A Project Report On

TRAVEL WORLD

FIND, BOOK AND EXPLORE

Mini project submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY
IN
INFORMATION TECHNOLOGY
(2022-2026)
BY

BASHABOINA ANEESH	**22241A12D9**
BANDARI SWAMY	**22241A12D8**
MAHAMOOD MOINUDDIN	**23245A1220**

Under the Esteemed Guidance
of
Mrs.T.N.P MADHURI
Assistant Professor

DEPARTMENT OF INFORMATION TECHNOLOGY
GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY
(AUTONOMOUS)
HYDERABAD

2024-25

CERTIFICATE

This is to certify that it is a bonafide record of Mini Project work entitled**"TRAVEL WORLD – FIND, BOOK AND EXPLORE (USING MERN STACK)"** done by **BASHABOINA ANEESH(22241A12D9), BANDARI SWAMY (22241A12D8), MAHAMOOD MOINUDDIN (23245A1220)** of **B. Tech** in the Department of Information of Technology, **Gokaraju Rangaraju Institute of Engineering and Technology** during the period 2022-2026 in the partial fulfillment of the requirements for the award of degree of **BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY** from GRIET, Hyderabad.

Mrs.T.N.P.MADHURI
Assistant Professor
Internal Guide

Dr. Y. J . Nagendra Kumar
Head of the Department

Project External

ACKNOWLEDGEMENT

We take the immense pleasure in expressing gratitude to our Internal guide**,** **Mrs. T.N.P. MADHURI, Assistant Professor, Dept of IT**, GRIET. We express our sincere thanks for his encouragement, suggestions and support, which provided the impetus and paved the way for the successful completion of the project work.

We wish to express our gratitude to **Dr. Y J Nagendra Kumar,** HOD IT, our Project Coordinators **Mr. P.K. Abhilash, Mrs. T Nishitha** and **Mr. K Sandeep** for their constant support during the project.

We express our sincere thanks to **Dr. Jandhyala N Murthy,** Director, GRIET**,** and **Dr. J. Praveen,** Principal, GRIET**,** for providing us the conductive environment for carrying through our academic schedules and project with ease.

We also take this opportunity to convey our sincere thanks to the teaching and non-teaching staff of GRIET College, Hyderabad.

Email: aneeshbashaboina2004@gmail.com

Contact No: 8074294033

Email: swamybandari34@gmail.com

Contact No: 9177795648

Email: moinuddin.hnza@gmail.com

Contact No. 9959596333

DECLARATION

This is to certify that the mini-project entitled "**TRAVEL WORLD – FIND, BOOK AND EXPLORE (USING MERN STACK)"** is a bonafide work done by us in partial fulfillment of the requirements for the award of the degree **BACHELOR OF TECHNOLOGY IN INFORMATION TECHNOLOGY** from Gokaraju Rangaraju Institute of Engineering and Technology, Hyderabad.

We also declare that this project is a result of our own effort and has not been copied or imitated from any source. Citations from any websites, books and paper publications are mentioned in the Bibliography.

This work was not submitted earlier at any other University or Institute for the award of any degree.

BASHABOINA ANEESH 22241A12D9

BANDARI SWAMY 22241A12D8

MAHAMOOD MOINUDDIN 23245A1220

TABLE OF CONTENTS

11 LIST OF DIAGRAMS

Abstract

The fast expansion of digital technology has revolutionized the tourism sector, and online booking systems have become a necessity for convenient trip planning. The present paper introduces a Tour and Travel Booking App built with the MERN stack (MongoDB, Express.js, React, Node.js) to offer users a contemporary, streamlined, and user-friendly interface for searching and booking travel experiences.The app provides major features like destination exploration, package booking, user authentication, real-time availability status, and secure payment integration. MongoDB is a scalable database to hold user profiles, bookings, and travel listings. Express.js and Node.js manage the backend, allowing efficient data retrieval and secure transactions. The frontend, developed with React, provides a dynamic and interactive interface, maximizing user engagement.A strong authentication mechanism based on JWT secures user information, whereas third-party APIs provide real-time price quotes, weather forecasts, and location-based recommendations. The app has a responsive UI for device accessibility, ensuring an effortless experience.Security features such as data encryption and role-based access control provide secure transactions and safeguard user privacy. Admin features also enable travel agencies to administer listings, monitor bookings, and update tour packages effectively.This study points out the strengths of utilizing the MERN stack for a high-performance and scalable travel booking solution. Future improvements are AI-powered personalized suggestions, chatbots for real-time customer support, and blockchain-based secure payments. The project seeks to make travel planning easier, enhance customer experience, and provide a trustworthy digital solution for the tourism sector.

Keywords:

Travel booking, MongoDB, Express.js, React, Node.js, secure payments, real-time data, user authentication, tailored suggestions, flight & hotel reservations, responsive UI,interactive interface.

1. INTRODUCTION

1.1 Introduction to Project

Discover the World with TRAVEL WORLD – FIND, BOOK AND EXPLORE

At **TRAVEL WORLD**, we believe that travel should be simple, stress-free, and truly memorable. Whether you're dreaming of a peaceful gateway, an adventurous road trip, or an immersive cultural experience, we're here to make it happen. From booking your stay and arranging your transport to uncovering the best local eats and must-see events, we take care of every detail—so all you have to do is enjoy the journey.

Our Services

• **Hotel Reservations:** Choose from a huge range of hotels, from low-budget options to Seven -star luxury hotels. With special promotions and free change options, we guarantee your stay is comfortable, irrespective of the destination.

• **Train Bookings:** Travel made easy. Whether you're commuting for business or heading out for leisure, our seamless train booking service gets you on board in just a few clicks. Enjoy scenic journeys, reliable schedules, and a smooth travel experience.

• **Restaurant Reservations:** Enjoy the best local food by making your reservation at highly rated restaurants. From everyday restaurants to fine dining, we have a wide range of eating places to fit every palate and taste.

• **Tour Packages:** Explore the world with our carefully curated tour packages. Whether you prefer to visit legendary landmarks, take part in action-packed activities, or indulge in a cultural experience, our tours are tailored to suit your interests and tempo.

• **Event Bookings:** Miss out no more on the local celebrations, concerts, sports games, and more. Through TRAVEL WORLD, booking tickets for the most trending events across the globe is so simple, allowing you to connect with the local culture of your travel destination like never before.

• **Vehicle Packages:** Ride in style and comfort with our car rental services. From automobiles and SUVs to limousine vehicles and personal chauffeurs, we have a collection of vehicles ranging from normal to luxurious ones. Whether you want to tour a city or drive out on a road trip, we have the right vehicle for you.

Why Choose Us?

- **One-Stop Shop:** Book all you require for your trip—hotels, trains, restaurants, tours, events, and cars—without ever leaving our site.

- **Customized Experience:** We provide recommendations based on your tastes, so your trip is created with your needs and wants in mind.
- **Simple & Secure Booking:** With our simple interface and secure payment methods, organizing your ideal holiday has never been simpler.
- **24/7 Support:** Our customer service team is available at your beck and call, providing a seamless and hassle-free experience from beginning to end.

Travel World: Special Package Deals

i. Platinum Package

- **Luxury Experience:** Five-star accommodations, first-class flight, and private guided tours.
- **Exclusive Perks:** VIP lounge access, personalized concierge services, and fine dining reservations.
- **Extended Duration:** Up to 15 days across multiple destinations with customized itineraries.
- **Best suited for:** Travelers in search of an upscale, hassle-free vacation.

ii. Gold Package

- **Comfort & Convenience:** Four-star stay and priority flight upgrades.
- **Guided Adventures:** Expert-guided tours and interactive cultural encounters.
- **Flexible Duration:** 10-day journeys with flexible itineraries.
- **Best suited for:** Leisure-seekers who want comfort along with adventure.

iii. Silver Package

- **Affordable & Enjoyable:** Economical stays with standard amenities.
- **Group Tours:** Shared trips and transportation for an economical outing.
- **Compact Duration:** 7-day trips with must-visit sightseeing points.
- **Suitable for:** Budget-conscious adventurers and explorers.

Each of the packages is specifically designed to cater to varying interests, offering a hassle-free and memorable experience for every traveler.

Project Overview & Core Functionality

The Tours and Travel World application enables users to explore, book, and manage travel packages seamlessly. Built using the MERN stack, it ensures a dynamic, responsive, and scalable experience.

i. Technology Stack

- **MongoDB:** NoSQL database for storing package details, user bookings, and payment records.
- **Express.js:** Backend framework to handle API requests and authentication.
- **React.js:** Frontend library for an intuitive and interactive user interface.
- **Node.js:** Runtime environment for executing server-side logic.

FIG 1

ii. Key Features

- **User Authentication:** Secure login/signup with JWT or Firebase authentication.
- **Travel Package Listings:** Display packages with images, descriptions, and pricing.
- **Booking System:** Customers are able to book packages with specified dates and choice.
- **Payment Integration:** Stripe/Razorpay for secure transactions.
- **Admin Dashboard:** Handle packages, reservations, and analytics.

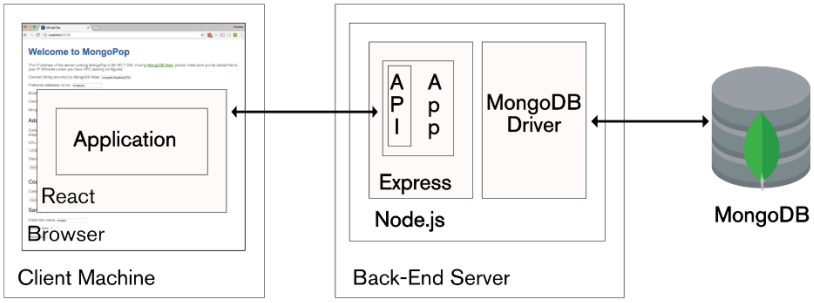
iii. API Structure

- **/api/users** → Control user authentication and profile information.
- **/api/packages** → CRUD actions for travel packages.
- **/api/bookings** → Process bookings and user itinerary.
- **/api/payments** → Handle secure transactions.

[FIG-2] API

iv. Application Architecture

- **Frontend (React)** → Uses React Router for navigation and Context API/Redux for state management.
- **Backend (Node & Express)** → Responsible for API logic, authentication, and database interaction.
- **Database (MongoDB)** → Saves user profiles, packages, and transactions.



[FIG-3] Application Architecture

v. Security Measures

- **Authentication:** JWT-based token validation to secure user sessions.
- **Data Encryption:** Secure storage of sensitive data such as passwords.
- **Error Handling:** Global middleware for catching and logging API errors.

vi. Deployment Strategy

- **Frontend Hosting:** Netlify or Vercel for performance optimization.
- **Backend Hosting:** AWS, DigitalOcean, or Render for scalable APIs.
- **Database:** MongoDB Atlas for cloud-based storage.
- **Containerization:** Docker for easy deployment and version control.

This framework gives a precise guide to creating and deploying a complete working MERN-based Tours and Travel application

1.2 Existing System

i. User Interface (Front-End)

- **Homepage:** Showcases top destinations, deals, and quick search.
- **Search & Filter:** Allow users to search by destination, budget, dates, and tour type.
- **Tour Listings:** Show packages with images, prices, duration, and brief descriptions.

- **Tour Details Page:** Complete itinerary, inclusions/exclusions, images, maps, and reviews.
- **Booking System:** Real-time booking form with calendar and traveler number.
- **User Account:** Login/signup, view previous bookings, save favorites.

ii. Admin Panel (Back-End)

- **Tour Management:** Edit/add/delete tour packages, prices, schedules.
- **Booking Management:** View, confirm, or cancel bookings; send notifications.
- **User Management:** Customer profiles and access levels management.
- **Content Management:** Homepage banners, blogs, FAQs management.
- **Reports:** Sales reports, customer activity, most booked tours.
- **Payments:** Integrated with gateways such as PayPal, Stripe for secure payments.

1.3 Proposed System

i. Core Features

- **Online Booking:** Users can book Trains, hotels, and tour packages.
- **Itinerary Management:** Automated trip planning with schedules and recommendations.
- **Payment Integration:** Secure transactions via credit and debit cards, digital wallets, and bank transfers.
- **Customer Support:** Chatbots and live assistance for queries and troubleshooting.

ii. Technologies Used

- **AI & Automation:** Chatbots for customer queries and recommendation engines.
- **Data Analytics**: Insights into customer preferences and travel trends.

iii. Technical Features

- **Security:** encryption, secure login or sign in, data protection.
- **SEO-Ready:** Optimized URLs, meta tags, quick loading, and image alt tags.
- **Multilingual/Multi-currency:** Optional support for worldwide users.

iv. Additional Features

- **Blog/Travel Tips Section:** Drives SEO and user interest.
- **Contact/Support:** Live chat, help center, contact form.
- **Reviews and Ratings:** Travelers can rate and comment on tours.

2. REQUIREMENT ENGINEERING

2.1 Hardware Requirements

- Processor – i5 and above (64-bit OS).
- Memory – 4GB RAM (8GB or above is recommended for better performance)
- Input devices – Keyboard, Mouse, Webcam

2.2 Software Requirements

i. Frontend (React.js)

- **react-router-dom** – For client-side routing between pages (Home, Tours, Booking, etc.)
- **axios** – For sending HTTP requests to the backend API
- **react-redux or @tanstack/react-query *(optional)*** – For managing global state or server-side data caching
- **tailwindcss** – Utility-first CSS framework for responsive design
- **react-hook-form** – better form handling
- **jwt-decode** – To decode JWT tokens and check user roles/expiry
- **react-cookie or js-cookie** – To store tokens securely in cookie.

ii. Backend(Node.js + Express.js) and Database

- **express** – Fast, unopinionated web framework for Node.js
- **cors** – Enables cross-origin requests (important for React frontend)
- **dotenv** – Loads environment variables from .env file
- **mongoose** – ODM (Object Data Modeling) library for MongoDB
- **jsonwebtoken** – For generating and verifying JWTs
- **multer** – Handles multipart/form-data (for tour image uploads)
- **cloudinary** – If using Cloudinary for image hosting

3. LITERATURE SURVEY

- **Currently Vrishank Dua , Aashray Agarwal , Tejna Khosla , K.C. Tripathi , M.L. Sharma in proposed Tours and Travels Booking System** .Over the years, digital innovation has reshaped the travel and tourism industry, moving it away from the traditional reliance on travel agents. Many earlier studies have highlighted the growing importance of platforms that focus on user needs, deliver real-time information, and offer intuitive, responsive interfaces. The emergence of online travel services marked a major shift toward automation in trip planning. Researchers have also pointed out the drawbacks of manual systems, such as delays and limited customization. To overcome these, the MERN stack—comprising MongoDB, Express.js, React, and Node.js—presents a powerful framework for building modern, integrated travel solutions. The use of secure login methods and smooth data handling has also been widely recognized as essential for enhancing trust and performance. Building on these insights, this study explores a MERN-based travel platform designed to deliver a more personalized and streamlined user experience

- **Tourism website using MERN stack and Augmented Reality developed by Dr. D. Thamaraiselvi, Pydikalva Srikanth, Ram Charan Tej V**. Their approach to transforming the travel and tourism space centers on using modern digital technologies to deliver smarter, more inclusive, and user-friendly services. Older systems were often dependent on manual processes and lacked the ability to update or respond in real time, which limited their effectiveness. In contrast, current travel platforms are built around responsive design, strong data protection, and smooth user navigation. The MERN stack—MongoDB, Express.js, React, and Node.js—has become a go-to solution for developing dynamic, scalable web applications. Researchers also highlight the growing need for secure logins, support for multiple languages, and features that make platforms accessible to users with disabilities. Taking inspiration from these advancements, this project introduces an interactive travel app that uses augmented reality (AR) to enhance user engagement and deliver a richer travel planning experience.

- **A Online Travel Portal Web Application Using MERN Stack proposed by Ms. Geetanjali , Siddh Kumar Singh , Shubham Chouhan, Kunal Thakur, Kajal Solanki.** Their approach is centered on leveraging cutting-edge technology to transform the travel industry, emphasizing personalization, automation, and accessibility. The integration of innovations such as metasearch engines, dynamic pricing algorithms, voice assistants, and blockchain has significantly enhanced the efficiency and transparency of travel services. Social media and user-generated content are increasingly shaping traveler decisions, while the rise of remote work has fueled the growth of flexible travel options. Additionally, the adoption of the MERN stack provides a robust framework for creating scalable, interactive travel platforms.

4. TECHNOLOGY

4.1 ABOUT MERN STACK

The MERN Stack is applied as a JavaScript(JS) technology to develop full- stack web applications. It has MongoDB as a data-storage engine; Express.js as a backend responsible for routing and API, React.js as a frontend where we develop dynamic & interactive UI; and Node.js as a server-side JavaScript environment. By the way of using JavaScript during the whole development process, MERN just makes it easier to develop efficient and cohesive applications. It is known for its scalability, flexibility, and speed – which makes it especially appropriate for interactive platforms such as travel apps, online stores and social sites.

4.1.1 MongoDB – No SQL

Database MongoDB is a categorized no sql database. It is a source-available cross-platform document-oriented database program. MongoDB relies on JSON-like documents referred to as BSON (binary JSON) with optional schemas. MongoDB is a flexible, non schema database in which developers are not required to outline the data structure prior to its use.

Key Features:

- It is a non-relational database, hence, it does not operate the regular tables and rows.
- Much is said about its scalability and its speed, which make it preferable for huge volumes of data
- Smoothly integrates with JavaScript and Node. js to fit modern web stacks such as MERN.
- Data is stored in a form of collections and documents like JSON making development easy.
- Supports replication and sharding, enabling distributed data storage and increased performance in application.

4.1.2 Express.js (Backend Framework)

Express.js a web application backend framework to create RESTful API's using Node.js. It makes it easier when developing APIs and when working with HTTP requests.

Key Features:

- Rules-based simple routing to control URL endpoints.
- Middleware to deal with requests, authentication, logging, etc…
- Built-in support for RESTful APIs.Lightweight and fast.

4.1.3 React.js (Frontend UI Library)

React.js is a frontend web application library used in building a UI based on the components. React can be used to develop single-page applications, mobile or server side applications. React makes performance better as only changing UI items are updated when using the virtual DOM without reloading the whole page.

Key Features:

- Virtual DOM for high and good performance.
- Component-based application .
- One way data binding that supports predictable data flow • Rich ecosystem or growths (Redux, React Router, Hooks).
- Active in RESTapi's or GraphQL.Node,js (JS Runtime).

4.1.4 Node.js

Node.js is a run time which can run as Windows, Linux, Unix, Mac OS and so on. Command Line Tools and server side scripting in JS are the tools used by node js developers. After it has been served in the server, with JavaScript functioning, dynamic web contents can be created long before being sent to the user's browser.

Key Features:

- Non-blocking, event-driven architecture.
- Fast execution using V8 engine.
- Works well for network applications on a scalable scale.… built-in modules (fs, http , path etc.).
- Usage of npm (Node Package Manager) as package management. Node.js is used in the development of the logic of the backend, as well as connecting to databases, serving content to the frontend.

[FIG-4] MERN STACK

4.2 Use Cases of MERN Stack

- Travel and tourism portals
- E-commerce websites
- Social networking sites
- Blogging platforms
- Real-time collaboration tools (chat apps, Trello clones)
- Content management systems (CMS)
- Online booking systems

4.3 Advantages of using MERN stack

- **Single Language Development:** Since JavaScript is used across the entire stack—from database interactions to backend logic and frontend interfaces—developers can build the whole application using one language, simplifying collaboration and maintenance.
- **Speed and Performance:** Node.js handles multiple requests simultaneously with its non-blocking, asynchronous architecture, while React uses a virtual DOM to quickly update only the necessary parts of the UI, boosting application responsiveness.
- **Component Reusability:** React promotes modular design through reusable components, which enhances code organization, maintainability, and reduces development time.
- **Flexible Data Modeling:** Being schema-less, MongoDB enables developers to manipulate dynamic and changing data structures, which is very convenient for agile development environment.
- **Rich Ecosystem**: The MERN stack has plenty of tools, plugins and frameworks library that promote faster development and easier integration of features.
- **Community Support:** Every element of the stack, from MongoDB to React to Express to Node.js, has an active large community that constantly updates the stack, pulls tutorials, and posts solutions to problems making it easy to learn and troubleshoot.

[FIG-5] Benefits of MERN Stack

5.DESIGN REQUIREMENT ENGINEERING

Understanding the Concept of UML

UML(Unified Modeling Language) is standardized visual language used for representation and designing software system. It serves the important function of aiding teams communicate ideas effectively by providing a set of diagrams and symbols which show some of the components of a system and how they interact and connect to each other. Whether it's the overall architecture or pieces of functionality, UML is a standardized way to visualize and document any aspect of a software project. UML is particularly useful on complex systems, as it acts as a communication tool between developers, analysts and stakeholders of a complex system. UML facilitates easier planning, organization, and management of the software development process through the support of object-oriented concepts and stimulation of good software engineering practices. In short it's not just a drawing tool at all but a common language that helps to cross the distance between technical teams and decision makers.

5.1 - UML DIAGRAMS:

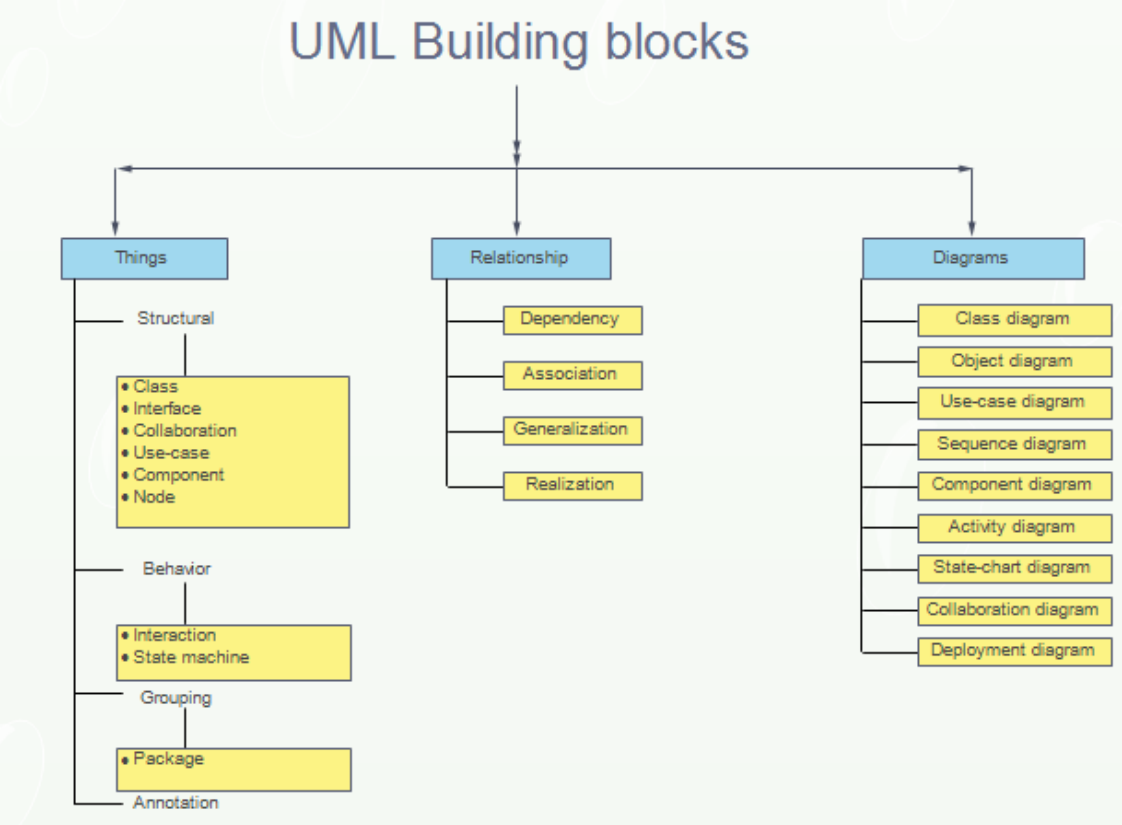
UML (Unified Modeling Language) is a flexible, platform-independent modeling language that can be used across various programming languages and development methodologies. Because it's a widely accepted standard, most developers are at least somewhat familiar with it. While not every engineer loves working with diagrams, UML can be particularly valuable in Agile environments, where it adds clarity without slowing down the pace of work.

Rather than viewing UML diagrams as optional or decorative, it's helpful to see them as part of meaningful project documentation. They offer real value by making complex systems easier to understand and communicate.

Here are a few practical ways UML diagrams support development teams:

- Helping a new team members by giving them a visual overview of how the system is structured.
- Making it easier to interpret or navigate or point the existing source code, especially in large projects.
- Assisting in planning and visualizing new features before diving into the actual coding.
- Bridging the gap between technical and non-technical stakeholders by presenting system interactions in a clear, accessible format.
- Supporting better collaboration between cross-functional teams by aligning everyone around a shared visual understanding of the system.

Diagrams in UML can be broadly classified…



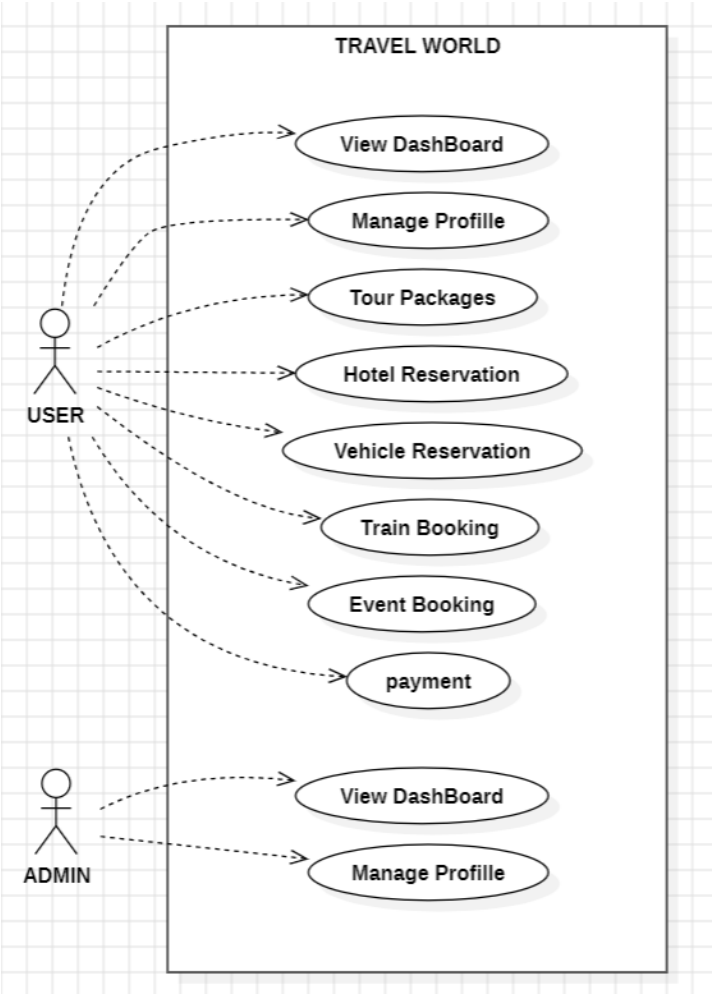
[FIG-6] UML DIAGRAM

5.2 Use Case Diagram

A Use Case is a visual diagram or description of an actual type of interaction with a particular system and a user (or another system) in place of system being designed. It describes the way the system acts when it answers a request made by the user or another system and focuses on the system's function and the purpose of the user**.**

Some of the Use Case Key aspects are:

1. **Actor:** An actor suggests a role that has interaction with the system. This can be either a human user (e.g/client, Admin) , another system e.g. Payment Gateway, or outside entity that interacts with the system.
2. **Use Case**: This is the main work that the system does in answer to an actor's request. Examples: "Book Hotel", "View Profile", "Reserve Restaurant".



[FIG-7] Use Case Diagram

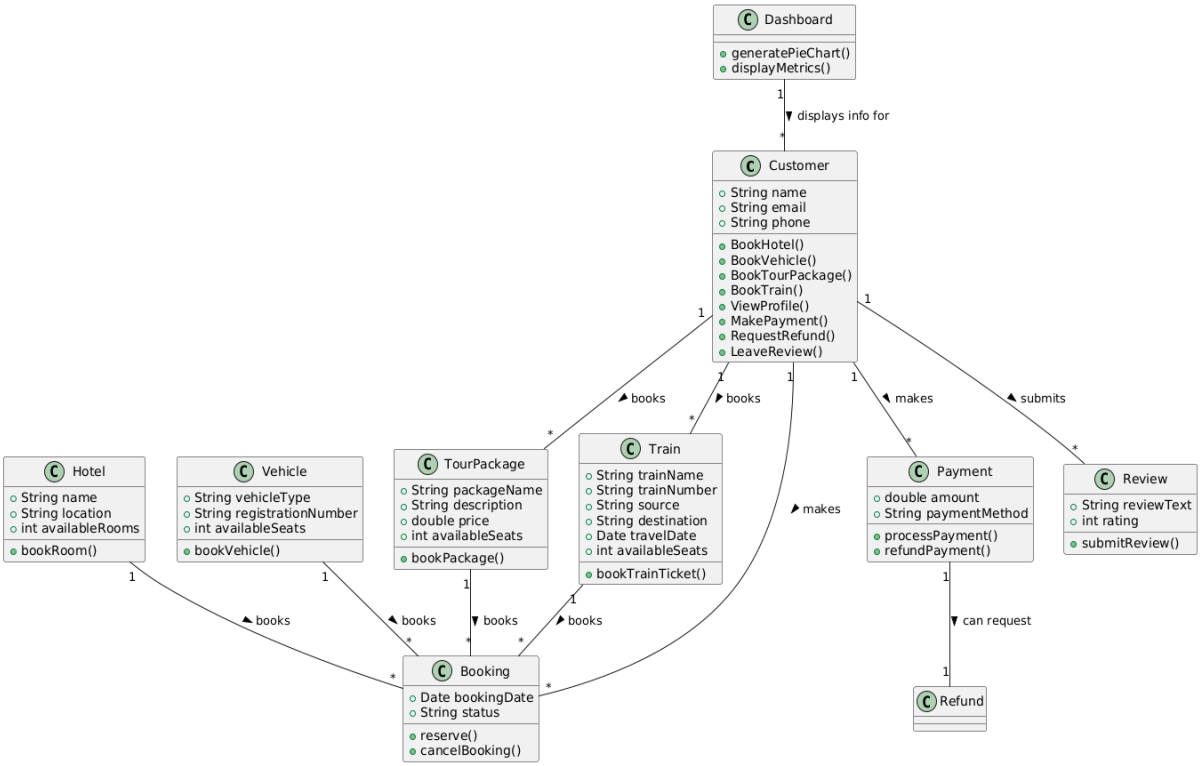
5.3 Class Diagram

A Class Diagram is an important part of object-oriented design as it visually organizes the structure of a system. It shows the classes, their members, methods, and the degrees of relationships. In a travel management system there may be critical classes such as:

- **User:** This class has critical user details like names, email and phone number. It provides opportunities for actions, such as BookHotel(), BookVehicle(), and LeaveReview(), and interfaces with various booking services such as Hotel, Vehicle, Tour, and Train
- **Booking:** This class manages user bookings, and it has booking date and status properties. Every booking is linked to a particular service, such as Hotel, Vehicle, Tour, Train etc.
- **Hotel, Vehicle, Tour, Train**: These are the services that are bookable. The properties of price, availability and location are found in each class. Book services such as bookRoom(), bookVehicle(), and bookTourPackage() allow customers to book.
- **Payment:** The Payment class handles transaction associated with bookings (amount and paymentMethod among others). It is associated with the Booking class to make payment for the services reserved.

- **Review:** This class enables customers to give feedback on the services they have received. It has attributes such as reviewText and rating that represent the reviews of customers left.
- **Dashboard:** This class provides a summary view of the customer's activity, revealing information about bookings, payments and reviews. It can create visual insights, e.g. pie charts and other metrics, which give a rapid overview of the customer's activity**.**

These classes interact in a way that represents the core functions of the travel management system. Customers book services, make payments, leave reviews, and monitor their activities through a dashboard. Each class offers specific actions that reflect the primary operations of booking, paying, and reviewing within the system.



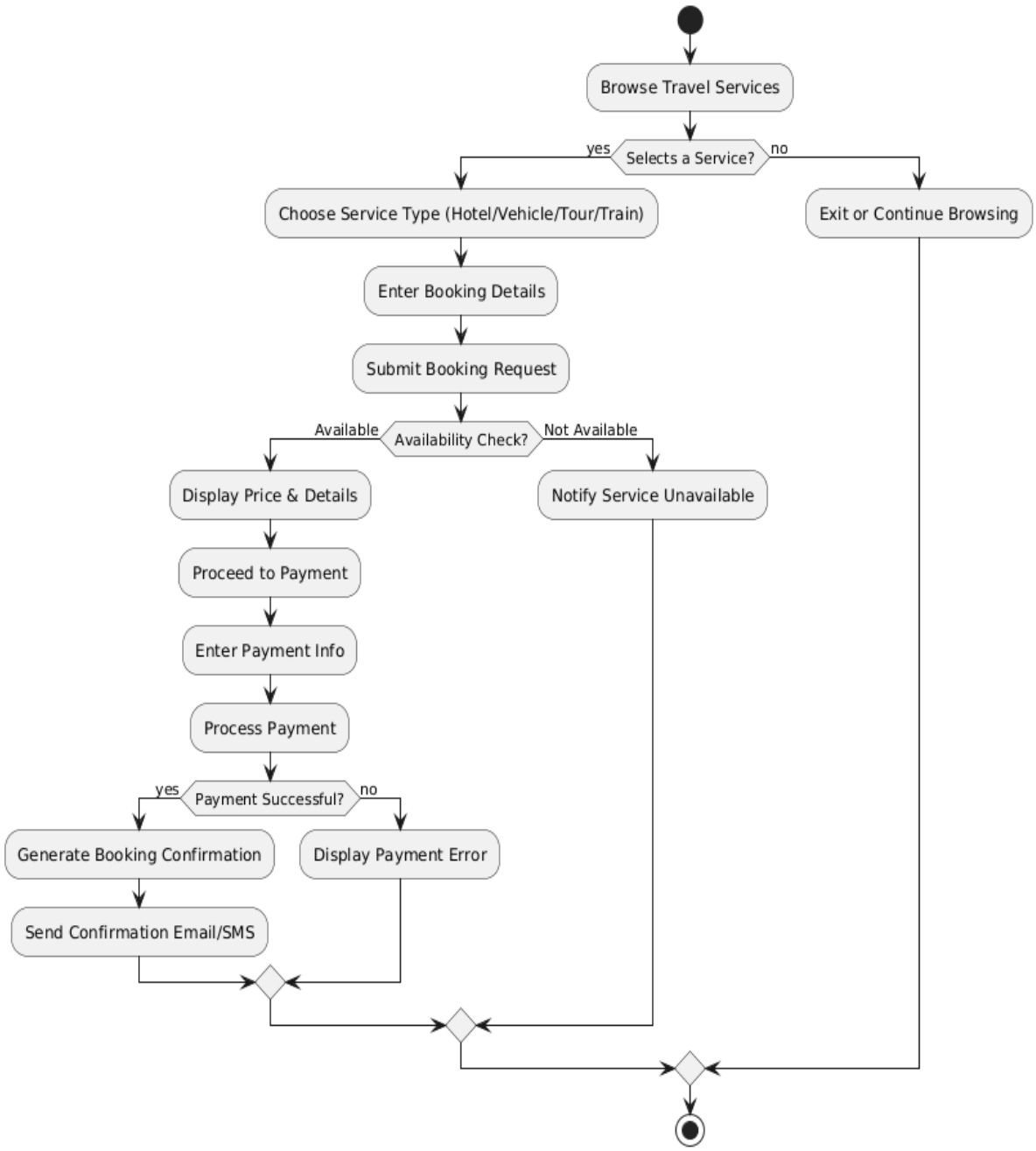
[FIG-8] CLASS DIAGRAM

5.4 Activity Diagram

An Activity Diagram is a visual tool used to illustrate the flow of actions or steps within a system. It outlines the process logic, including sequences, decisions, and parallel actions, making it easier to understand how a user interacts with different features.

For the **Travel World**, the activity diagram will map the process the work flow step by step such as:

- Starting the system (initial node)
- Log in or sign up
- Browsing the services (hotels, tours, vehicles, etc.)
- Booking a packages and services
- Making payment
- Receiving confirmation
- Optionally, submitting a review or viewing the dashboard
- Sign out



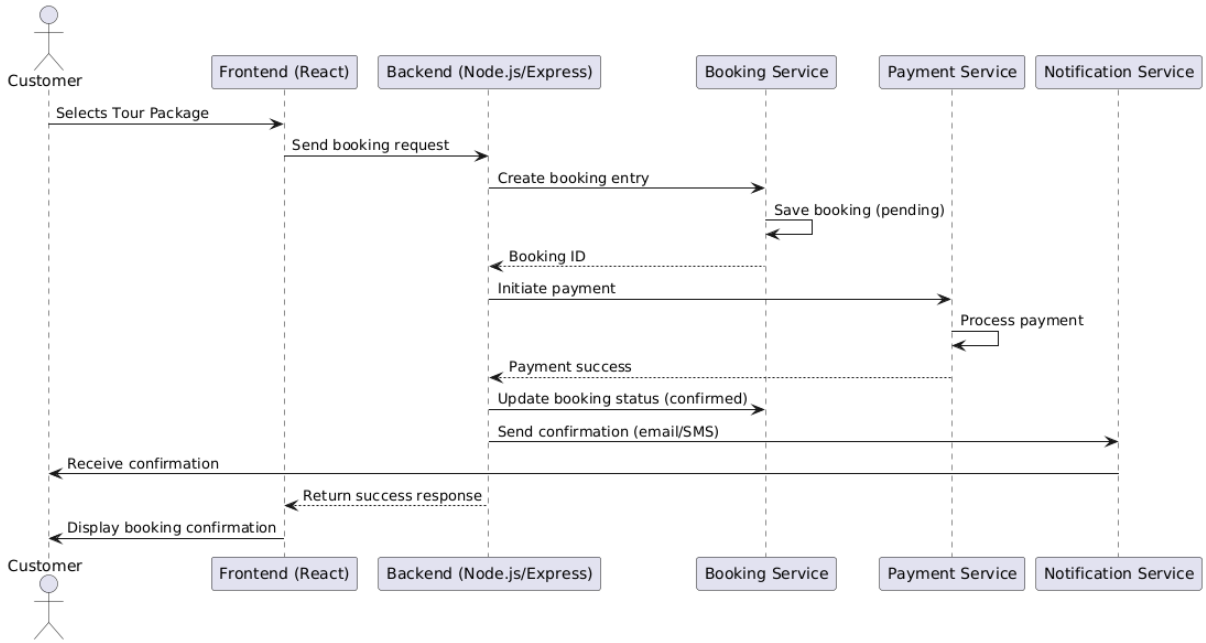
[FIG-9] Activity Diagram

5.5 Sequence Diagram

Another of the UML diagrams of collection is the collection sequence diagrams also frequently known as the occurrence diagrams that present the sequential flow of your gadgets. This includes the very threads of your character pieces and the schemes and dialogues of characters with your gizmos along with the conversations amongst these gizmos to fulfil a particular purpose**.**

Actors & Components:

- User.
- Travel Portal (Frontend).
- Booking Service.
- Payment Gateway.
- Confirmation/notification Service.

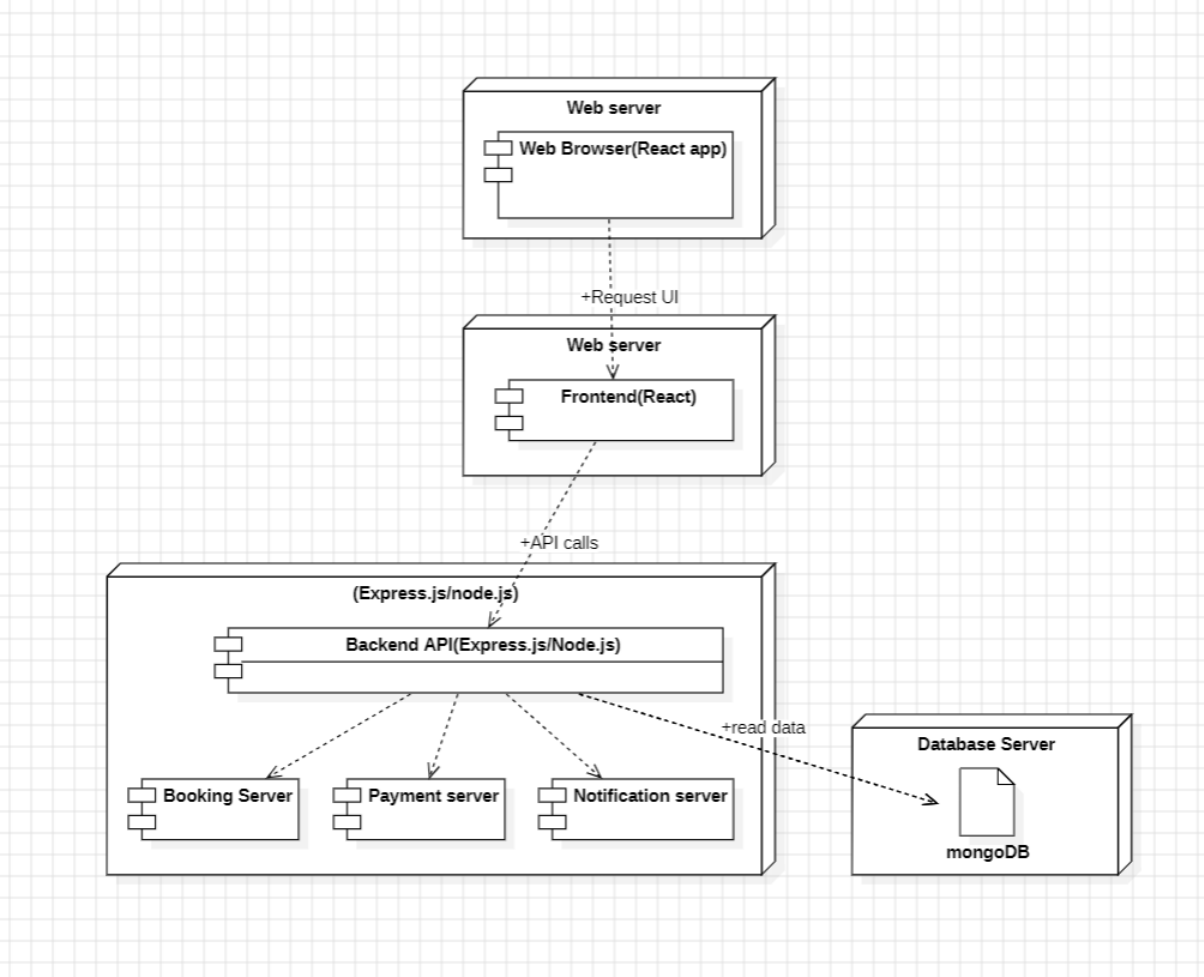


[FIG-10] Sequence Diagram

5.6 Deployment

A Deployment Diagram gives a visual view of how the software components are actually distributed on hardware in a system. It presents the way nodes—servers, client machines, or mobile devices—and the software artifacts (databases, web apps, or APIs) placed thereupon, were situated. Most straightforward, deployment diagram depicts physical connections between different sections of a distributed system that facilitates identification of how modules communicate through networks or devices. These diagrams are particularly useful if for designing a software which needs to be executed smoothly across several physical platforms

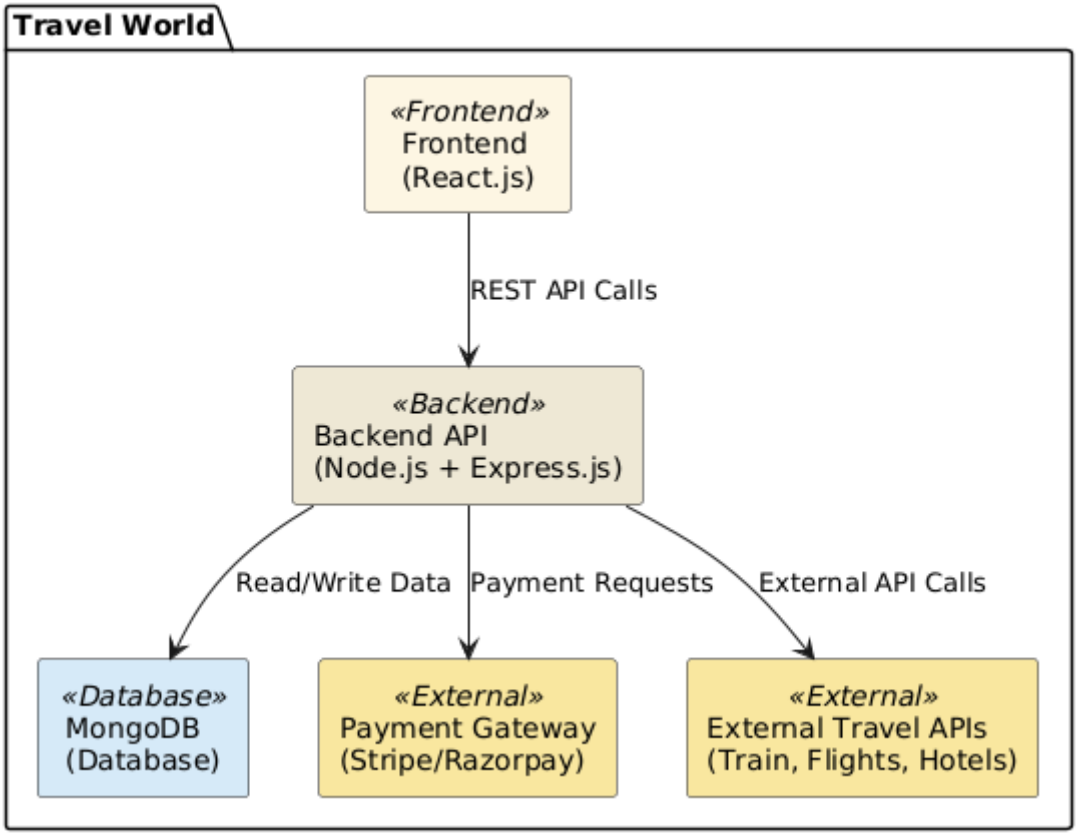
and therefore each portion of application needs to be appropriately mapped and needs to interact well with other parts of application.



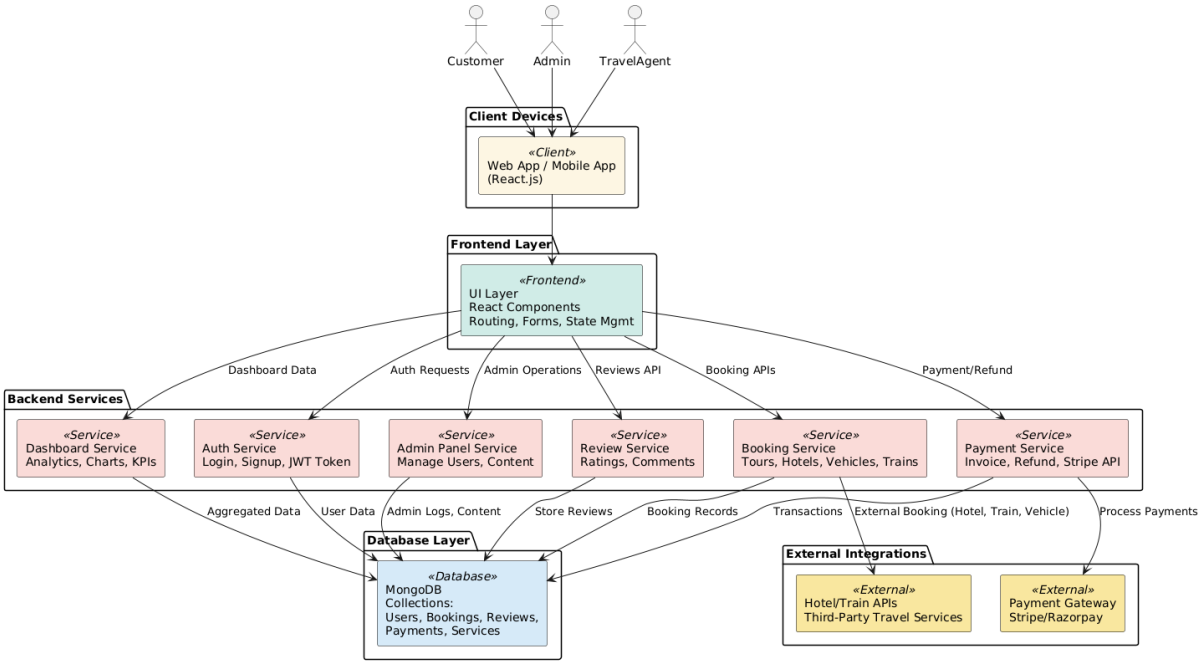
[FIG – 11] Deployment Diagram

5.7 Architecture

The TravelWorld architecture developed with the help of the MERN stack is divided by multiple layers: Client, frontend ,backend, database and external services. The platform is accessible by users, who include customers, admins and travel agents, through a web or mobile interface built out of React.js. The frontend uses the Node.js several backend services that serve for the various functions such as authentication, booking, payment, reviews, and dashboard. MongoDB serves as the primary database where all critical information i.e user details, bookings, and transactions is stored in. The system also has a connection to external services such as payment gateways (e.g. Stripe) and travel related APIs for real time capabilities.



[FIG-12] Basic Architecture



[FIG-13] Travel World Architecture

6. Implementation

6.1 Backend

6.1.1 Config

6.1.2 Controllers

i. AuthControllers.js:

authController.js

Travely > backend > controllers > authController.js > ...

```
const User = require("../models/userModel");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const { createError } = require("../middleware/error");
const nodemailer = require("nodemailer");
// const cloudinary = require("..//utils/cloudinary.js");  // Add this import
const multer = require("multer");
const cloudinary = require("cloudinary").v2;
const streamifier = require("streamifier");

const generateToken = (payload) => {
  const token = jwt.sign(payload, "secretKey", { expiresIn: "1h" });
  return token;
};

const verifyToken = (token) => {
  try {
    const decoded = jwt.verify(token, "secretKey");
    return decoded;
  } catch (err) {
    throw new Error("Invalid token");
  }
};

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

const registerUser = async (req, res, next) => {
  try {
    const { name, email, country, mobile, password, type } = req.body;
    let imgUrl = "";

    if (req.file) {
      const streamUpload = (req) => {
```

authController.js

Travely > backend > controllers > authController.js > ...

```
const registerUser = async (req, res, next) => {
    if (req.file) {
      const streamUpload = (req) => {
        return new Promise((resolve, reject) => {
          const stream = cloudinary.uploader.upload_stream(
            { folder: "user_images" },
            (error, result) => {
              if (result) resolve(result);
              else reject(error);
            }
          );
          streamifier.createReadStream(req.file.buffer).pipe(stream);
        });
      };

      const result = await streamUpload(req);
      imgUrl = result.secure_url;
    }

    const salt = await bcrypt.genSalt(10);
    const hash = await bcrypt.hash(password, salt);

    const newUser = new User({
      name,
      email,
      country,
      mobile,
      password: hash,
      type,
      img: imgUrl,
    });

    await newUser.save();
    res.status(200).send("User created successfully");
  } catch (error) {
    next(error);
  }
```

authController.js

Travely > backend > controllers > authController.js > ...

```
78  const loginUser = async (req, res, next) => {
79    try {
80      const user = await User.findOne({ email: req.body.email });
81      if (!user) {
82        return res.status(404).send("User not found");
83      }
84
85      const isMatch = await bcrypt.compare(req.body.password, user.password);
86
87      if (!isMatch) {
88        return res.status(404).send("wrong password");
89      }
90
91      //create the token
92      const token = jwt.sign(
93        { id: user, isAdmin: user.isAdmin },
94        process.env.JWT
95      );
96
97      const { password, isAdmin, ...otherDetails } = user._doc;
98      res
99        .cookie("access_token", token, {
100         httpOnly: true,
101       })
102       .status(200)
103       .json({ details: { ...otherDetails }, isAdmin, token });
104   } catch (error) {
105     next(error);
106   }
107 };
108
109 // @desc    Logout user
110 // @route   POST /api/logout
111 // @access  Private
112 const logoutUser = (req, res) => {
113   res.clearCookie("access_token"); // clear the access_token cookie
114   req.session.destroy(); // destroy the session
```

authController.js

Travely > backend > controllers > authController.js > ...

```
119 const resetpasswordrequest = async (req, res) => {
120   const { email } = req.body;
121
122   try {
123     const user = await User.findOne({ email });
124     if (!user) {
125       return res.status(404).json({ message: "User not found" });
126     }
127
128     const token = generateToken({ userId: user._id });
129     const resetLink = `http://localhost:3000/reset-password?token=${token}`;
130
131     // create reusable transporter object using the default SMTP transport
132     let transporter = nodemailer.createTransport({
133       host: "smtp.office365.com",
134       port: 587,
135       secure: false,
136       auth: {
137         user: "isurusanka98@gmail.com",
138         pass: "HGTim@98",
139       },
140       tls: {
141         ciphers: "SSLv3",
142       },
143     });
144
145     // send mail with defined transport object
146     let info = await transporter.sendMail({
147       from: "isurusanka98@gmail.com",
148       to: email,
149       subject: "Reset Password",
150       text: `Please click on the following link to reset your password: ${resetLink}`,
151     });
152
153     console.log("Message sent: %s", info.messageId);
154
155     res.json({ message: "Reset password email sent", token: token });
```

authController.js

Travely > backend > controllers > authController.js > ...

```
162  //rest password
163  const resetpassword = async (req, res) => {
164    const { token, password } = req.body;
165
166    try {
167      const { userId } = verifyToken(token);
168      const user = await User.findById(userId);
169      if (!user) {
170        return res.status(404).json({ message: "User not found" });
171      }
172
173      var salt = await bcrypt.genSaltSync(10);
174      var hash = await bcrypt.hashSync(password, salt);
175
176      user.password = hash;
177      await user.save();
178
179      res.json({ message: "Password reset successfully" });
180    } catch (error) {
181      console.error(error);
182      res.status(500).json({ message: "Internal server error" });
183    }
184  };
185
186  const checkEmailExists = async (req, res, next) => {
187    try {
188      const { email } = req.query;
189      const user = await User.findOne({ email });
190      if (user) {
191        return res.status(409).json({ message: "Email already exists" });
192      }
193      return res.status(200).json({ message: "Email is available" });
194    } catch (error) {
195      next(error);
196    }
197  };
198
```

authController.js

Travely > backend > controllers > authController.js > ...

```
186  const checkEmailExists = async (req, res, next) => {
187    try {
188      const { email } = req.query;
189      const user = await User.findOne({ email });
190      if (user) {
191        return res.status(409).json({ message: "Email already exists" });
192      }
193      return res.status(200).json({ message: "Email is available" });
194    } catch (error) {
195      next(error);
196    }
197  };
198
199  module.exports = {
200    registerUser,
201    loginUser,
202    logoutUser,
203    resetpasswordrequest,
204    resetpassword,
205    checkEmailExists,
206  };
207
```

ii. UserController.js:

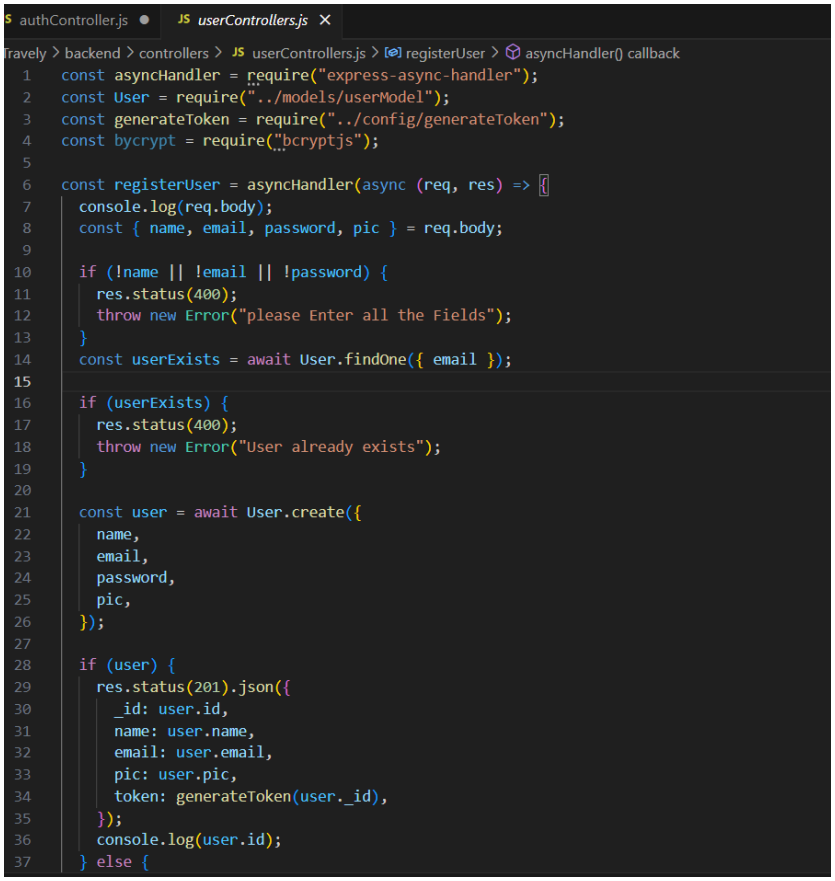
Travely > backend > controllers > JS userController.js > ...

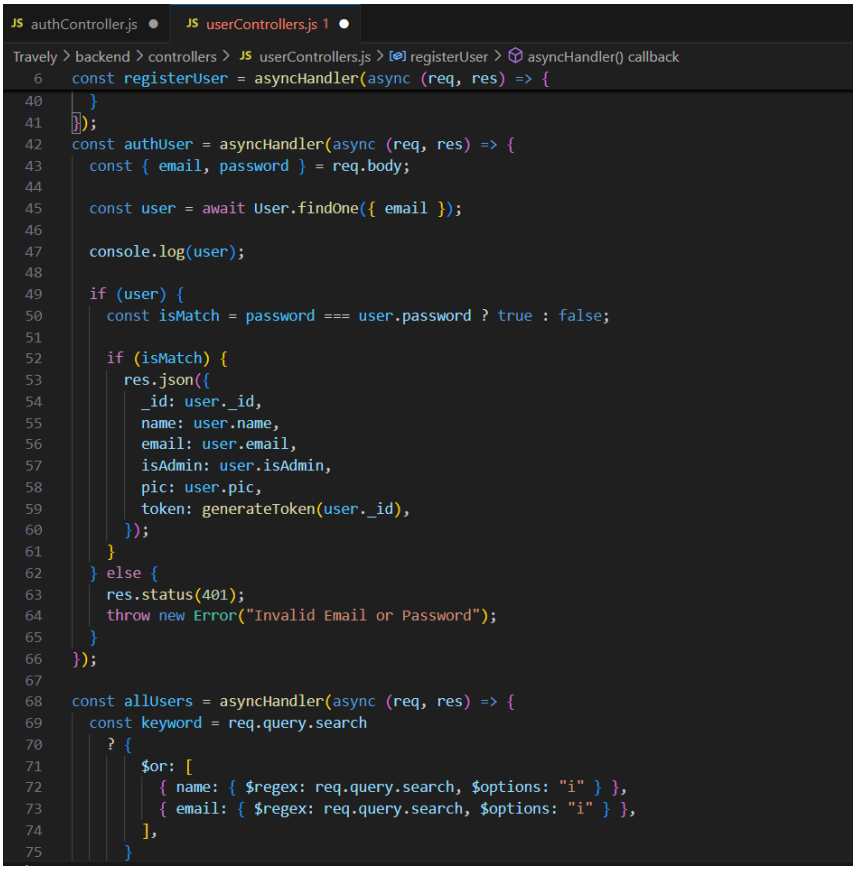
```
const User = require("../models/userModel");

// @desc    Update a User
// @route   PUT /api/users/:id
// @access
const updateUser = async (req, res) => {
  try {
    const updatedUser = await User.findByIdAndUpdate(
      req.params.id,
      {
        $set: req.body,
      },
      { new: true }
    );
    res.status(200).json(updatedUser);
  } catch (error) {
    res.status(404).json({ message: error.message });
  }
};

// @desc    Delete a User
// @route   DELETE /api/users/:id
// @access
const deleteUser = async (req, res) => {
  try {
    await User.findByIdAndDelete(req.params.id);
    res.status(200).json("User has been deleted");
  } catch (error) {
    res.status(404).json({ message: error.message });
  }
};

// @desc    Get a User
// @route   GET /api/users/:id
// @access
const getUser = async (req, res) => {
  try {
```

Travely > backend > controllers > JS userController.js > ...

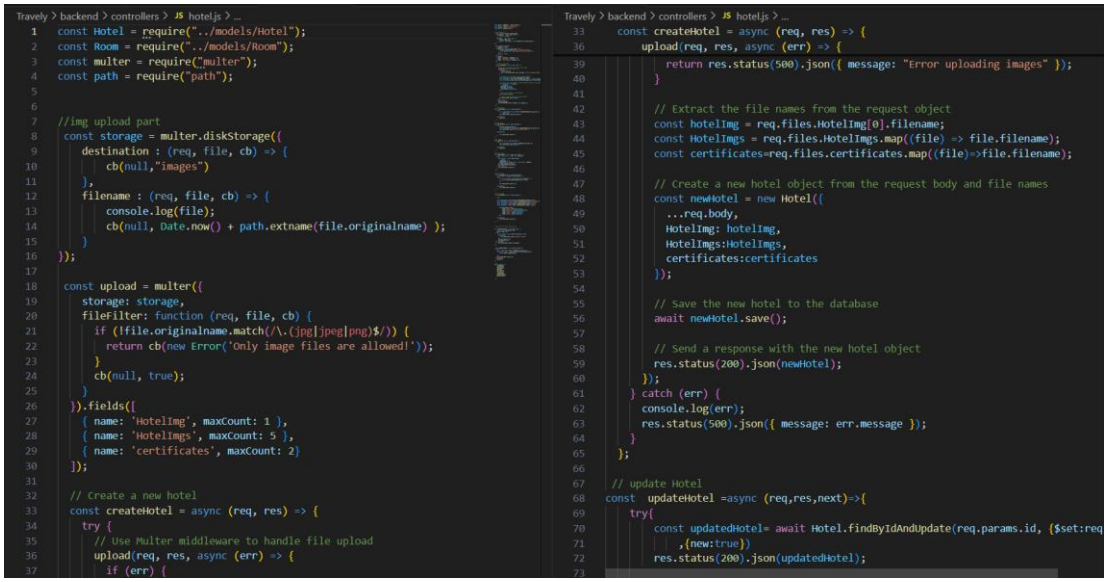
```
// @desc    Get a User
// @route   GET /api/users/:id
// @access
const getUser = async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    res.status(200).json(user);
  } catch (error) {
    res.status(404).json({ message: error.message });
  }
};

// @desc    Get all Users
// @route   GET /api/hotels
// @access
const getAllUsers = async (req, res) => {
  try {
    const users = await User.find();
    res.status(200).json(users);
  } catch (error) {
    res.status(404).json({ message: error.message });
  }
};

module.exports = {
  updateUser,
  deleteUser,
  getUser,
  getAllUsers,
};
```

iii. UserControllers.js:



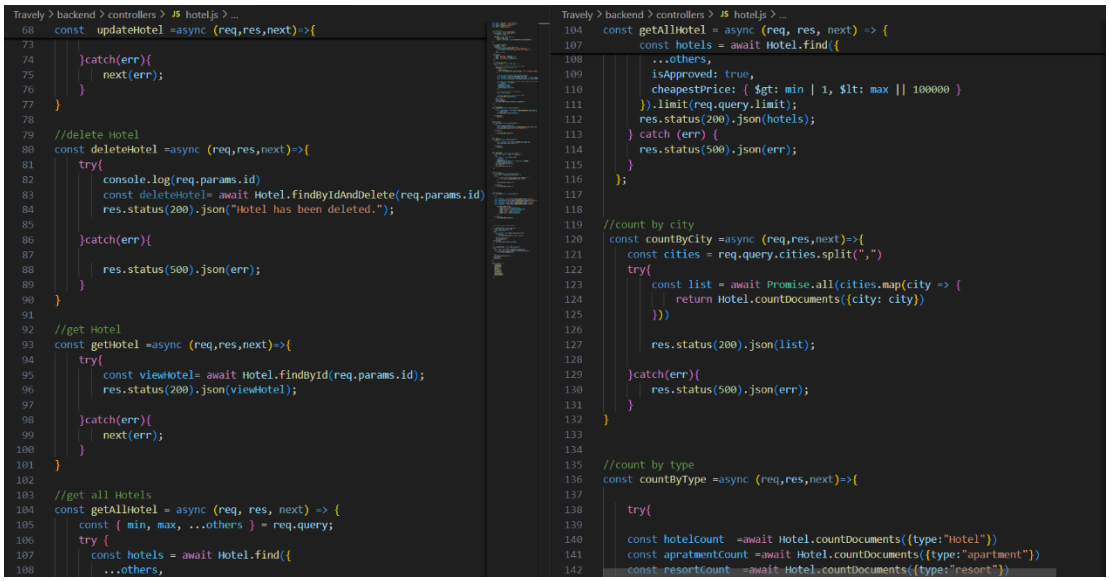


iv. Hotel.js



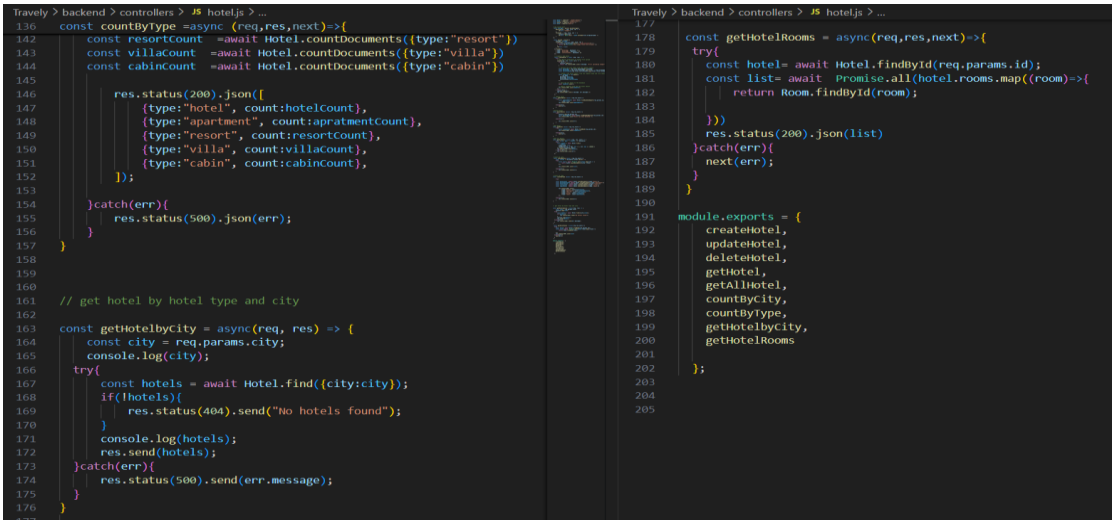
```
Travely > backend > controllers > JS hotel.js > ...
const Hotel = require("../models/Hotel");
const Room = require("../models/Room");
const multer = require("multer");
const path = require("path");


//img upload part
const storage = multer.diskStorage({
    destination : (req, file, cb) => {
        cb(null,"images")
    },
    filename : (req, file, cb) => {
        console.log(file);
        cb(null, Date.now() + path.extname(file.originalname) );
    }
});

const upload = multer({
    storage: storage,
    fileFilter: function (req, file, cb) {
      if (!file.originalname.match(/\.(jpg|jpeg|png)$/)) {
        return cb(new Error('Only image files are allowed!'));
      }
      cb(null, true);
    }
  }).fields([
    { name: 'HotelImg', maxCount: 1 },
    { name: 'HotelImgs', maxCount: 5 },
    { name: 'certificates', maxCount: 2}
  ]);

  // Create a new hotel
  const createHotel = async (req, res) => {
    try {
      // Use Multer middleware to handle file upload
      upload(req, res, async (err) => {
        if (err) {
```

```
Travely > backend > controllers > JS hotel.js > ...
    const createHotel = async (req, res) => {
      upload(req, res, async (err) => {
        if (err) {
          return res.status(500).json({ message: "Error uploading images" });
        }

        // Extract the file names from the request object
        const hotelImg = req.files.HotelImg[0].filename;
        const HotelImgs = req.files.HotelImgs.map((file) => file.filename);
        const certificates=req.files.certificates.map((file)=>file.filename);

        // Create a new hotel object from the request body and file names
        const newHotel = new Hotel({
          ...req.body,
          HotelImg: hotelImg,
          HotelImgs:HotelImgs,
          certificates:certificates
        });

        // Save the new hotel to the database
        await newHotel.save();

        // Send a response with the new hotel object
        res.status(200).json(newHotel);
      });
    } catch (err) {
      console.log(err);
      res.status(500).json({ message: err.message });
    }
  };

  // update Hotel
const  updateHotel =async (req,res,next)=>{
    try{
        const updatedHotel= await Hotel.findByIdAndUpdate(req.params.id, {$set:req
            ,{new:true})
        res.status(200).json(updatedHotel);
```



```
Travely > backend > controllers > JS hotel.js > ...
const  updateHotel =async (req,res,next)=>{

    }catch(err){
        next(err);
    }
}

//delete Hotel
const deleteHotel =async (req,res,next)=>{
    try{
        console.log(req.params.id)
        const deleteHotel= await Hotel.findByIdAndDelete(req.params.id)
        res.status(200).json("Hotel has been deleted.");

    }catch(err){

        res.status(500).json(err);
    }
}

//get Hotel
const getHotel =async (req,res,next)=>{
    try{
        const viewHotel= await Hotel.findById(req.params.id);
        res.status(200).json(viewHotel);

    }catch(err){
        next(err);
    }
}

//get all Hotels
const getAllHotel = async (req, res, next) => {
    const { min, max, ...others } = req.query;
    try {
      const hotels = await Hotel.find({
        ...others,
```

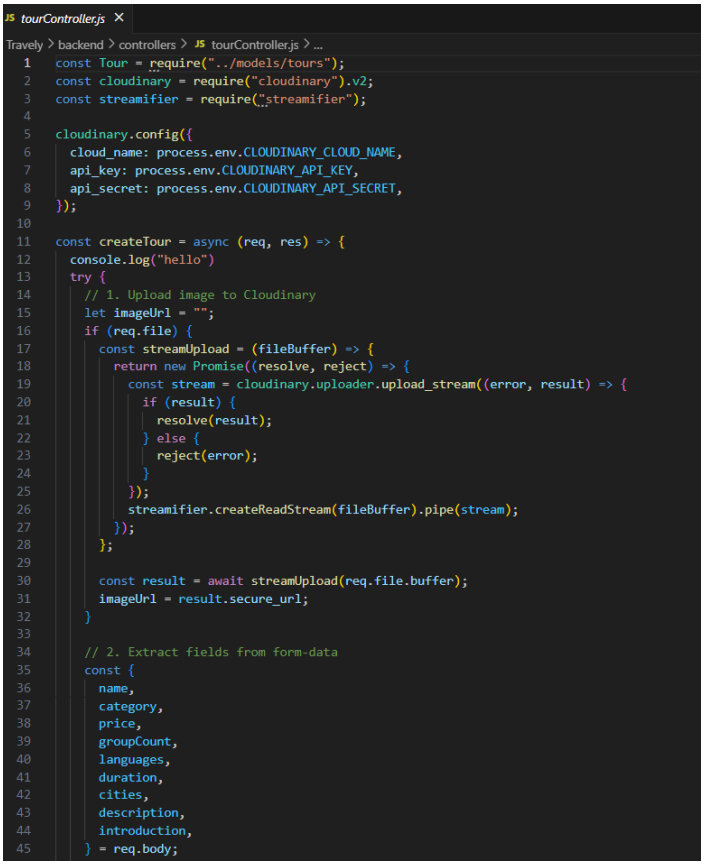
```
Travely > backend > controllers > JS hotel.js > ...
const getAllHotel = async (req, res, next) => {
      const hotels = await Hotel.find({
        ...others,
        isApproved: true,
        cheapestPrice: { $gt: min | 1, $lt: max || 100000 }
      }).limit(req.query.limit);
      res.status(200).json(hotels);
    } catch (err) {
      res.status(500).json(err);
    }
  };


//count by city
 const countByCity =async (req,res,next)=>{
    const cities = req.query.cities.split(",")
    try{
        const list = await Promise.all(cities.map(city => {
            return Hotel.countDocuments({city: city})
        }))

        res.status(200).json(list);

    }catch(err){
        res.status(500).json(err);
    }
}


//count by type
const countByType =async (req,res,next)=>{

    try{

    const hotelCount  =await Hotel.countDocuments({type:"Hotel"})
    const apratmentCount =await Hotel.countDocuments({type:"apartment"})
    const resortCount  =await Hotel.countDocuments({type:"resort"})
```



```
Travely > backend > controllers > JS hotel.js > ...
const countByType =async (req,res,next)=>{
    const resortCount  =await Hotel.countDocuments({type:"resort"})
    const villaCount  =await Hotel.countDocuments({type:"villa"})
    const cabinCount  =await Hotel.countDocuments({type:"cabin"})

        res.status(200).json([
            {type:"hotel", count:hotelCount},
            {type:"apartment", count:apratmentCount},
            {type:"resort", count:resortCount},
            {type:"villa", count:villaCount},
            {type:"cabin", count:cabinCount},
        ]);

    }catch(err){
        res.status(500).json(err);
    }
}


// get hotel by hotel type and city

const getHotelbyCity = async(req, res) => {
    const city = req.params.city;
    console.log(city);
   try{
      const hotels = await Hotel.find({city:city});
      if(!hotels){
          res.status(404).send("No hotels found");
      }
      console.log(hotels);
      res.send(hotels);
   }catch(err){
      res.status(500).send(err.message);
   }
}
```

```
Travely > backend > controllers > JS hotel.js > ...

const getHotelRooms = async(req,res,next)=>{
 try{
    const hotel= await Hotel.findById(req.params.id);
    const list= await  Promise.all(hotel.rooms.map((room)=>{
        return Room.findById(room);

    }))
    res.status(200).json(list)
 }catch(err){
    next(err);
 }
}

module.exports = {
    createHotel,
    updateHotel,
    deleteHotel,
    getHotel,
    getAllHotel,
    countByCity,
    countByType,
    getHotelbyCity,
    getHotelRooms

};
```

v.ToursController.js



```
tourController.js
Travely > backend > controllers > tourController.js > ...
const Tour = require("../models/tours");
const cloudinary = require("cloudinary").v2;
const streamifier = require("streamifier");

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

const createTour = async (req, res) => {
  console.log("hello")
  try {
    // 1. Upload image to Cloudinary
    let imageUrl = "";
    if (req.file) {
      const streamUpload = (fileBuffer) => {
        return new Promise((resolve, reject) => {
          const stream = cloudinary.uploader.upload_stream((error, result) => {
            if (result) {
              resolve(result);
            } else {
              reject(error);
            }
          });
          streamifier.createReadStream(fileBuffer).pipe(stream);
        });
      };

      const result = await streamUpload(req.file.buffer);
      imageUrl = result.secure_url;
    }

    // 2. Extract fields from form-data
    const {
      name,
      category,
      price,
      groupCount,
      languages,
      duration,
      cities,
      description,
      introduction,
    } = req.body;
```

```
tourController.js
Travely > backend > controllers > tourController.js > ...
const createTour = async (req, res) => {
    // 3. Create tour object
    const newTour = new Tour({
      name,
      category,
      price,
      groupCount,
      languages,
      duration,
      cities,
      description,
      introduction,
      img: imageUrl, // saved Cloudinary URL
    });

    const savedTour = await newTour.save();

    res.status(200).json({
      status: "Success",
      message: "Tour added successfully",
      data: {
        tour: savedTour,
      },
    });
  } catch (err) {
    console.error(err.message);
    res.status(400).json({
      status: "Fail",
      message: err.message,
    });
  }
};

//getAll tour list
const getAllTours = async (req, res) => {
  try {
    const allTours = await Tour.find();
    res.status(200).send(allTours);
  } catch (err) {
    console.log(err.message);
    res.status(404).json({
      status: "unsuccess",
      message: err.message,
    });
  }
```

```
const getToursByName = async (req, res) => {
  try {
    const name = req.params.name;

    // Find all tours with the given name (case-insensitive match)
    const tours = await Tour.find({ name: { $regex: new RegExp(`^${name}$`, "i") } });

    if (tours.length === 0) {
      return res.status(404).json({
        status: "Not Found",
        message: `No tours found with name: ${name}`,
      });
    }

    res.status(200).json({
      status: "Success",
      results: tours.length,
      data: {
        tours,
      },
    });
  } catch (err) {
    console.error(err.message);
    res.status(500).json({
      status: "Error",
      message: err.message,
    });
  }
};

const getTour = async (req, res) => {
  try {
    const uID = req.params.id;

    const oneTour = await Tour.findById(uID);
    res.status(200).json({
      status: "Success",
      data: {
        oneTour,
      },
    });
  } catch (err) {
    console.log(err.message);
    res.status(404).json({
      status: "unsuccess",
```

Travely > backend > controllers > JS tourController.js > ...

```
const updateTour = async (req, res) => {
      status: "Success",
      data: {
        tour: updatedTour,
      },
    });
  } catch (err) {
    console.log(err.message);
    res.status(404).json({
      status: "unsuccess",
      message: err.message,
    });
  }
};

//delete a tour
const deleteTour = async (req, res) => {
  try {
    const UID = req.params.id;
    const deletedTour = await Tour.findByIdAndDelete(UID);

    res.status(204).json({
      status: "Success",
      data: {
        old: deletedTour,
        tour: "Null",
      },
    });
  } catch (err) {
    console.log(err.message);
    res.status(404).json({
      status: "unsuccess",
      message: err.message,
    });
  }
};

module.exports = {
  createTour,
  updateTour,
  getAllTours,
  getTour,
  deleteTour,
  getToursByName,
};
```

vi. Trainctrls:

```
const multer = require("multer");
const Train = require("../models/Train");
const path = require("path");

const storage = multer.diskStorage({
    destination : (req,file,cb)=> {
        cb(null,"images")
    },
    filename : (req,file,cb)=>{
        console.log(file);
        cb(null,Date.now()+path.extname(file.originalname));
    }
});

const upload = multer({
    storage: storage,
    fileFilter: function (req, file, cb) {
      if (!file.originalname.match(/\.(jpg|jpeg|png)$/)) {
        return cb(new Error('Only image files are allowed!'));
      }
      cb(null, true);
    }
  }).fields([
    { name: 'trainMainImg', maxCount: 1 }
  ]);

// add a train admin
const addTrain = async(req,res)=>{
    try {
        upload(req,res,async(err)=>{
            if(err){
                console.log(err.message);
                return res.status(500).json({message:"error in uploading"})
            }

            const newTrain = new Train({
                ...req.body,
               // trainMainImg : req.files.trainMainImg[0].filename
            })

            await newTrain.save();
            res.status(200).json(newTrain);
        });
    } catch(err){
        console.log(err);
```

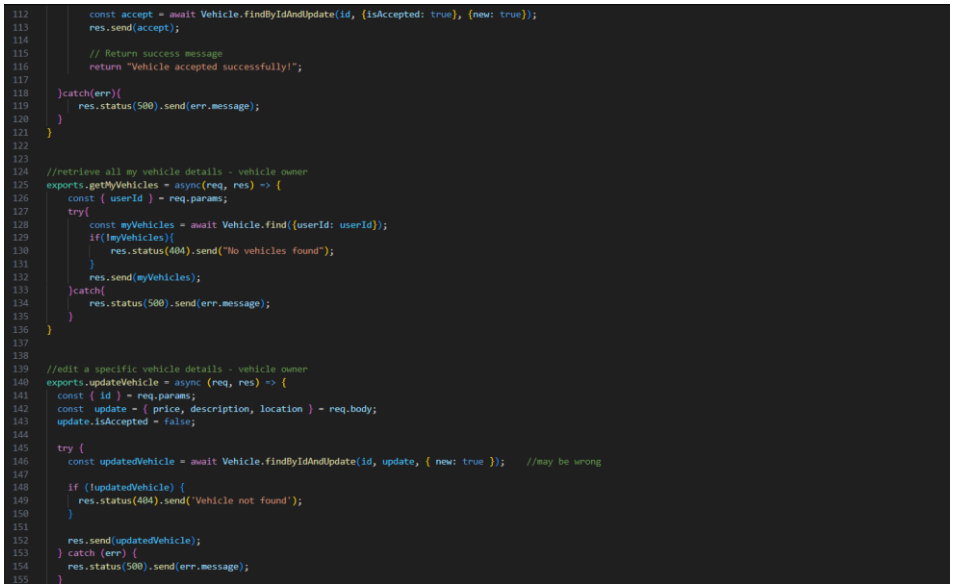
```
// get all trains admin -- just for now
const getAllTrains = async(req,res)=>{
    await Train.find().then((trains)=>{
        res.json(trains);
    }).catch((err)=>{
        res.send(err.message);
    })
}

// get one trains
const getSingleTrain = async (req,res)=>{
    const id = req.params.id;

    const train = await Train.findById(id).then((train)=>{
        res.json(train);
    }).catch((err)=>{
        res.json(err.message)
    })
}

// delete train  - admin
const deleteTrain = async(req,res)=>{
    const id = req.params.id;

    await Train.findByIdAndDelete(id).then(()=>{
        res.json("deleted");
    }).catch((err)=>{
        res.json(err.message);
    })
}


// update train -admin
const updateTrain = async(req,res)=>{
    const id = req.params.id;

    const {trainName,from,to,arrivalTime,depatureTime,date,price,
        noOfSeats ,description,trainMainImg,MaxBagage,classType,cancelCharges} = req.body;

    const updatedTrain = {trainName,from,to,arrivalTime,depatureTime,date,price,
        noOfSeats ,description,trainMainImg,MaxBagage,classType,cancelCharges
    }

    await Train.findByIdAndUpdate(id,updatedTrain).then(()=>{
```

```
const updateTrain = async(req,res)=>{
    await Train.findByIdAndUpdate(id,updatedTrain).then(()=>{
        //const fetehced =  Train.findById(id);
        res.json({status : "Updated",train:updatedTrain});
    }).catch((err)=>{
        res.json(err.message);
    })

}

// fetch trains by from and To of search component
const getTrainFromTo =  async (req,res) =>{
    const {from,to} = req.params;
    try{
        const trains = await Train.find({from : from , to : to})
        if(!trains){
            res.status(404).send("no trains");
        }
        res.send(trains)
    }catch(err){
        res.status(500).send(err.message);
    }
}

module.exports= {
    addTrain,
    getAllTrains,
    getSingleTrain,
    deleteTrain,
    updateTrain,
    getTrainFromTo
}
```

vii. VehiclesControllers:

```
const multer = require("multer");
const Vehicle = require("../models/Vehicle");
//const AcceptedVehicle = require("../models/AcceptedVehicle");
const path = require("path");

//image uploading path to diskStorage
const storage = multer.diskStorage({
    destination : (req, file, cb) => {
        cb(null, "images")
    },
    filename : (req, file, cb) => {
        console.log(file);
        cb(null, Date.now() + path.extname(file.originalname) );
    }
});

const upload = multer({
    storage: storage,
    fileFilter: function (req, file, cb) {
      if (!file.originalname.match(/\.(jpg|jpeg|png)$/)) {
        return cb(new Error('Only image files are allowed!'));
      }
      cb(null, true);
    }
  }).fields([
    { name: 'vehicleMainImg', maxCount: 1 },
    { name: 'insuranceImgs', maxCount: 2 },
    { name: 'vehicleImgs', maxCount: 5 }
  ]);



//add a vehicle - postman checked. working well.
exports.addVehicle = async(req, res) => {

    try {
        upload(req, res, async(err) => {
            if (err) {
                console.log(err);
                return res.status(500).json({ message: "Error uploading images" });
            }


            const newVehicle = new Vehicle({
                ...req.body,
```

```
//get all vehicles - vehicle admin            //postman checked. working well.
exports.getAllVehicle = async(req, res) => {
    try{
        const vehicles = await Vehicle.find();
        res.send(vehicles);
    }catch (err) {
        res.status(500).send(err.message);
    }
}


//get a specific vehicle by id - vehicle admin       postman checked. working well.
exports.getSpecificvehicle = async(req, res) => {
    const {id} = req.params;
    try{
        const specificVehicle = await Vehicle.findById({_id: id});     //without _id, it will not work ???
        res.send(specificVehicle);
    }catch(err){
        res.status(500).send(err.message);
    }
}



//delete a specific vehicle by id (Reject a vehicle from pending collection) - vehicle admin
exports.deleteVehicle = async (req, res) => {
    const { id } = req.params;
    try {
      const vehicle = await Vehicle.findByIdAndDelete(id);
      if (!vehicle) {
        res.status(404).send('Vehicle not found');
      } else {
        res.send(vehicle);
      }
    } catch (err) {
      res.status(500).send(err.message);
    }
  };


//accept a vehicle by isAccepted change to true - vehicle admin
exports.acceptVehicle = async(req, res) => {
    const {id} = req.params;
    try{
        const vehicle = await Vehicle.findById(id);
```



```
        const accept = await Vehicle.findByIdAndUpdate(id, {isAccepted: true}, {new: true});
        res.send(accept);

        // Return success message
        return "Vehicle accepted successfully!";

  }catch(err){
      res.status(500).send(err.message);
  }
}


//retrieve all my vehicle details - vehicle owner
exports.getMyVehicles = async(req, res) => {
    const { userId } = req.params;
    try{
        const myVehicles = await Vehicle.find({userId: userId});
        if(!myVehicles){
            res.status(404).send("No vehicles found");
        }
        res.send(myVehicles);
    }catch{
        res.status(500).send(err.message);
    }
}


//edit a specific vehicle details - vehicle owner
exports.updateVehicle = async (req, res) => {
  const { id } = req.params;
  const  update = { price, description, location } = req.body;
  update.isAccepted = false;

  try {
    const updatedVehicle = await Vehicle.findByIdAndUpdate(id, update, { new: true });    //may be wrong

    if (!updatedVehicle) {
      res.status(404).send('Vehicle not found');
    }

    res.send(updatedVehicle);
  } catch (err) {
    res.status(500).send(err.message);
  }
```

```
exports.availableVehicles = async (req, res) => {

 try{
     const availablevehicles = await Vehicle.aggregate([
         {
             $lookup: {
                 from: "reservations",
                 localField: "_id",
                 foreignField: "vehicleId",
                 as: "reservations"
             }
         },
         {
             $match: {
                 $and: [
                     { "reservations.pickupDate": { $lt: new Date(req.body.pickupDate) } },
                     { "reservations.returnDate": { $lt: new Date(req.body.pickupDate) } },
                     { "reservations.pickupDate": { $gt: new Date(req.body.returnDate) } },
                     { "reservations.returnDate": { $gt: new Date(req.body.returnDate) } },
                     { "location": req.body.location },
                     { "vehicleType": req.body.vehicleType }
                 ]
             }
         }
     ]);

     res.send(availablevehicles);
  }catch(err){
     res.status(500).send(err.message);

    }

}


//retrieve vehicles by vehicle type and location - vehicle owner
exports.getVehiclesByTypeAndLocation = async(req, res) => {
    const { vehicleType, location } = req.params;
    try{
        const vehicles = await Vehicle.find({vehicleType: vehicleType, location: location});
        if(!vehicles){
            res.status(404).send("No vehicles found");
        }
        res.send(vehicles);
    }catch(err){
```

vehicleController.js

Travely > backend > controllers > vehicleController.js > ...

```
//retrieve vehicles by vehicle type
exports.getVehicleByType = async (req, res) => {
    const {vehicleType} = req.params;
    try{
        const vehicles = await Vehicle.find({vehicleType: vehicleType});
        if(!vehicles){
            res.status(404).send("No vehicles found");
        }
        res.send(vehicles);
    }catch(err){
        res.status(500).send(err.message);
    }
}

//retrieve vehicles by location
exports.getVehicleByLocation = async (req, res) => {
    const {location} = req.params;
    try{
        const vehicles = await Vehicle.find({location: location});
        if(!vehicles){
            res.status(404).send("No vehicles found");
        }
        res.send(vehicles);
    }catch(err){
        res.status(500).send(err.message);
    }
}

```

6.1.3 Middleware

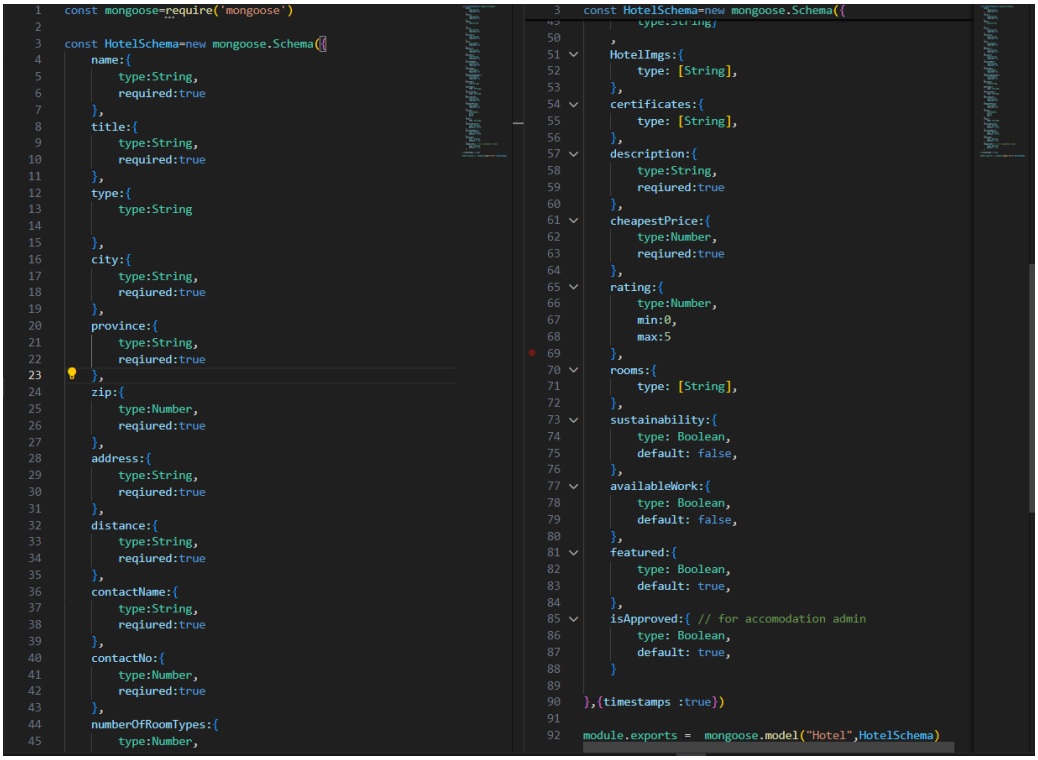
i. authMiddleWare.js

```
const jwt = require('jsonwebtoken');
//normal user
const User = require('../models/userModel.js');

const userMiddleware = async (req, res, next) => {
  try {
    const token = req.cookies.access_token;
    if (!token) {
      return res.status(401).json({ message: 'Invalid Token' });
    }
    const decoded = jwt.verify(token, 'mekarahasak');
    req.user = decoded.id;
    next();
  } catch (err) {
    console.error(err);
    return res.status(401).json({ message: 'Invalid Token' });
  }
};
//admin
const adminMiddleware = async (req, res, next) => {
  try {
    const token = req.cookies.access_token;
    if (!token) {
      return res.status(401).json({ message: 'Invalid Token' });
    }
    const decoded = jwt.verify(token, 'mekarahasak');
    const user = await User.findById(decoded.id, { isAdmin: 1 });
    console.log(user);
    if (!user) {
      return res.status(401).json({ message: 'User not found' });
    }
    if (!user.isAdmin) {
      return res.status(403).json({ message: 'Access denied' });
    }
    req.user = decoded.id;
    next();
  } catch (err) {
    console.error(err);
    return res.status(401).json({ message: 'Invalid Token' });
  }
};
//activity organizer
const organizerMiddleware = async (req, res, next) => {
  try {
    const token = req.cookies.access_token;
```

```
//activity organizer
const organizerMiddleware = async (req, res, next) => {
  try {
    const token = req.cookies.access_token;
    if (!token) {
      return res.status(401).json({ message: 'Invalid Token' });
    }
    const decoded = jwt.verify(token, 'mekarahasak');
    const user = await User.findById(decoded.id, { type: 1 });
    if (!user) {
      return res.status(401).json({ message: 'User not found' });
    }
    if (user.type !== 'eventOrganizer') {
      return res.status(403).json({ message: 'Access denied' });
    }
    req.user = decoded.id;
    next();
  } catch (err) {
    console.error(err);
    return res.status(401).json({ message: 'Invalid Token' });
  }
};

module.exports = {
  userMiddleware,
  adminMiddleware,
  organizerMiddleware,
};

```

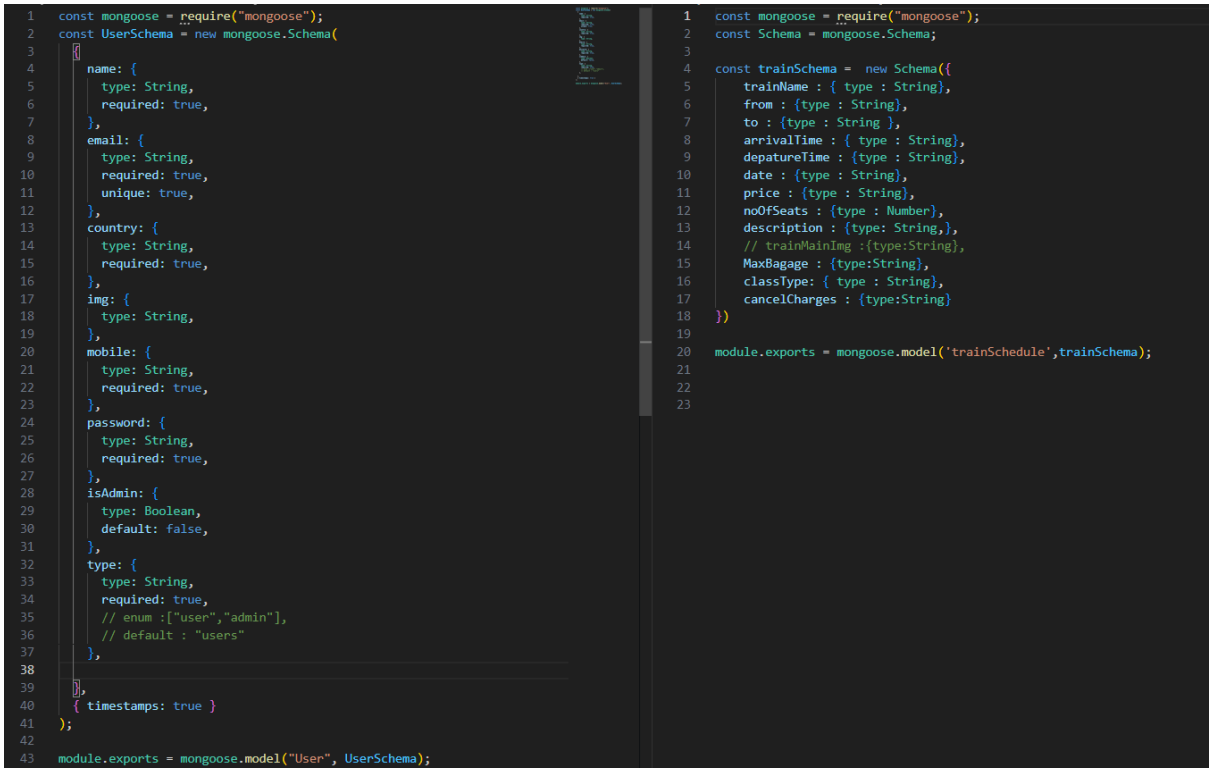
6.1.4 Models

i. .Hotel.js



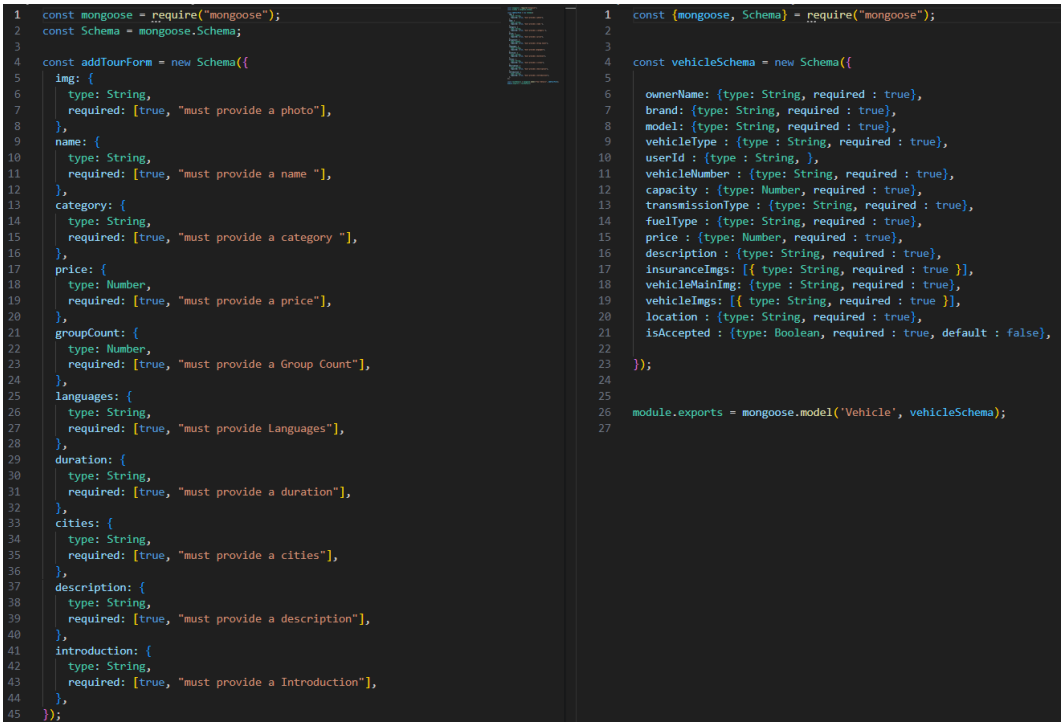
```
const mongoose=require('mongoose')

const HotelSchema=new mongoose.Schema({
    name:{
        type:String,
        required:true
    },
    title:{
        type:String,
        required:true
    },
    type:{
        type:String

    },
    city:{
        type:String,
        reqiured:true
    },
    province:{
        type:String,
        reqiured:true
    },
    zip:{
        type:Number,
        reqiured:true
    },
    address:{
        type:String,
        reqiured:true
    },
    distance:{
        type:String,
        reqiured:true
    },
    contactName:{
        type:String,
        reqiured:true
    },
    contactNo:{
        type:Number,
        reqiured:true
    },
    numberOfRoomTypes:{
        type:Number,
    ,
    HotelImgs:{
        type: [String],
    },
    certificates:{
        type: [String],
    },
    description:{
        type:String,
        reqiured:true
    },
    cheapestPrice:{
        type:Number,
        reqiured:true
    },
    rating:{
        type:Number,
        min:0,
        max:5
    },
    rooms:{
        type: [String],
    },
    sustainability:{
        type: Boolean,
        default: false,
    },
    availableWork:{
        type: Boolean,
        default: false,
    },
    featured:{
        type: Boolean,
        default: true,
    },
    isApproved:{ // for accomodation admin
        type: Boolean,
        default: true,
    }

},{timestamps :true})

module.exports =  mongoose.model("Hotel",HotelSchema)
```

ii.usermodel,js , Train.js , tours.js , vehcile.js ,rooms.js



```
const mongoose = require("mongoose");
const UserSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    country: {
      type: String,
      required: true,
    },
    img: {
      type: String,
    },
    mobile: {
      type: String,
      required: true,
    },
    password: {
      type: String,
      required: true,
    },
    isAdmin: {
      type: Boolean,
      default: false,
    },
    type: {
      type: String,
      required: true,
      // enum :["user","admin"],
      // default : "users"
    },

  },
  { timestamps: true }
);

module.exports = mongoose.model("User", UserSchema);
```

```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const trainSchema =  new Schema({
    trainName : { type : String},
    from : {type : String},
    to : {type : String },
    arrivalTime : { type : String},
    depatureTime : {type : String},
    date : {type : String},
    price : {type : String},
    noOfSeats : {type : Number},
    description : {type: String,},
    // trainMainImg :{type:String},
    MaxBagage : {type:String},
    classType: { type : String},
    cancelCharges : {type:String}
})

module.exports = mongoose.model('trainSchedule',trainSchema);
```



```
const mongoose = require("mongoose");
const Schema = mongoose.Schema;

const addTourForm = new Schema({
  img: {
    type: String,
    required: [true, "must provide a photo"],
  },
  name: {
    type: String,
    required: [true, "must provide a name "],
  },
  category: {
    type: String,
    required: [true, "must provide a category "],
  },
  price: {
    type: Number,
    required: [true, "must provide a price"],
  },
  groupCount: {
    type: Number,
    required: [true, "must provide a Group Count"],
  },
  languages: {
    type: String,
    required: [true, "must provide Languages"],
  },
  duration: {
    type: String,
    required: [true, "must provide a duration"],
  },
  cities: {
    type: String,
    required: [true, "must provide a cities"],
  },
  description: {
    type: String,
    required: [true, "must provide a description"],
  },
  introduction: {
    type: String,
    required: [true, "must provide a Introduction"],
  },
});
```

```
const {mongoose, Schema} = require("mongoose");


const vehicleSchema = new Schema({

  ownerName: {type: String, required : true},
  brand: {type: String, required : true},
  model: {type: String, required : true},
  vehicleType : {type : String, required : true},
  userId : {type : String, },
  vehicleNumber : {type: String, required : true},
  capacity : {type: Number, required : true},
  transmissionType : {type: String, required : true},
  fuelType : {type: String, required : true},
  price : {type: Number, required : true},
  description : {type: String, required : true},
  insuranceImgs: [{ type: String, required : true }],
  vehicleMainImg: {type : String, required : true},
  vehicleImgs: [{ type: String, required : true }],
  location : {type: String, required : true},
  isAccepted : {type: Boolean, required : true, default : false},

});


module.exports = mongoose.model('Vehicle', vehicleSchema);
```

```
const mongoose=require('mongoose')

const RoomSchema=new mongoose.Schema({
    title:{
        type:String,
        required:true
    },
    price:{
        type:Number,
        required:true
    },
    description:{
        type:String,
        reqiured:true
    },
    maxPeople:{
        type:Number,
        reqiured:true
    },
    roomNumbers: [{number : Number, unavailableDates :{type: [Date]}}],

})
module.exports =  mongoose.model("Room",RoomSchema)
```

6.1.5 Utils

Travely > backend > utils > cloudinary.js > ...

```
import { v2 as cloudinary } from "cloudinary";
import dotenv from "dotenv";

dotenv.config();

cloudinary.config({
  cloud_name: process.env.CLOUDINARY_CLOUD_NAME,
  api_key: process.env.CLOUDINARY_API_KEY,
  api_secret: process.env.CLOUDINARY_API_SECRET,
});

export default cloudinary;

```

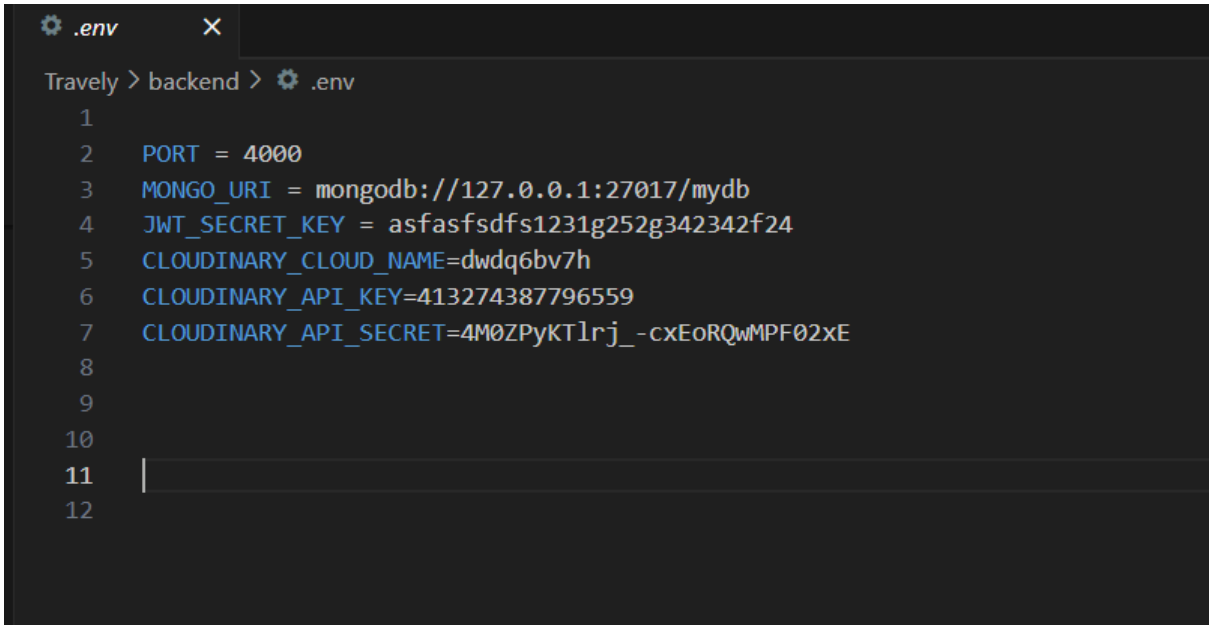
Travely > backend > utils > multer.js > ...

```
import multer from "multer";

const storage = multer.memoryStorage();
const upload = multer({ storage });

export default upload;

```

Travely > backend > utils > error.js > ...

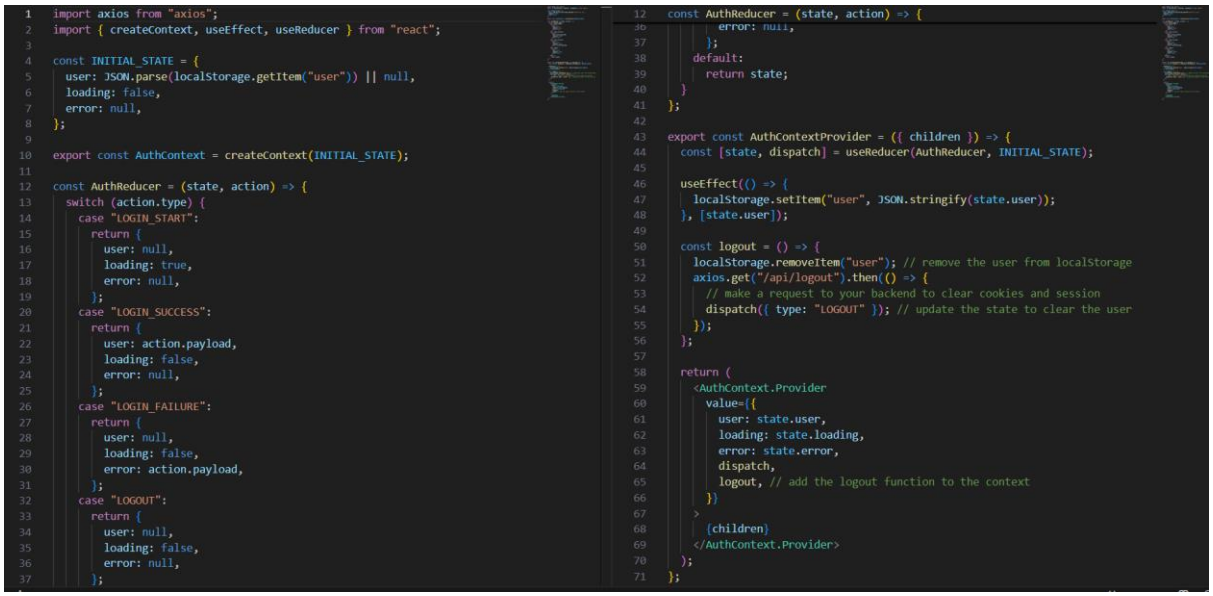
```
export const createError =(status, message)=>{
    const err=new Error();
    err.status=status;
    err.message=message;
    return err;
};
```

6.1.6. .eny



6.2 Frontend

6.2.1 authcontent.js

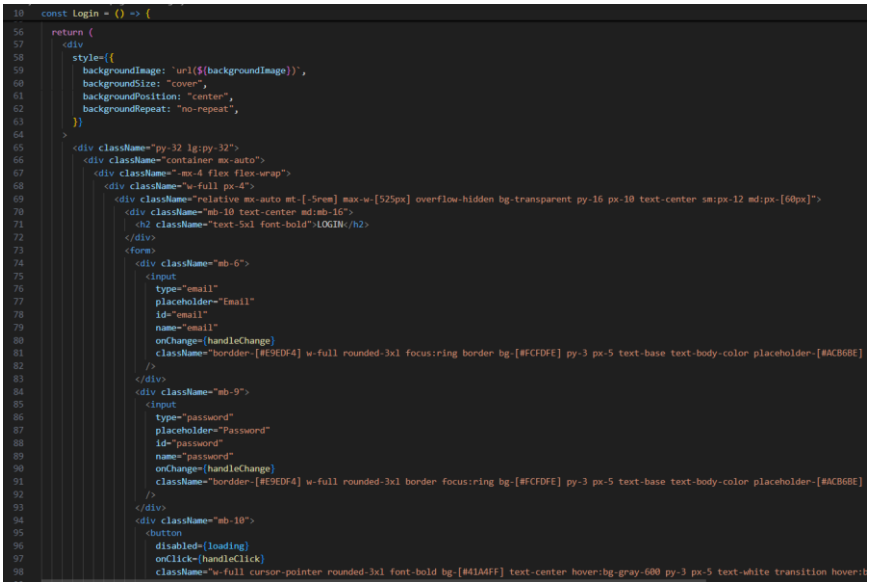


```
import axios from "axios";
import { createContext, useEffect, useReducer } from "react";

const INITIAL_STATE = {
  user: JSON.parse(localStorage.getItem("user")) || null,
  loading: false,
  error: null,
};

export const AuthContext = createContext(INITIAL_STATE);

const AuthReducer = (state, action) => {
  switch (action.type) {
    case "LOGIN_START":
      return {
        user: null,
        loading: true,
        error: null,
      };
    case "LOGIN_SUCCESS":
      return {
        user: action.payload,
        loading: false,
        error: null,
      };
    case "LOGIN_FAILURE":
      return {
        user: null,
        loading: false,
        error: action.payload,
      };
    case "LOGOUT":
      return {
        user: null,
        loading: false,
        error: null,
      };
```

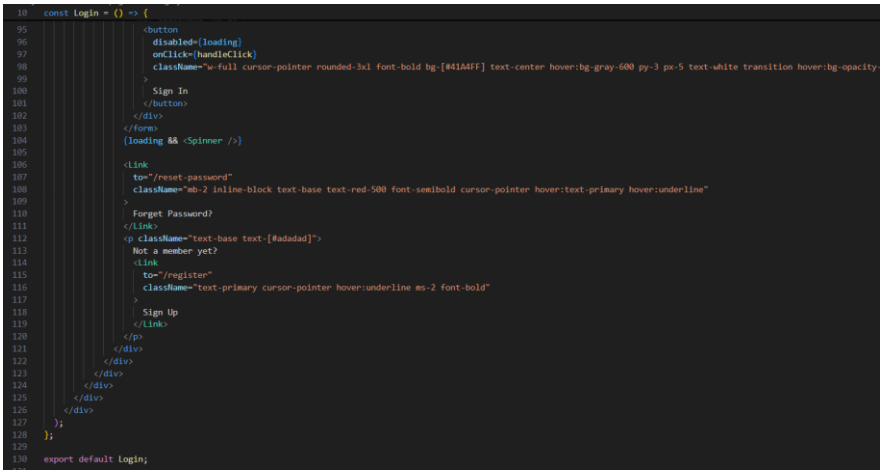
```
const AuthReducer = (state, action) => {
        error: null,
      };
    default:
      return state;
  }
};

export const AuthContextProvider = ({ children }) => {
  const [state, dispatch] = useReducer(AuthReducer, INITIAL_STATE);

  useEffect(() => {
    localStorage.setItem("user", JSON.stringify(state.user));
  }, [state.user]);

  const logout = () => {
    localStorage.removeItem("user"); // remove the user from localStorage
    axios.get("/api/logout").then(() => {
      // make a request to your backend to clear cookies and session
      dispatch({ type: "LOGOUT" }); // update the state to clear the user
    });
  };

  return (
    <AuthContext.Provider
      value={{
        user: state.user,
        loading: state.loading,
        error: state.error,
        dispatch,
        logout, // add the logout function to the context
      }}
    >
      {children}
    </AuthContext.Provider>
  );
};
```

6.2.2 Login.jsx

```jsx
import axios from "axios";
import { useContext, useState } from "react";
import { useNavigate } from "react-router-dom";
import { AuthContext } from "../context/authContext";
import { Link } from "react-router-dom";
import Swal from "sweetalert2";
import backgroundImage from "../assets/images/bg.jpg";
import Spinner from "../components/spinner/LoadingSpinner";

const Login = () => {
  const [credentials, setCredentials] = useState({
    email: undefined,
    password: undefined,
  });

  const [loading2, setLoading2] = useState(false);

  const { loading, error, dispatch } = useContext(AuthContext);

  const navigate = useNavigate();

  const handleChange = (e) => {
    setCredentials((prev) => ({ ...prev, [e.target.id]: e.target.value }));
  };

  const handleClick = async (e) => {
    e.preventDefault();
    dispatch({ type: "LOGIN_START" });

    if (!credentials.email || !credentials.password) {
      Swal.fire("Please enter your email and password", "", "error");
    }
    if (!/\S+@\S+\.\S+/.test(credentials.email)) {
      Swal.fire("Please enter a valid email address", "", "error");
    }
    try {
      setLoading2(true);
      const res = await axios.post("auth/login", credentials);
      dispatch({ type: "LOGIN_SUCCESS", payload: res.data.details });
      setLoading2(false);
      if (res.data.isAdmin === true) {
        navigate("/admin");
      } else if (res.data.details.type == "financeManager") {
        navigate("/finance");
      } else if (res.data.isAdmin === false) {
```



```jsx
const Login = () => {
  return (
    <div
      style={{
        backgroundImage: `url(${backgroundImage})`,
        backgroundSize: "cover",
        backgroundPosition: "center",
        backgroundRepeat: "no-repeat",
      }}
    >
      <div className="py-32 lg:py-32">
        <div className="container mx-auto">
          <div className="-mx-4 flex flex-wrap">
            <div className="w-full px-4">
              <div className="relative mx-auto mt-[-5rem] max-w-[525px] overflow-hidden bg-transparent py-16 px-10 text-center sm:px-12 md:px-[60px]">
                <div classNamе="mb-10 text-center md:mb-16">
                  <h2 className="text-5xl font-bold">LOGIN</h2>
                </div>
                <form>
                  <div className="mb-6">
                    <input
                      type="email"
                      placeholder="Email"
                      id="email"
                      name="email"
                      onChange={handleChange}
                      className="bordder-[#E9EDF4] w-full rounded-3xl focus:ring border bg-[#FCFDFE] py-3 px-5 text-base text-body-color placeholder-[#ACB6BE]
                    />
                  </div>
                  <div className="mb-9">
                    <input
                      type="password"
                      placeholder="Password"
                      id="password"
                      name="password"
                      onChange={handleChange}
                      className="bordder-[#E9EDF4] w-full rounded-3xl border focus:ring bg-[#FCFDFE] py-3 px-5 text-base text-body-color placeholder-[#ACB6BE]
                    />
                  </div>
                  <div className="mb-10">
                    <button
                      disabled={loading}
                      onClick={handleClick}
                      className="w-full cursor-pointer rounded-3xl font-bold bg-[#41A4FF] text-center hover:bg-gray-600 py-3 px-5 text-white transition hover:
```



```jsx
const Login = () => {
                    <button
                      disabled={loading}
                      onClick={handleClick}
                      className="w-full cursor-pointer rounded-3xl font-bold bg-[#41A4FF] text-center hover:bg-gray-600 py-3 px-5 text-white transition hover:bg-opacity-
                    >
                      Sign In
                    </button>
                  </div>
                </form>
                {loading && <Spinner />}

                <Link
                  to="/reset-password"
                  className="mb-2 inline-block text-base text-red-500 font-semibold cursor-pointer hover:text-primary hover:underline"
                >
                  Forget Password?
                </Link>
                <p className="text-base text-[#adadad]">
                  Not a member yet?
                  <Link
                    to="/register"
                    className="text-primary cursor-pointer hover:underline ms-2 font-bold"
                  >
                    Sign Up
                  </Link>
                </p>
              </div>
            </div>
          </div>
        </div>
      </div>
    </div>
  );
};

export default Login;
```

6.2.3 Register.jsx

```jsx
import axios from "axios";
import { useState } from "react";
import { Link, useNavigate } from "react-router-dom";
import DriveFolderUploadOutlinedIcon from "@mui/icons-material/DriveFolderUploadOutlined";
import Swal from "sweetalert2";
import backgroundImage from "../assets/images/bg.jpg";
import Spinner from "../components/spinner/LoadingSpinner";

const Register = () => {
  const [loading2, setLoading2] = useState(false);
  const [file, setFile] = useState(null);

  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [mobile, setMobile] = useState("");
  const [country, setCountry] = useState("");
  const [type, setType] = useState("traveler");
  const [password, setPassword] = useState("");

  const navigate = useNavigate();

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading2(true);

    const formData = new FormData();
    formData.append("name", name);
    formData.append("email", email);
    formData.append("mobile", mobile);
    formData.append("country", country);
    formData.append("type", type);
    formData.append("password", password);
    if (file) formData.append("img", file);

    try {
      await axios.post("http://localhost:5000/api/auth/register", formData, {
        headers: {
          "Content-Type": "multipart/form-data",
        },
      });

      localStorage.setItem("travely_user", JSON.stringify({
        name:name,
        mobile:mobile,
        country: country,
```

```jsx
const Register = () => {
  const handleSubmit = async (e) => {
      Swal.fire("Registered successfully!", "", "success");
      navigate("/");
    } catch (err) {
      Swal.fire({
        icon: "error",
        title: "Registration failed",
        text: err.response?.data?.message || err.message,
      });
    } finally {
      setLoading2(false);
    }
  };

  return (
    <div
      style={{
        backgroundImage: `url(${backgroundImage})`,
        backgroundSize: "cover",
        backgroundPosition: "center",
        backgroundRepeat: "no-repeat",
      }}
    >
      <div className="py-10 lg:py-20 px-16 lg:px-96 md:px-64 flex flex-col text-center">
        <h2 className="text-5xl font-bold mb-8">SIGN UP</h2>

        <div className="mb-6 flex sm:flex-row justify-center">
          <img
            className="rounded-full"
            src={
              file
                ? URL.createObjectURL(file)
                : "https://icon-library.com/images/no-image-icon/no-image-icon-0.jpg"
            }
            alt="avatar"
            style={{ width: "120px", height: "120px" }}
          />
        </div>

        <form onSubmit={handleSubmit}>
          <div className="mb-6 flex justify-center items-center">
            <label htmlFor="file">
              Click to add profile picture <DriveFolderUploadOutlinedIcon />
            </label>
```

```
9   const Register = () => {
91                  <input
92                    type="file"
93                    id="file"
94                    name="img"
95                    style={{ display: "none" }}
96                    accept="image/*"
97                    onChange={(e) => setFile(e.target.files[0])}
98                  />
99                </div>
100
101               <input
102                 placeholder="Name"
103                 type="text"
104                 value={name}
105                 onChange={(e) => setName(e.target.value)}
106                 className="mb-6 w-full rounded-3xl border bg-[#FCFDFE] py-3 px-5"
107               />
108
109               <input
110                 placeholder="Email"
111                 type="text"
112                 value={email}
113                 onChange={(e) => setEmail(e.target.value)}
114                 className="mb-6 w-full rounded-3xl border bg-[#FCFDFE] py-3 px-5"
115               />
116
117               <input
118                 placeholder="Mobile"
119                 type="tel"
120                 value={mobile}
121                 onChange={(e) => setMobile(e.target.value)}
122                 className="mb-6 w-full rounded-3xl border bg-[#FCFDFE] py-3 px-5"
123               />
124
125               <input
126                 placeholder="Country"
127                 type="text"
128                 value={country}
129                 onChange={(e) => setCountry(e.target.value)}
130                 className="mb-6 w-full rounded-3xl border bg-[#FCFDFE] py-3 px-5"
131               />
132
133               <select
134                 value={type}
```

```
9   const Register = () => {
138                 <option value="traveler">Traveler</option>
139                 <option value="hotelOwner">Hotel Owner</option>
140                 <option value="vehicleOwner">Vehicle Owner</option>
141                 <option value="resturentOwner">Restaurant Owner</option>
142                 <option value="tourGuide">Tour Guide</option>
143                 <option value="eventOrganizer">Event Organizer</option>
144               </select>
145
146               <input
147                 placeholder="Password"
148                 type="password"
149                 value={password}
150                 onChange={(e) => setPassword(e.target.value)}
151                 className="mb-9 w-full rounded-3xl border bg-[#FCFDFE] py-3 px-5"
152               />
153
154               <button
155                 type="submit"
156                 className="mb-10 w-full font-bold rounded-3xl bg-[#41A4FF] py-3 px-5 text-white hover:bg-gray-600"
157               >
158                 Sign Up
159               </button>
160             </form>
161
162             {loading2 && <Spinner />}
163
164             <p className="text-base text-[#adadad] pb-20">
165               Already a member yet?
166               <Link
167                 to="/login"
168                 className="text-primary hover:underline ms-2 font-bold"
169               >
170                 Sign In
171               </Link>
172             </p>
173           </div>
174         </div>
175       );
176     };
177
178     export default Register;
```

6.2.4 hotelHome.jsx

```jsx
import React, { useContext } from "react";
import Navbar from "../../components/navbar/Navbar";
import Footer from "../../components/footer/Footer";
import { HotelHero } from "../../components/hotel/HotelHero";
import { HotelSearchBar } from "../../components/hotel/HotelSearchBar";
import HotelCard from "../../components/hotel/HotelCard";
import { useLocation } from "react-router-dom";
import { Link } from "react-router-dom";
import SearchCard from "../../components/hotel/SearchCard";
import { AuthContext } from "../../context/authContext";

export const HotelHome = () => {
  const location = useLocation();
  const {data, date} = location.state ?? {};


  return (
    <div>
      <HotelHero />
      <HotelSearchBar />
      {data?.map((item) => (
        <SearchCard
        name={item.name}
        city={item.city}
        cheapestPrice={item.cheapestPrice}
        HotelImg={item.HotelImg}
        _id= {item._id}
        date={date}
        />


      ))}
      <h1 className="ml-10 mt-5  md:text-2xl  font-bold   text-[#272727]">
        Hotels guests love
      </h1>
      <HotelCard />

    </div>
  );
};

```

6.2.5 TourHome.jsx

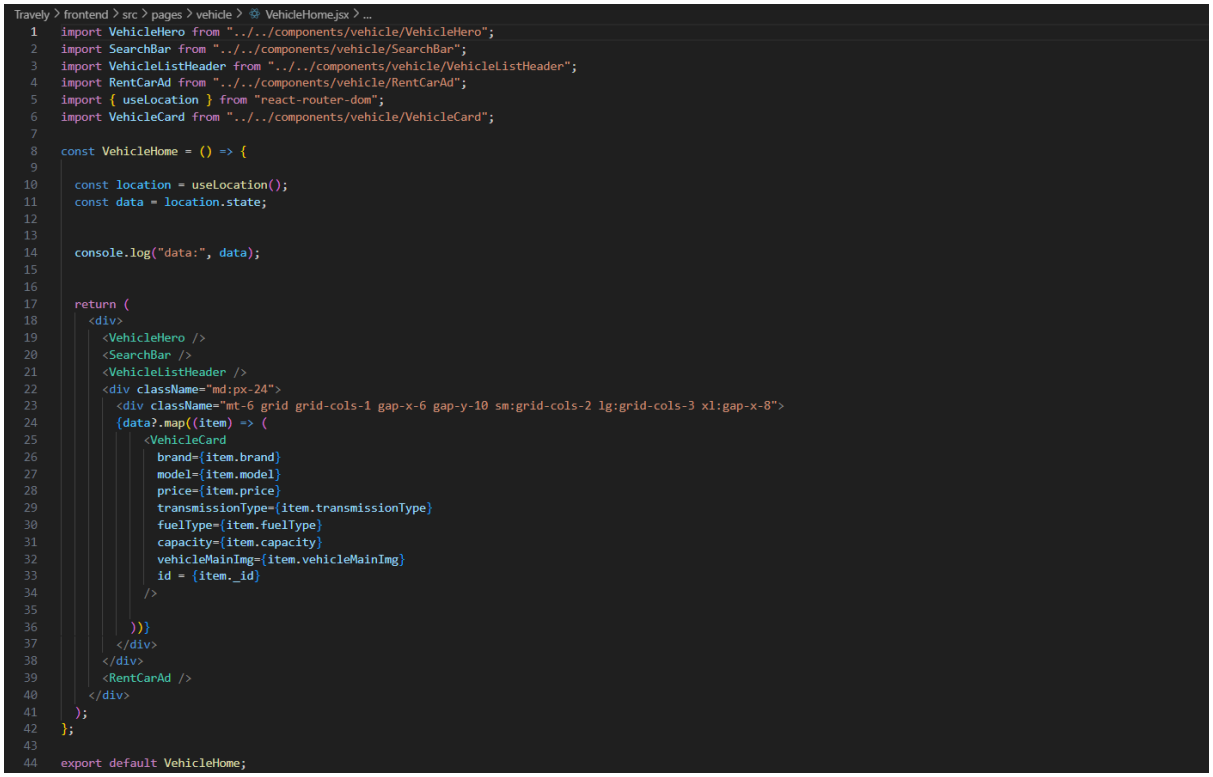
```jsx
import React from "react";

import HeroTour from "./HeroTour";
import HiddenPlaces from "./HiddenPlaces";

import ServiceCard from "./Services/ServiceCard";
import TourCategories from "./Services/ServiceCategories";
import TourNav from "../../components/navbar/TourNav";
import { Link } from "react-router-dom";
import { AiOutlineRight } from "react-icons/ai";
import CustomForm from "./Services/CustomForm";

import welcome from "../../assets/Tour/Tour-Welcome.jpg";

const Home = () => {
  return (
    <div>
      <HeroTour />

      {/* Navigated menu start*/}
      <nav class="bg-grey-light w-full rounded-md pl-20 pt-10">
        <ol class="list-reset flex">
          <li>
            <Link
              to={"/"}
              class="text-primary transition duration-150 ease-in-out hover:text-primary-600 focus:text-primary-600
            >
              Home
            </Link>
          </li>
          <li>
            <AiOutlineRight className="mt-1 mx-2" />
          </li>
          <li>
            <Link
              to={"#"}
              class="text-primary transition duration-150 ease-in-out hover:text-primary-600 focus:text-primary-600
            >
              Tour Packages
            </Link>
          </li>
          <li>
            <AiOutlineRight className="mt-1 mx-2" />
          </li>
          <li class="text-neutral-500 dark:text-neutral-400">
```

```jsx
const Home = () => {
  return (
    <div>
      <div className="mx-auto max-w-2xl px-4  sm:px-6  lg:max-w-7xl lg:px-8">
        {/* welcome image */}
        <img src={welcome} alt="" />
        <h1 className="text-4xl mt-10 mb-10 ml-2">Tour Categories</h1>
        <TourCategories />
      </div>
      {/* Service Card Brief start */}
      <div className="mx-auto max-w-2xl px-4 py-16 sm:px-6 sm:py-24 lg:max-w-7xl lg:px-8">
        <h1 className="text-4xl mb-10 ml-2">Perfect Picks For You</h1>
        <ServiceCard />
        {/* <div className="flex justify-center items-center">
          <button
            type="button"
            data-te-ripple-init
            data-te-ripple-color="light"
            class="mt-20 inline-block rounded-xl bg-primary px-6 pb-2 pt-2.5 text-3xl font-medium uppercase leading-normal text-white shadow-[
          >
            <Link to={"/tourdetails"}></Link>
            View More
          </button>
        </div> */}
      </div>
      {/* Service Card Brief end*/}

      <div className="mx-auto max-w-2xl px-4 py-16 sm:px-6  lg:max-w-7xl lg:px-8">
        <h1 className="text-black uppercase text-center pt-0 mt-0 text-5xl">
          don't fit All these Packages to the  your unique interests and"
          (property) JSX.IntrinsicElements.h1: React.DetailedHTMLProps<React.HTMLAttributes<HTMLHeadingElement>, HTMLHeadingElement>
        </h1>
      </div>

      {/* customer form */}
      <div>
        <CustomForm />
      </div>
      <div>
        <HiddenPlaces />
      </div>
    </div>
  );
};

export default Home;
```

6.2.6 TrainHome,jsx

```jsx
import React ,{useState,useEffect} from "react";
import TrainHero from "../../components/train/TrainHero";
import SearchBar from "../../components/train/SearchBar";
import TrainCard from "../../components/train/TrainCard";
import TrainListheader from "../../components/train/TrainListheader";
import BookTrainAd from "../../components/train/BookTrainAd";
import { useLocation } from "react-router-dom";

const TrainHome = () => {

    const location = useLocation();
    const data = location.state;

    console.log("data",data);

    return (
        <div>
            <TrainHero />
            <SearchBar />
            <TrainListheader />
            <div className="md:px-24">
                <div className="flex flex-wrap flex-col md:flex-row lg:mx-16 gap-[30px]">
                    {
                        data?.map((item)=>(
                            <TrainCard
                                trainName ={item.trainName}
                                from = {item.from}
                                to = {item.to}
                                arrivalTime={item.arrivalTime}
                                depatureTime = {item.depatureTime}
                                noOfSeats = {item.noOfSeats}
                                id={item._id}
                                price={item.price}
                            />
                        ))
                    }

                </div>
            </div>
            <BookTrainAd/>

        </div>
    )
}
export default TrainHome;
```

6.2.7 Vehcile.jsx

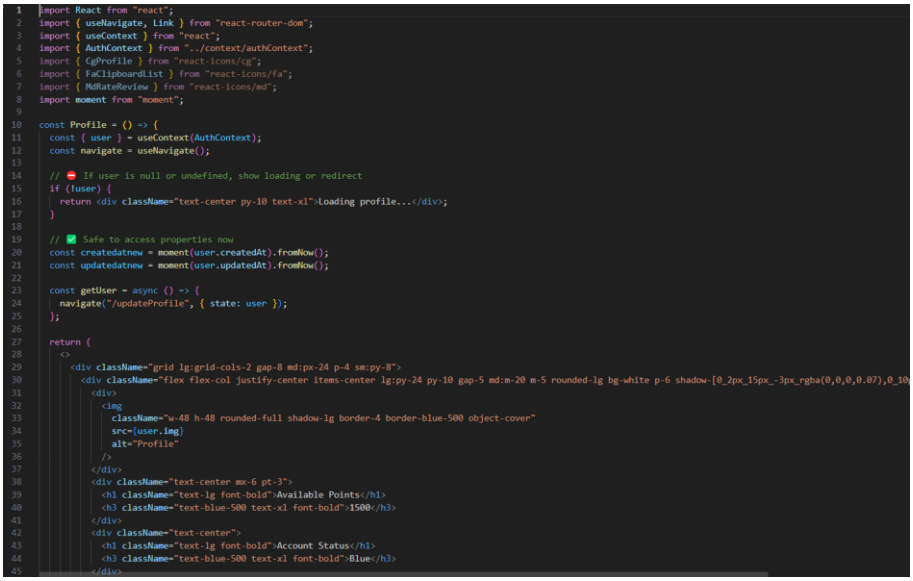


```
Travely > frontend > src > pages > vehicle > VehicleHome.jsx > ...
import VehicleHero from "../../components/vehicle/VehicleHero";
import SearchBar from "../../components/vehicle/SearchBar";
import VehicleListHeader from "../../components/vehicle/VehicleListHeader";
import RentCarAd from "../../components/vehicle/RentCarAd";
import { useLocation } from "react-router-dom";
import VehicleCard from "../../components/vehicle/VehicleCard";

const VehicleHome = () => {

  const location = useLocation();
  const data = location.state;


  console.log("data:", data);


  return (
    <div>
      <VehicleHero />
      <SearchBar />
      <VehicleListHeader />
      <div className="md:px-24">
        <div className="mt-6 grid grid-cols-1 gap-x-6 gap-y-10 sm:grid-cols-2 lg:grid-cols-3 xl:gap-x-8">
        {data?.map((item) => (
            <VehicleCard
              brand={item.brand}
              model={item.model}
              price={item.price}
              transmissionType={item.transmissionType}
              fuelType={item.fuelType}
              capacity={item.capacity}
              vehicleMainImg={item.vehicleMainImg}
              id = {item._id}
            />

        ))}
        </div>
      </div>
      <RentCarAd />
    </div>
  );
};

export default VehicleHome;
```

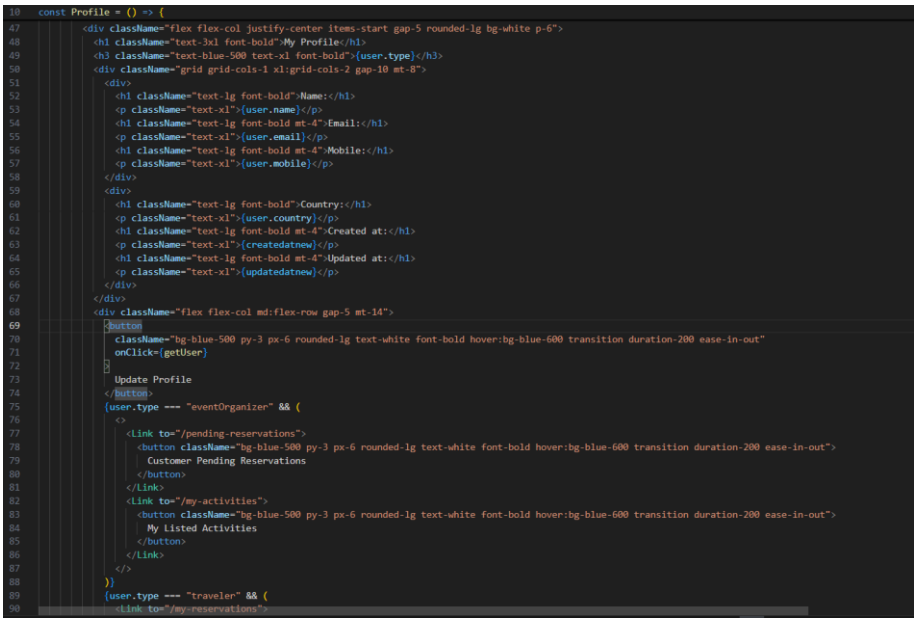
6.2.8 Home.jsx

```
import React from "react";
// import { useContext } from "react";
// import { AuthContext } from "../context/authContext";

import Aboutus from "../components/aboutus/Aboutus";
import Hero2 from "../components/hero2/Hero2";
import Services from "../components/services/Services";
import Hero3 from "../components/hero2/Hero3";

const Home = () => {
  // const { user } = useContext(AuthContext);

  return (
    <div>
      <Hero2 />
      <Aboutus />
      <Services />
      <Hero3 />
    </div>
  );
};

export default Home;
```

6.2.9 profile.jsx



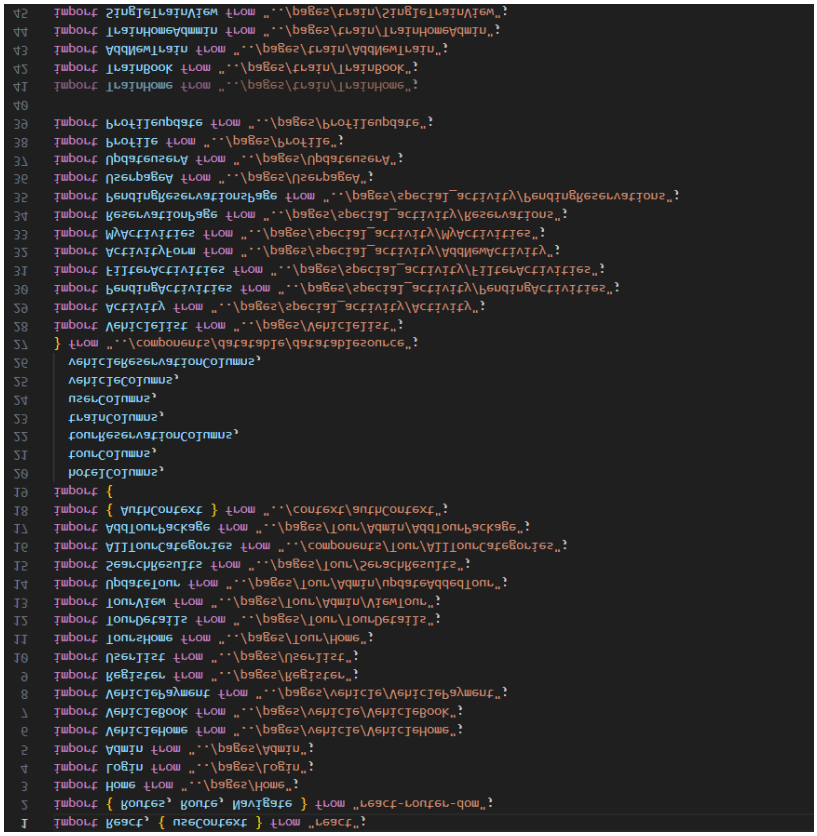
```jsx
import React from "react";
import { useNavigate, Link } from "react-router-dom";
import { useContext } from "react";
import { AuthContext } from "../context/authContext";
import { CgProfile } from "react-icons/cg";
import { FaClipboardList } from "react-icons/fa";
import { MdRateReview } from "react-icons/md";
import moment from "moment";

const Profile = () => {
  const { user } = useContext(AuthContext);
  const navigate = useNavigate();

  // ⛔ If user is null or undefined, show loading or redirect
  if (!user) {
    return <div className="text-center py-10 text-xl">Loading profile...</div>;
  }

  // ✅ Safe to access properties now
  const createdatnew = moment(user.createdAt).fromNow();
  const updatedatnew = moment(user.updatedAt).fromNow();

  const getUser = async () => {
    navigate("/updateProfile", { state: user });
  };

  return (
    <>
      <div className="grid lg:grid-cols-2 gap-8 md:px-24 p-4 sm:py-8">
        <div className="flex flex-col justify-center items-center lg:py-24 py-10 gap-5 md:m-20 m-5 rounded-lg bg-white p-6 shadow-[0_2px_15px_-3px_rgba(0,0,0,0.07),0_10
          <div>
            <img
              className="w-48 h-48 rounded-full shadow-lg border-4 border-blue-500 object-cover"
              src={user.img}
              alt="Profile"
            />
          </div>
          <div className="text-center mx-6 pt-3">
            <h1 className="text-lg font-bold">Available Points</h1>
            <h3 className="text-blue-500 text-xl font-bold">1500</h3>
          </div>
          <div className="text-center">
            <h1 className="text-lg font-bold">Account Status</h1>
            <h3 className="text-blue-500 text-xl font-bold">Blue</h3>
          </div>
```



```jsx
const Profile = () => {
        <div className="flex flex-col justify-center items-start gap-5 rounded-lg bg-white p-6">
          <h1 className="text-3xl font-bold">My Profile</h1>
          <h3 className="text-blue-500 text-xl font-bold">{user.type}</h3>
          <div className="grid grid-cols-1 xl:grid-cols-2 gap-10 mt-8">
            <div>
              <h1 className="text-lg font-bold">Name:</h1>
              <p className="text-xl">{user.name}</p>
              <h1 className="text-lg font-bold mt-4">Email:</h1>
              <p className="text-xl">{user.email}</p>
              <h1 className="text-lg font-bold mt-4">Mobile:</h1>
              <p className="text-xl">{user.mobile}</p>
            </div>
            <div>
              <h1 className="text-lg font-bold">Country:</h1>
              <p className="text-xl">{user.country}</p>
              <h1 className="text-lg font-bold mt-4">Created at:</h1>
              <p className="text-xl">{createdatnew}</p>
              <h1 className="text-lg font-bold mt-4">Updated at:</h1>
              <p className="text-xl">{updatedatnew}</p>
            </div>
          </div>
          <div className="flex flex-col md:flex-row gap-5 mt-14">
            <button
              className="bg-blue-500 py-3 px-6 rounded-lg text-white font-bold hover:bg-blue-600 transition duration-200 ease-in-out"
              onClick={getUser}
            >
              Update Profile
            </button>
            {user.type === "eventOrganizer" && (
              <>
                <Link to="/pending-reservations">
                  <button className="bg-blue-500 py-3 px-6 rounded-lg text-white font-bold hover:bg-blue-600 transition duration-200 ease-in-out">
                    Customer Pending Reservations
                  </button>
                </Link>
                <Link to="/my-activities">
                  <button className="bg-blue-500 py-3 px-6 rounded-lg text-white font-bold hover:bg-blue-600 transition duration-200 ease-in-out">
                    My Listed Activities
                  </button>
                </Link>
              </>
            )}
            {user.type === "traveler" && (
              <Link to="/my-reservations">
```

```jsx
const Profile = () => {
              <Link to="/my-reservations">
                <button className="bg-blue-500 py-3 px-6 rounded-lg text-white font-bold hover:bg-blue-600 transition duration-200 ease-in-out">
                  My Reservations
                </button>
              </Link>
            )}
            {user.isAdmin && (
              <Link to="/pending-activities">
                <button className="bg-blue-500 p-3 rounded-xl text-white font-bold">
                  Pending Activities
                </button>
              </Link>
            )}
          </div>
        </div>
      </div>
    </>
  );
};

export default Profile;
```

6.2.10 Routers



```
import AddPassengerDetails from "../pages/train/AddPassengerDetails";
import DoUpdateTrain from "../pages/train/DoUpdateTrain";

import Adduser from "../pages/Adduser";
import { HotelHome } from "../pages/hotel/HotelHome";
import AddHotel from "../pages/hotel/AddHotel";
import { AddRoom } from "../pages/hotel/AddRoom";
import UpdateHotel from "../pages/hotel/UpdateHotel";
import AddVehicle from "../pages/vehicle/AddVehicle";
import EditVehicle from "../pages/vehicle/EditVehicle";
import HotelView from "../components/hotel/HotelView";
import HotelOverView from "../components/hotel/HotelOverview";
import VehicleView from "../pages/vehicle/VehicleView";

import RestaurentForm from "../pages/Restaturant/RestaurantForm";
import HadminView from "../pages/hotel/HadminView";
import HotelReserve from "../components/hotel/HotelReserve";
import Hotellist from "../pages/Hotellist";
import Tourlist from "../pages/Tourlist";
import Trainlist from "../pages/Trainlist";
import ContactUs from "../pages/ContactUs";
import HotelBook from "../pages/hotel/HotelBook";
import ResetPassword from "../pages/ResetPassword";
import Tourreservations from "../pages/Tourreservations";
import Vehiclereservation from "../pages/Vehiclereservation";
import ReviewTickets from "../pages/train/ReviewTickets";
import ReviewPanel from "../pages/train/ReviewPanel";
import MyTickets from "../pages/train/MyTickets";
import MyOneTicket from "../pages/train/MyOneTicket";
import TravelerHome from "../pages/train/TravelerHome";

import { Main } from "../pages/Main";
import Refund from "../components/Refund";
import RefundReq from "../components/RefundReq";
import RefundUpdate from "../components/RefundUpdate";
import { SalaryCalculation } from "../pages/SalaryCalculation";
import { EmployeeList } from "../pages/EmployeeList";
import { SalarySheet } from "../pages/SalarySheet";
import { FinanceHealth } from "../pages/FinanceHealth";

const RouteTour = () => {
  const ProtectedRoute = ({ children }) => {
    const { user } = useContext(AuthContext);

    if (!user) {
```

```
return (
  <Routes>
    <Route path="/" element={<Home />} />
    <Route path="/login" element={<Login />} />
    <Route path="/reset-password" element={<ResetPassword />} />

    <Route
      path="/admin"
      element={
        <ProtectedRoute>
          <Admin />
        </ProtectedRoute>
      }
    />
    <Route path="/register" element={<Register />} />

    <Route
      path="/users"
      element={
        <ProtectedRoute>
          <Userlist columns={userColumns} />
        </ProtectedRoute>
      }
    />
    <Route
      path="/hotels"
      element={
        <ProtectedRoute>
          <Hotellist columns={hotelColumns} />
        </ProtectedRoute>
      }
    />
    <Route
      path="/tours"
      element={
        <ProtectedRoute>
          <Tourlist columns={tourColumns} />
        </ProtectedRoute>
      }
    />
    <Route
      path="/tourreservation/all"
      element={
        <ProtectedRoute>
```

```
const RouteTour = () => {
          </ProtectedRoute>
        }
      />
      <Route
        path="/train"
        element={
          <ProtectedRoute>
            <Trainlist columns={trainColumns} />
          </ProtectedRoute>
        }
      />
      <Route
        path="/vehicle"
        element={
          <ProtectedRoute>
            <Vehiclelist columns={vehicleColumns} />
          </ProtectedRoute>
        }
      />
      <Route
        path="/vehiclereservation"
        element={
          <ProtectedRoute>
            <Vehiclereservation columns={vehicleReservationColumns} />
          </ProtectedRoute>
        }
      />

      <Route path="/userpage" element={<UserpageA />} />
      <Route path="/update" element={<UpdateuserA />} />
      <Route path="/profile" element={<Profile />} />
      <Route path="/updateProfile" element={<Profileupdate />} />
      <Route path="/adduser" element={<Adduser />} />

      <Route path="/vehicles" element={<VehicleHome />} />
      <Route path="/vehicle/book/:id" element={<VehicleBook />} />
      <Route path="/vehicle/payment/" element={<VehiclePayment />} />
      <Route path="/vehicle/add" element={<AddVehicle />} />
      <Route path="/vehicle/edit/:id" element={<EditVehicle />} />
      <Route path="/vehicle/view/" element={<VehicleView />} />

      <Route path="/tours/home" element={<ToursHome />} />
      <Route path="/tours/:id" element={<TourDetails />} />
      <Route
```

```
188        <Route path="/addtour" element={<AddTourPackage />} />
189        <Route path="/tour/view" element={<TourView />} />
190        <Route path="/tour/update" element={<UpdateTour />} />
191
192        <Route path="/sunandbeach" element={<AllTourCategories />} />
193        <Route path="/hikingandtrekking" element={<AllTourCategories />} />
194        <Route path="/wildsafari" element={<AllTourCategories />} />
195        <Route path="/special" element={<AllTourCategories />} />
196        <Route path="/cultural" element={<AllTourCategories />} />
197        <Route path="/festival" element={<AllTourCategories />} />
198
199        <Route path="/contactus" element={<ContactUs />} />
200
201        <Route path="/add-new-activity" element={<ActivityForm />} />
202        <Route path="/add-new-activity/:id" element={<ActivityForm />} />
203        <Route path="/pending-activities" element={<PendingActivities />} />
204        <Route
205          path="/pending-reservations"
206          element={<PendingReservationsPage />}
207        />
208        <Route path="/events" element={<FilterActivities />} />
209        <Route path="/activities/:id" element={<Activity />} />
210        <Route path="/my-activities" element={<MyActivities />} />
211        <Route path="/my-reservations" element={<ReservationPage />} />
212
213        <Route path="/train/book/:id" element={<TrainBook />} />
214        <Route path="/admintrain/add" element={<AddNewTrain />} />
215        <Route path="/adminTrain" element={<TrainHomeAdmmin />} />
216        <Route path="/adminTrain/:id" element={<SingleTrainView />} />
217        <Route path="/passengerDet" element={<AddPassengerDetails />} />
218        <Route path="/train/book/:id" element={<TrainBook />} />
219        <Route path="/train/update/:id" element={<DoUpdateTrain />} />
220        <Route path="/TrainHome" element={<TravelerHome />} />
221        <Route path="/adminTrain/reviewTicket/:id" element={<ReviewTickets />} />
222        <Route path="/adminTrain/reviewPanel" element={<ReviewPanel />} />
223        <Route path="/train/MyTickets" element={<MyTickets />} />
224        <Route path="/train/MyTickets/:id" element={<MyOneTicket />} />
225        <Route path="/addrestaurant" element={<RestaurentForm />} />
226
227        <Route path="/train/book/:id" element={<TrainBook />} />
228        <Route path="/train/add" element={<AddNewTrain />} />
229        <Route path="/adminTrain" element={<TrainHomeAdmmin />} />
230        <Route path="/adminTrain/:id" element={<SingleTrainView />} />
```

```
86  const RouteTour = () => {
230        <Route path="/adminTrain/:id" element={<SingleTrainView />} />
231        <Route
232          path="/train/book/passengerDet"
233          element={<AddPassengerDetails />}
234        />
235
236        <Route path="/hotelhome" element={<HotelHome />} />
237        <Route path="/hotels/new" element={<AddHotel />} />
238        <Route path="/rooms/new/:id" element={<AddRoom />} />
239        <Route path="/hotels/update/:id" element={<UpdateHotel />} />
240        <Route path="/hotel/:id" element={<HotelView />} />
241        <Route path="/hoteloverview/:id" element={<HotelOverView />} />
242        <Route path="/hoteladmin" element={<HadminView />} />
243        <Route path="/hotelreserve/:id" element={<HotelReserve />} />
244        <Route path="/hotelbooking" element={<HotelBook />} />
245
246        <Route path="/addrestaurants" element={<RestaurentForm />} />
247        <Route path="/finance" element={<Main />} />
248        <Route path="/finance/salary" element={<SalaryCalculation />} />
249        <Route path="/finance/employee" element={<EmployeeList />} />
250        <Route path="/finance/salarySheet" element={<SalarySheet />} />
251        <Route path="/finance/FinanceHealth" element={<FinanceHealth />} />
252        {<Route path="/finance/refund" element={<Refund />} />}
253        {<Route path="finance/addRefund" element={<RefundReq />} />}
254        {<Route path="finance/updateRefund/:id" element={<RefundUpdate />} />}
255      </Routes>
256    );
257  };
258
259  export default RouteTour;
260
```

7. SOFTWARE TESTING

Software testing in the Tours and travel industry ensures that booking platforms, payment gateways, itinerary management, and customer support systems function smoothly. Since travel applications handle complex transactions, real-time updates, and user interactions, rigorous testing is essential.

Types of Testing for Travel Systems

- **Functional Testing** – Verifies that booking, payment, and search functionalities work as expected.
- **Integration Testing** – Ensures seamless communication between third-party services like airlines, hotels, and car rentals.
- **Security Testing** – Protects customer data, payment information, and API interactions from cyber threats.
- **Performance Testing** – Evaluates system responsiveness under high traffic conditions, especially during peak travel seasons.
- **Usability Testing** – Assesses user experience, ensuring intuitive navigation and accessibility..

7.1. Unit Testing

Unit testing in the tours and travel industry helps to establish whether the different software pieces (booking systems, payment gateways, itinerary management, and customer support tools among others) works correctly before the release of the integrated complete system. There are intricate transactions, real time updates, and interaction with users when dealing with travel platforms and for this reason, thorough unit testing is necessary to prevent reliability and efficiency atrophy.

Unit Testing: Important Points of Tours & Travel World Software

- **Booking System Validation** – Flight, Hotel and Tour reservations process verification.
- **Payment Gateway Testing** – Making sure safe transactions and error-coping for a range of payment methods.
- **Itinerary Management** – Ensuring travel plans implement dynamic updates and rant across platforms.
- **User Authentication & Security** – Login, user profiles, and data protection defences' testing.
- API integration – Making smooth transmission of information between third party services such as airlines, hotels and car rentals.

7.2 Integration Testing

The integration testing in the tours and travel industry confirms that various software modules, including booking systems, gateway payment, itinerary management and customer support, function as a single integrated system. Given that various services are intertwined on travel platforms, integration testing is essential to avoid error in transactions, data synchronization and interaction. Main Issues of Integration Testing in Travel Software Integration testing of

travel software deals with testing different components of the developed application. One component of the developed application (e.g., a support ticket) should function correctly as long as it is used in combination with other components.

Key Aspects of Integration Testing in Travel Software

- **Booking System & Payment Gateway** – Making sure reservations and payments function permeating across various providers.
- **API Integration** – Checking the ease of communication between third-party facilities such as airlines, hotels and car rentals.
- **Data Consistency**; Ensuring check that customer details, itineraries and pricing is accurately reflected in platforms.
- **User Experience Validation** – Creation of smooth transition between various modules with no glitches.
- **Security & Authentication** – Login, user profiles and protection of data mechanisms under test. Integration testing ensures travel businesses deliver a seamless experience by identifying issue before they affect customers.

7.3 Functional Testing

The purpose of the functional testing is to check that a software system meets its requirements by testing its features and behaviors. In the tours and travel industry, functional testing has very important role to play in delivering the desired results i.e. booking system, payout gateway, itinerary management and customer support tool should work as specified.

Functional Testing in Travel Software

- **Booking System Validation** – Validating to the users is it possible to make searches, make selection and book flights, hotels and tours without any errors.
- **Payment Processing** (i.e. checking secure transactions in terms of refunds).
- **Search & Filtering** – Ensuring that the users can filter the results by filters such as price, location, and availability.
- Login, account management, and security capabilities testing under User Authentication & Profiles.
- **Third-Party API Integration** – Making it possible to communicate successfully with airlines, hotels and car rental services.

7.4 Security Testing

Testing in tours and travel industry security guarantees that booking platforms, payment gateways and customer's information are secure from cyber threat. Unlike regular applications, travel applications involve handling of this information that is sensitive as well as financial transactions and third party integration and hence robust security testing is important.

Some Major Points on Security Testing in Travel Software

- **Data Protection & Privacy** – Protecting customer's data, payment information, and travel track record stored and sent.

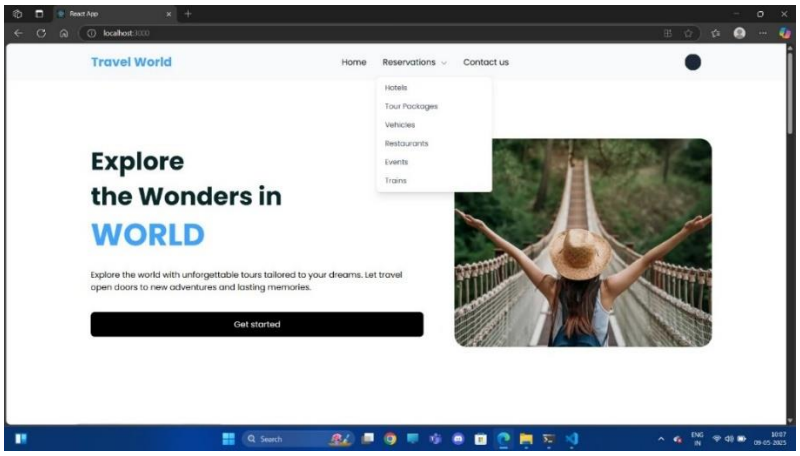
- **Payment Security** – checking cipher protocols and fraud detection instruments for transactions.
- **API Security**–Checking vulnerabilities in third-party integration to airlines, hotels and car-rental services.
- **Authentication && Access Control:** Testings of login procedures, multi factor authentication, and role based access.
- **Network Security** – Detection of vulnerabilities associated with unauthorized access, leaks derelict and denial of service attacks.

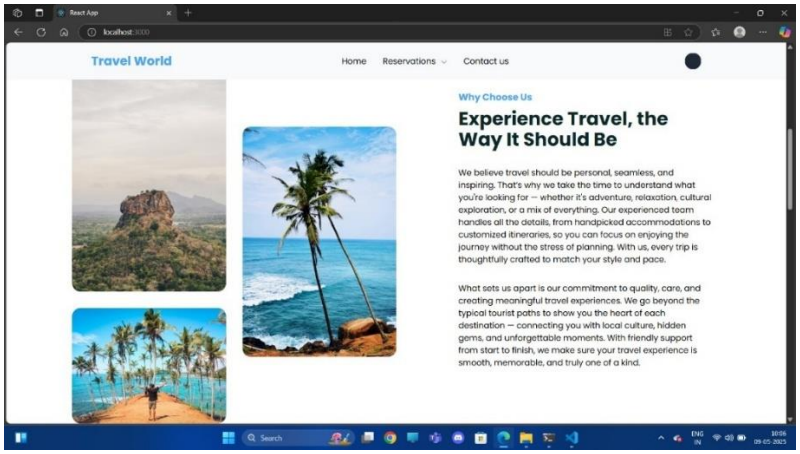
7.5 Testing on system

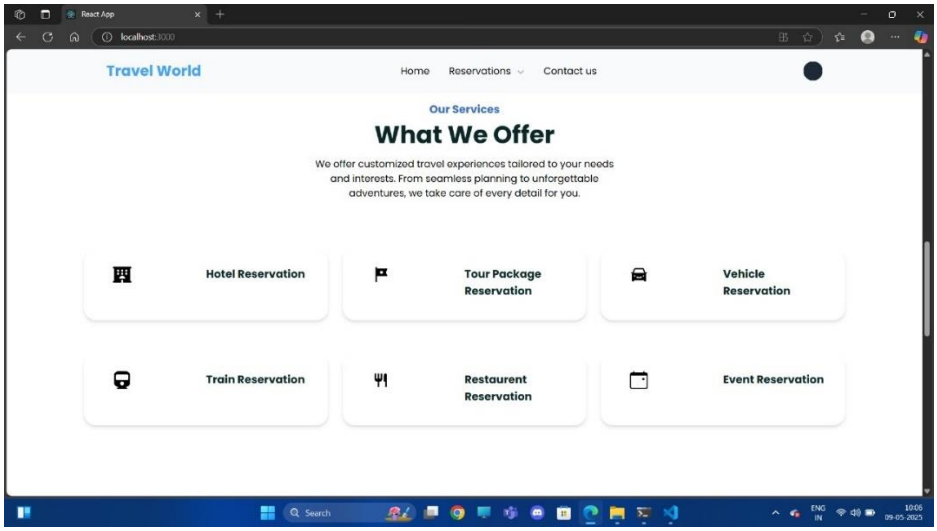
A unit is defined as a single behaviour exhibited by the system under test (SUT), usually corresponding to a requirement[definition needed]. While a unit may correspond to a single function or module (in procedural programming) or a single method or class (in object-oriented programming), functions/methods and modules/classes do not necessarily correspond to units. From the system requirements perspective only the perimeter of the system is relevant, thus only entry points to externally visible system behaviours define units

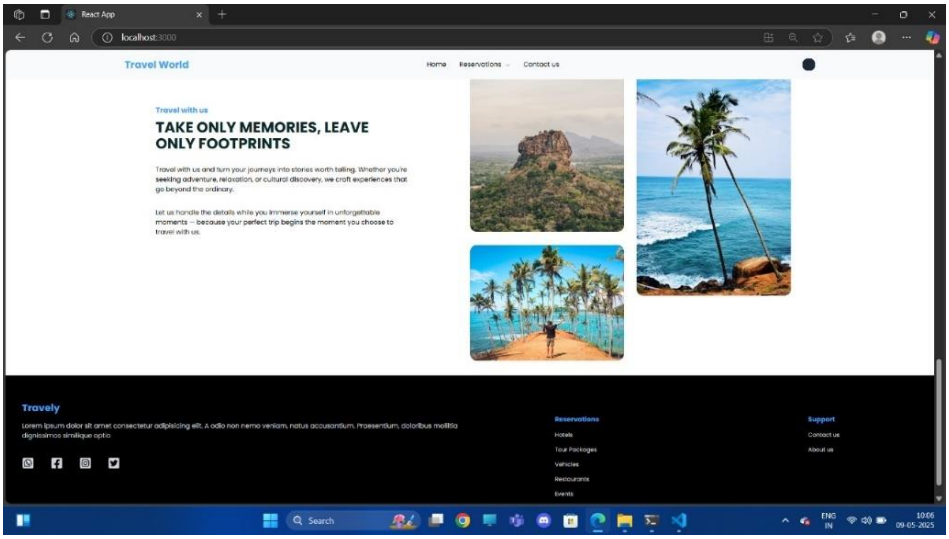
8. RESULTS

8.1 Home

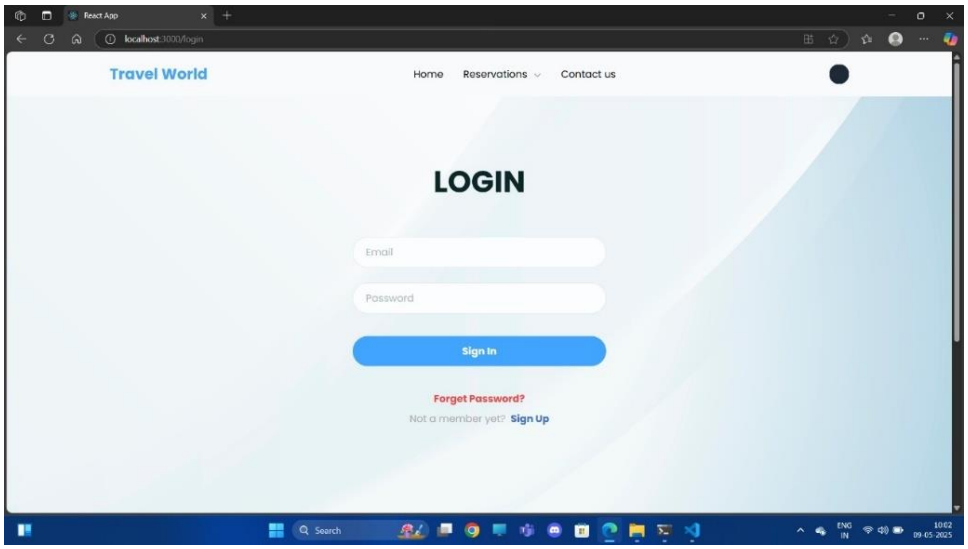


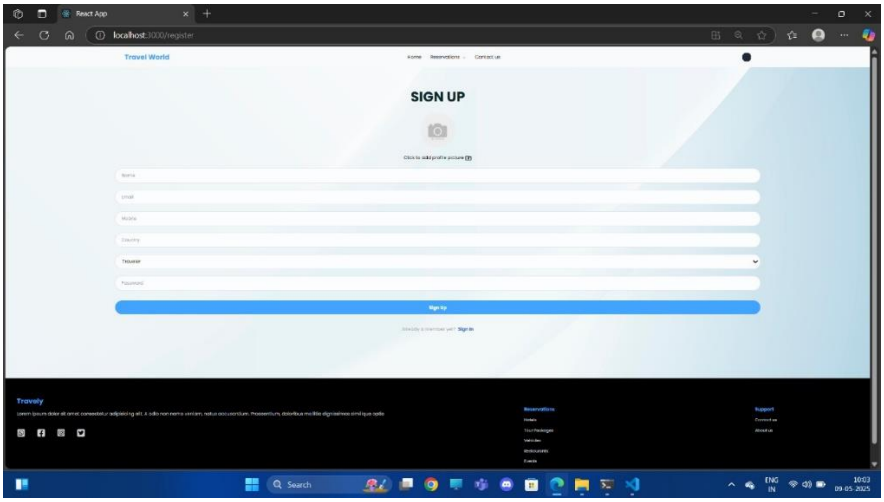




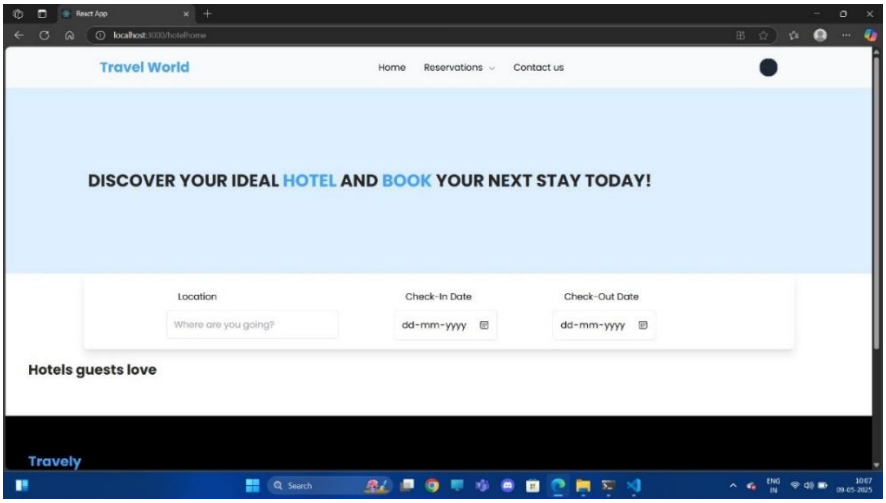


8.2 Sign in & Sign out

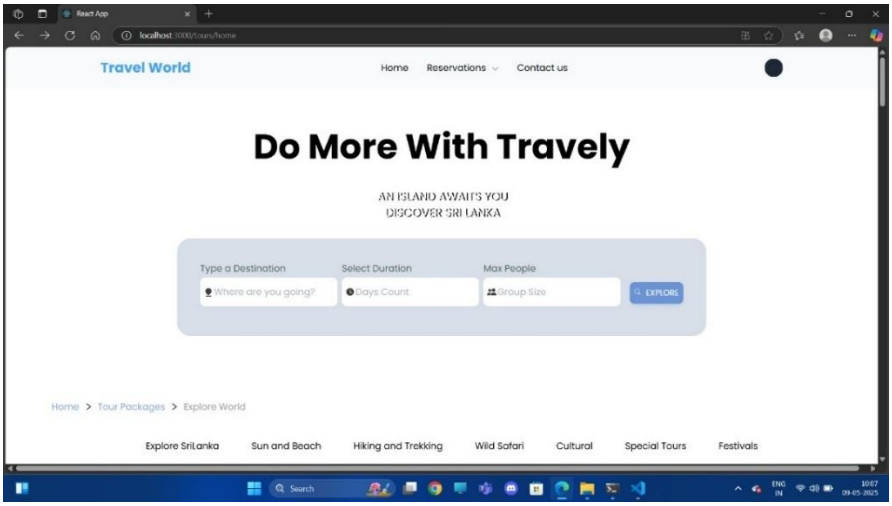


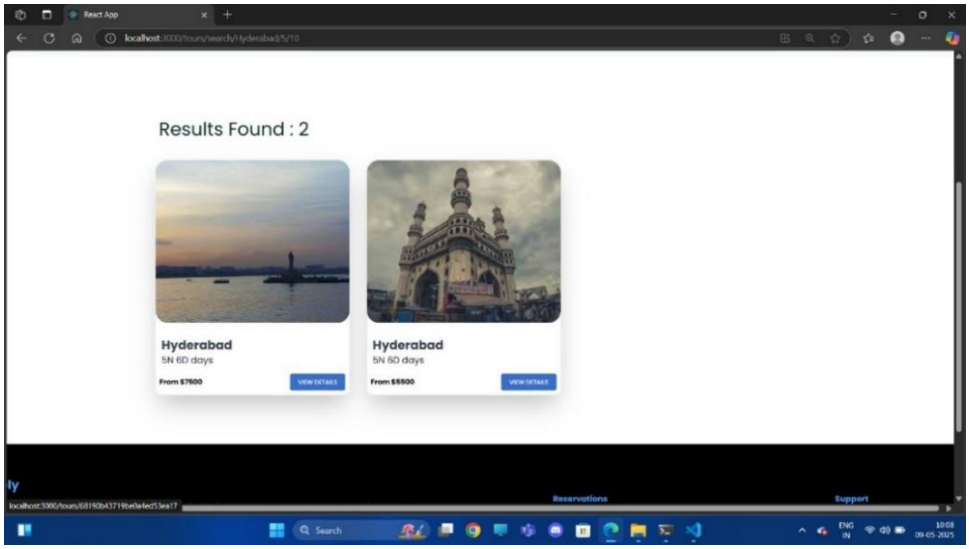


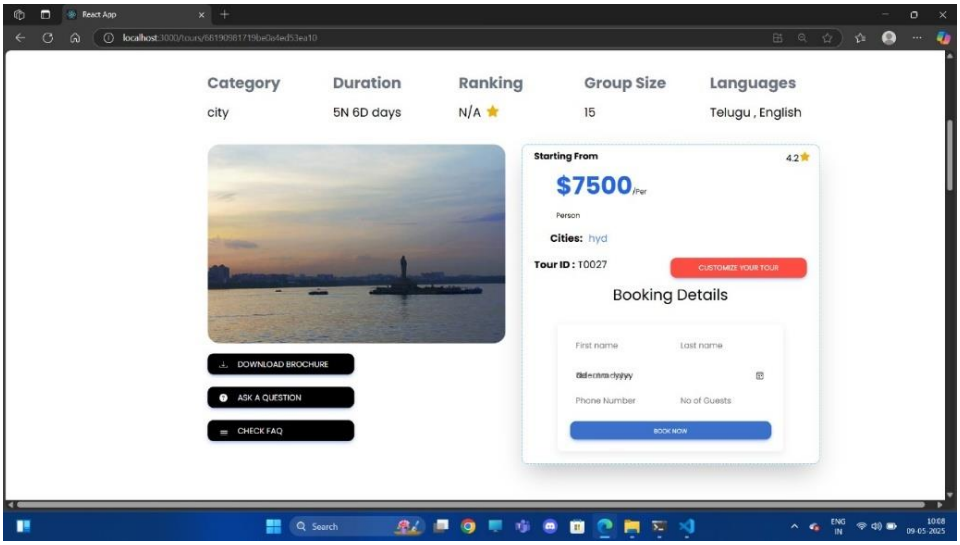
8.2 Hotel Booking



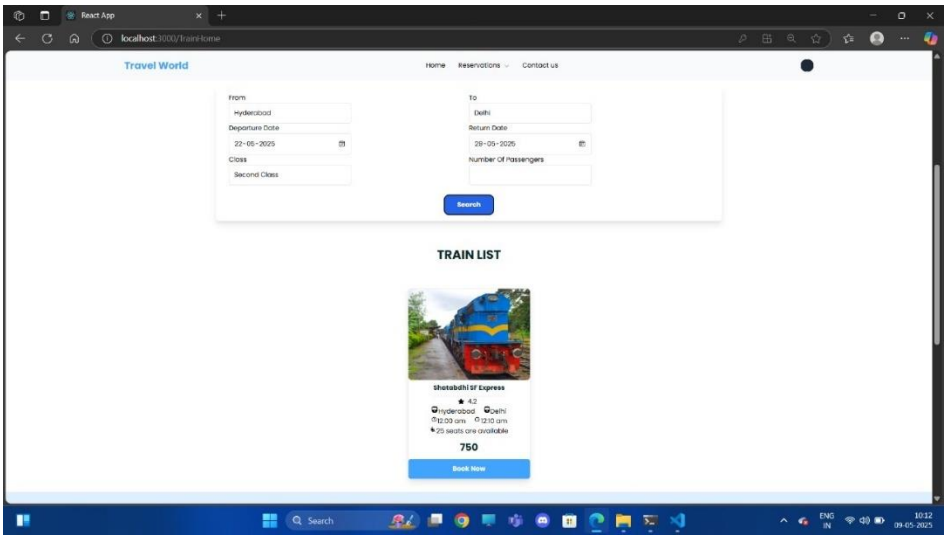
8.3 Tour Booking



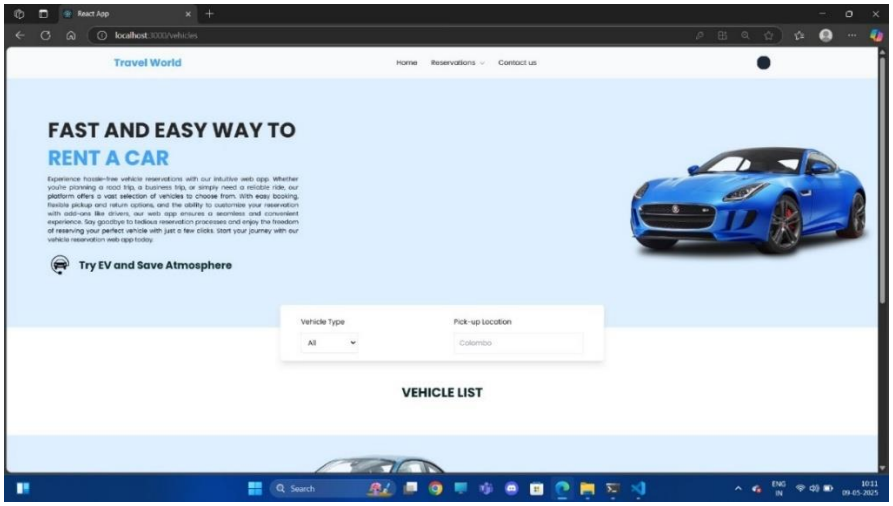




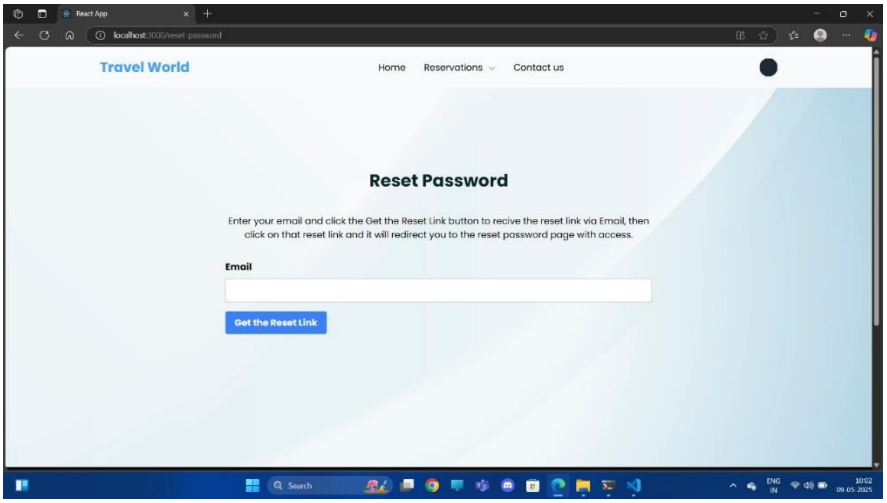
8.4 Train Booking



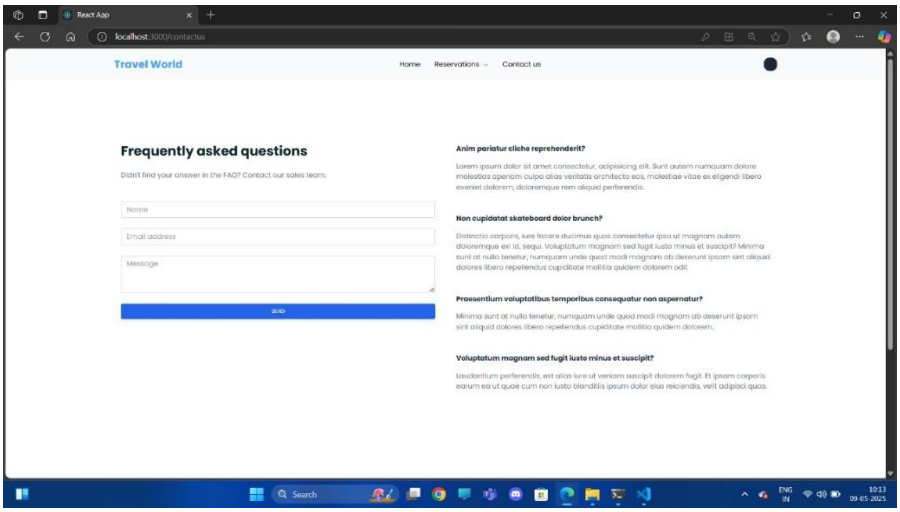
8.5 Vehcile Booking



8.6 Reset Password



8.7 Contant us



9.CONCLUSION AND FUTURE ENHANCEMENTS

9.1 Conclusion

Travel World tours and travels website offers an up-todate, effective, and customerfriendly interface to customers to search, book, and manage tour packages. Built on the MERN stack (MongoDB, Express.js, React.js, Node.js), the system features a responsive frontend interface, secure backend services and a flexible database structure. Users can search and book a detailed tour with payment integration and interact with real-time dynamic content. Admin users enjoy a useful dashboard for handling tours, bookings and customer apps.

This project seems to solve the key issues on the side of the travelers and the service providers helping simplify the travel booking, and enhance the general availability and convenience.

9.2 Future Enchancements

For additional improvement of the system and the fulfillment of increased user expectations, it is proposed to implement the following future changes:

- **AI-Powered Tour Recommendations:** Embed machine models learning to recommend custom tour packages according to people's history, preferences and seasonal trends.
- **Progressive Web App (PWA**): Support Make offline capabilities possible and mobile app-like behavior, which will increase usability in particular to frequent travelers.
- **Multi-Language & Multi-Currency:** Support Integrate translation ability for language and the currency conversion in order to reach the international audience, and increase the scope of its expansion.
- **Voice Search Functionality:** Integrate voice enabled search to increase the interaction of visually impaired users and mobile usability.
- **Dynamic Pricing Engine Automatically:** change tour price depending on demand, availability, season and user location to increase sales and competitiveness.
- **Travel Insurance and Visa Services**: Partner with third party provider to deliver the travel insurance and visa processing as a part of the booking flow.
- **Augmented Reality (AR) Tour Previews**: Give the customers the ability to look through destinations and accommodations virtually prior to booking them.
- **Loyalty & Referral Program**: Introduce the reward point system and referral incentives to enhance user retention as well as the activity level of the platform.

By bringing in these upgrades, Travel World can metamorphose into a smart, scalable travel management ecosystem that gives the users a smooth and enhanced experience planning their travels.

10. BIBLIOGRAPHY

- **Sharma, R. K. (2020).** *Web application development using MERN stack.* New Delhi: Khanna Publishers.
- **Gupta, S., & Mehta, A. (2019).** *E-commerce technologies and applications*. Mumbai: Tech Knowledge Publications.
- **Ministry of Tourism, Government of India. (2023).** *Incredible India*. Retrieved from https://www.incredibleindia.org/.
- Taneja, M. (2022). Digital marketing strategies for travel businesses in India. *Journal of Business and Tourism*, 8(2), 34–42.
- **India Brand Equity Foundation (IBEF). (2023).** *Tourism and Hospitality Industry Report.* Retrieved from https://www.ibef.org
- **Verma, P., & Singh, D. (2022).** *Adoption of MERN stack for scalable web applications. International Journal of Web Engineering*, 5(3), 45–52.
- **Bhatia, A. (2021).** *Full-Stack Web Development with MongoDB, Express, React, and Node.js.* Bengaluru: Tech Press.
- **Joshi, V., & Patel, R. (2020).** *E-commerce Concepts and Applications.* Pune: Nirali Prakashan.
- **Desai, K. (2021).** *Modern Web Development: Full Stack with JavaScript.* New Delhi: BPB Publications.
- **Reddy, S., & Kapoor, A. (2021).** *Leveraging digital marketing in the Indian travel sector. Journal of Tourism Research and Applications*, 6(1), 12–25.
- **Sharma, N. (2020).** *Role of web technologies in the growth of Indian e-commerce. International Journal of Computer Applications*, 177(25), 22–27.