setHoverMapSrc] = useState(null); // New state
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [destination, setDestination] = useState('');
  const [source, setSource] = useState('');
  const [suggestions, setSuggestions] = useState([]);
  const [showSourceInput, setShowSourceInput] = useState(false);
  const [showNearbyStationsInput, setShowNearbyStationsInput] =
useState(false);
  const [nearbyStations, setNearbyStations] = useState([]);
  const [metroStations, setMetroStations] = useState([]);

  useEffect(() => {
    const fetchStations = async () => {
      try {
              const { data } = await
axios.get(`${import.meta.env.VITE_API_URL}/api/stations`);
        setMetroStations(data);
      } catch (error) {
        console.error('Failed to fetch stations', error);
```

```
      }

    };

    fetchStations();

  }, []);



  useEffect(() => {

setMapSrc(`https://maps.google.com/maps?q=23.8103,90.4125&t=&z=12&ie=UTF8&iwloc=&output=embed`);

    setLoading(false);

  }, []);



  const handleSearchLocation = () => {

    if (destination) {

setMapSrc(`https://maps.google.com/maps?q=${destination}&t=&z=15&ie=UTF8&iwloc=&output=embed`);

      setError('');

    } else {

      setError(t('enter_destination_error'));

    }

  };



  const handleGetDirections = (dest = destination) => {

    console.log('handleGetDirections called. Source:', source, 'Destination:', dest); // Debug log

    if (source && dest) { // Use dest instead of destination
```

```
setMapSrc(`https://maps.google.com/maps?saddr=${source}&daddr=${dest}&t=&z=15
&ie=UTF8&iwloc=&output=embed`);

      setError('');

    } else {

      setError(t('enter_source_destination_error'));

    }

  };



  const handleFindNearbyStations = async () => { // Added async

    if (source) {

      try {

        // Make API call to backend

              const response = await
axios.post(`${import.meta.env.VITE_API_URL}/api/stations/nearby`, { location:
source });

        setNearbyStations(response.data);

        setError('');

      } catch (err) {

        console.error('Error fetching nearby stations from backend:', err);

        setError(err.response?.data?.message ||
t('failed_fetch_nearby_stations'));

        setNearbyStations([]); // Clear previous results on error

      }

    } else {

      setError(t('enter_location_nearby_stations_error'));

    }
```

```
};


const handleCancelJourney = () => {
  window.location.reload();
};


const handleGoBack = () => {
  setSource('');
  setDestination('');
  setShowSourceInput(false);
  setError('');
  setNearbyStations([]);
};


const handleSourceChange = (e) => {
  setSource(e.target.value);
  if (error) {
    setError('');
  }
};


const handleDestinationChange = (e) => {
  const value = e.target.value;
  setDestination(value);
```

```
    if (error) {
      setError('');
    }
    if (value) {
      const filteredSuggestions = metroStations.filter(station =>
        station.name.toLowerCase().includes(value.toLowerCase())
      );
      setSuggestions(filteredSuggestions);
    } else {
      setSuggestions([]);
    }
  };

  const handleSuggestionClick = (suggestion) => {
    setDestination(suggestion.name);
    setSuggestions([]);
  };

  const handleStationHover = (stationName) => {
    if (source) { // Only show hover directions if source is available

setHoverMapSrc(`https://maps.google.com/maps?saddr=${source}&daddr=${stationName}&t=&z=15&ie=UTF8&iwloc=&output=embed`);
    }
  };
```

```
  const handleStationLeave = () => {
    setHoverMapSrc(null);
  };

  return (
    <div className="min-h-screen bg-background">
      <div className="container mx-auto px-4 py-8">
        <motion.div
          initial={{ opacity: 0, y: 20 }}
          animate={{ opacity: 1, y: 0 }}
          transition={{ duration: 0.5 }}
          className="relative"
        >
          <div className="mb-8">
            <h1 className="text-4xl font-bold mb-2">Metro Map</h1>
            <p className="text-muted-foreground">Explore the metro network and plan your routes.</p>
          </div>

          {user && user.role !== 'admin' && (
            <div className="absolute top-1 right-4 z-10 bg-white p-3 rounded-md shadow-md flex items-center space-x-2">
              {(showSourceInput || mapSrc.includes('saddr=')) && (
                <input
```

```
                type="text"
                placeholder={t('enter_your_location')}
                value={source}
                onChange={handleSourceChange}
                className="w-64 p-2 border border-gray-300 rounded-md"
                readOnly={mapSrc.includes('saddr=')} // Added readOnly
              />
            )}
            {(!showNearbyStationsInput || mapSrc.includes('saddr=')) && (
              <div className="relative">
                <input
                  type="text"
                  placeholder={t('enter_destination')}
                  value={destination}
                  onChange={handleDestinationChange}
                  className="w-64 p-2 border border-gray-300 rounded-md"
                  readOnly={mapSrc.includes('saddr=')} // Added readOnly
                />
                {suggestions.length > 0 && (
                  <ul className="absolute z-10 w-full bg-white border
border-gray-300 rounded-md mt-1">
                    {suggestions.map(suggestion => (
                      <li
                        key={suggestion.name}
                        onClick={() => handleSuggestionClick(suggestion)}
```

```
                        className="p-2 cursor-pointer hover:bg-gray-200"
                      >
                        {suggestion.name}
                      </li>
                    ))}
                  </ul>
                )}
              </div>
            )}
            {mapSrc.includes('saddr=') ? (
              <button
                onClick={handleCancelJourney}
                className="w-auto p-2 bg-transparent text-red-500
rounded-md border-2 border-red-500 font-bold transition-all duration-300
ease-in-out px-4 py-2 hover:bg-red-500 hover:text-white"
              >
                {t('cancel_journey')}
              </button>
            ) : ( // Existing logic for other buttons
              <>
                {!showSourceInput && !showNearbyStationsInput ? (
                  <>
                    <button
                      onClick={handleSearchLocation}
                      className="w-auto p-2 bg-transparent text-blue-500
rounded-md border-2 border-blue-500 font-bold transition-all duration-300
```

```
ease-in-out px-4 py-2 hover:bg-blue-500 hover:text-white"

                        >

                          {t('search')}

                        </button>

                        <button

                          onClick={() => setShowSourceInput(true)}

                          className="w-auto p-2 bg-transparent text-green-700
rounded-md border-2 border-green-700 font-bold transition-all duration-300
ease-in-out px-4 py-2 hover:bg-green-700 hover:text-white"

                        >

                          {t('go_to_location')}

                        </button>

                        <button

                          onClick={() => setShowNearbyStationsInput(true)}

                          className="w-auto p-2 bg-transparent text-purple-500
rounded-md border-2 border-purple-500 font-bold transition-all duration-300
ease-in-out px-4 py-2 hover:bg-purple-500 hover:text-white"

                        >

                          {t('find_nearby_station')}

                        </button>

                      </>

                    ) : showSourceInput ? (

                      <>

                        <button

                          onClick={() => handleGetDirections()} // Call without
arguments

                          className="w-auto p-2 bg-transparent text-blue-500
```

```
rounded-md border-2 border-blue-500 font-bold transition-all duration-300
ease-in-out px-4 py-2 hover:bg-blue-500 hover:text-white"

                        >

                          {t('get_directions')}

                        </button>

                        <button

                          onClick={handleGoBack}

                          className="w-auto p-2 bg-transparent text-gray-500
rounded-md border-2 border-gray-500 font-bold transition-all duration-300
ease-in-out px-4 py-2 hover:bg-gray-500 hover:text-white"

                        >

                          {t('go_back')}

                        </button>

                      </>

                    ) : (

                      <div className="flex items-center space-x-2">

                        <input

                          type="text"

                          placeholder={t('enter_your_location')}

                          value={source}

                          onChange={handleSourceChange}

                          className="w-64 p-2 border border-gray-300
rounded-md"

                        />

                        <button

                          onClick={handleFindNearbyStations}

                          className="w-auto p-2 bg-transparent text-purple-500
```

```
rounded-md border-2 border-purple-500 font-bold transition-all duration-300
ease-in-out px-4 py-2 hover:bg-purple-500 hover:text-white"
                      >
                        {t('search_nearby')}
                      </button>
                      <button
                        onClick={() => {
                          setShowNearbyStationsInput(false);
                          setNearbyStations([]); // Clear nearby stations
                        }}
                        className="w-auto p-2 bg-transparent text-gray-500
rounded-md border-2 border-gray-500 font-bold transition-all duration-300
ease-in-out px-4 py-2 hover:bg-gray-500 hover:text-white"
                      >
                        {t('cancel')}
                      </button>
                    </div>
                  )}
                </>
              )}
            </div>
          )}


          {loading && (
            <div className="flex items-center justify-center
h-[60vh]">{t('loading_map')}</div>
```

```
          )}


          {error && (
            <div className="absolute top-1/2 left-1/2 -translate-x-1/2
-translate-y-1/2 bg-red-500 text-white p-4 rounded-md shadow-md z-20">
              {error}
            </div>
          )}


          {!loading && (
            <div className="relative w-full h-[70vh] flex rounded-md
overflow-hidden mt-10 border border-gray-300 border-2">
              {(hoverMapSrc || mapSrc) && (
                <iframe
                  width="100%"
                  height="100%"
                  style={{ border: 0 }}
                  src={hoverMapSrc || mapSrc}
                  allowFullScreen
                ></iframe>
              )}


              {showNearbyStationsInput && nearbyStations.length > 0 && (
                <div className="absolute right-4 top-5 w-64 bg-white p-4
rounded-md shadow-md z-10 overflow-y-auto max-h-[calc(100%-100px)]">
                  <h3 className="text-lg font-semibold
```

```
mb-2">{t('nearby_stations')}</h3>

                  <ul>

                    {nearbyStations.map(station => (

                      <li

                        key={station.name}

                        onClick={() => {

                          setDestination(station.name);

                          handleGetDirections(station.name); // Pass the
station name directly

                          setShowNearbyStationsInput(false); // Hide the
nearby stations input after selecting

                        }}

                        onMouseEnter={() => handleStationHover(station.name)}
// Hover handler

                        onMouseLeave={handleStationLeave} // Leave handler

                        className="p-2 cursor-pointer hover:bg-gray-100
border-b border-gray-200 last:border-b-0"

                      >

                        {station.name} ({t('distance')}: {station.distance}
km)

                      </li>

                    ))}

                  </ul>

                </div>

              )}

            </div>

          )}
```

```
        </motion.div>

      </div>

    </div>

  );

};


export default Map;
```

State Management:

- Tracks mapSrc and hoverMapSrc for the embedded map URL, loading and error for UI states, and destination, source, suggestions, nearbyStations, and metroStations for location data.
- Controls visibility of input fields (showSourceInput, showNearbyStationsInput) for a step-by-step user experience.

useEffect Hooks:

- Fetches metro stations on mount to populate autocomplete suggestions.
- Initializes the map with a default location and removes loading state.

Functionality:

- Search Location: Updates the map to focus on the entered destination using Google Maps embed URL parameters.
- Get Directions: Generates a route from source to destination, requiring both fields to be filled.
- Find Nearby Stations: Posts the source location to the API to retrieve nearby stations, displaying them in a dropdown.
- Cancel Journey/Go Back: Resets the map and inputs or reloads the page.
- Input Handling: Updates source/destination states, filters metro station suggestions, and handles clicks on suggestions.
- Hover Effects: Previews directions to nearby stations on hover, resetting when the mouse leaves.

UI Components:

- Features a header with a title and description, animated with Framer Motion.
- Includes input fields for source and destination, with buttons for search, directions, and nearby station lookup.
- Displays an iframe for the map, overlaid with a nearby stations list when applicable, and error/loading messages.

- Buttons have hover effects and are styled with Tailwind CSS for a modern look.

User Restrictions: Limits functionality to non-admin users, showing inputs and buttons only when appropriate.

Error and Loading:

- Shows errors (e.g., missing inputs, API failures) in a centered alert.
- Displays a loading message during initial map setup.

3. UserDashboard.jsx

```
import React, { useState, useEffect } from 'react';

import { Link, useNavigate } from 'react-router-dom';

import { useTranslation } from 'react-i18next';

import { useAuth } from '../../context/AuthContext';

import axios from 'axios';

import { ArrowRight, Clock, Ticket, MapPin, TrendingUp, Wallet, Shield,
Cross, DollarSignIcon } from 'lucide-react';

import timeImage from '../../assets/time.png';

import ticketImage from '../../assets/ticket.png';

import mapImage from '../../assets/map.png';

import { Icon } from 'lucide-react';

import { motion, AnimatePresence } from "framer-motion";

import image0 from '../../assets/Hero/0.jpg';

import image1 from '../../assets/Hero/1.png';

import image2 from '../../assets/Hero/2.jpg';

import image3 from '../../assets/Hero/3.jpg';

import image4 from '../../assets/Hero/4.jpg';

import image5 from '../../assets/Hero/5.png';

import timeImage2 from '../../assets/timeu.png';
```

```
import ticketImage2 from '../../assets/ticket2.png';




const Button = ({ className, variant, size, asChild = false, ...props }) => {
  const Comp = asChild ? 'div' : 'button';
  return <Comp className={`${variant} ${size} ${className}`} {...props} />;
};

const Card = ({ className, ...props }) => (
  <div className={`card ${className}`} {...props} />
);

const ImageWithFallback = ({ src, alt, ...props }) => {
  const [error, setError] = useState(false);
  const handleError = () => setError(true);
  return error ? (
    <div className="image-fallback" {...props}>
      Error
    </div>
  ) : (
    <img src={src} alt={alt} onError={handleError} {...props} />
  );
};
```

```
const UserHero = () => {
  const { t, i18n } = useTranslation();
  const { user, token } = useAuth();
  const [currentSlide, setCurrentSlide] = useState(0);
  const [hoveredButton, setHoveredButton] = useState(null);
  const [fine, setFine] = useState(0);
  const [tripsThisMonth, setTripsThisMonth] = useState(0);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchProfile = async () => {
      if (user && token) {
        try {
          setLoading(true);
          const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/auth/profile`, {
            headers: {
              'x-auth-token': token,
            },
          });
          const userData = response.data;
          setFine(userData.fine || 0);
          setTripsThisMonth(userData.tripsThisMonth || 0);
```

```
        } catch (err) {
          setError(err.response?.data?.message || t('failed_fetch_profile'));
        } finally {
          setLoading(false);
        }
      }
    };

    fetchProfile();

    const intervalId = setInterval(fetchProfile, 5000); // Poll every 5
seconds

    return () => clearInterval(intervalId); // Cleanup on unmount
  }, [user, token, t]);

  const heroImages = [
    image0, image1, image2, image3, image4, image5
  ];

  const buttonImages = {
    book: ticketImage2,
    schedule: timeImage2,
    map: mapImage
  };
```

```
// Auto-slide effect
useEffect(() => {
  const interval = setInterval(() => {
    setCurrentSlide((prev) => (prev + 1) % heroImages.length);
  }, 5000);

  return () => clearInterval(interval);
}, [heroImages.length]);

const handleMouseEnter = (image) => {
  setHoveredButton(image);
};

const handleMouseLeave = () => {
  setHoveredButton(null);
};

return (
  <section className="relative bg-background overflow-hidden">
    <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8 py-16 lg:py-24">
      <div className="grid lg:grid-cols-2 gap-12 items-center">
        <div className="space-y-8">
          <div className="space-y-4">
            <h1 className="text-4xl lg:text-5xl xl:text-6xl text-foreground
```

```
max-w-xl">

                {t('welcome_back')}, {user?.name || 'User'}!

              </h1>

              <p className="text-lg font-bold text-muted-foreground
max-w-lg">

                {t('user_hero_description')}

              </p>

            </div>



            <div className="flex flex-col sm:flex-row gap-4">

                <Link to="/book-tickets" onMouseEnter={() =>
handleMouseEnter(buttonImages.book)} onMouseLeave={handleMouseLeave}>

                    <Button size="lg" className="hover:bg-black
hover:text-white rounded-md transition-all duration-300 ease-in-out px-4 py-2
border-black border-2">

                        <div className="flex items-center justify-center">

                        <Ticket className="h-4 w-7" />

                        </div>

                        <div className="flex items-center justify-center">

                        {i18n.language === "en" ? "Book New Trip " : "নতুন ট্রিপ
বুক করুন "}

                        <ArrowRight className="h-4 w-4" />

                        </div>

                    </Button>

                </Link>

                <Link to="/search-schedules" onMouseEnter={() =>
handleMouseEnter(buttonImages.schedule)} onMouseLeave={handleMouseLeave}>
```

```
                    <Button size="lg" className="hover:bg-black
hover:text-white rounded-md transition-all duration-300 ease-in-out px-4 py-2
border-black border-2">

                        <div className="flex items-center justify-center">

                        <Clock className="h-4 w-7" />

                        </div>

                        <div className="flex items-center justify-center">

                        {i18n.language === "en" ? "Live Schedules " : "লাইভ
সময়সূচী "}

                        < ArrowRight className="h-4 w-4" />

                        </div>

                    </Button>

                </Link>

                <Link to="/map" onMouseEnter={() =>
handleMouseEnter(buttonImages.map)} onMouseLeave={handleMouseLeave}>

                    <Button size="lg" className="hover:bg-black
hover:text-white rounded-md transition-all duration-300 ease-in-out px-4 py-2
border-black border-2">

                        <div className="flex items-center justify-center">

                        <MapPin className="h-4 w-7" />

                        </div>

                        <div className="flex items-center justify-center">

                        {i18n.language === "en" ? "Find Stations " : "স্টেশন
খুঁজুন "}

                        <ArrowRight className="h-4 w-4" />

                        </div>

                    </Button>
```

```
                </Link>

            </div>



            <div className="grid grid-cols-3 gap-4 pt-4">

              <Card className="p-4 text-center bg-gradient-to-br from-blue-50
to-blue-100 border-blue-200 rounded-md border-b-4">

                <div className="flex items-center justify-center mb-2">

                  <TrendingUp className="h-5 w-5 text-blue-600" />

                </div>

                <div className="text-2xl font-bold
text-blue-600">{tripsThisMonth}</div>

                <div className="text-sm font-bold text-blue-600">

                  {i18n.language === "en" ? "Trips This Month" : "এই মাসে যাত্রা"}

                </div>

              </Card>

              <Card className="p-4 text-center bg-gradient-to-br
from-green-50 to-green-100 border-green-200 rounded-md border-b-4">

                <div className="flex items-center justify-center mb-2">

                  <Wallet className="h-5 w-5 text-green-600" />

                </div>

                <div className="text-2xl font-bold text-green-600">

                  {user && user.rapidPassId ? `৳${user.passBalance}` : 'N/A'}

                </div>

                <div className="text-sm font-bold text-green-600">

                  {i18n.language === "en" ? "Rapid Pass Balance" : "র‍্যাপিড পাস
ব্যালেন্স"}
```

```
                </div>

              </Card>

              <Card className="p-4 text-center bg-gradient-to-br from-red-50
to-red-100 border-red-200 rounded-md border-b-4">

                <div className="flex items-center justify-center mb-2">

                  <DollarSignIcon className="h-5 w-5 text-red-600" />

                </div>

                <div className="text-2xl font-bold
text-red-600">৳{fine.toFixed(2)}</div>

                <div className="text-sm font-bold text-red-600">

                  {i18n.language === "en" ? "Due Fines" : "জরিমানা বাকি"}

                </div>

              </Card>

            </div>

          </div>



          <div className="relative">

            <div className="relative overflow-hidden rounded-2xl shadow-2xl">

              <AnimatePresence mode="wait">

                <motion.img

                  key={hoveredButton || currentSlide}

                  src={hoveredButton || heroImages[currentSlide]}

                  alt="Metro Station"

                  className="w-full h-96 lg:h-[500px] object-cover center
rounded-2xl"

                  initial={{ opacity: 0 }}
```

```
                animate={{ opacity: 1 }}
                exit={{ opacity: 0 }}
                transition={{ duration: 0.5 }}
              />
            </AnimatePresence>

            <div className="absolute inset-0 bg-gradient-to-t from-black/20
to-transparent" />
          </div>
        </div>
      </div>
    </section>
  );
};

const MyTicketsSection = () => {
  const { t } = useTranslation();
  const { user, token } = useAuth();
  const [tickets, setTickets] = useState([]);
  const [loadingTickets, setLoadingTickets] = useState(true);
  const [ticketsError, setTicketsError] = useState('');

  useEffect(() => {
    const fetchTickets = async () => {
```

```
      if (user && token) {
        try {
          setLoadingTickets(true);
          const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/user/tickets`, {
            headers: {
              'x-auth-token': token,
            },
          });
          setTickets(response.data.slice(0, 3));
        } catch (err) {
          setTicketsError(err.response?.data?.message ||
t('failed_to_fetch_ticket_details'));
        } finally {
          setLoadingTickets(false);
        }
      }
    };
    fetchTickets();
  }, [user, token]);


  if (!user) {
    return null; // Don't render if user is not logged in
  }
```

```
  return (
    <section id="my-tickets" className="py-16 lg:py-24 bg-gray-100">
      <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">
        <h2 className="text-3xl lg:text-4xl text-foreground text-center
mb-2">
          {t('my_tickets')}
        </h2>
        <div className="text-center mb-8">
          {t("txt_2")}<Link to="/booking-history" className="text-blue-500
hover:underline">
            {t('here')}
          </Link>
        </div>
        {loadingTickets ? (
          <div className="text-center">{t('loading')}</div>
        ) : ticketsError ? (
          <div className="text-center text-red-500">{ticketsError}</div>
        ) : tickets.length > 0 ? (
          <div className="grid md:grid-cols-2 lg:grid-cols-3 gap-6">
            {tickets.map((ticket) => (
              <div key={ticket._id} className="bg-white p-6 rounded-lg
shadow-md">
                <p><strong>{t('ticket_id')}</strong> {ticket._id}</p>
                <p><strong>{t('train')}</strong>
{ticket.schedule.trainName}</p>
                <p><strong>{t('from')}</strong>
{ticket.schedule.sourceStation}</p>
```

```
                <p><strong>{t('to')}</strong>
{ticket.schedule.destinationStation}</p>

                <p><strong>{t('departure')}</strong>
{ticket.schedule.departureTime}</p>

                <p><strong>{t('arrival')}</strong>
{ticket.schedule.arrivalTime}</p>

                <p><strong>{t('fare')}</strong> {ticket.amount}
{t('bdt')}</p>

                <p><strong>{t('status')}</strong> {ticket.paymentStatus}</p>

              </div>

            ))}

          </div>

        ) : (

          <p className="text-center
text-gray-500">{t('no_tickets_found')}</p>

        )}

      </div>

    </section>

  );

};



const Features = () => {

    const { t } = useTranslation();

    const navigate = useNavigate();



    const handleBookNow = () => {

        navigate('/book-tickets');
```

```
  };


  const features = [
    {
      icon: Clock,
      title: t('feature1_title'),
      description: t('feature1_description'),
      image: timeImage,
      action: t('view_schedules'),
      path: '/search-schedules'
    },
    {
      icon: Ticket,
      title: t('feature2_title'),
      description: t('feature2_description'),
      image: ticketImage,
      action: t('book_now'),
      onClick: handleBookNow
    },
    {
      icon: MapPin,
      title: t('feature3_title'),
      description: t('feature3_description'),
      image:
"https://images.unsplash.com/photo-1736117703288-3f918a212c31?crop=entropy&cs
=tinysrgb&fit=max&fm=jpg&ixid=M3w3Nzg4Nzd8MHwxfHNlYXJjaHwxfHxtZXRybyUyMG1hcCU
```

```
yMHRyYW5zcG9ydGF0aW9ufGVufDF8fHx8MTc1NTIzODI4MXww&ixlib=rb-4.1.0&q=80&w=1080&
utm_source=figma&utm_medium=referral",

        action: t('find_stations'),

        path: '/map'

      }

    ];



    return (

      <section id="features" className="py-16 lg:py-24 bg-muted/30">

        <div className="max-w-7xl mx-auto px-4 sm:px-6 lg:px-8">

          <div className="text-center space-y-4 mb-16">

            <h2 className="text-3xl lg:text-4xl text-foreground">

              {t('features_section_title')}

            </h2>

            <p className="text-lg text-muted-foreground max-w-2xl mx-auto">

              {t('features_section_description')}

            </p>

          </div>



          <div className="grid md:grid-cols-3 gap-8">

            {features.map((feature, index) => {

              const IconComponent = feature.icon;

              return (

                <Card key={index} className="bg-white rounded-lg shadow-md
overflow-hidden group transition-all duration-300 ease-in-out
hover:shadow-xl">
```

```
                <div className="relative h-48 overflow-hidden">
                  <ImageWithFallback
                    src={feature.image}
                    alt={feature.title}
                    className="w-full h-full object-cover rounded-t-lg transition-transform duration-300 ease-in-out group-hover:scale-105 object-position-top"
                  />
                </div>

                <div className="p-6">
                  <h3 className="text-xl font-bold mb-2">{feature.title}</h3>
                  <p className="text-gray-600 mb-4">{feature.description}</p>
                  {feature.path ? (
                    <Link to={feature.path} className="w-full">
                      <Button variant="outline" className="w-full border-gray-200 text-black rounded-md hover:bg-black hover:text-white flex items-center justify-center">
                        {feature.action}
                        <ArrowRight className="h-4 w-4 ml-2" />
                      </Button>
                    </Link>
                  ) : (
                    <Button onClick={feature.onClick} variant="outline" className="w-full border-gray-200 text-black rounded-md hover:bg-black hover:text-white flex items-center justify-center">
```

```
                        {feature.action}
                        <ArrowRight className="h-4 w-4 ml-2" />
                      </Button>
                    )}
                  </div>
                </Card>
              );
            })}
          </div>
        </div>
      </section>
    );
  };


export default function UserDashboard() {
  return (
    <div className="min-h-screen bg-background">
      <main>
        <UserHero />
        <MyTicketsSection />
        {/* <Features /> */}
      </main>
    </div>
  );
```

```
}
```

The UserDashboard component is a centralized hub for users with the following components:

- **UserHero:**
  - **Purpose:** Welcomes the user with a personalized greeting and provides quick access to key actions.
  - **Functionality:**
    - Fetches user profile data (fine, trips this month) every 5 seconds via GET /api/auth/profile.
    - Displays a slideshow of hero images with hover effects on action buttons, using AnimatePresence for smooth transitions.
    - Offers buttons to book tickets, view schedules, and find stations, with language-specific labels (English/Bangla).
    - Shows stats in cards: trips this month, rapid pass balance, and due fines, styled with gradients.
  - **State:** Manages current slide, hovered button, fine, trips, loading, and error **states.**
- **MyTicketsSection:**
  - **Purpose:** Displays a summary of the user's recent tickets**.**
  - **Functionality:**
    - Fetches up to three tickets via GET /api/user/tickets on mount.
    - Renders ticket details (ID, train, route, times, fare, status) in a grid, with a link to full history.
    - Handles loading and error states, hiding if the user is not logged in.
  - **State:** Tracks tickets, loading, and error states.
- **Features:**
  - **Purpose: Highlights key application features (schedules, booking, station finder).**
  - **Functionality:**
    - Displays a grid of cards with icons, titles, descriptions, and images, using ImageWithFallback for error handling.
    - Provides action buttons (links or clicks) to navigate to respective pages.
    - Uses Tailwind CSS for hover effects and responsive design.
  - **State:** Relies on static feature data with dynamic routing.
- **General Features:**
  - **Authentication:** Uses AuthContext to access user and token data, ensuring protected routes.
  - **Internationalization:** Supports multi-language content via useTranslation.
  - **Navigation:** Integrates Link and useNavigate for seamless page transitions.
  - **Custom Components:** Includes reusable Button and Card components, with

ImageWithFallback for robust image handling.
    - **Animations:** Employs Framer Motion for fade-in effects and slideshow transitions.
- **Flow:**
    - Loads user data and tickets on mount, polling profile data periodically.
    - Renders the hero section with interactive buttons, followed by ticket history.
    - Allows navigation to other pages (e.g., booking, schedules, map) based on user actions.

4. PaymentSuccess.jsx

```
import React, { useState, useEffect } from 'react';
import { useSearchParams } from 'react-router-dom';
import axios from 'axios';
import { useTranslation } from 'react-i18next';

const API = import.meta.env.VITE_API_URL;

const PaymentSuccess = () => {
  const { t } = useTranslation();
  const [searchParams] = useSearchParams();
  const ticketId = searchParams.get('ticket');
  const [ticket, setTicket] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchTicket = async () => {
      if (!ticketId) {
```

```
      setError('No ticket ID found in URL.');
      setLoading(false);
      return;
    }

    try {
      const token = localStorage.getItem('token');
      const res = await axios.get(`${API}/api/user/tickets/${ticketId}`, {
        headers: {
          Authorization: token ? `Bearer ${token}` : undefined,
        },
      });
      setTicket(res.data);
      console.log('API response ticket data:', res.data);
    } catch (err) {
      setError('Failed to fetch ticket details.');
    } finally {
      setLoading(false);
    }
  };

  fetchTicket();
}, [ticketId]);
```

```
  if (loading) {
    return <div className="container mx-auto p-4">{t('loading')}</div>;
  }

  if (error) {
    return <div className="container mx-auto p-4 text-red-500">{error}</div>;
  }

  return (
    <div className="container mx-auto p-4 max-w-7xl">
      <h1 className="text-2xl font-bold mb-4">{t('payment_successful')}!</h1>
      {ticket && (
        <div className="bg-white p-6 rounded-lg shadow-md flex">
          <div className="w-1/2 pr-4">
            <h2 className="text-2xl font-semibold
mb-4">{t('your_ticket_details')}</h2>
            <p className="text-lg"><strong>{t('ticket_id')}</strong>
{ticket._id}</p>
            <p className="text-lg"><strong>{t('from')}</strong>
{ticket.schedule.sourceStation}</p>
            <p className="text-lg"><strong>{t('to')}</strong>
{ticket.schedule.destinationStation}</p>
            <p className="text-lg"><strong>{t('fare')}</strong>
{ticket.amount} {t('bdt')}</p>
            <p className="text-lg"><strong>{t('time_of_booking')}</strong>
{new Date(ticket.createdAt).toLocaleString('en-US', {
              year: 'numeric',
```

```
              month: 'long',
              day: 'numeric',
              hour: '2-digit',
              minute: '2-digit'
            })}</p>
          </div>
          {ticket.qrCode && (
            <div className="w-1/2 flex flex-col items-center justify-center">
              <h3 className="text-2xl font-semibold
mb-4">{t('scan_qr_code')}</h3>
              <img src={ticket.qrCode} alt="Ticket QR Code" className="w-64
h-64" />
              <button
                onClick={() => {
                  const link = document.createElement('a');
                  link.href = ticket.qrCode;
                  link.download = `ticket_qr_${ticket._id}.png`;
                  document.body.appendChild(link);
                  link.click();
                  document.body.removeChild(link);
                }}
                className="mt-4 bg-blue-500 hover:bg-blue-700 text-white
font-bold py-2 px-4 rounded"
              >
                {t('download_qr_code')}
              </button>
```

```
              <button
                onClick={() => {
                  const scanUrl =
`${API}/api/qr/scan?ticketId=${ticket._id}`;
                  window.open(scanUrl, '_blank');
                }}
                className="mt-4 bg-green-500 hover:bg-green-700 text-white
font-bold py-2 px-4 rounded"
              >
                {t('simulate_scan')}
              </button>
            </div>
          )}
        </div>
      )}
    </div>
  );
};


export default PaymentSuccess;
```

The PaymentSuccess component is a post-payment interface with the following features:

- **State Management:**
  - Tracks ticket for fetched data, loading for API call status, and error for any issues.
  - Uses ticketId from URL search parameters to identify the specific ticket.
- **useEffect Hook:**
  - Fetches ticket details on mount if a ticketId is present, using a token from localStorage

for authentication.
    - Handles errors (e.g., missing ID, API failure) and sets loading state accordingly.
- **Functionality:**
    - Data Fetch: Retrieves ticket data including source, destination, fare, and QR code if generated.
    - Display: Shows a success message, ticket details formatted with toLocaleString, and a QR code section if applicable.
    - **Actions:**
        - Allows downloading the QR code as a PNG file using a dynamically created link.
        - Opens a new tab to simulate QR scanning via /api/qr/scan with the ticket ID.
- **UI Components:**
    - Features a centered container with a bold success header and a flex layout for ticket details and QR code.
    - Includes styled buttons for download and scan simulation with hover effects.
    - Displays loading or error messages as needed.
- **Error Handling:**
    - Checks for a valid ticketId and handles API errors with user-friendly messages.
    - Falls back to a loading or error state if data retrieval fails**.**
- **Internationalization:** Uses useTranslation to support multi-language labels (e.g., "BDT" for Bangladeshi Taka).
- **Flow:**
    - On load, checks for ticketId and fetches data.
    - Renders a success page with details and QR options if successful, or an error/loading message otherwise.

5. StationManagement.jsx

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';
import { useAuth } from '../../context/AuthContext';
import Input from '../ui/Input';
import Button from '../ui/Button';


const StationManagement = () => {
  const { token } = useAuth();
```

```
  const [stations, setStations] = useState([]);

  const [name, setName] = useState('');

  const [serial, setSerial] = useState('');

  const [latitude, setLatitude] = useState('');

  const [longitude, setLongitude] = useState('');

  const [editingStation, setEditingStation] = useState(null);

  const [loading, setLoading] = useState(false);

  const [error, setError] = useState('');



  useEffect(() => {

    const fetchStations = async () => {

      try {

        setLoading(true);

              const response = await
axios.get(`${import.meta.env.VITE_API_URL}/api/stations`);

        setStations(response.data);

      } catch (err) {

        setError('Failed to fetch stations.');

      } finally {

        setLoading(false);

      }

    };

    fetchStations();

  }, []);
```

```
useEffect(() => {
  if (!editingStation) {
    if (stations.length > 0) {
      const maxSerial = Math.max(...stations.map(s => s.serial));
      setSerial(String(maxSerial + 1));
    } else {
      setSerial('1');
    }
  }
}, [stations, editingStation]);


const handleAddStation = async () => {
  try {
    setLoading(true);
    setError('');
    const config = {
      headers: {
        'Content-Type': 'application/json',
        'x-auth-token': token,
      },
    };
    const { data } = await axios.post(
              `${import.meta.env.VITE_API_URL}/api/stations`,
      { name, serial: Number(serial), latitude, longitude },
```

```
      config
    );
    setStations([...stations, data]);
    setName('');
    setLatitude('');
    setLongitude('');
  } catch (err) {
    setError(err.response?.data?.message || 'Failed to add station.');
  } finally {
    setLoading(false);
  }
};

const handleUpdateStation = async () => {
  try {
    setLoading(true);
    setError('');
    const config = {
      headers: {
        'Content-Type': 'application/json',
        'x-auth-token': token,
      },
    };
    const { data } = await axios.put(
```

```
`${import.meta.env.VITE_API_URL}/api/stations/${editingStation._id}`,
        { name, serial: Number(serial), latitude, longitude },
        config
      );
      setStations(
        stations.map((s) => (s._id === editingStation._id ? data : s))
      );
      setEditingStation(null);
      setName('');
      setLatitude('');
      setLongitude('');
    } catch (err) {
      setError(err.response?.data?.message || 'Failed to update station.');
    } finally {
      setLoading(false);
    }
  };


  const handleDeleteStation = async (id) => {
    try {
      setLoading(true);
      setError('');
      const config = {
        headers: {
```

```
          'x-auth-token': token,
        },
      };
            await
axios.delete(`${import.meta.env.VITE_API_URL}/api/stations/${id}`, config);
      setStations(stations.filter((s) => s._id !== id));
    } catch (err) {
      setError(err.response?.data?.message || 'Failed to delete station.');
    } finally {
      setLoading(false);
    }
  };

  const startEditing = (station) => {
    setEditingStation(station);
    setName(station.name);
    setSerial(station.serial);
    setLatitude(station.latitude);
    setLongitude(station.longitude);
    setTimeout(() => {
      window.scrollTo({
        top: 0,
        behavior: 'smooth'
      });
      document.documentElement.scrollTop = 0;
```

```
      document.body.scrollTop = 0;
    }, 0);
  };

  const cancelEditing = () => {
    setEditingStation(null);
    setName('');
    setLatitude('');
    setLongitude('');
  };

  return (
    <div className="container mx-auto p-4">
      <h1 className="text-2xl font-bold mb-4">Station Management</h1>
      {error && <p className="text-red-500 bg-red-100 p-3 rounded
mb-4">{error}</p>}
      <div className="bg-white p-6 rounded-lg shadow-md mb-6">
        <h2 className="text-xl font-semibold mb-4">
          {editingStation ? 'Edit Station' : 'Add New Station'}
        </h2>
        <div className="grid grid-cols-1 md:grid-cols-3 gap-4">
          <Input
            type="text"
            placeholder="Station Name"
            value={name}
```

```
            onChange={(e) => setName(e.target.value)}
          />
          <Input
            type="number"
            placeholder="Serial"
            value={serial}
            onChange={(e) => setSerial(e.target.value)}
          />
          <Input
            type="number"
            placeholder="Latitude"
            value={latitude}
            onChange={(e) => setLatitude(e.target.value)}
          />
          <Input
            type="number"
            placeholder="Longitude"
            value={longitude}
            onChange={(e) => setLongitude(e.target.value)}
          />
        </div>
        <div className="flex justify-end mt-4">
          {editingStation && (
            <Button onClick={cancelEditing} className="mr-2 hover:scale-105
transition-transform duration-200" variant="secondary">
```

```
            Cancel
          </Button>
        )}
        <Button
          onClick={editingStation ? handleUpdateStation : handleAddStation}
          disabled={loading}
          className="hover:scale-105 transition-transform duration-200"
        >
          {loading
            ? editingStation
              ? 'Updating...'
              : 'Adding...'
            : editingStation
            ? 'Update Station'
            : 'Add Station'}
        </Button>
      </div>
    </div>


    <div className="bg-white p-6 rounded-lg shadow-md">
      <h2 className="text-xl font-semibold mb-4">Existing Stations</h2>
      {loading && stations.length === 0 ? (
        <p>Loading stations...</p>
      ) : (
```

```
          <div className="overflow-x-auto">

            <table className="min-w-full bg-white">

              <thead className="bg-gray-200">

                <tr>

                  <th className="py-2 px-4 border-b text-center">Name</th>

                  <th className="py-2 px-4 border-b text-center">Serial</th>

                  <th className="py-2 px-4 border-b
text-center">Latitude</th>

                  <th className="py-2 px-4 border-b
text-center">Longitude</th>

                  <th className="py-2 px-4 border-b text-center">Actions</th>

                </tr>

              </thead>

              <tbody>

                {stations.map((station) => (

                  <tr key={station._id}>

                    <td className="py-2 px-4 border-b
text-center">{station.name}</td>

                    <td className="py-2 px-4 border-b
text-center">{station.serial}</td>

                    <td className="py-2 px-4 border-b
text-center">{station.latitude}</td>

                    <td className="py-2 px-4 border-b
text-center">{station.longitude}</td>

                    <td className="py-2 px-4 border-b text-center">

                      <Button

                        onClick={() => startEditing(station)}
```

```
                        className="mr-2 hover:scale-105 transition-transform
duration-200"
                        size="sm"
                      >
                        Edit
                      </Button>
                      <Button
                        onClick={() => handleDeleteStation(station._id)}
                        disabled={loading}
                        variant="danger"
                        size="sm"
                        className="hover:scale-105 transition-transform
duration-200"
                      >
                        Delete
                      </Button>
                    </td>
                  </tr>
                ))}
              </tbody>
            </table>
          </div>
        )}
      </div>
    </div>
```

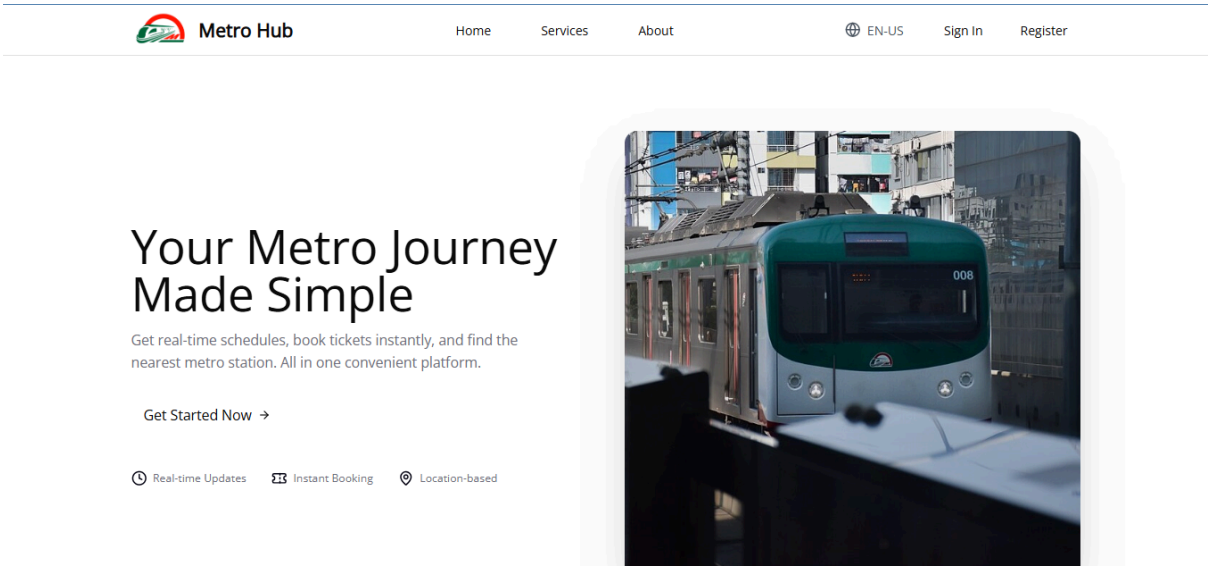
```
  );
};


export default StationManagement;
```

The StationManagement component is an admin interface with the following features:

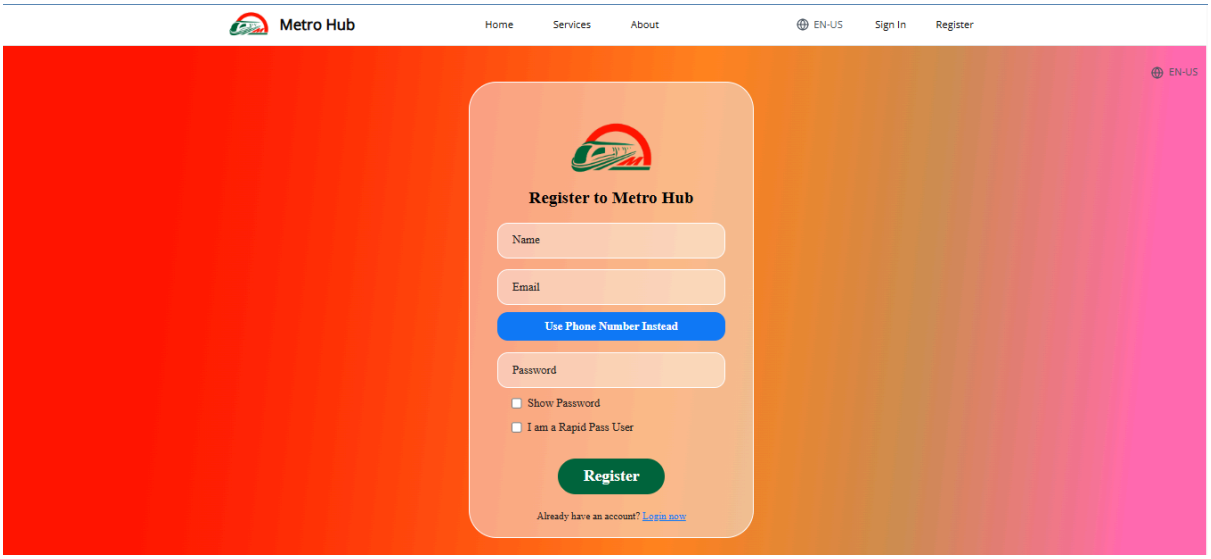
- **State Management:**
  - Tracks stations for the list of stations, name, serial, latitude, and longitude for form inputs, and editingStation for edit mode.
  - Manages loading for API calls and error for feedback.
- **useEffect Hooks:**
  - Fetches stations on mount via GET /api/stations.
  - Updates the serial number based on the highest existing value when not editing.
- **Functionality:**
  - Add Station: Posts new station data to /api/stations with validation and resets the form on success.
  - Update Station: Puts updated data to /api/stations/:id, updating the stations list and exiting edit mode.
  - Delete Station: Deletes a station via /api/stations/:id, filtering it from the list.
  - Edit Mode: Populates the form with selected station data and scrolls to the top.
  - Cancel Edit: Resets the form and exits edit mode.
- **UI Components:**
  - Features a header and two sections: a form for adding/editing and a table for existing stations.
  - Uses custom Input and Button components with Tailwind CSS for styling.
  - Displays errors in a red alert and loading states in the table or form buttons.
- **Authentication:**
  - Uses a token from AuthContext for all API requests, ensuring admin-only access.
- **Interactivity:**
  - Buttons have hover scale effects and disable during loading.
  - Table is responsive with horizontal scrolling for small screens.
- **Flow:**
  - Loads stations on mount, auto-sets serial for new entries.
  - Allows adding a new station or editing an existing one, with immediate UI updates.
  - Supports deletion with confirmation via button click.

7. User Manual

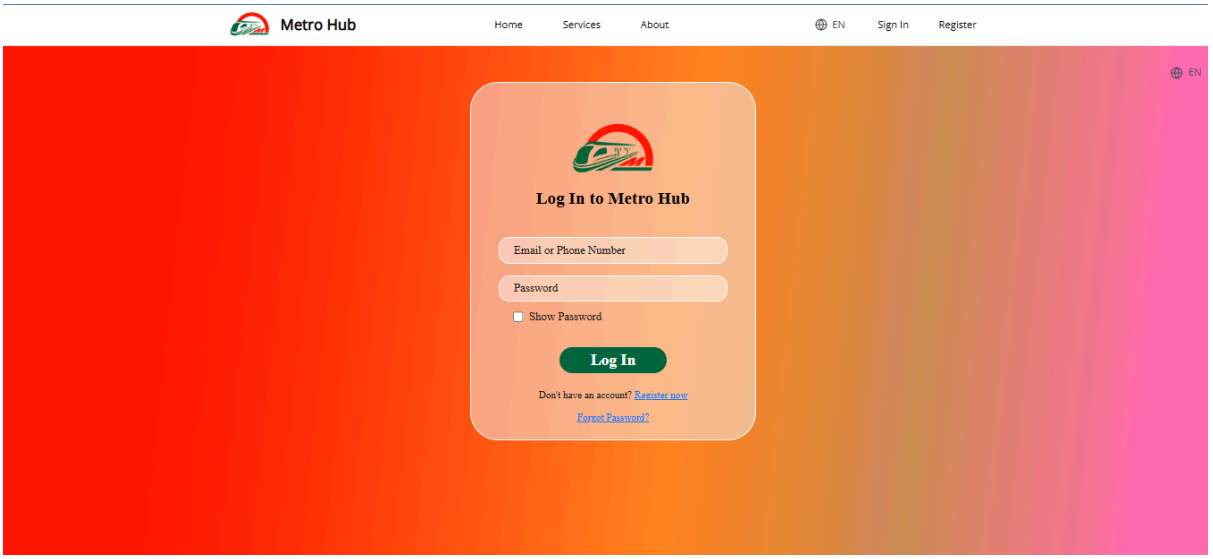
1. **The first page:**
    - **Start at the Top:** Head to the navigation bar and choose "Home," "Services," or "About." Switch language to EN-US, or click "Sign In" or "Register" to get started.
    - **Explore the Main Section:** Focus on the headline "Your Metro Journey Made Simple" for an overview. Read the description for real-time schedules, instant ticket booking, and location-based services.
    - **Take Action:** Click the "Get Started Now" button to begin using the platform.
    - **Check Features:** Glance at the bottom icons for quick access to real-time updates, instant booking, and location-based tools, with the metro train image on the right for context.



2. **If the User clicks on register:**
    - **Access the Registration Form:** Focus on the "Register to Metro Hub" section.
    - **Enter Details:** Input your "Name," "Email," and "Password" in the respective fields. Click "Use Phone Number Instead" if preferred to switch to phone-based registration.
    - **Additional Options:** Check "Show Password" to view your password or select "I am a Rapid Pass User" if applicable.
    - **Submit Registration:** Click the green "Register" button to create your account. If you already have an account, click "Login now" to sign in.



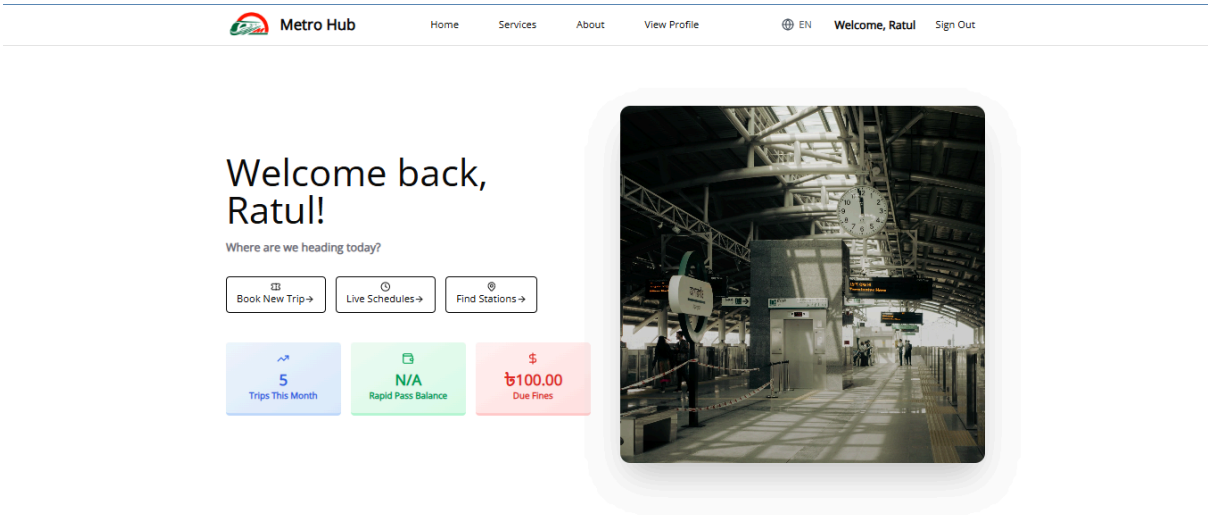
3. **If the user already registered:**
    - **Access the Login Interface**: Direct your attention to the central "Log In to Metro Hub" section, distinguished by the Metro Hub logo against a vibrant gradient backdrop.
    - **Input Credentials:** Enter your "Email or Phone Number" and "Password" into the designated fields. Activate the "Show Password" checkbox to reveal your password for verification.
    - **Execute Login:** Depress the green "Log In" button to authenticate your session. Should you encounter difficulties, click "Forgot Password?" to initiate recovery, or select "Register now" if you lack an account to establish one.



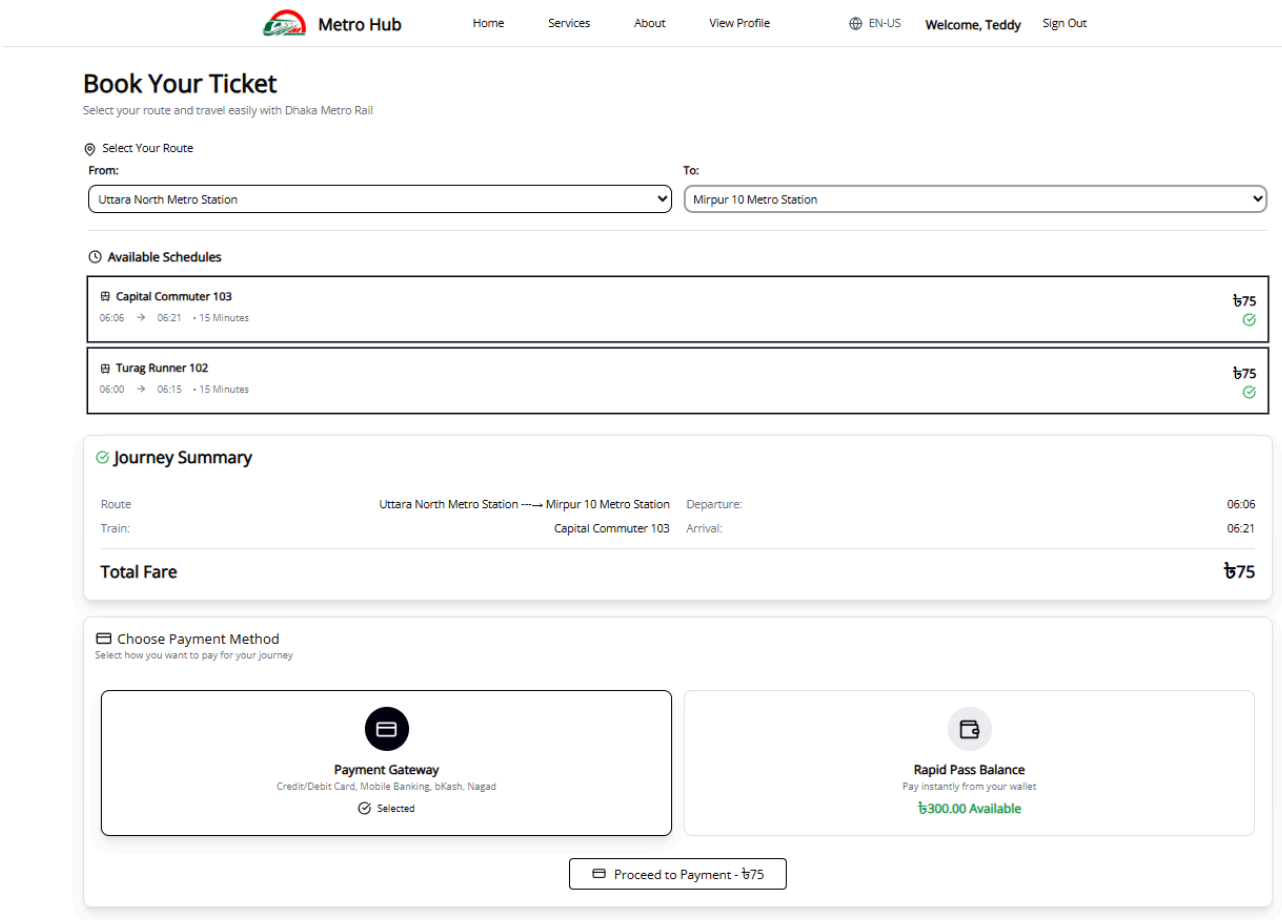
4. **After logging in:**
    - **Welcome Dashboard:** Focus on the "Welcome back, Ratul!" greeting, which personalizes your experience, accompanied by the prompt "Where are we heading today?"
    - **Plan Your Journey:** Select "Book New Trip" to start a new reservation, "Live Schedules" for real-time updates, or "Find Stations" to locate metro stops.
    - **Review Account Details:** Check "Trips This Month" (currently 5), "Rapid

Pass Balance" (N/A), and "Due Fines" (¥100.00) to manage your account status.

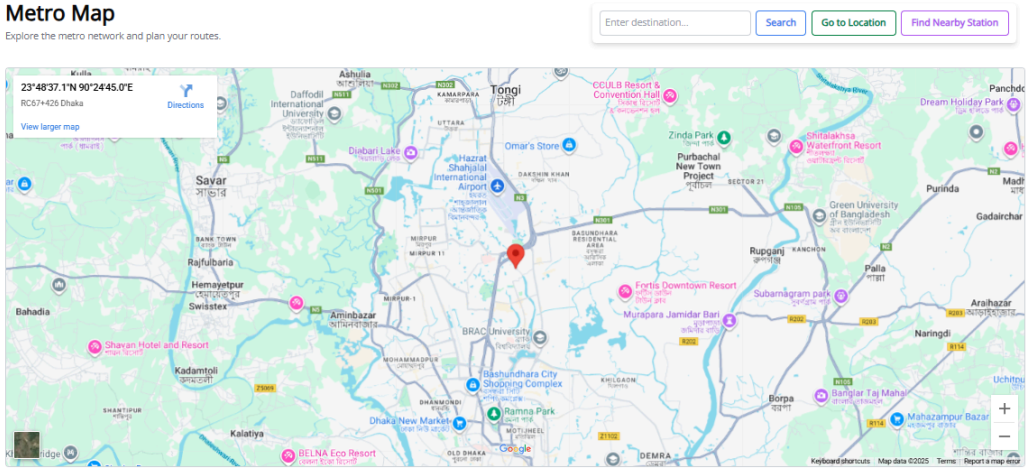
- **Visual Context:** Observe the station image on the right for an immersive view of the metro environment.



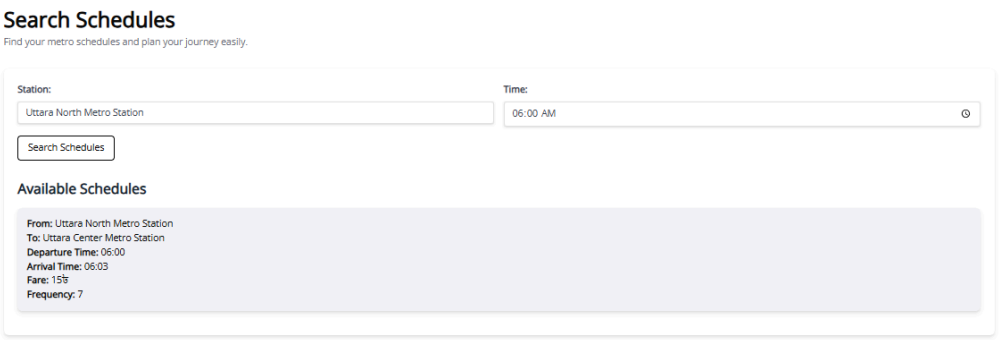
5. **If the user chooses book your ticket:**
    - **Select Route:** Under "Select Your Route," for an example choose "Uttara Center Metro Station" as the "From" location and "Kazipara Metro Station" as the "To" destination from the dropdown menus.
    - **Review Schedules:** Scroll to "Available Schedules" to pick a time, such as "06:09 → 06:24, 15 Minutes" (Capital Commuter 103) or "06:03 → 06:18, 15 Minutes" (Turag Runner 102), and select your preferred option.
    - **Check Summary:** Review the "Journey Summary" for the route (Uttara Center Metro Station to Kazipara Metro Station via Capital Commuter 103) and departure/arrival times (06:09 to 06:24).
    - **Confirm Fare:** Note the "Total Fare" of ¥75 and proceed to "Choose Payment Method" to select your payment option. If you are a rapid pass user you can pay from your deposit no need to choose payment gateway.
    - **Finalize Payment:** Click the "Proceed to Payment" button to complete the 75tk transaction securely via Credit/Debit Card, Mobile Banking, or bKash/Nagad.



6. **If the user chooses Find station:**
    - **Enter Metro Map:** Navigate to the "Metro Map" section, where the prompt "Explore the metro network and plan your routes" introduces the interface.
    - **Locate a Station:** Click "Find Nearby Station" to identify stations close to your current location, marked by a red pin at 23°48'37.1"N 90°24'45.0"E (RC-674-26 Dhaka). A pop-up provides "Directions" and "View larger map" options.
    - **Search Options:** Use the "Enter destination" field to input a specific location, then click "Search" or "Go to location" to explore, with the map displaying key stations and landmarks like Uttara, Mirpur, and Old Dhaka.



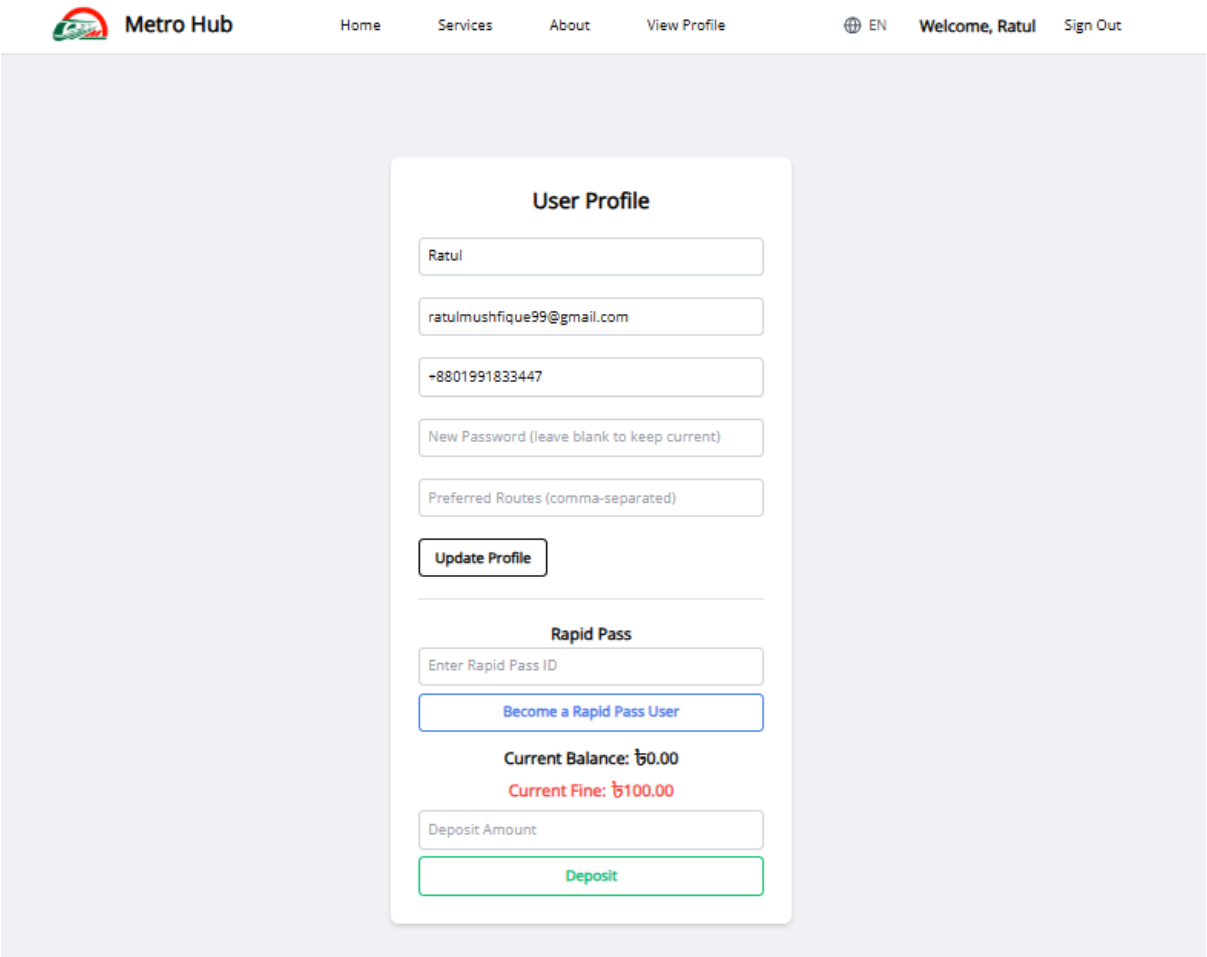
7. **If the user chooses View schedules:**
    - **Enter Search Criteria:** Navigate to the "Search Schedules" section, where the prompt "Find your metro schedules and plan your journey easily" guides you. For example, select "Uttara North Metro Station" from the "Station" dropdown and set the "Time" to 06:00 AM, then click "Search Schedules."
    - **Review Available Schedules:** Explore the "Available Schedules" section, which displays details for the selected station, including: From: Uttara North Metro Station To: Uttara Center Metro Station Departure Time: 06:00 Arrival Time: 06:03 Fare: 15tk Frequency: 7



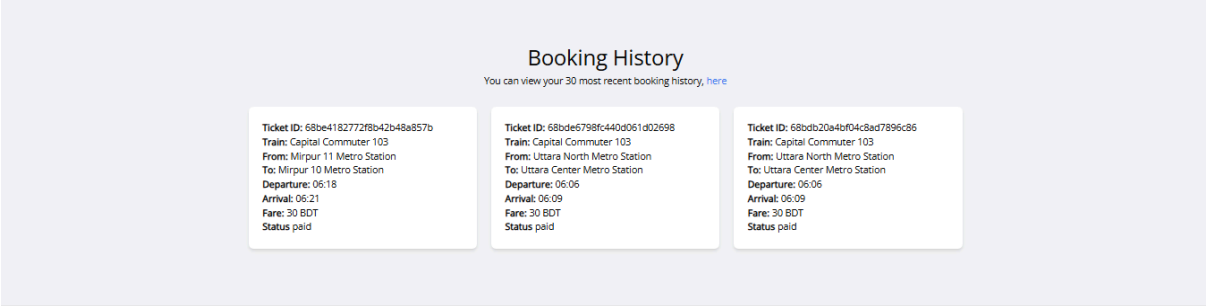
8. **If the user selects view profile:**
    - **View Personal Details:** Navigate to the "User Profile" section, where you can review and edit your information like for this user: "Ratul" (name), "ratulmusfiq99@gmail.com" (email), "+880199133447" (phone), "New Password" (leave blank to keep current), and "Preferred Routes" (comma-separated).
    - **Update Information:** Click the "Update Profile" button to save any changes to your details.
    - **Manage Rapid Pass:** Scroll to the "Rapid Pass" section. Enter a "Rapid Pass

ID" if applicable, or click "Become a Rapid Pass User" to register. Check your "Current Balance: 0.00tk" and "Current Fines: 100.00tk."

- **Add Funds:** Input a "Deposit Amount" and click the "Deposit" button to add funds to your account.

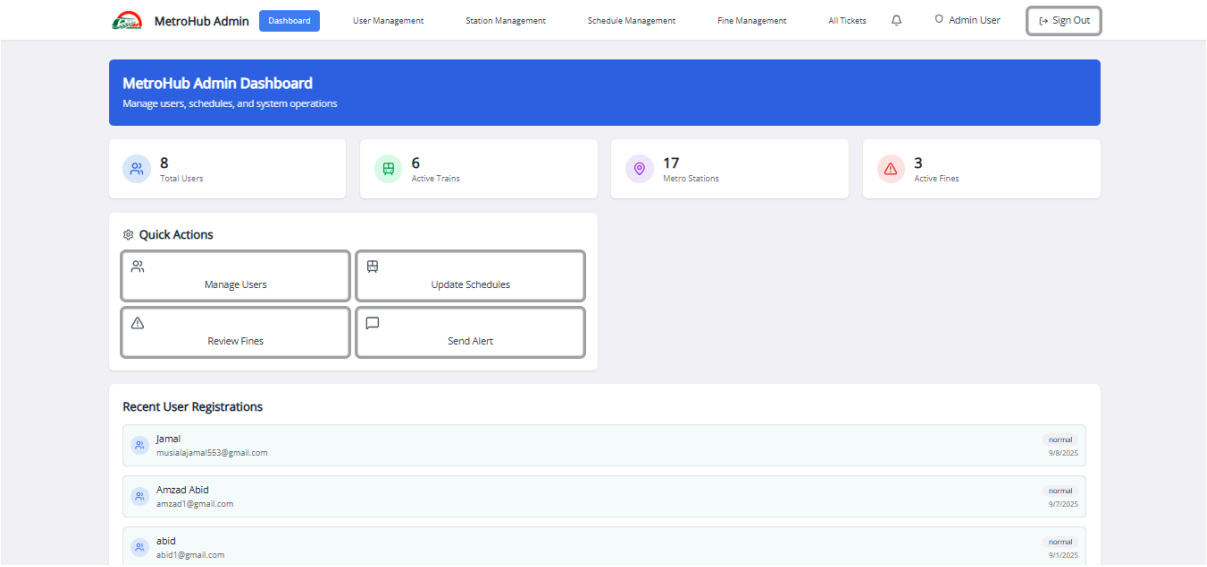


9. **For every user, they can see this booking history after logging:**
    - **Access History:** Start at the top navigation bar and select "Home," "Services," "About," or "View Profile." Adjust language to EN, or click "Sign Out" to exit. Note the "Welcome, Ratul" greeting.
    - **View Booking History:** Navigate to the "Booking History" section, where the prompt "You can view your 30 most recent booking history, here" introduces your past transactions.



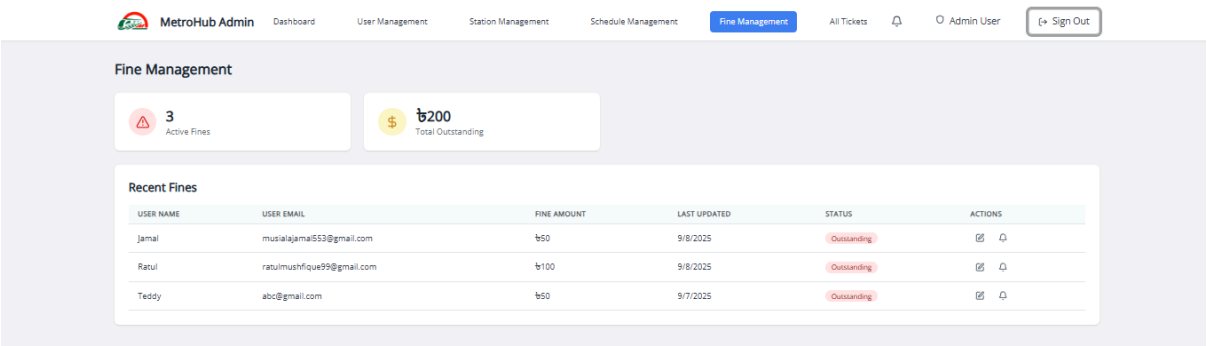
10. This is the admin dashboard:

- **Access Dashboard:** Start at the top navigation bar and select "Dashboard," "User Management," "Station Management," "Schedule Management," or "Fine Management." Adjust language to EN, or click "Admin User" and then "Sign Out" to exit.
- **Overview Section:** Navigate to the "MetroHub Admin Dashboard" header, which states "Manage users, schedules, and system operations." Review key metrics: Total Users, Active Trains, Metro Stations, and Active Fines.
- **Perform Quick Actions:** Use the "Quick Actions" panel to "Manage Users," "Update Schedules," "Review Fines," or "Send Alert" by clicking the respective buttons.
- **Check Recent Registrations:** Scroll to "Recent User Registrations" to view the latest sign-ups



11. Fine Management done by Admin:

- **Overview Section:** Navigate to the "Fine Management" section, which displays 3 Active Fines and a Total Outstanding amount of 200.
- **Review Recent Fines:** Scroll to "Recent Fines" to view the list of outstanding fines.

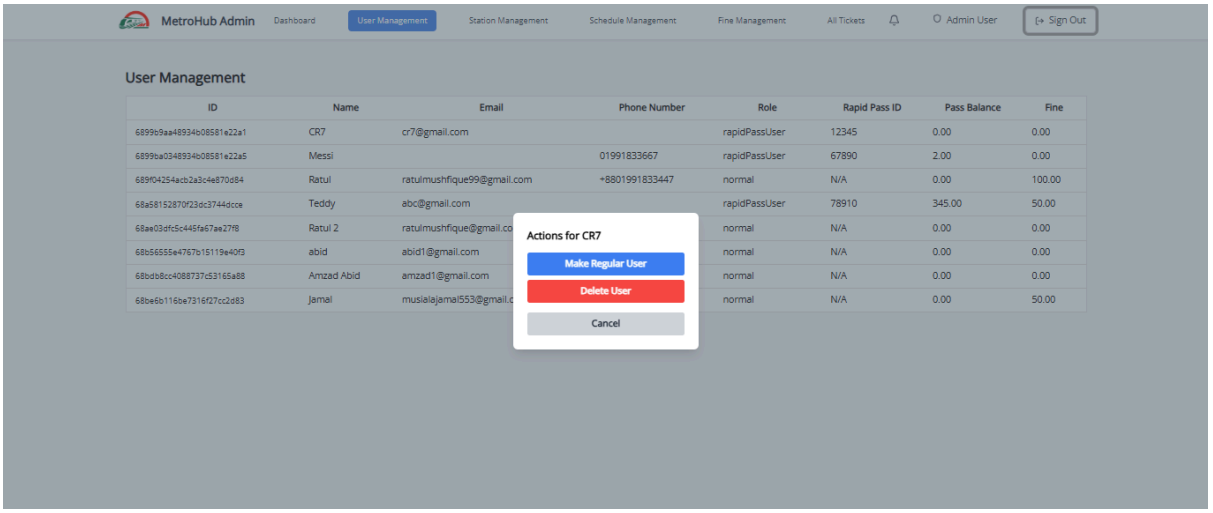


12. Admin manages the users:

- **View User Table:** Navigate to the "User Management" section, which

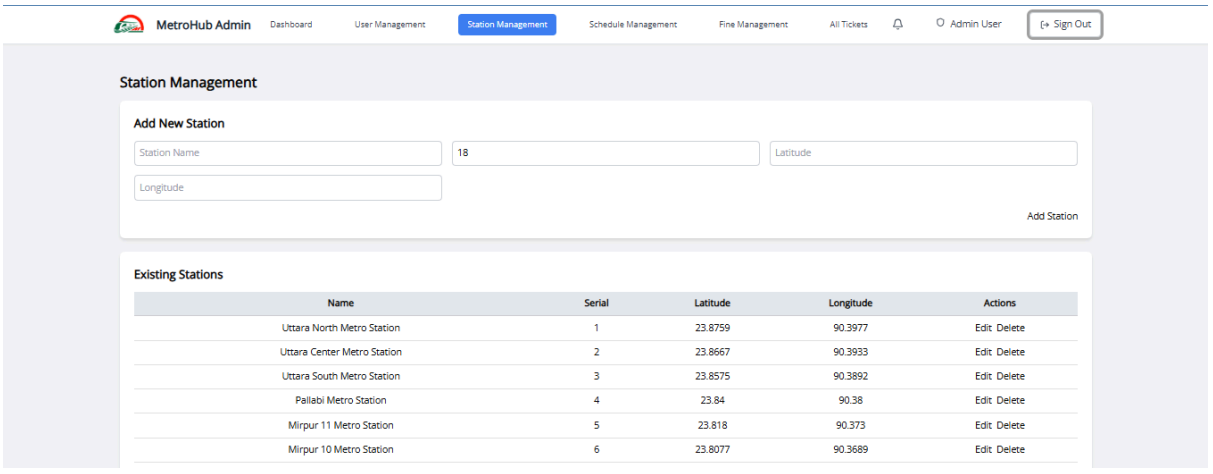
displays a table with columns for User ID, Name, Email, Phone Number, Role, Rapid Pass ID, Balance, and Fine.

- **Perform Actions:** Click the actions button (e.g., for user "CR7") to open a popup menu with options: "Make Regular User" (blue button) to upgrade the user's role, "Delete User" (red button) to remove the account, or "Cancel" (gray button) to close the menu without changes.



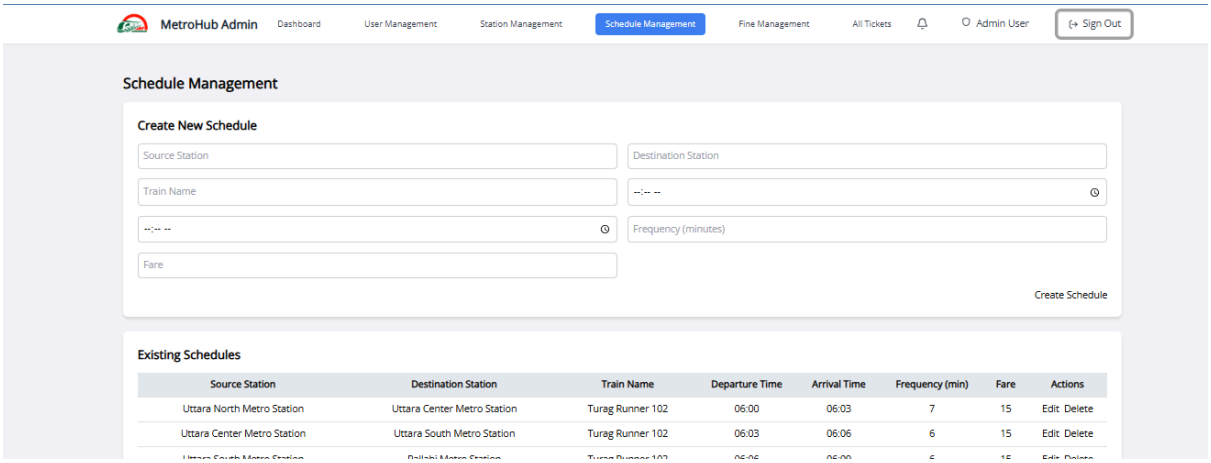
13. Admin managing the stations:

- **Add New Station:** Navigate to the "Add New Station" section. Enter a "Station Name" (e.g., Uttara Center Metro Station), "Serial" (e.g., 18), "Latitude" (e.g., 23.8579), and "Longitude" (e.g., 90.3777), then click "Add Station" to create a new entry.
- **Manage Existing Stations:** Scroll to "Existing Stations" to view the current list like: Uttara North Metro Station: Serial 1, Latitude 23.8579, Longitude 90.3777 (Edit/Delete)



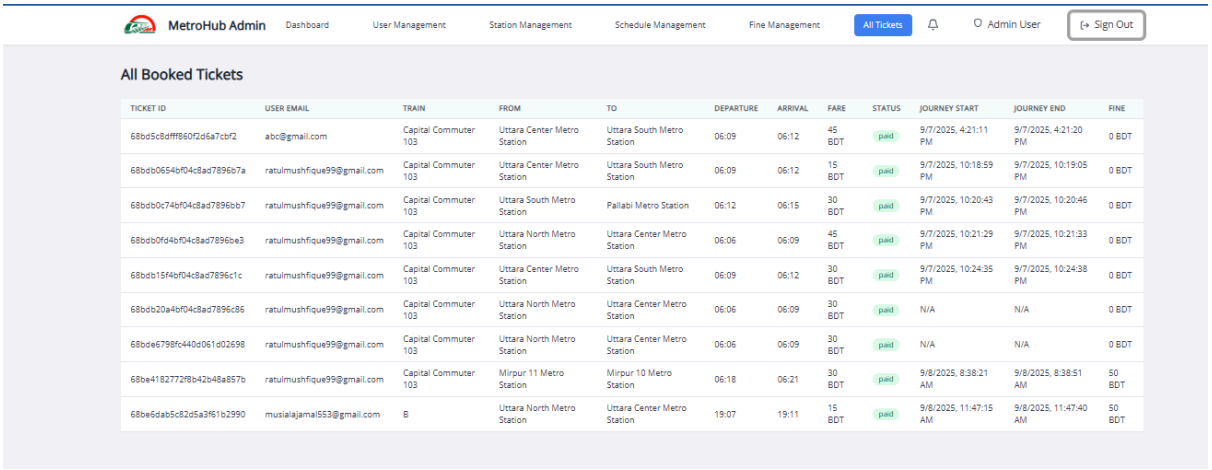
14. Admin managing the schedule:

Create New Schedule: Navigate to the "Create New Schedule" section. Enter the "Source Station," "Destination Station," "Train Name," "Frequency (minutes)," and "Fare," then click "Create Schedule" to add a new schedule.



15. Admin can see all booked tickets:

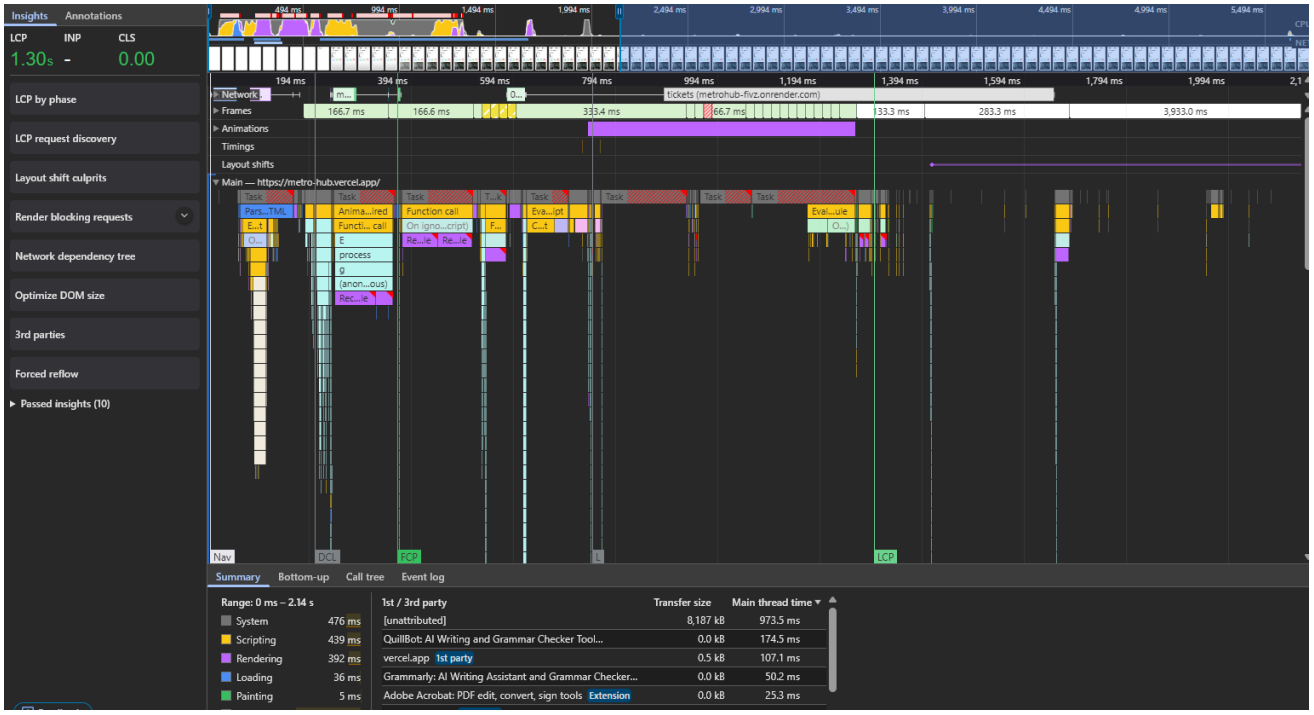
Navigate to the "All Booked Tickets" section, which displays a table with columns for Ticket ID, User Email, Train, From, To, Departure, Arrival, Fare, Status, Journey Start, Journey End, and Fine.



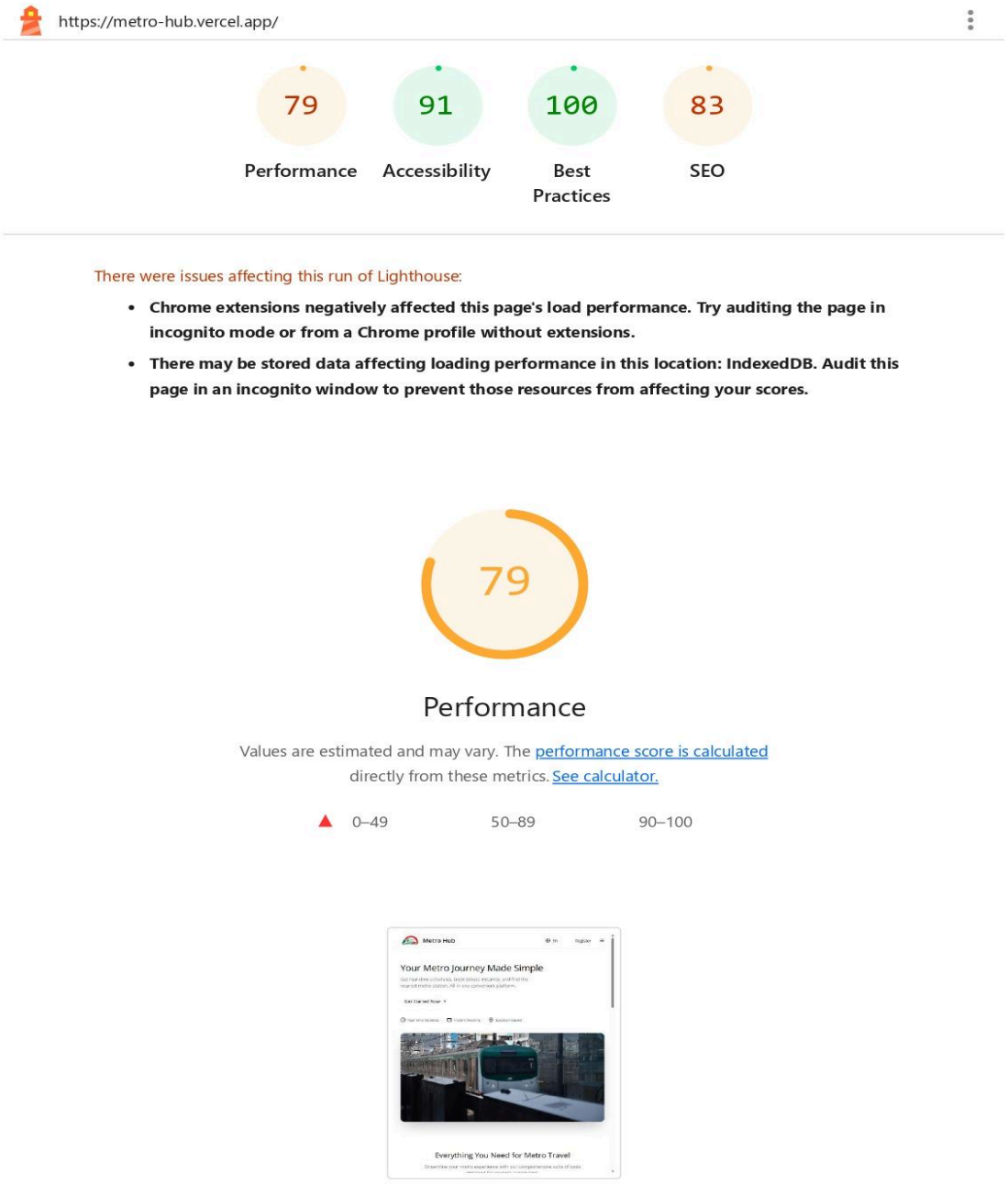
8. Performance and Network Analysis

Generate Performance and Network analysis report using lighthouse DevTool. Add screenshots of the report. Also, attach a screenshot of your system in any mobile viewport. **The system UI must be responsive.**

Performance:



Lighthouse:


https://metro-hub.vercel.app/
79
Performance
91
Accessibility
100
Best Practices
83
SEO
There were issues affecting this run of Lighthouse:
Chrome extensions negatively affected this page's load performance. Try auditing the page in incognito mode or from a Chrome profile without extensions.
There may be stored data affecting loading performance in this location: IndexedDB. Audit this page in an incognito window to prevent those resources from affecting your scores.
79
Performance
Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.
0–49
50–89
90–100
Metro Hub
Your Metro Journey Made Simple
Everything You Need for Metro Travel

METRICS Expand view

First Contentful Paint
0.5 s

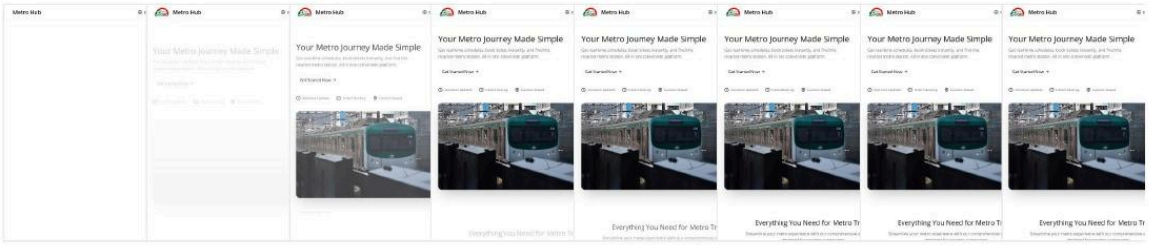
Largest Contentful Paint
1.3 s

▲ Total Blocking Time
390 ms

Cumulative Layout Shift
0.001

Speed Index
1.1 s

View Treemap



Later this year, insights will replace performance audits. Learn more and provide feedback here.

Try insights

Show audits relevant to: All FCP LCP TBT CLS

DIAGNOSTICS

- ▲ Reduce the impact of third-party code — Third-party code blocked the main thread for 370 ms
- ▲ Preconnect to required origins — Est savings of 290 ms
- ▲ Serve images in next-gen formats — Est savings of 6,716 KiB
- ▲ Properly size images — Est savings of 7,277 KiB
- ▲ Reduce unused JavaScript — Est savings of 6,104 KiB
- ▲ Largest Contentful Paint element — 1,250 ms
- Preload Largest Contentful Paint image

- Image elements do not have explicit `width` and `height`
- Minify JavaScript — Est savings of 45 KiB
- Avoid serving legacy JavaScript to modern browsers — Est savings of 30 KiB
- Avoid enormous network payloads — Total size was 8,178 KiB
- Avoid long main-thread tasks — 4 long tasks found
- Avoid large layout shifts — 1 layout shift found
- User Timing marks and measures — 2 user timings
- Avoid chaining critical requests — 2 chains found

More information about the performance of your application. These numbers don't directly affect the Performance score.

PASSED AUDITS (23) Show

Accessibility

These checks highlight opportunities to improve the accessibility of your web app. Automatic detection can only detect a subset of issues and does not guarantee the accessibility of your web app, so manual testing is also encouraged.

NAMES AND LABELS

- Buttons do not have an accessible name
- Links do not have a discernible name

These are opportunities to improve the semantics of the controls in your application. This may enhance the experience for users of assistive technology, like a screen reader.

ADDITIONAL ITEMS TO MANUALLY CHECK (10) Show

These items address areas which an automated testing tool cannot cover. Learn more in our guide on conducting an accessibility review.

PASSED AUDITS (24) Show

NOT APPLICABLE (31) Show

100

Best Practices

TRUST AND SAFETY

- Ensure CSP is effective against XSS attacks
- Ensure proper origin isolation with COOP
- Mitigate clickjacking with XFO or CSP

GENERAL

- Missing source maps for large first-party JavaScript

PASSED AUDITS (13) Show

NOT APPLICABLE (4) Show

83

SEO

These checks ensure that your page is following basic search engine optimization advice. There are many additional factors Lighthouse does not score here that may affect your search ranking, including performance on Core Web Vitals. Learn more about Google Search Essentials.

CONTENT BEST PRACTICES

Document does not have a meta description

Format your HTML in a way that enables crawlers to better understand your app's content.

CRAWLING AND INDEXING

robots.txt is not valid — 14 errors found

To appear in search results, crawlers need access to your app.

ADDITIONAL ITEMS TO MANUALLY CHECK (1) Show

Run these additional validators on your site to check additional SEO best practices.

PASSED AUDITS (7) Show

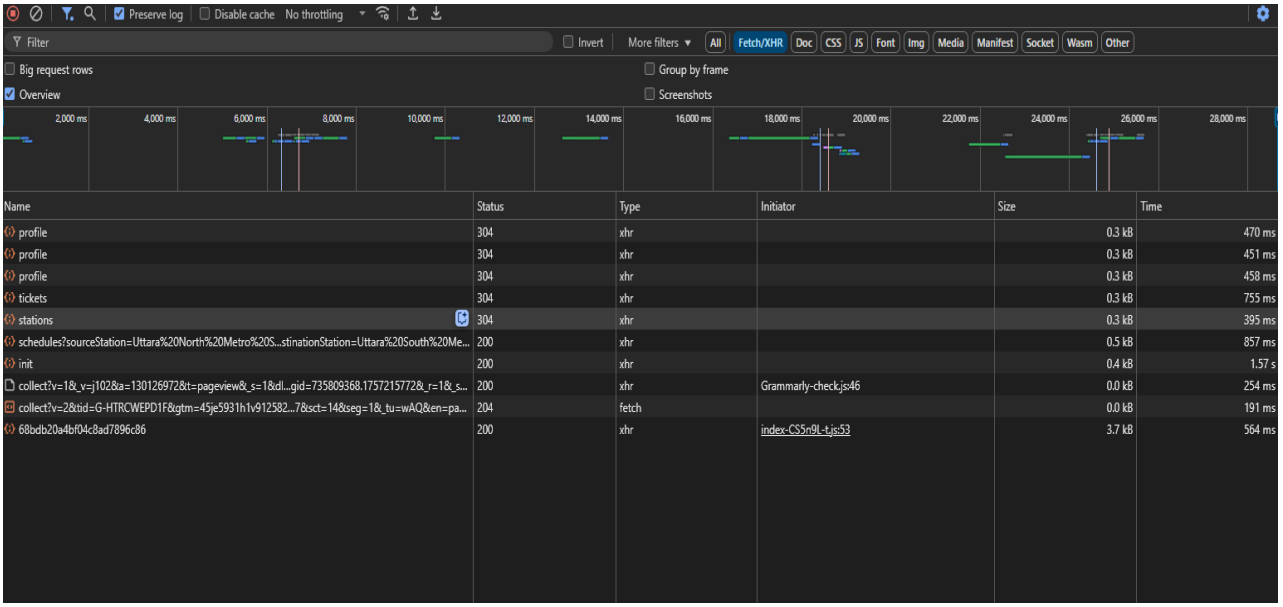
NOT APPLICABLE (1) Show

Captured at Sep 7, 2025, 9:58 PM GMT+6
Initial page load
Emulated Desktop with Lighthouse 12.6.1
Custom throttling
Single page session
Using Chromium 139.0.0.0 with devtools

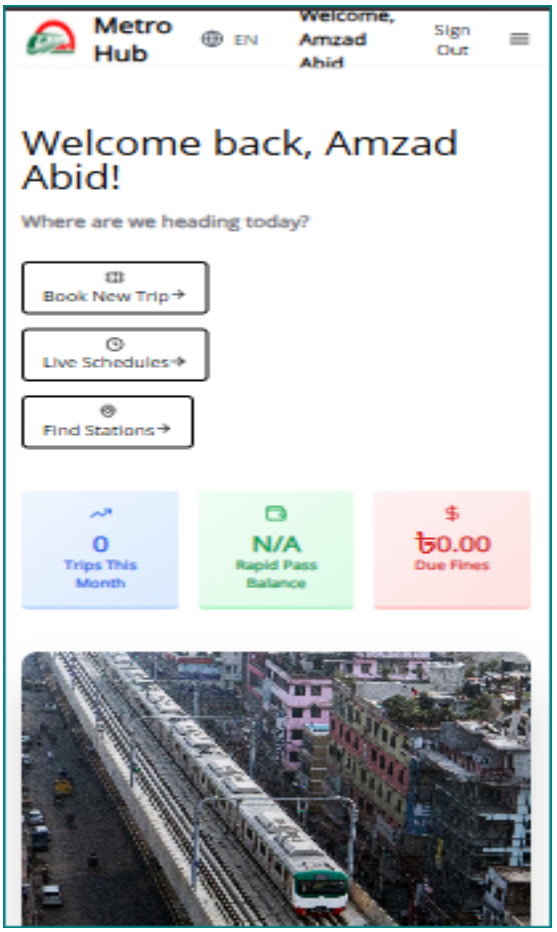
Generated by **Lighthouse** 12.6.1 | File an issue

47

NetworkAnalysis:



System UI responsiveness:



9. Github Repo [Public] Link:

[github repo]

10. Link of Deployed Project:

[Metro-Hub]

11. Individual Contribution

Group member - 01	
Name: Md. Ratul Mushfique	Student ID: 21301418
Functional Requirements which are developed by this member:	
1. Module 1: Users can register and log in using email/mobile phone and password. There is a security question section for those who have forgotten their password.	

.
2. Module 2: Google Maps API integration helps users visualize nearby metro stations and walking directions.
3. Module 3: Users can book metro tickets with online payment methods (SSLCOMMERZ gateway or rapid pass option), download QR code for the ticket, scan QR code (simulation of entering and exiting metro station) and view booking history from their dashboard. Admins can also view tickets that are booked.
4. Module 3: Users receive gmail confirmation for booked rides using Nodemail

Group member - 02	
Name: Sadman Safiur Rahman	Student ID: 21301411
Functional Requirements which are developed by this member:	
1. Module 1: Admins can manage user roles and permissions (e.g., regular user, rapid pass user, remove user etc).	
2. Module 2: Users can search for metro schedules by metro station, and time, and view real-time train updates done by Admin.	
3. Module 2: Users can view various options, sections and ticket fares in multiple languages (e.g., English and Bangla). Users can switch the app language with a toggle.	
4. Module 3: Users receive fines when they are in station after a certain period of time, added to the profile. If the QR code scanned (indicating exit from station) 2nd time before that time limit then journey has ended, no fine.	

Group member - 03	
Name: Kazi Amzad Abid	Student ID: 21301750
Functional Requirements which are developed by this member:	

1. Module 1: Users can update their profile including name, phone number, emails, passwords and preferred routes.
2. Module1 :Users can set themselves as rapid pass users and can also deposit money in the pass.
3. Module 2: Admins can create, update, or delete metro train schedules, station lists and station info
4. Module 3: For metro pass users, ticket costs are automatically deducted from their balance upon booking.

12. References

Choudhury, S. (2023). ***Build and deploy a full stack movie ticket booking app using MERN*** **[Video]. YouTube. https://www.youtube.com/watch?v=Pez37wmUaQM**

Kumar, A. (2023). ***Build a fullstack booking app using MERN*** **[Video]. YouTube. https://www.youtube.com/watch?v=MpQbwtSiZ7E**

Singh, R. (2022). ***MERN stack full tutorial & project | Complete all-in-one course*** **[Video]. YouTube. https://www.youtube.com/watch?v=CvCiNeLnZ00**