Software Requirements Specification

for

Smart bus ticket booking system

Prepared by Jayashree S

13-09-2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to define the requirements for the *Bus Ticket Booking System* . This system is designed to allow users to conveniently book, cancel, and manage bus tickets online, reducing the dependency on manual ticketing. It will serve both passengers and administrative users by operations such as schedule management, fare calculation, and seat allocation. This document focuses on the core functionalities of the system, including ticket booking, viewing schedules, processing payments, and handling cancellations.

1.2 Document Conventions

This SRS document follows IEEE standard formatting for software specifications. Section titles are written in bold, important terms are *italicized*.

1.3 Intended Audience and Reading Suggestions

This SRS is for admin to develop and handle the Bus Ticket Booking System by understanding the details of the system's functionality and interfaces of the system and for passengers and helper (luggage handler) and drivers .They are suggested to begin with the *Introduction* section to understand the product scope, followed by the Overall Description and Specific Requirements.

1.4 Product Scope

The Bus Ticket Booking System is a web-based application designed to simplify the process of reserving, managing, and canceling bus tickets. It allows passengers to check schedules, select seats, make secure payments, and receive digital tickets. The system also supports administrative tasks like route management, fare settings, and daily reporting. The goal is to reduce manual operations, minimize booking errors, and improve overall user experience.

1.5 References

- Express.js Documentation – https://expressjs.com/
- React.js Official Documentation – https://react.dev/
- Node.js Official Documentation – https://nodejs.org/en/docs
- MongoDB Documentation – https://www.mongodb.com/docs/

2. Overall Description

2.1 Product Perspective

The Bus Ticket Booking System is a web application. It is designed for use by private bus operator to automate and manage ticket reservations, schedules. It can function independently or integrate with third-party payment gateways and notification services.

2.2 Product Functions

- User access (Admin, Driver, Passenger)
- Ticket booking and cancellation
- Seat selection and fare calculation
- Schedule management
- Announcements for delays/cancellations
- Payment reports
- Driver assignment and route mapping

2.3 User Classes and Characteristics

Bus Ticket Booking System → Admin / Driver / Passenger
UC1: Book Ticket

- **Actor:** Passenger
- **Secondary Actor:** Admin
- **Precondition:** Passenger must be logged in
- **Trigger:** Passenger selects "Book Ticket"

Main Success Scenario:

1. Passenger selects route, date, and bus
2. System displays available seats
3. Passenger selects seat and enters passenger details
4. System processes payment
5. System confirms booking and generates e-ticket
6. System updates seat availability and notifies admin

Exception Scenarios:

a) No seats available
➔ System shows unavailability and suggests alternatives
b) Payment failure
➔ System cancels booking and shows retry option
c) Server/network error
➔ System shows error and allows retry

UC2: Cancel Ticket

- **Actor:** Passenger
- **Secondary Actor:** Admin
- **Precondition:** Ticket must be booked
- **Trigger**: Passenger selects "Cancel Ticket"

Main Success Scenario:

1. Passenger selects a ticket from booking history
2. System checks cancellation policy and eligibility
3. System cancels ticket and initiates refund
4. Seat becomes available for booking
5. Passenger and admin receive cancellation confirmation

Exception Scenarios:

a) Cancellation deadline passed
➔ System denies cancellation and notifies passenger
b) Refund process failed
➔ System notifies passenger and marks refund as pending

UC3: Assign Driver to Bus

- **Actor:** Admin
- **Precondition:** Admin must be logged in
- **Trigger:** Admin selects "Assign Driver"

Main Success Scenario:

1. Admin selects a bus and available driver
2. System updates assignment
3. Driver receives assignment notification

Exception Scenarios:

a) No driver available

➔ System shows error and waits for availability

UC4: View Booking Reports

- **Actor:** Admin
- **Precondition:** System contains booking data
- **Trigger:** Admin selects "View Reports"

Main Success Scenario:

1. Admin selects date range or route
2. System fetches and displays booking details
3. Admin views/downloads reports

Exception Scenarios:

a) No data for selection

➔ System shows "No records found"

b) Database error

➔ System displays technical error message

UC5: Check Bus Schedule

- Actor: Passenger
- Precondition: Passenger must be logged in
- Trigger: Passenger selects "Check Schedule"

Main Success Scenario:

1. Passenger selects source and destination
2. System displays list of buses with timing and availability

Exception Scenarios:

a) No buses available

➔ System shows alternative routes or time slots

b) Server timeout

➔ System shows error and retry option

UC6: Post Announcements (Delays, Maintenance, etc.)

- Actor: Admin
- Use Case ID: UC6
- Precondition: Admin must be logged in
- Primary Scenario: Admin posts an announcement
- Trigger: Admin selects "Post Announcement"

Main Success Scenario:

1. Admin writes announcement message
2. System posts announcement to dashboard/feed
3. Affected passengers and drivers receive notification

Exception Scenarios:

a) Network issue
➔ System shows failed notification and allows retry

2.4 Operating Environment

- Web Browsers: Chrome, Firefox, Edge
- Frontend: HTML, CSS, JavaScript, React.js
- Backend: Node.js or Express.js
- Database: MongoDB

2.5 Design and Implementation Constraints

- Secure transactions via HTTPS.
- Mobile-friendly user interface
- Frontend Framework: React.js

2.6 User Documentation

- Online Help via call
- User Manuals (PDF format)

2.7 Assumptions and Dependencies

- Users have access to stable internet connection
- Users have basic familiarity with online ticket booking interfaces
- Application runs on supported browsers

3. External Interface Requirements

3.1 User Interfaces

*The application provides a responsive **web-based GUI** accessible through desktops, tablets, and mobile devices.*

- ***Sample screens**: Login/Signup page, Bus Search, Seat Selection, Booking Confirmation, Admin Dashboard.*
- ***GUI Standards**:*
  - *Consistent color scheme (React + Tailwind CSS).*
  - *Navigation bar on all screens.*
  - *Standard buttons: Book Now, Cancel, Logout, Help.*
  - *Error messages displayed in modal or inline form (e.g., "Invalid password" or "Seat already booked").*
  - *Responsive layout using **Flexbox/Grid** ensuring cross-device usability.*
- ***Shortcuts and UX**:*
  - *Autofocus on primary input fields.*
  - *Real-time validation for fields like email/password.*
- ***Components**:*
  - *User Interface for passengers.*
  - *Admin Interface for bus management.*

3.2 Hardware Interfaces

The system is **web-based** and requires no specialized hardware beyond standard computing devices.

- **Client Device Requirements**:
  - Minimum 2 GB RAM, 1.5 GHz processor, updated web browser.
- **Server Requirements**:
  - Node.js backend hosted on server with at least 4 GB RAM, 2-core CPU.
- **Supported Devices**: Desktops, laptops, smartphones.
- **Communication Protocols**: HTTPS for secure communication.

3.3 Software Interfaces

- **Operating System**: Works on Windows, Linux, or macOS.
- **Frontend**: React.js + Tailwind CSS.
- **Backend**: Node.js with Express.js.
- **Database**: MongoDB (NoSQL document database).

- **Authentication**: JWT for secure user sessions, bcrypt for password hashing.
- **Real-Time Updates**: Socket.IO for instant seat updates.
- **External Libraries**:
    - Mongoose (for MongoDB connection).
    - Axios (for API communication).
    - Nodemon (development tool).

Data flow examples:

- Input: User selects bus → API call → MongoDB retrieves bus details.
- Output: Backend sends bus info & seat status to frontend.

3.4 Communications Interfaces

- **Protocols**: HTTP/HTTPS for API communication.
- **Real-Time Messaging**: Socket.IO (WebSocket protocol).
- **Data Formatting**: JSON used for all client-server communication.
- **Security**:HTTPS encryption,JWT-based secure token transfer.
- **Browser Compatibility**: Chrome, Firefox, Safari, Edge.

4. System Features

4.1 User Authentication and Authorization

4.1.1 Description and Priority:

- Handles user registration, login, and role-based access.
- **Priority**: High.

4.1.2 Stimulus/Response Sequences:

- User enters credentials → Backend verifies → JWT issued → User redirected to dashboard.

4.1.3 Functional Requirements:

- REQ-1: The system must hash passwords using bcrypt before storing.
- REQ-2: The system must issue JWT tokens for login sessions.
- REQ-3: The system must validate expired/invalid tokens and deny access.

4.2 Bus Search and Booking

4.2.1 Description and Priority:

- Allows users to search for buses, view availability, and book seats.
- **Priority**: High.

4.2.2 Stimulus/Response Sequences:

- User enters route and date → System fetches buses → User selects bus and seat → Booking confirmed.

4.2.3 Functional Requirements:

- REQ-4: System must display available buses for entered route/date.
- REQ-5: System must prevent double booking of the same seat.
- REQ-6: System must store booking details in MongoDB.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- System should handle at least **100 concurrent users** with minimal delay.
- API response time must be under **2 seconds** for normal load.
- Real-time updates via Socket.IO should propagate within **1 second**.

5.2 Safety Requirements

- Data loss prevention by maintaining backups of MongoDB.
- Prevent booking conflicts by using **transaction locks** during booking.

5.3 Security Requirements

- Passwords must never be stored in plain text.
- JWT tokens must expire after a set time (e.g., 1 hour).

- Only authenticated users can access booking system.

5.4 Software Quality Attributes

- **Usability**: Simple navigation and mobile-friendly UI.
- **Scalability**: System should scale horizontally with additional servers.
- **Reliability**: Booking system should be consistent and fault-tolerant.
- **Maintainability**: Codebase modular and well-documented.

5.5 Business Rules

- Only registered users can book tickets.
- One seat cannot be booked by multiple users simultaneously.

6. Other Requirements

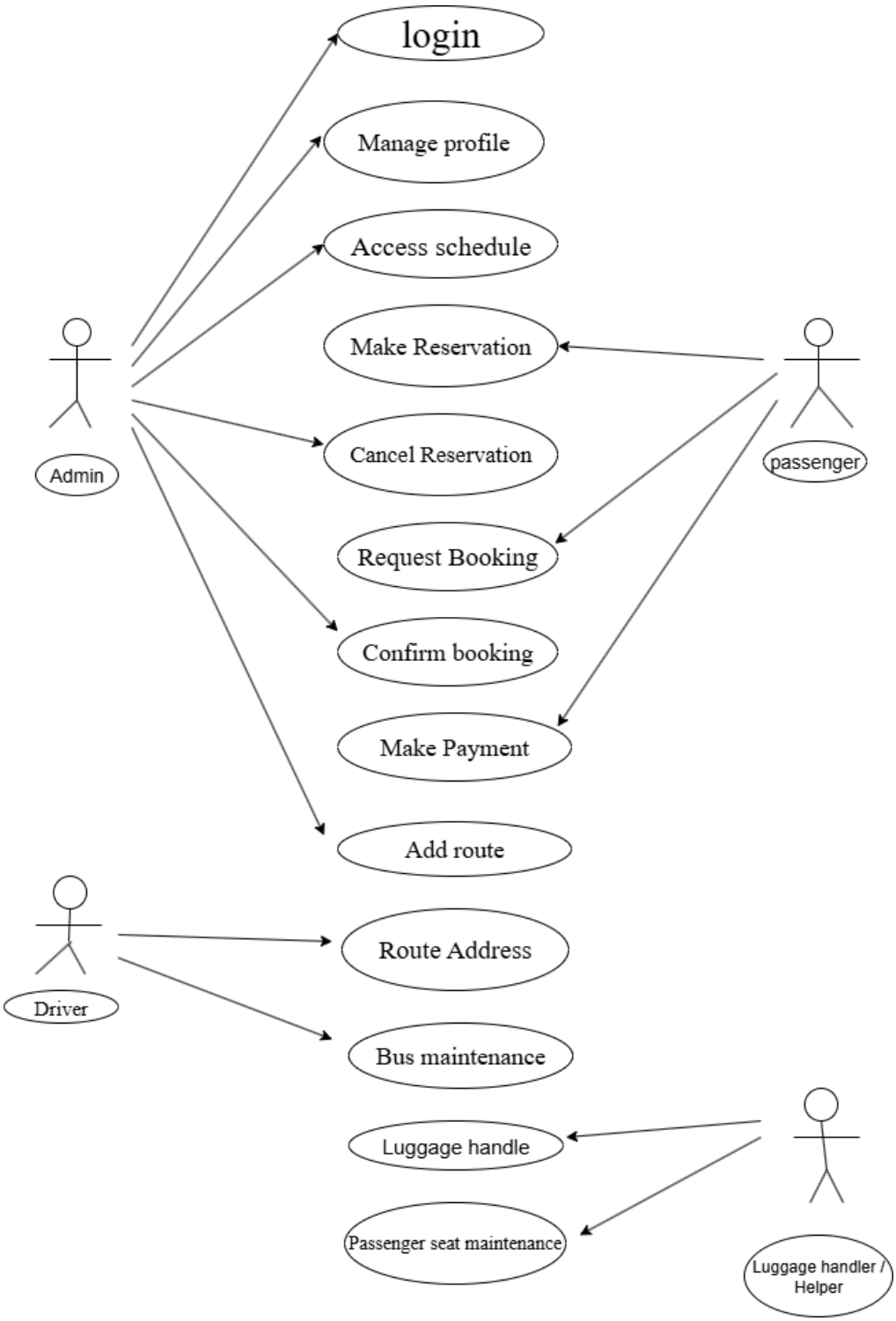
- Database must store **user, bus, route, and booking details**.
- Support for **internationalization** (future enhancement: multiple languages).
- Deployment must comply with **data privacy regulations (GDPR-like)**.

Appendix A: Glossary

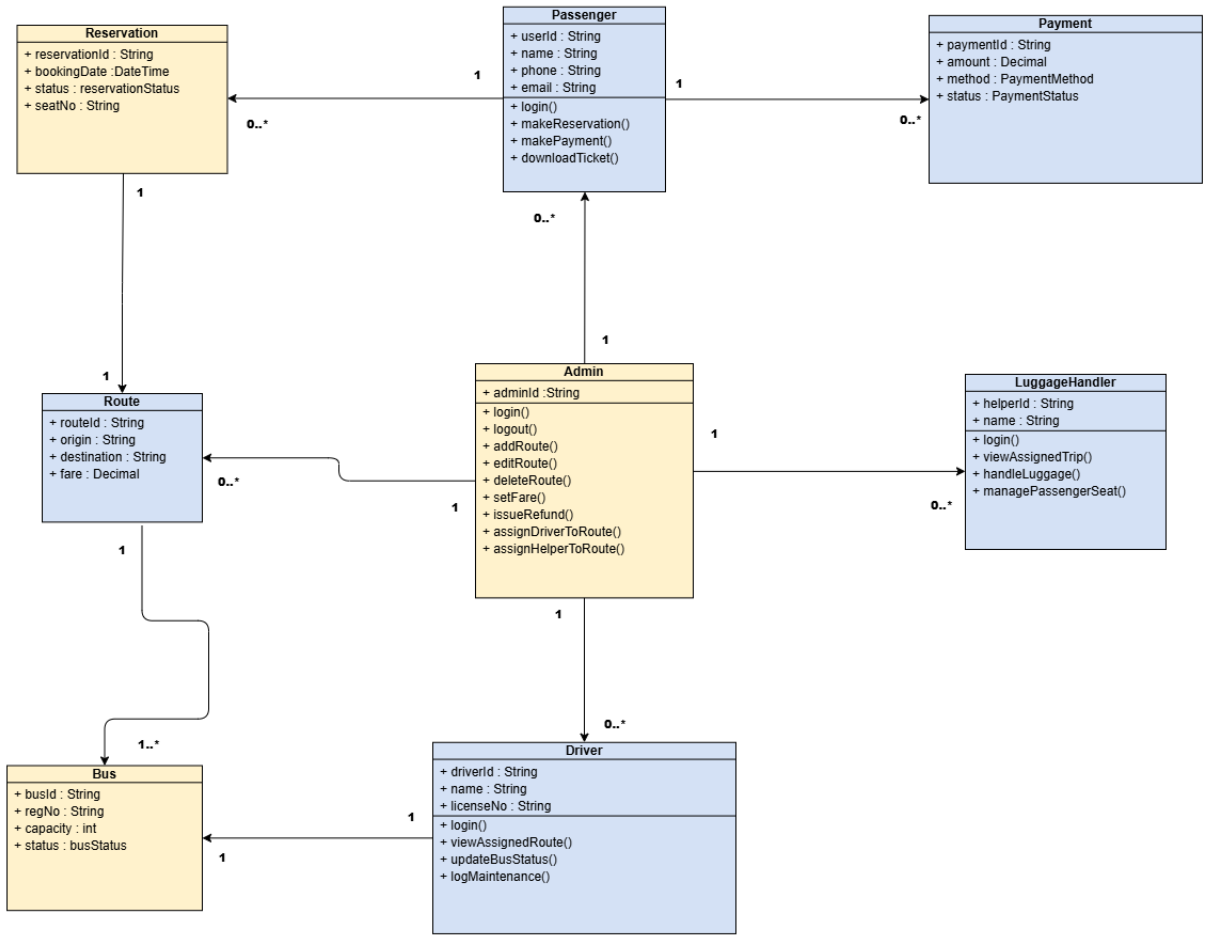
- MERN: MongoDB, Express.js, React.js, Node.js.
- JWT: JSON Web Token, used for secure authentication.
- CRUD: Create, Read, Update, Delete.
- Socket.IO: Library for real-time, bidirectional communication.
- UI/UX: User Interface/User Experience.
- API: Application Programming Interface.

Appendix B: Analysis Models

Use case diagram

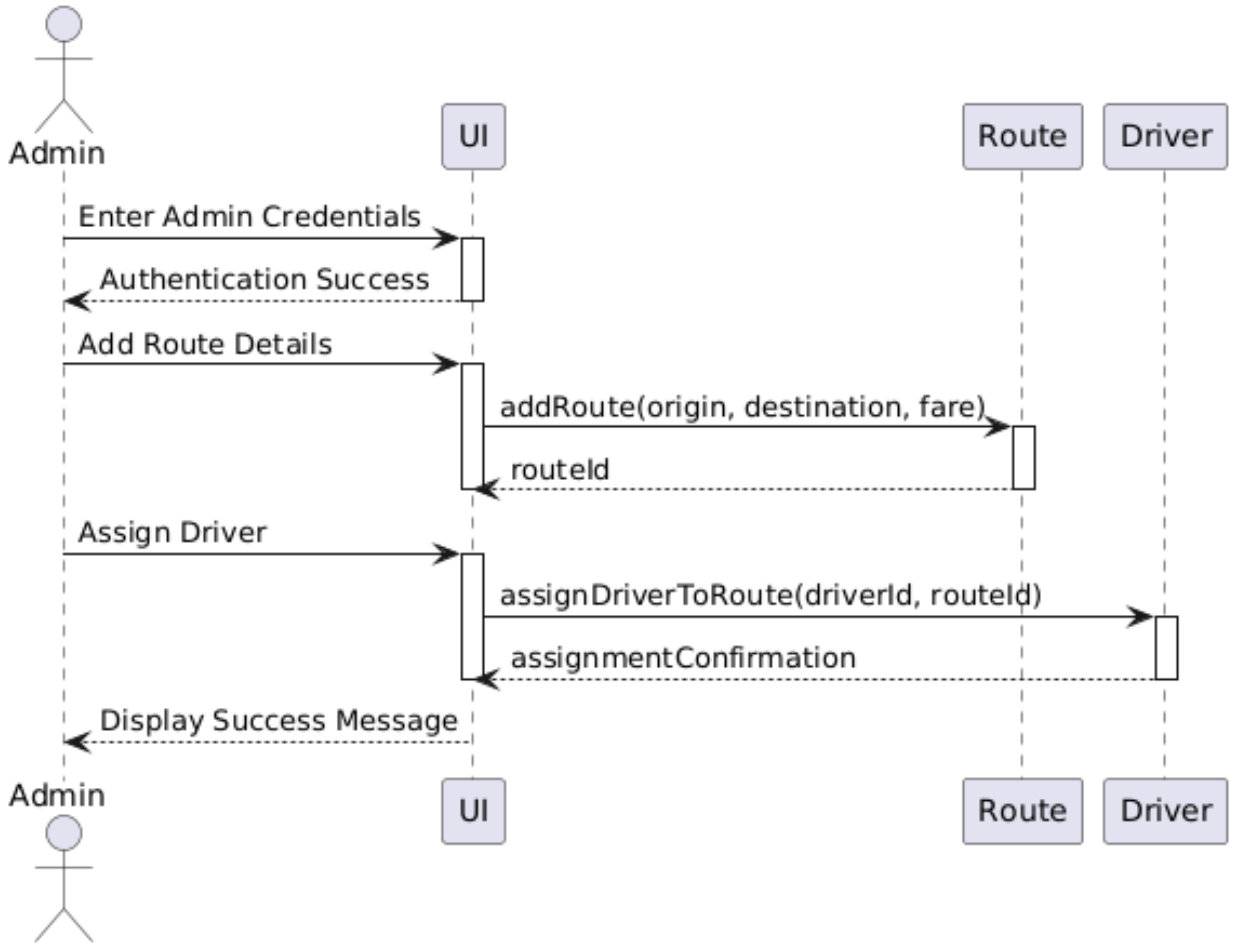


Class diagram:



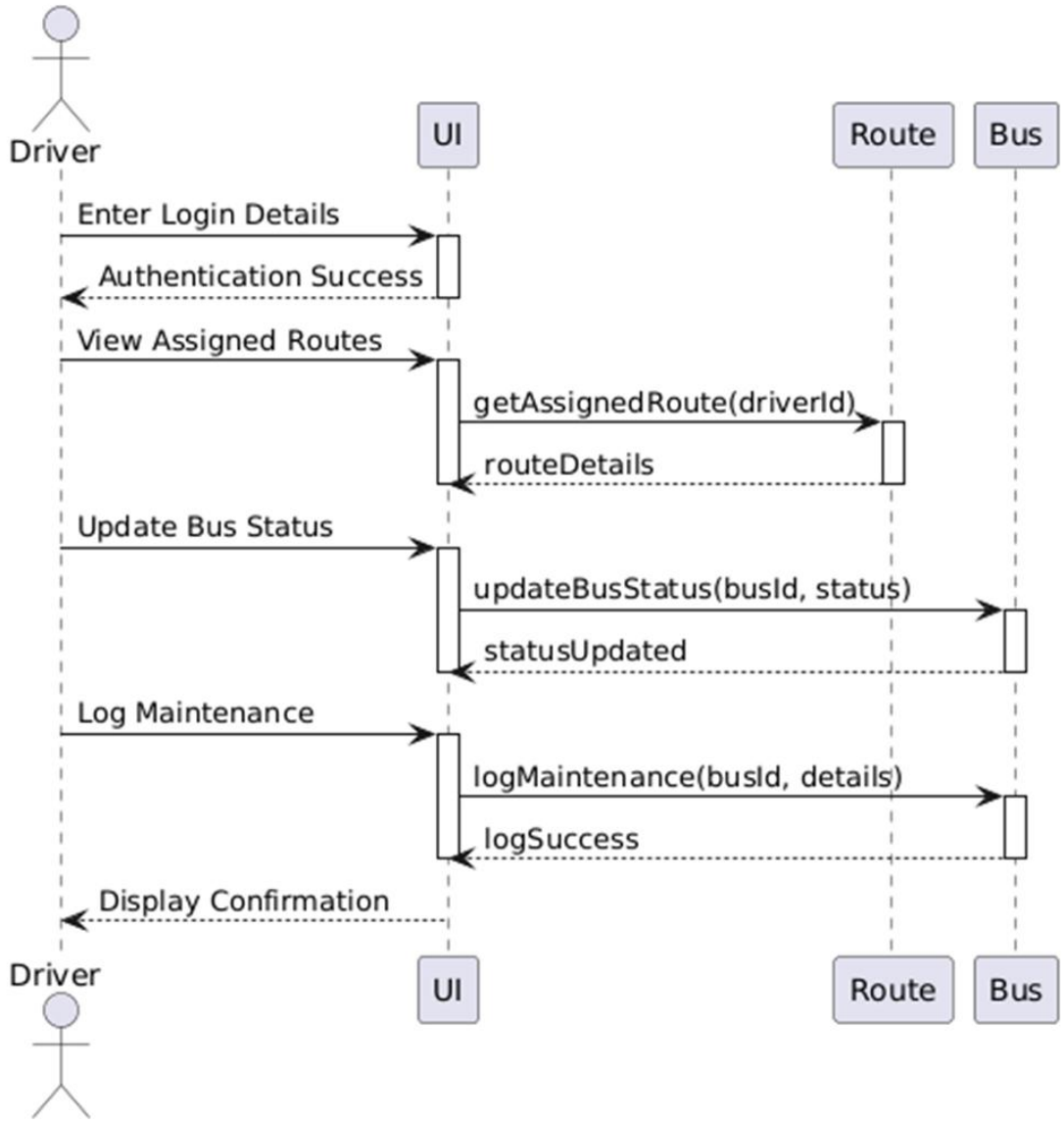
Sequence diagram

ADMIN



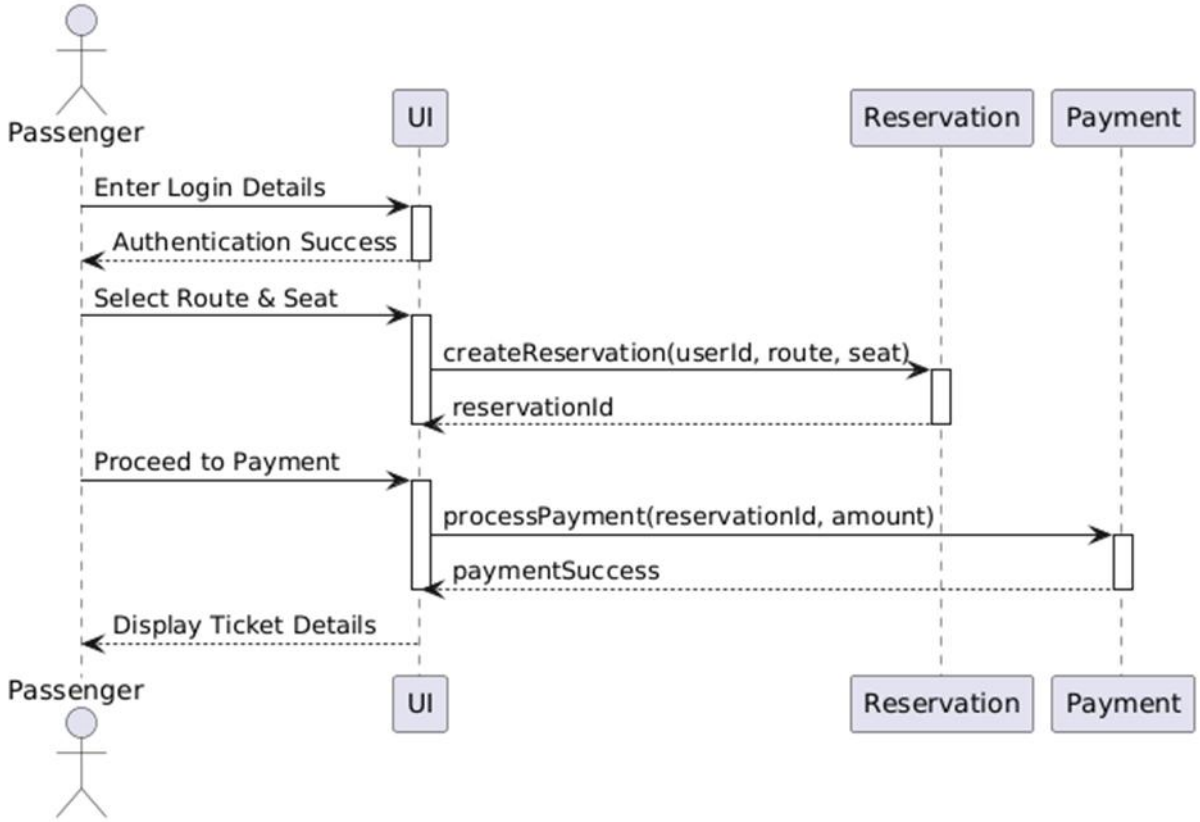
Sequence diagram

DRIVER

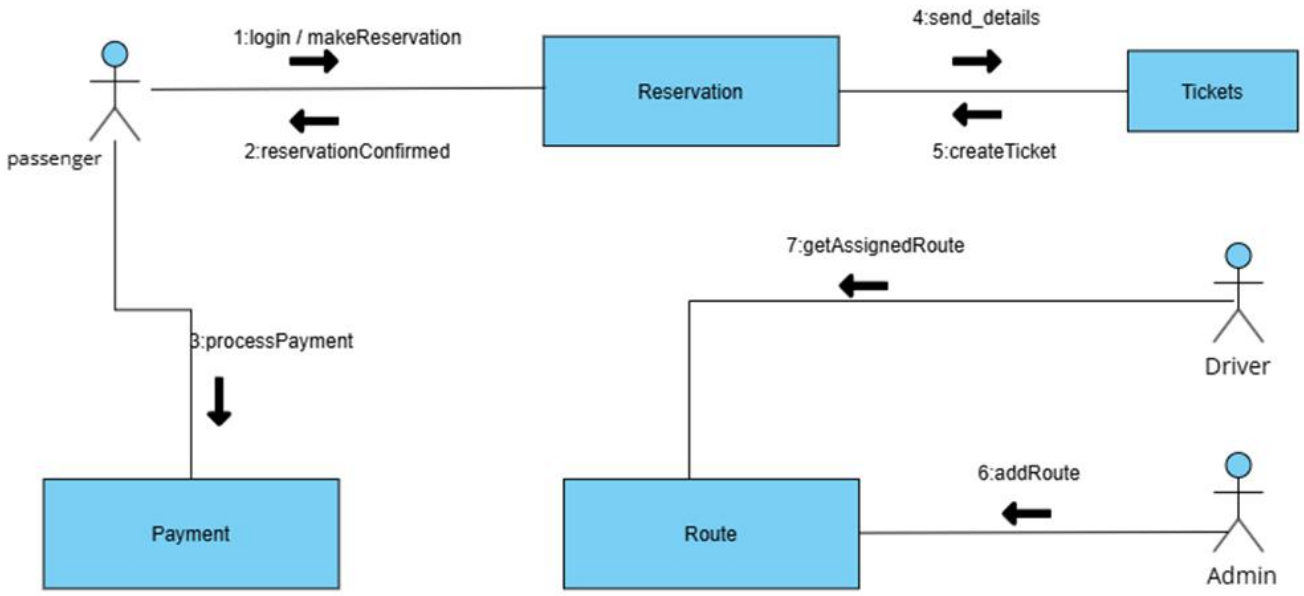


Sequence diagram

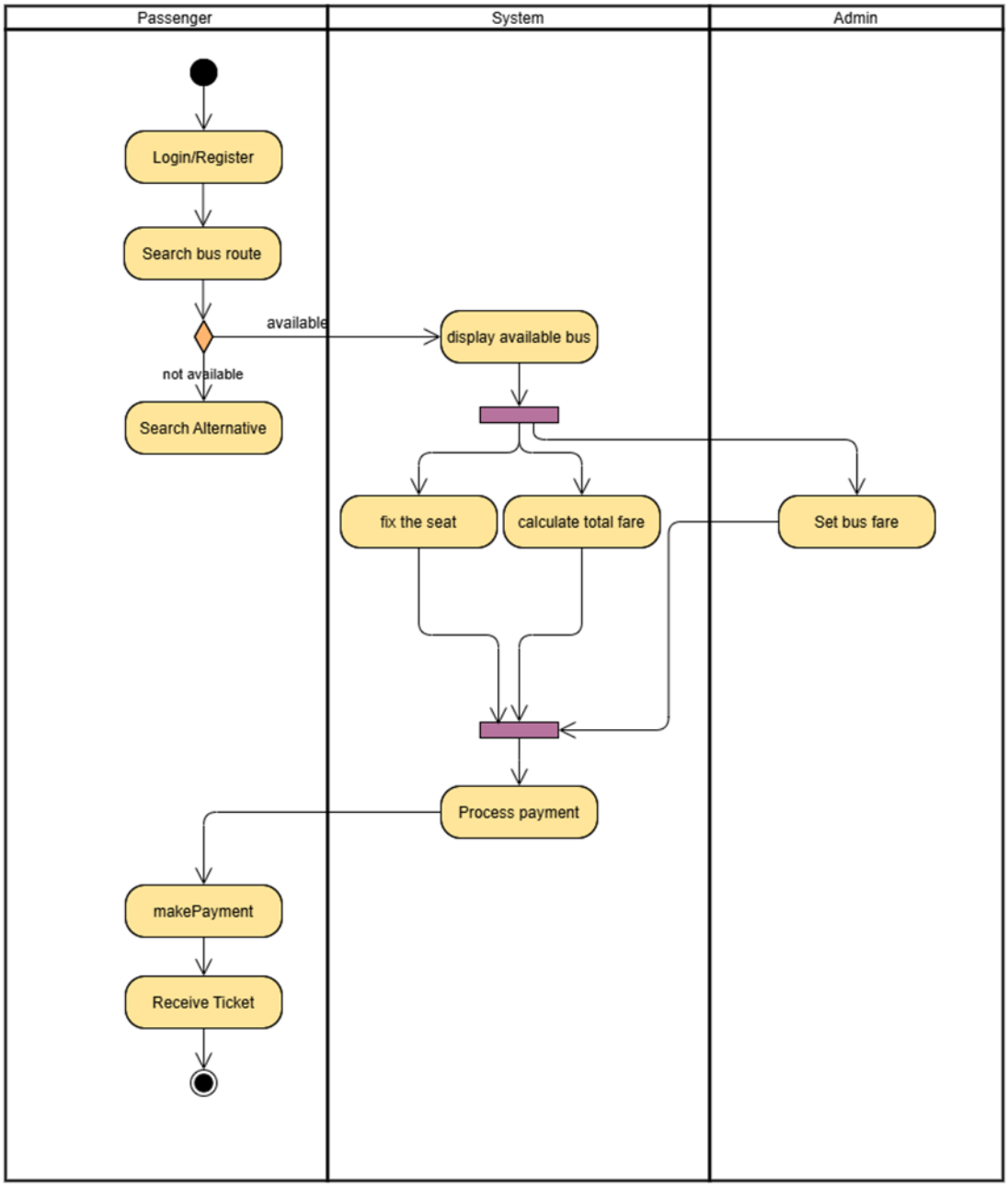
PASSENGER



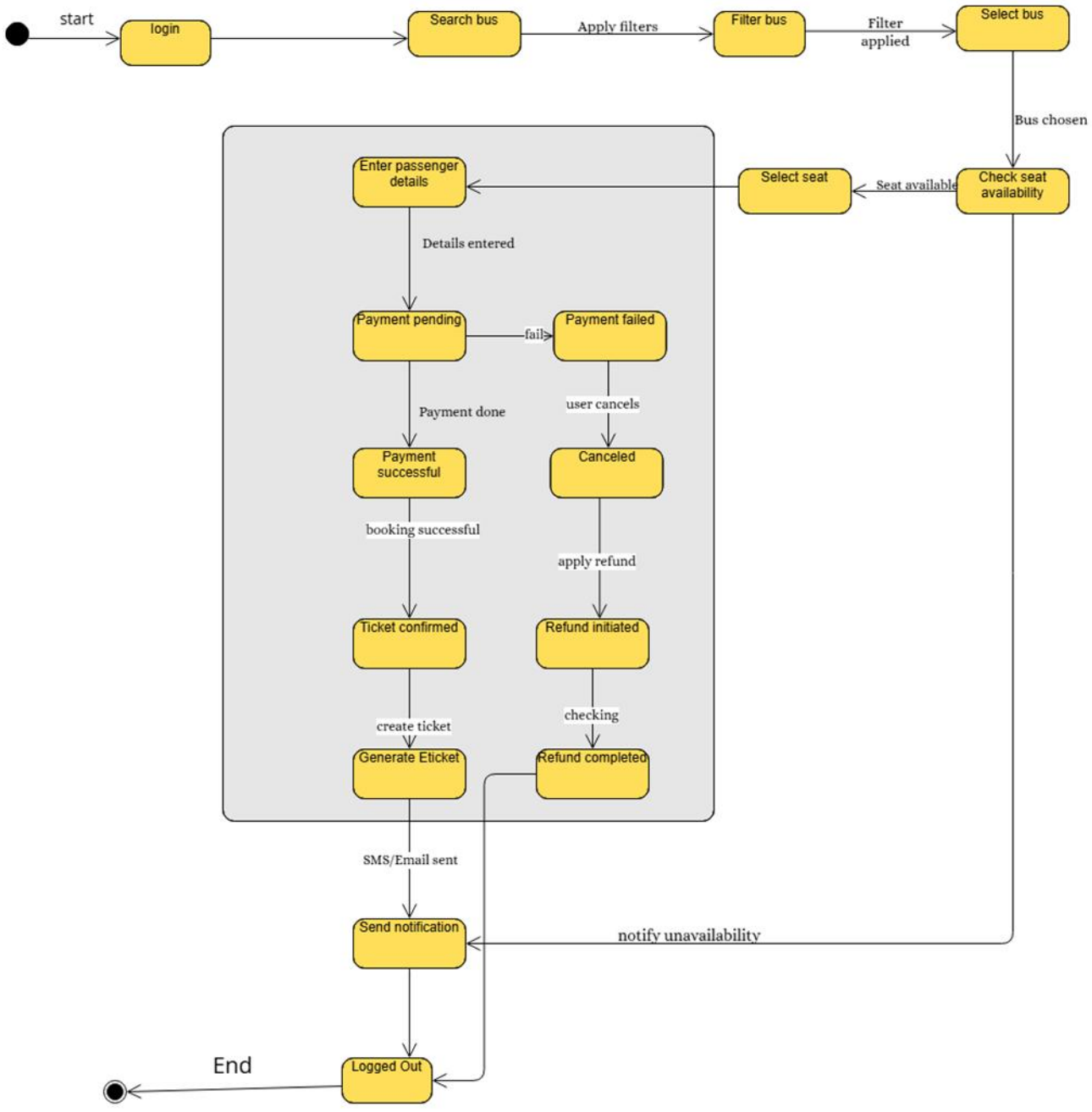
Colloboration diagram



Activity diagram



State chart diagram



Source code

Homepage.js

```
import React from 'react'
import './homepage.css'
export default function Homepage({ history }) {
   const enterSite = e => {
      e.preventDefault()
      history.push('/login')
   }

   return (
      <div className='container maint-cnt'>
         <div className="header-nav">
            <span className="mytext1"> Unique Travels </span>
         </div>

         <div className="">
         </div>

         <div className="container">
            <div className="slogan">
               <h1>
                  <span>always Travel</span>
                  <div className="message">
                     <div className="word1">Uniquely</div>
                     <div className="word2">Safely</div>
                     <div className="word3">with a smile</div>
                  </div>
               </h1>
            </div>

            <a href="/#" onClick={e => enterSite(e)} className="mainBtn">
               <svg width="277" height="62">
                  <defs>
                     <linearGradient id="grad1">
                        <stop offset="0%" stopColor="#FF8282" />
                        <stop offset="100%" stopColor="#E178ED" />
                     </linearGradient>
                  </defs>
                  <rect x="5" y="5" rx="25" fill="none" stroke="url(#grad1)" width="266"
height="50"></rect>
               </svg>
               <span >Get Started!</span>
            </a>
         </div>
      </div>
   )
}
```

Loginfunctions.js

```
import axios from 'axios'

export function logUserIn(userCredentials) {
   let apiUrl = 'http://localhost:8080/login'
   return axios.post(apiUrl,userCredentials, {
      headers: {
         'Content-Type': 'application/json'
      }
   })
}

export function loadRoutes(){
   const authToken = sessionStorage.getItem('authToken' || '')
   let apiUrl = `http://localhost:8080/user/profile?secret_token=${authToken}`
   return axios.get(apiUrl)
}

export function getCurrentUserDetails(authToken){
   const token =  authToken
   let apiUrl = `http://localhost:8080/user/profile?secret_token=${token}`
   return axios.get(apiUrl)
}
```

TicketPage.js

```
import React from 'react'
import './TicketPage.css'
export default function TicketPage({ history }) {

   const handleSignOut = e => {
      e.preventDefault()
      sessionStorage.removeItem('authToken')
      localStorage.removeItem('reservedSeats')
      localStorage.removeItem('nameData')
      localStorage.clear()
      history.push('/')
   }
   const handleBookAgainIcon = e => {
      e.preventDefault()
      history.push('/routes')
   }
   const getLocationData = () => {
      let from = localStorage.getItem("start")
      let to = localStorage.getItem("destination")
      return (
         <div>
            <p>From:  {from}</p>
            <p>To:  {to}</p>
         </div>
      )
   }
```

```
    const getPassengerName = () => {
        let nameArray = localStorage.getItem("nameData")
        let names = JSON.parse(nameArray)
        return names.map((name, idx) => {
            return (
                <div key={idx}>
                    <p className="names">{name}</p>
                </div>
            )
        })
    }
    const getSeatNumbers = () => {
        let noArray = localStorage.getItem("reservedSeats")
        let arr = JSON.parse(noArray)
        return arr.map((element, idx) => {
            return (
                <div key={idx}>
                    <p classsName="seatNo">{element}</p>
                </div>
            )
        })
    }
    const getIdNumber = () => {
        let tokenData = localStorage.getItem("selectedBusId")
        return (
            <p className="idData">
                {tokenData}
            </p>
        )
    }
    const getDateValue = () => {
        let dat = localStorage.getItem("date")
        return <p>On: {dat}, 10 AM (Hourly commute)</p>
    }
    return (

        <div className="container">
            <div>
                <nav className="mb-4 navbar navbar-expand-lg navbar-dark bg-unique hm-gradient">
                    <a href="/#" className="navbar-brand Company-Log">UT</a>
                    <button className="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarSupportedContent-3" aria-controls="navbarSupportedContent-3" aria-expanded="false" aria-
label="Toggle navigation">
                        <span className="navbar-toggler-icon"></span>
                    </button>
                    <div className="collapse navbar-collapse" id="navbarSupportedContent-3">
                        <ul className="navbar-nav ml-auto nav-flex-icons ic">
                            <li className="nav-item">
                                <a href="/#" className="nav-link waves-effect waves-light" onClick={e =>
handleBookAgainIcon(e)}>Book Again</a>
                            </li>
                            <li className="nav-item">
                                <a href="/#" className="nav-link waves-effect waves-light" onClick={e =>
handleSignOut(e)}>Sign-Out</a>
```

```
                    </li>
                  </ul>
                </div>
              </nav>
            </div>
            <div className="tpMain">
              <article className="ticket">
                <header className="ticket__wrapper">
                  <div className="ticket__header">
                    1  UNIQUE TRAVELS
                  </div>
                </header>
                <div className="ticket__divider">
                  <div className="ticket__notch"></div>
                  <div className="ticket__notch ticket__notch--right"></div>
                </div>
                <div className="ticket__body">
                  <section className="ticket__section">
                    {getLocationData()}
                    {getSeatNumbers()}
                    <p>Your seats are together <span>{getDateValue()}</span></p>
                  </section>
                  <section className="ticket__section">
                    <h3>Passenger Names</h3>
                    {getPassengerName()}
                  </section>
                  <section className="ticket__section">
                    <h3>Payment Method</h3>
                    <p>Credit Card</p>
                  </section>
                </div>
                <footer className="ticket__footer">
                  <p>Transaction-ID</p>
                  {getIdNumber()}
                </footer>
              </article>
            </div>

        </div>

    )
}
```

App.js

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';
import 'bootstrap/dist/css/bootstrap.min.css'
import Homepage from './components/Homepage/Homepage'
import RouteSelection from './components/RouteSelection/RouteSelection'
import LogOrsign from './components/Login-Signup/LogOrsign'
import Signup from './components/Login-Signup/Signup'
import Profile from './components/Profile/Profile'
import TicketPage from './components/TicketPage/TicketPage'
```

```
import './App.css';

function App() {
  return (
    <div className="App">
      <Router>
        <Switch>
          <Route path="/" exact render={props => <Homepage {...props} />} />
          <Route path="/login" render={props => <LogOrsign {...props} />} />
          <Route path="/register" render={props => <Signup {...props} />} />
          <Route path="/routes" exact render={props => <RouteSelection {...props} />} />
          <Route path="/profile" exact render={props => <Profile {...props} />} />
          <Route path="/getTicket" exact render={props => <TicketPage {...props} />} />
        </Switch>
      </Router>
    </div>

  );
}

export default App;
```

Models
Buses.js

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

const BusSchema = new Schema({
    companyName: {
        type: String
    },
    busType: {
        type: String
    },
    busNumber: {
        type: String
    },
    startCity: {
        type: String
    },
    destination: {
        type: String
    },
    totalSeats: {
        type: String
    },
    availableSeats: {
        type: String
    },
    pricePerSeat: {
        type: String
    }
}, {collection: "buses"})
```

```
const bus = mongoose.model('bus', BusSchema)

module.exports = bus;
```

User.js

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

const UserSchema = new Schema({
   name: {
      type: String,
      required: true,
   },
   email: {
      type: String,
      required: true,
   },
   password: {
      type: String,
      required: true,
   },
   mobile: {
      type: String,
      required: true,
   },
   gender: {
      type: String,
      required: true,
   },
   dob: {
      type: Date,
      required: false,
   },
});

const User = mongoose.model("user", UserSchema);

module.exports = User;
```

passport.js

```
const passport = require("passport");
const localStrategy = require("passport-local").Strategy;
const User = require("../models/User");

passport.use(
   "login",
   new localStrategy({
         usernameField: "email",
         passwordField: "password",
      },
      async(email, password, done) => {
```

```
        try {
          const user = await User.findOne({ email });
          if (!user) {
            return done(null, false, { message: "User not found" });
          }

          const validate = await user.isValidPassword(password);
          if (!validate) {
            return done(null, false, { message: "Wrong Password" });
          }
          return done(null, user, { message: "Logged in Successfully" });
        } catch (error) {
          return done(error);
        }
      }
    )
);
```

login.js

```
const express = require("express");
const passport = require("passport");
const User = require("../models/User");
const jwt = require("jsonwebtoken");
var bcrypt = require("bcrypt");

const router = express.Router();

router.post("/login", async(req, res, next) => {
  const { email, password } = req.body;
  try {
    User.findOne({ email: email }, (err, doc) => {
      console.log(doc);
      if (err) {} else {
        if (!doc) {} else {
          bcrypt.compare(password, doc.password, function(error, response) {
            console.log(response);
            const token = jwt.sign({ doc }, "top_secret");
            res.status(200).json({ token });
          });
        }
      }
    });
  } catch (error) {}

});

module.exports = router;
```

register.js

```
var express = require('express');
var router = express.Router();
var User = require('../models/User')
var bcrypt = require('bcrypt');
var moment = require('moment');
var bodyParser = require('body-parser')
router.get('/', (req, res) => {
   res.send("Register Here")
});
var jsonParser = bodyParser.json()

router.post('/', jsonParser, async (req, res) => {
   //Hash Password
   const hashPassword = await bcrypt.hash(req.body.password, 10)
   let user = {
      name: req.body.name,
      email: req.body.email,
      password: hashPassword,
      mobile: req.body.mobile,
      gender: req.body.gender,
      dob: moment(req.body.dob).format('YYYY-MM-DD')
   }
   let newUser = new User(user)
   newUser.save((err, reslut) => {
      if (err) console.log(err)
      else res.status(201).json(reslut)
   })
});
module.exports = router;
```

routeSelection.js

```
var express = require('express');
var router = express.Router();
var bus = require('../models/Buses');
router.post('/', (req, res) => {

   bus.find({ 'startCity': req.body.startCity, 'destination': req.body.destination }).exec((err, bus) => {
      if (err) {
         res.json({ status: false, message: "error while searching" })
      }
      else res.json({ bus })
   })
})

router.post('/', (req, res) => {

   bus.findOne({ _id: req.body.bId }, (err, bus) => {
      if (err) {
         res.json({ status: false, message: "error while searching with ID" })
      }
```

```
        else
            res.json({ bus })
    })
})

module.exports = router;
```

app.js

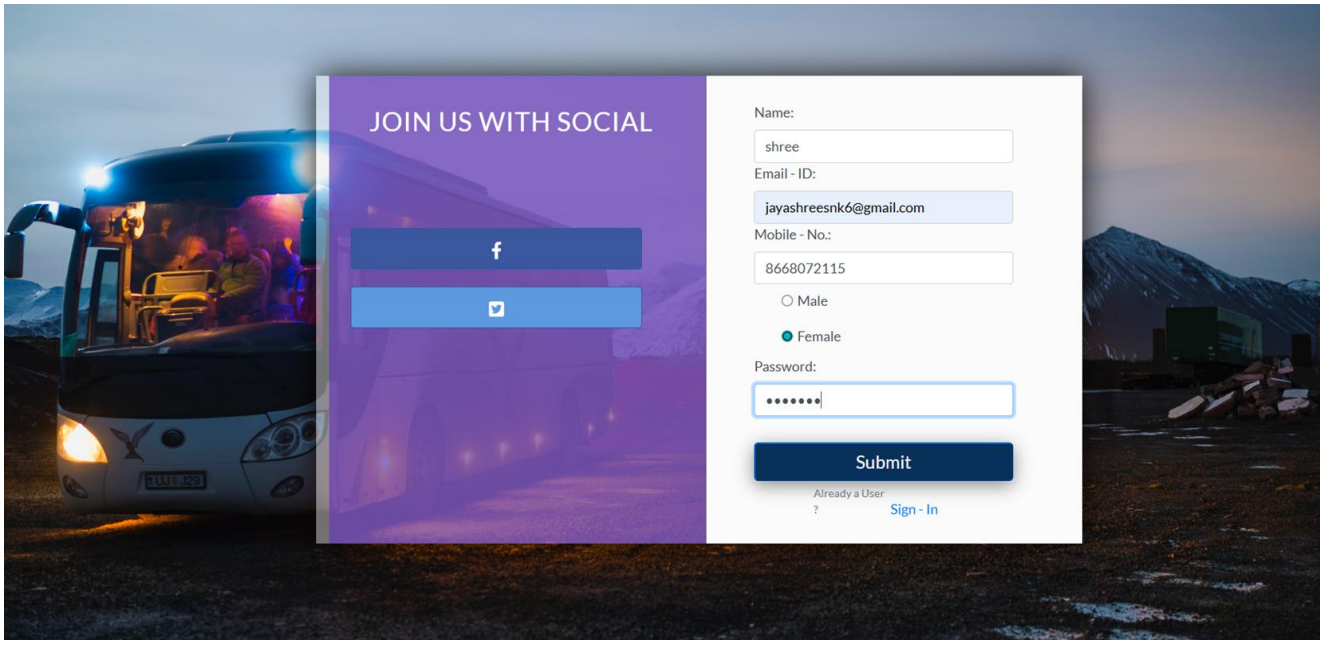
```
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');
var mongoose = require('mongoose');
var passport = require('passport');
const cors = require('cors')
var app = express();
require('./auth/auth');
const login = require('./routes/login')
const loggedInPage = require('./routes/loggedInUser');
const bookingRoute = require('./routes/routeSelection')
var registerRouter = require('./routes/register');
const DB_URL = require('./config/keys').MongoURI;
mongoose.connect(DB_URL, {
    useNewUrlParser: true,
    useUnifiedTopology: true
})
    .then(() => {
        console.log("Connected to MongoDB")
    })
    .catch(err => {
        throw err
    })
app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
app.use(cors())
app.use('/', login);
app.use('/booking', bookingRoute);
app.use('/register', registerRouter);
app.use('/user', passport.authenticate('jwt', { session: false }), loggedInPage);

module.exports = app;
```

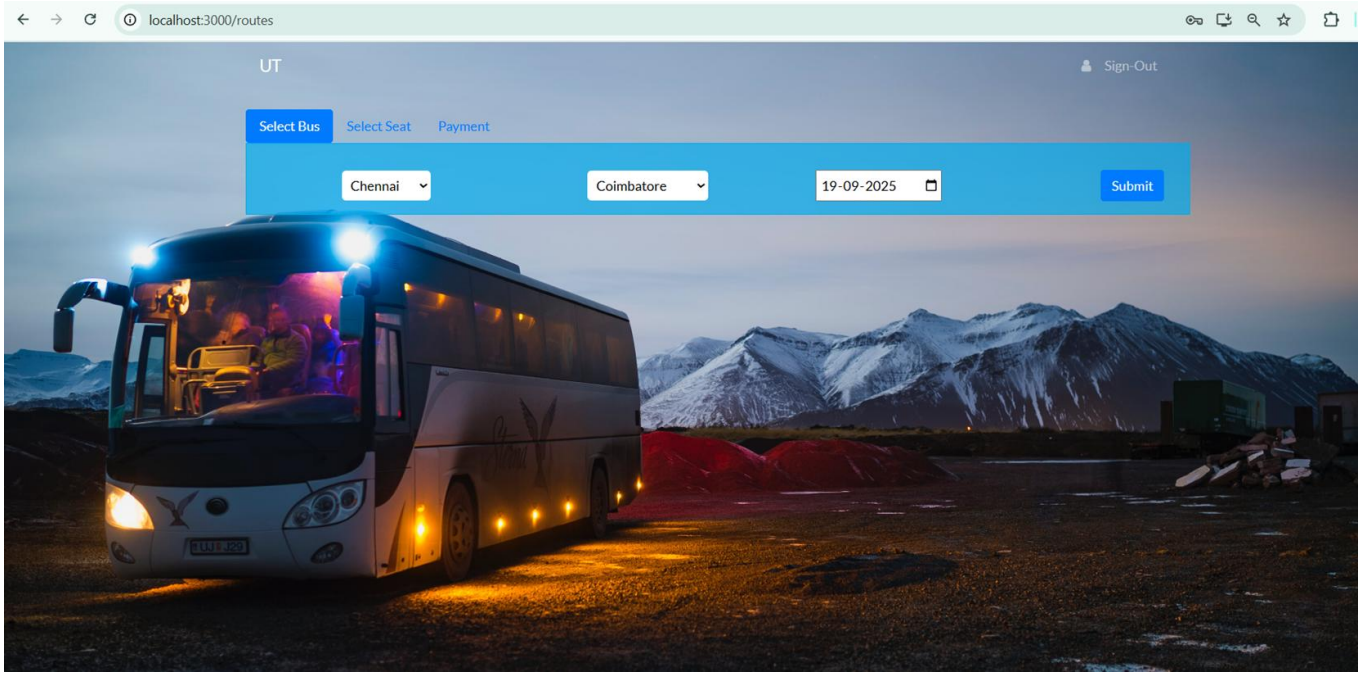
OUTPUT :

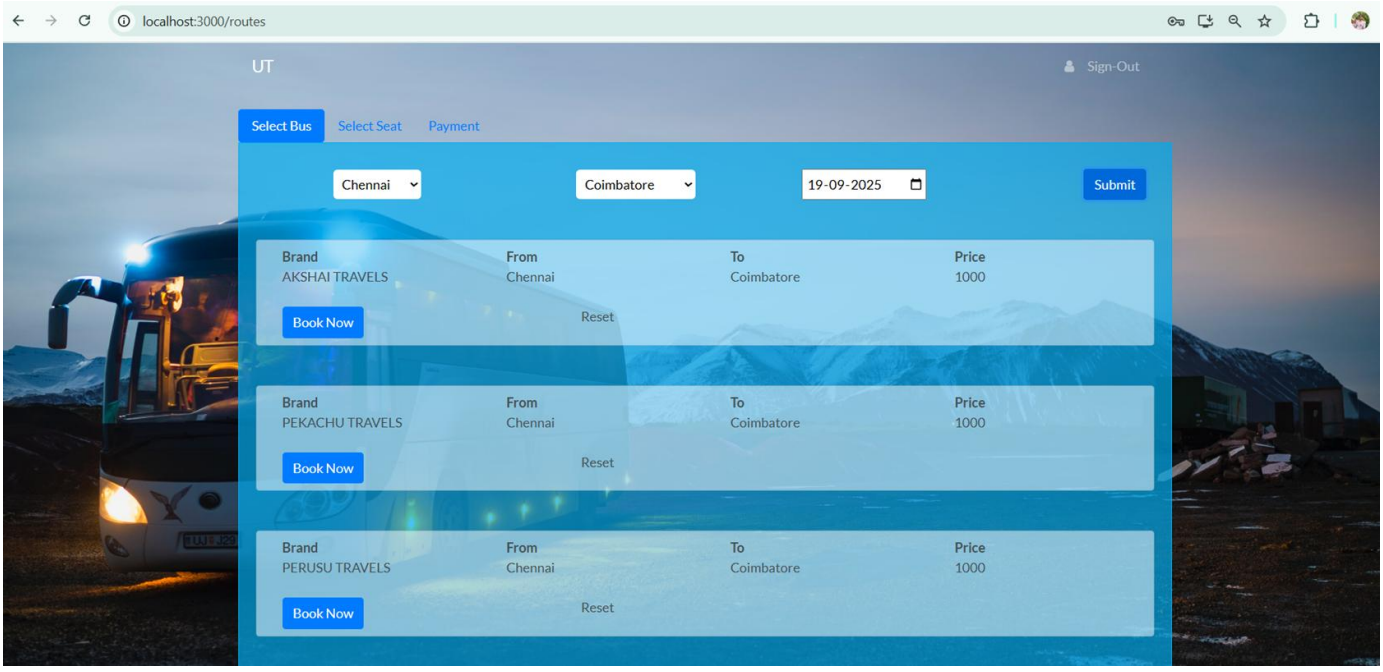
7.1 Homepage

7.2 Login page

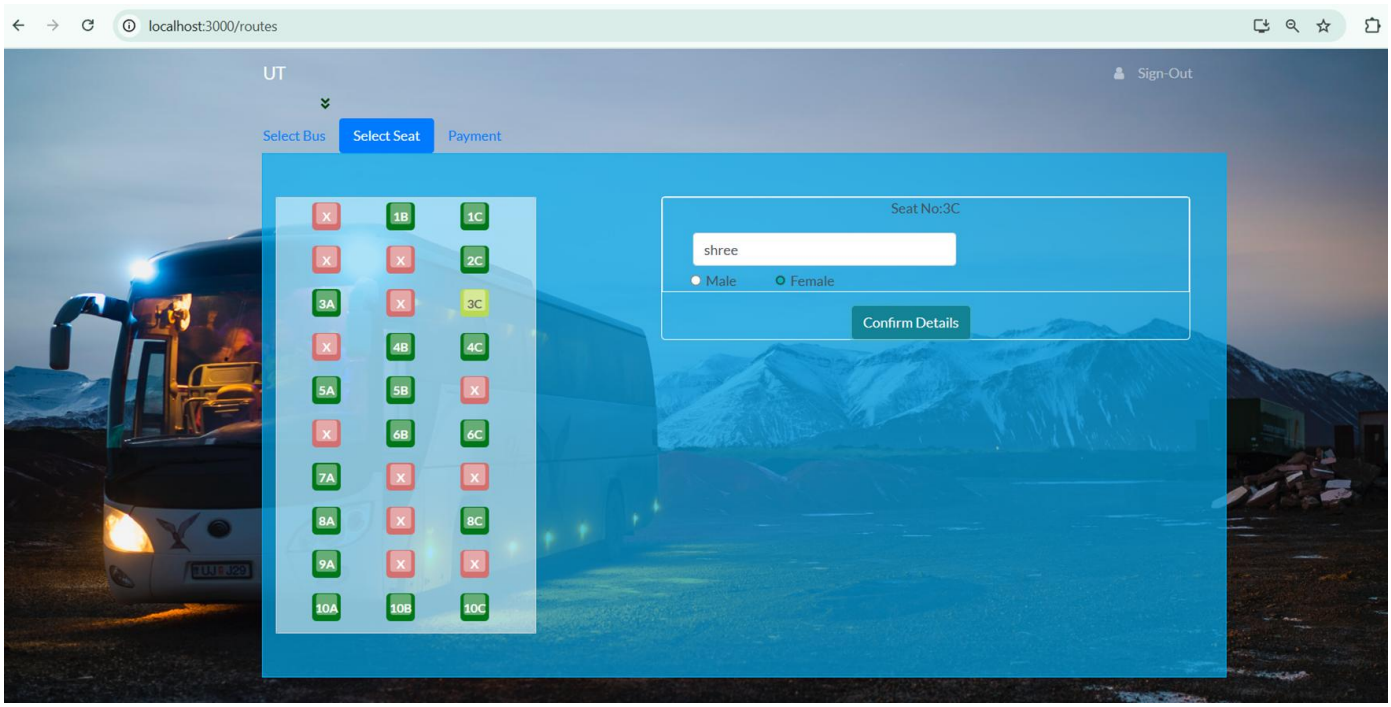


7.3 bus selection

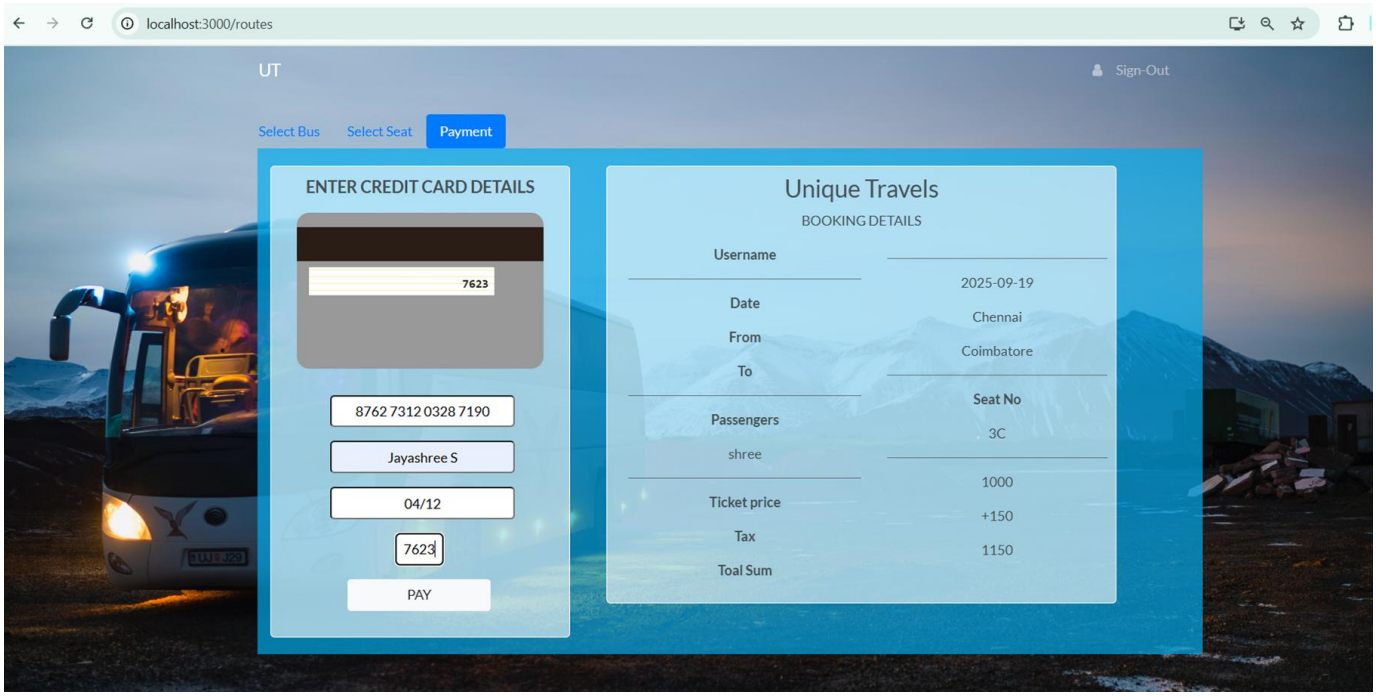




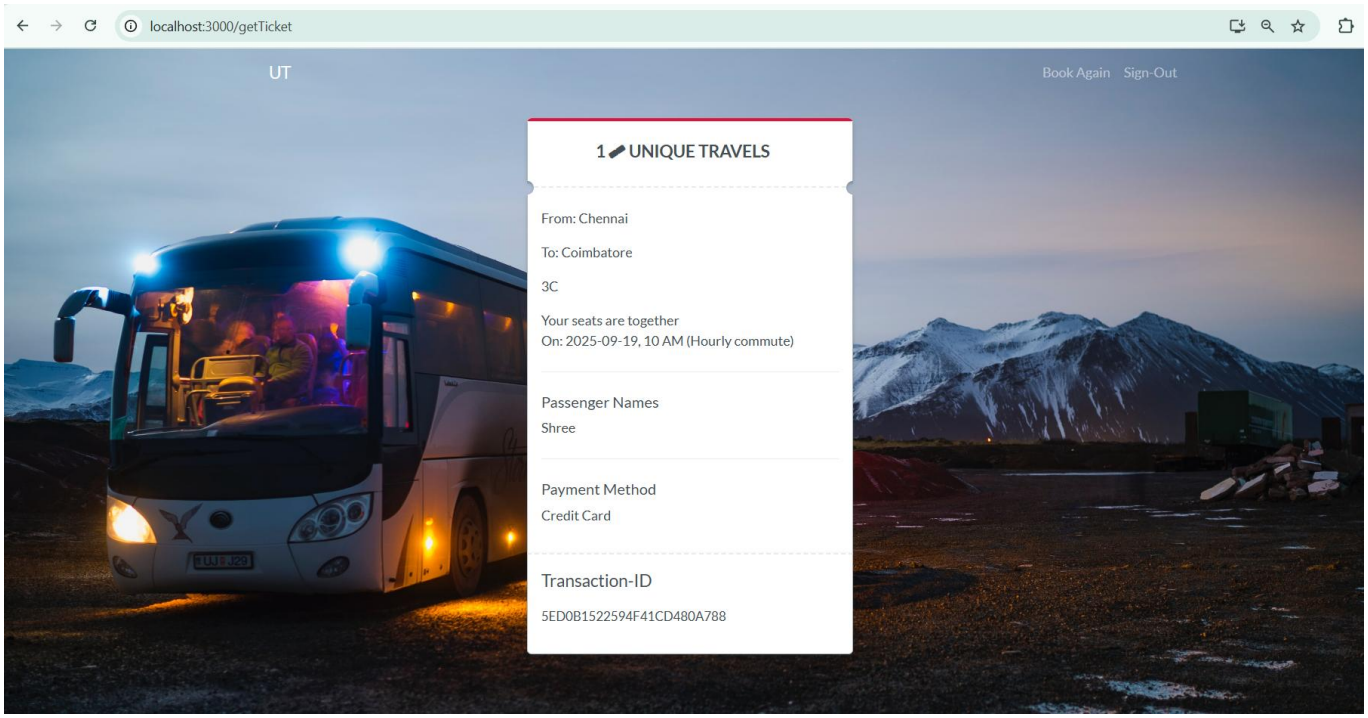
7.4 seat selection



7.5 payment


localhost:3000/routes
UT
Sign-Out
Select Bus
Select Seat
Payment
ENTER CREDIT CARD DETAILS
7623
8762 7312 0328 7190
Jayashree S
04/12
7623
PAY
Unique Travels
BOOKING DETAILS
Username
Date
2025-09-19
From
Chennai
To
Coimbatore
Seat No
Passengers
3C
shree
Ticket price
1000
Tax
+150
Toal Sum
1150

7.6 seat booked details



Appendix C: To Be Determined List

1. *Payment Gateway Integration*
    - *TBD: Which payment gateway(s) will be integrated (Stripe, Razorpay, PayPal, COD).*
    - *TBD: Transaction handling and refund policies.*
2. *Cloud Deployment Platform*
    - *TBD: Final hosting environment (AWS, GCP, Azure, or on-premises).*
    - *TBD: Deployment architecture (single server vs. load-balanced cluster).*
3. *Scalability Limits*
    - *TBD: Maximum number of concurrent users the system must reliably support in production.*
4. *Internationalization*
    - *TBD: Supported languages and currency formats.*
    - *TBD: Localization requirements (date/time formats, translations).*
5. *Mobile Application*
    - *TBD: Whether a React Native mobile app will be developed alongside the web version.*
6. *Security Enhancements*
    - *TBD: Final decision on two-factor authentication (2FA) for login.*
    - *TBD: Data encryption standards for sensitive fields beyond passwords (e.g., booking history, payment data).*
    -
7. *Regulatory Compliance*
    - *TBD: Compliance with specific regional laws (e.g., GDPR, Indian IT Act, PCI DSS for payments).*
8. *Backup & Recovery Strategy*
    - *TBD: Frequency of MongoDB backups (daily, hourly, real-time replication).*
    - *TBD: Disaster recovery plan and RTO/RPO metrics.*