A Project Report On

# “EasyLuxury Go – Bus Booking System”

by

#### **AKSHIT GAJERA (23SS02IT058)**

#### **SMIT GADHIYA (23SS02IT056)**

#### **HARSHIL GUJARATI (22SS02IT069)**

under the mentorship of **Mr. ANSHUL GOUR** Assistant Professor, School of Engineering

**NOVEMBER 2025**

**P P SAVANI SCHOOL OF ENGINEERING** **P P SAVANI UNIVERSITY** **NH NO.: 8, VILLAGE: DHAMDOD, TA: MANGROL, NEAR KOSAMBA, SURAT – 394 125, (GUJARAT).**

## CERTIFICATE

This is to certify that the Project Report titled “EasyLuxury Go – Bus Booking System” submitted by:

#### **AKSHIT GAJERA (23SS02IT058)**

#### **SMIT GADHIYA (23SS02IT056)**

#### **HARSHIL GUJARATI (22SS02IT069)**

to the **P P SAVANI UNIVERSITY** for the partial fulfillment of the subject credit requirements is a bonafide work carried out by the students under my supervision and guidance.

This is to further certify that I have been supervising the Major/Minor Project of the above-mentioned students.

The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree, diploma, or titles.

**Sign of Faculty Mentor:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Name of Faculty Mentor:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Date:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## ACKNOWLEDGEMENT

We sincerely thank our faculty mentor **Mr. Anshul Gour**, Assistant Professor, School of Engineering, P. P. Savani University, for his constant support and valuable guidance throughout this project.  
We are also grateful to all faculty members of the Computer Engineering Department for their advice and encouragement.  
Finally, we extend our thanks to our families and friends for their patience and motivation during this academic journey.

**Name of Students:**

* AKSHIT GAJERA (22SS02IT058)
* SMIT GADHIYA (22SS02IT056)
* HARSHIL GUJARATI (22SS02IT069)

## **1.1 Purpose**

The purpose of this Software Requirement Specification (SRS) is to describe the complete functionality of the **EasyLuxury Go – Bus Booking System**.  
It serves as a reference for developers, project managers, QA testers, faculty evaluators, and stakeholders.  
This SRS follows the **IEEE 830-1998 standard**, covering system requirements, design constraints, and quality attributes for effective development and testing.

## **1.2 Project Scope**

### **1.2.1 Overview**

**EasyLuxury Go** is a web-based system that automates bus ticket booking and management through a digital platform.  
It includes:

* **Customer Features:** Bus search, seat selection, secure payment (Razorpay), e-ticket download, live tracking, booking history, and cancellation.
* **Admin Features:** Fleet management, route setup, pricing, staff approval, trip scheduling, and performance analytics.
* **Staff Features:** Real-time location sharing, passenger list management, and trip updates.

### **1.2.2 Benefits**

* **Customers:** 24/7 access, instant booking, tracking, easy cancellation, and digital tickets.
* **Administrators:** Centralized control, analytics-based decisions, and reduced manual work.
* **Staff:** Simplified workflow, digital passenger handling, and real-time updates.

### **1.2.3 Out of Scope**

The current version excludes mobile apps, offline bookings, multilingual support, inter-system integration, dynamic pricing, and AI recommendations.

## **1.3 Project Objectives**

The key goals of **EasyLuxury Go** are:

* **Modernize Bus Booking:** Enable 24/7 online booking and reduce booking time to <5 minutes.
* **Enhance User Experience:** Offer real-time seat view, responsive UI, and instant confirmations.
* **Improve Operations:** Automate trip scheduling, track buses, and reduce manual errors.
* **Ensure Security:** Use JWT authentication, encrypted payments, and data backup for 99% uptime.
* **Scalability:** Deploy cloud-based MERN architecture for flexible expansion.
* **Academic Learning:** Apply full-stack development principles using modern web technologies.

## **1.4 System Overview**

### **1.4.1 Architecture**

The system uses a **three-tier MERN architecture**:

1. **Presentation Layer (Frontend):** React.js SPA with Tailwind CSS, real-time Socket.IO, deployed on **Vercel**.
2. **Application Layer (Backend):** Express.js + Node.js REST API handling logic, authentication, and real-time communication, hosted on **Render**.
3. **Data Layer (Database):** MongoDB Atlas with Mongoose ODM for data modeling and backup.

### **1.4.2 System Features Summary**

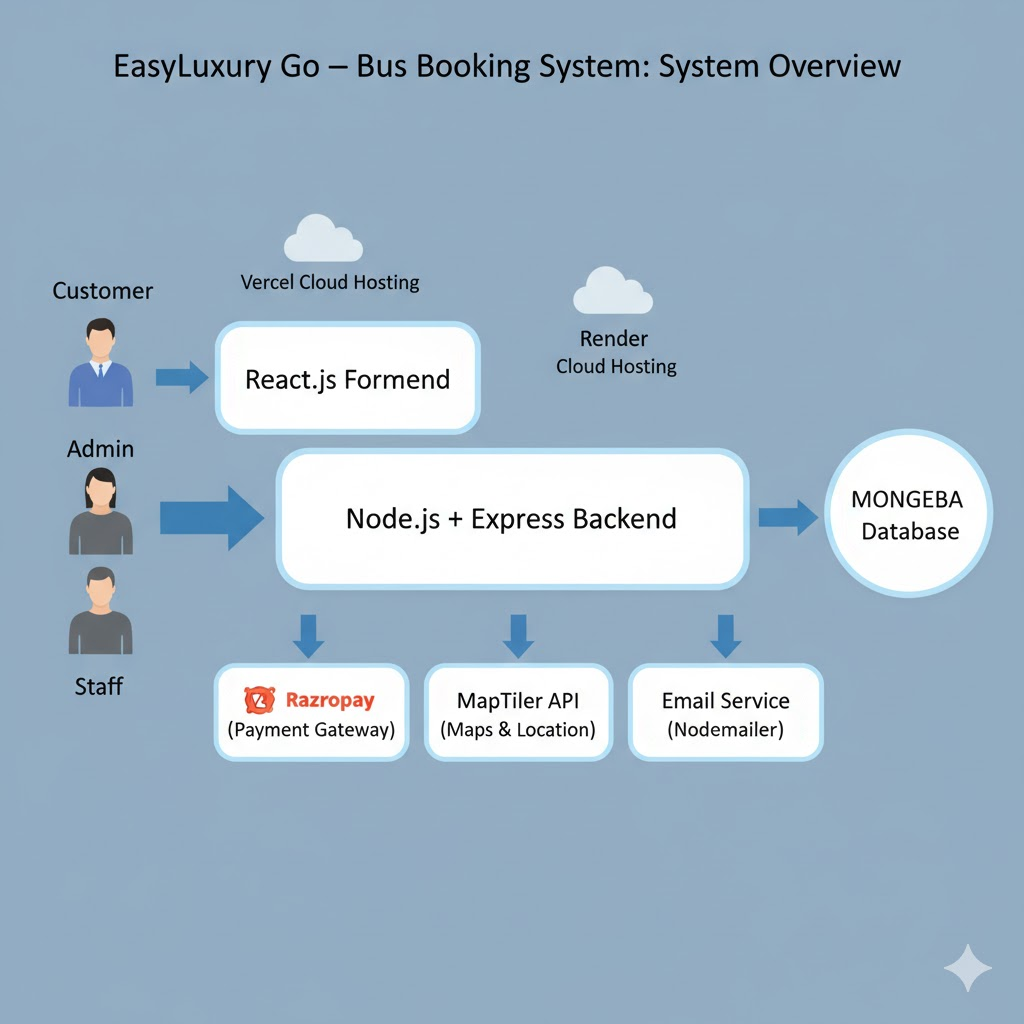
| **Module** | **Key Features** |
| --- | --- |
| **Customer** | Registration, Trip Search, Seat Selection, Payment, Ticket Download, Tracking |
| **Admin** | Dashboard, Bus/Route Management, Scheduling, Staff & User Control, Analytics |
| **Staff** | Dashboard, Location Sharing, Passenger Handling, Trip Updates |

### **1.4.3 Technology Ecosystem**

Integrations: **Razorpay (Payments)**, **MapTiler (Maps)**, **Nodemailer (Emails)**, **MongoDB Atlas (Database)**, **Vercel (Frontend)**, **Render (Backend)**.

## **1.5 Technology Stack**

| **Layer** | **Technology** | **Version** | **Purpose** |
| --- | --- | --- | --- |
| **Frontend** | React.js, Vite, Tailwind CSS, React Router, Axios, Socket.IO Client | Latest | User Interface, Client Routing, Real-time Updates |
| **Backend** | Node.js, Express.js, Mongoose, JWT, bcryptjs, Socket.IO | Latest | API Development, Auth, Data Validation |
| **Database** | MongoDB, MongoDB Atlas | Cloud | Data Storage and Management |
| **External APIs** | Razorpay, MapTiler, Nodemailer | - | Payments, Map Rendering, Notifications |
| **Deployment** | Vercel, Render | Cloud | Hosting and Continuous Deployment |



## 2. System Analysis

## **2.1 Identification of Need**

### **2.1.1 Current Challenges**

Traditional bus booking systems in India suffer from:

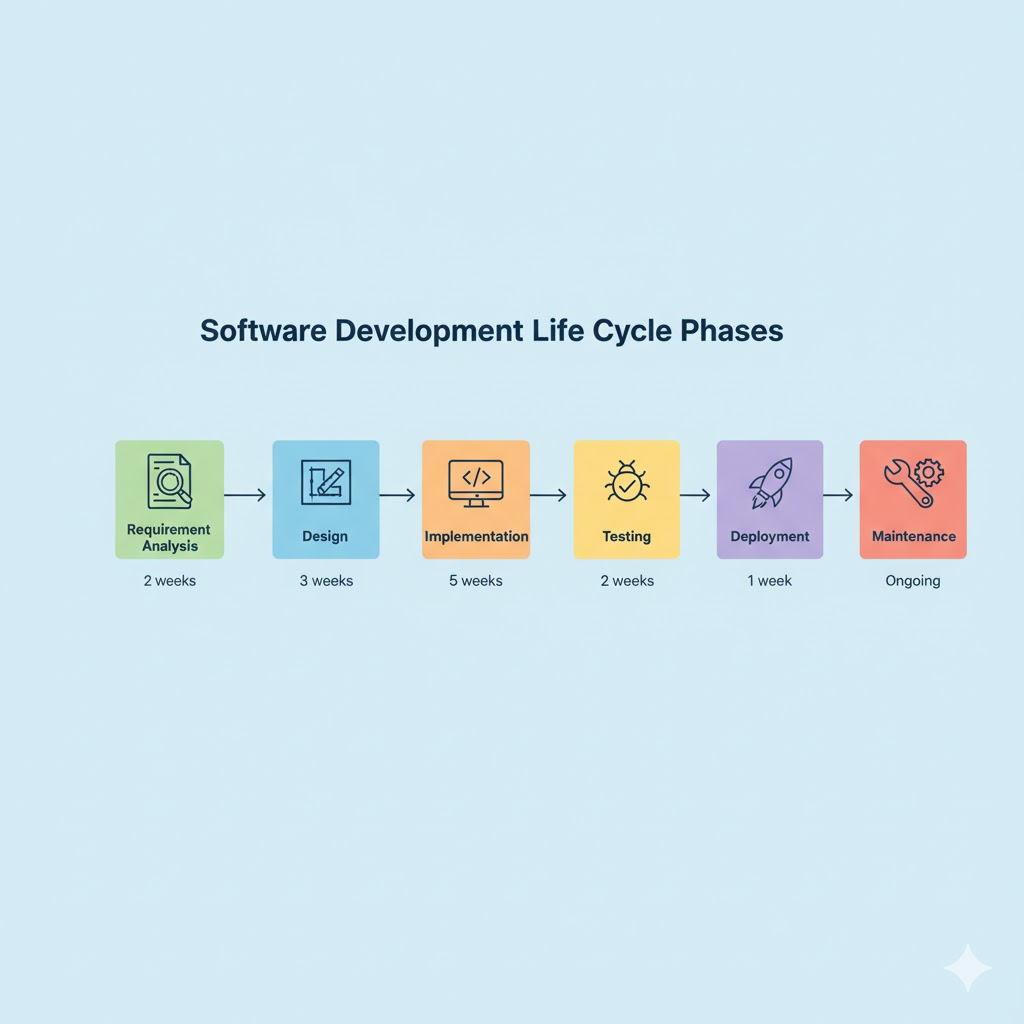
* Long waiting queues and limited booking hours.
* No real-time seat availability or bus location tracking.
* Manual cash transactions with safety and transparency issues.
* Poor cancellation policies and lack of booking history.

## **2.1 Identification of Need**

### **2.1.1 Current Challenges**

Traditional bus booking systems in India suffer from:

* Long waiting queues and limited booking hours.
* No real-time seat availability or bus location tracking.
* Manual cash transactions with safety and transparency issues.
* Poor cancellation policies and lack of booking history.



#### 2.2.2 Technology Investigation

|  |  |  |
| --- | --- | --- |
| **Technology** | **Chosen** | **Reason** |
| **Frontend** |  |  |
| React.js | ✓ | Component reusability, large ecosystem, performance |
| Angular | ✗ | Steep learning curve, overkill for project size |
| Vue.js | ✗ | Smaller community compared to React |
| **Backend** |  |  |
| Node.js | ✓ | JavaScript everywhere, non-blocking I/O, scalable |
| Django | ✗ | Python-based, separate language from frontend |
| Spring Boot | ✗ | Java-based, heavier footprint |
| **Database** |  |  |
| MongoDB | ✓ | Flexible schema, JSON-native, cloud support |
| PostgreSQL | ✗ | Rigid schema not suitable for evolving requirements |
| MySQL | ✗ | Less suitable for hierarchical data |

#### 2.2.3 Stakeholder Interviews

* **Bus Operators Feedback:**
  + Need for automated scheduling
  + Requirement for staff monitoring
  + Demand for analytics and reports
* **Passenger Feedback:**
  + Priority: Real-time bus tracking
  + Concern: Payment security
  + Desire: Easy cancellation process

### 2.3 Feasibility Study

#### 2.3.1 Technical Feasibility

* **Development Team Skills:**
  + ✓ Team has expertise in JavaScript, React, Node.js
  + ✓ Familiar with MongoDB and REST APIs
  + ✓ Experience with Git version control
  + ✓ Understanding of cloud deployment
* **Technology Availability:**
  + ✓ All chosen technologies are open-source and free
  + ✓ Comprehensive documentation available
  + ✓ Active community support
  + ✓ Free-tier cloud services available (Vercel, Render, MongoDB Atlas)
* **Integration Requirements:**
  + ✓ Razorpay SDK available with good documentation
  + ✓ MapTiler API accessible with free tier
  + ✓ Socket.IO well-documented and widely used
* **Development Tools:**
  + ✓ VS Code IDE (free)
  + ✓ Postman for API testing (free)
  + ✓ Git/GitHub for version control (free)
  + ✓ MongoDB Compass for database management (free)
* **Conclusion:** ✅ **Technically Feasible** - All required technologies, tools, and skills are available

#### 2.3.2 Economic Feasibility

* **Development Costs:** | Resource | Cost (₹) | Details | | :--- | :--- | :--- | | Development Team | 0 | Student project (academic) | | Software Licenses | 0 | All open-source technologies | | Development Tools | 0 | Free IDEs and tools | | Cloud Services (Dev) | 0 | Free tier usage | | Domain Name | 500 | Optional (.com domain) | | **Total Development** | **₹500** | **Minimal investment** |
* **Operational Costs (Monthly):** | Service | Free Tier | Paid Tier (if scaled) | | :--- | :--- | :--- | | Vercel Frontend | ✓ Unlimited | ₹1,500/month | | Render Backend | ✓ 750 hrs/month | ₹2,000/month | | MongoDB Atlas | ✓ 512 MB | ₹3,500/month | | MapTiler API | ✓ 100K requests | ₹2,500/month | | Razorpay | 2% transaction fee | 2% (standard) | | **Total** | **₹0** | **₹9,500/month** |
* **Conclusion:** ✅ **Economically Feasible** - Minimal investment, potential for profitability

#### 2.3.3 Operational Feasibility

* **User Acceptance:**
  + Target users (18-45 age group) are tech-savvy
  + Growing preference for online transactions
  + Similar apps (food delivery, cab booking) widely accepted
* **Training Requirements:** | User Type | Training Needed | Duration | | :--- | :--- | :--- | | Customers | None (intuitive UI) | Self-service | | Staff | Location sharing, passenger check-in | 1 hour | | Administrators | System management, analytics | 2-3 hours |
* **Conclusion:** ✅ **Operationally Feasible** - Easy to adopt, minimal training, maintainable

**Table 2.1: Feasibility Analysis Summary**

|  |  |  |
| --- | --- | --- |
| **Feasibility Type** | **Status** | **Confidence Level** |
| Technical | ✅ Feasible | 95% |
| Economic | ✅ Feasible | 90% |
| Operational | ✅ Feasible | 85% |
| **Overall** | **✅FEASIBLE** | **90%** |
|  | | | |  |  |

**2.4 Project Planning**

Phased development followed agile iterations:

* **Phase 1:** Requirement Analysis – 2 weeks
* **Phase 2:** System Design – 3 weeks
* **Phase 3:** Implementation – 5 weeks
* **Phase 4:** Testing – 2 weeks
* **Phase 5:** Deployment – 1 week
* **Phase 6:** Maintenance – Ongoing

Tools: VS Code, GitHub, Postman, MongoDB Compass.

## **2.5 Project Scheduling**

### **2.5.1 PERT Chart**

Depicts interdependent tasks such as requirement analysis, design, implementation, and testing — identifying the critical path to ensure timely completion.

### **2.5.2 Gantt Chart**

Represents project timeline, task duration, and resource allocation used for weekly progress tracking.

## **2.6 Software Requirement Specification (SRS)**

### **2.6.1 Functional Requirements (FRs)**

* **Customer Module (FR-1–21):** Registration, Trip Search, Seat Booking, Razorpay Payment, PDF Ticket, Live Tracking.
* **Admin Module (FR-22–32):** Dashboard, Fleet & Route Management, Scheduling, Staff & User Control, Promo Codes.
* **Staff Module (FR-33–38):** Location Sharing, Passenger Management, Trip Updates.

### **2.6.2 Non-Functional Requirements (NFRs)**

* **Performance:** API <2s, Page Load <3s.
* **Reliability:** 99% uptime, backup integrity.
* **Security:** JWT, bcrypt, HTTPS, rate limiting.
* **Usability:** Responsive UI, accessibility compliance.
* **Maintainability:** Modular code, Git control, detailed logs.
* **Portability:** Cloud deployment, cross-browser support.

### 2.7 Software Engineering Paradigm applied

## **2.7 Software Engineering Paradigm**

### **Methodology: Incremental + Agile Hybrid**

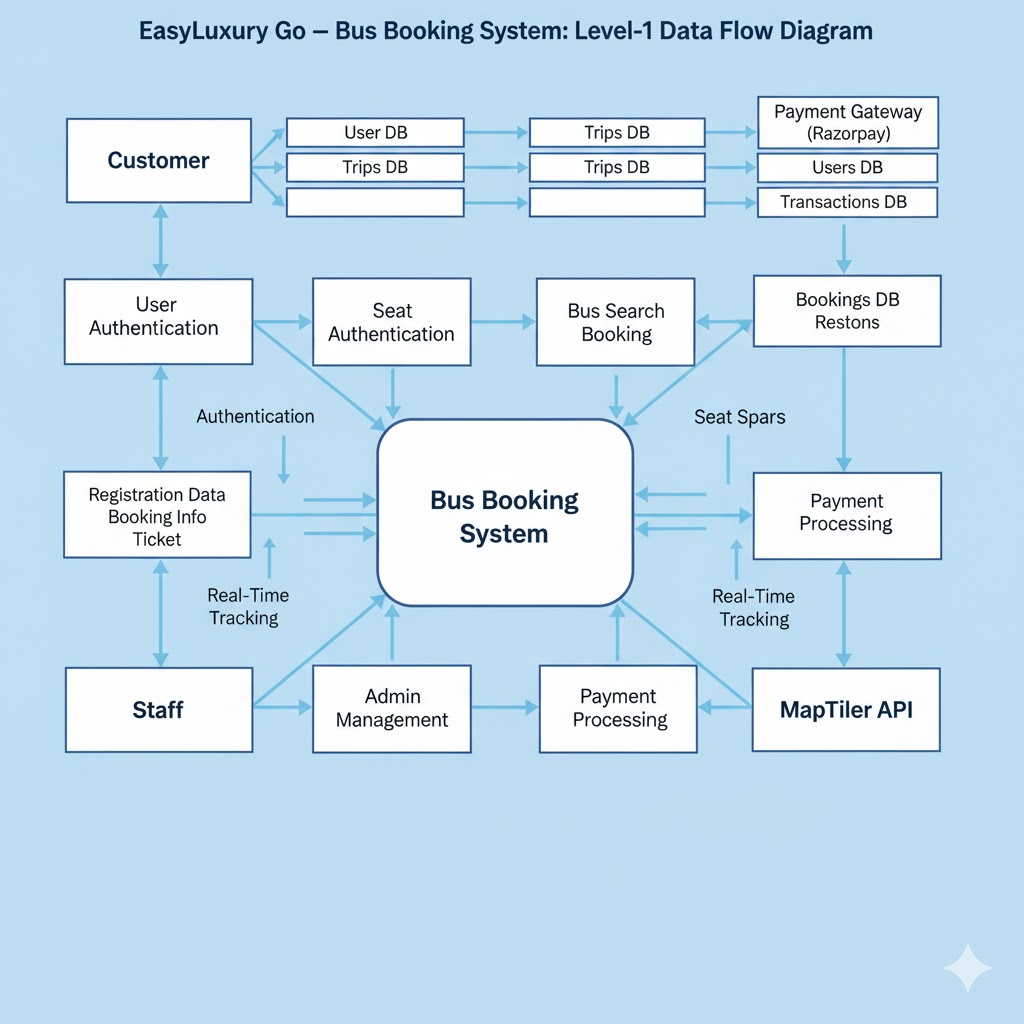
* Developed in phases — each increment adds key functionality.
* Regular testing, Git-based continuous integration, and faculty feedback loops.
* Agile sprints of 2 weeks enabled adaptive planning and user validation.

### 2.8 Data Models (DFD, ERD, Use Case, etc.)

#### 2.8.1 Data Flow Diagrams

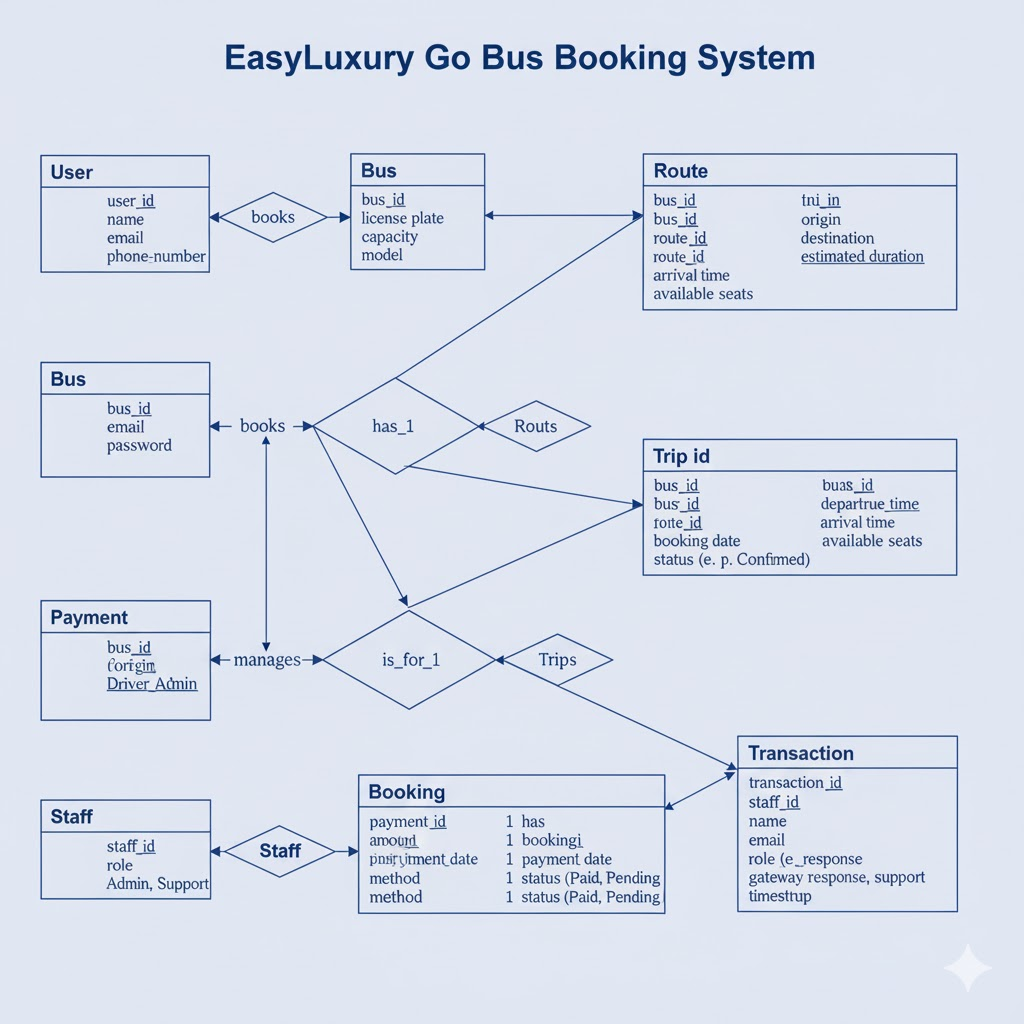
### **2.8.1 DFDs (Figures 2.1, 2.2)**

* Show system-level and module-level data flow between users, booking processes, payment, and admin operations.



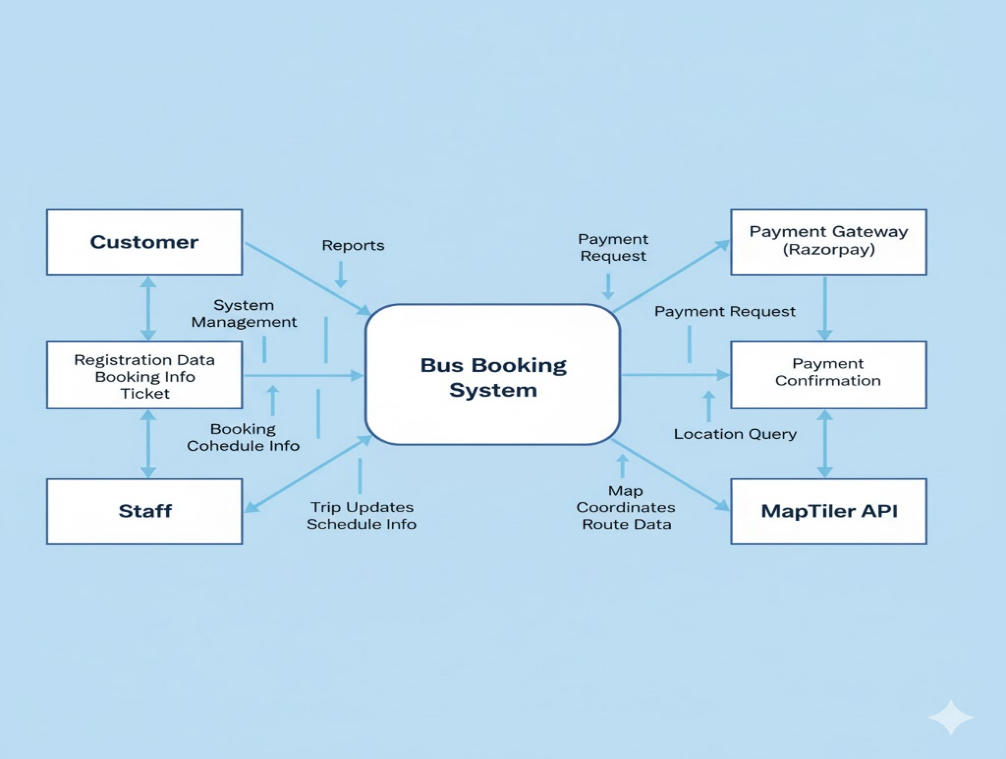
### **2.8.2 ER Diagram (Figure 5.1)**

Illustrates entity relationships: Users ↔ Bookings ↔ Trips ↔ Buses ↔ Routes.



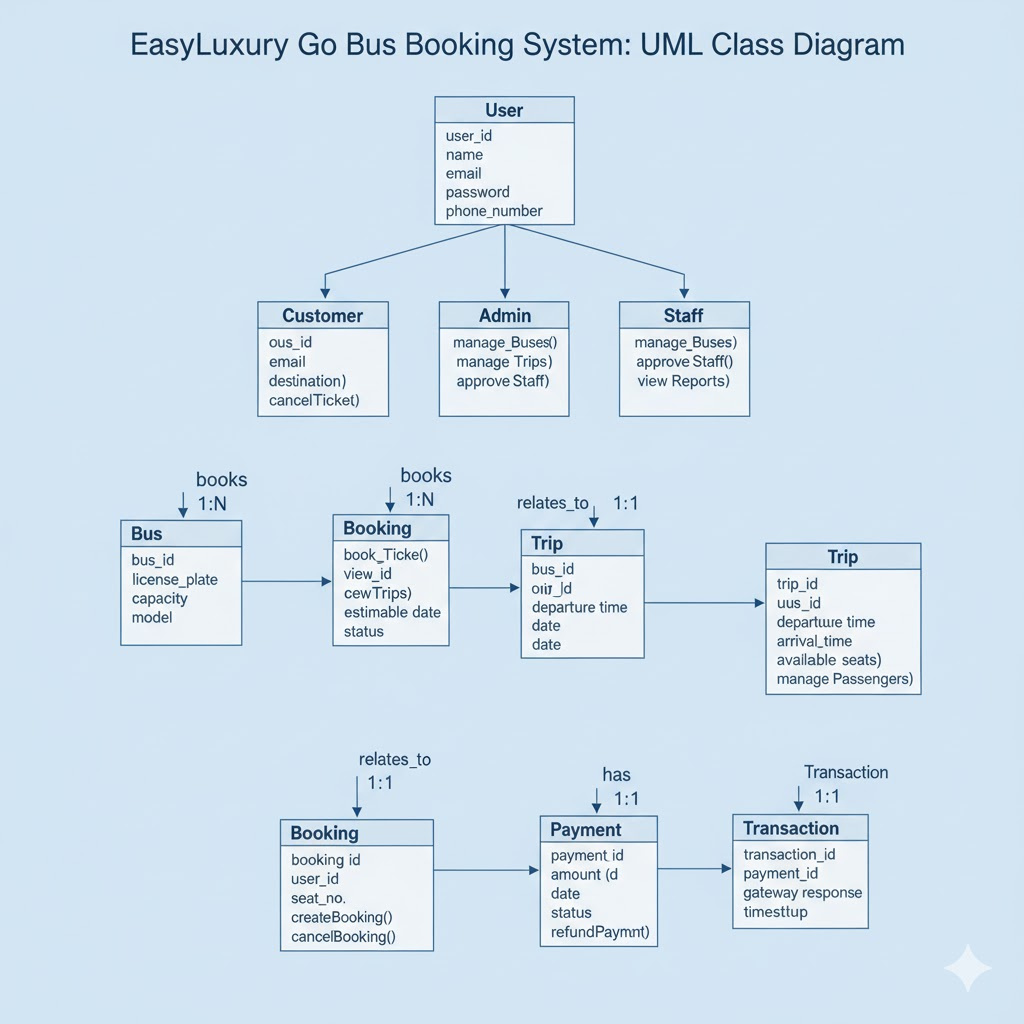
### **2.8.3 Use Case Diagram (Figure 5.2)**

Visualizes actor interactions for Customers, Admins, and Staff across booking and monitoring features.



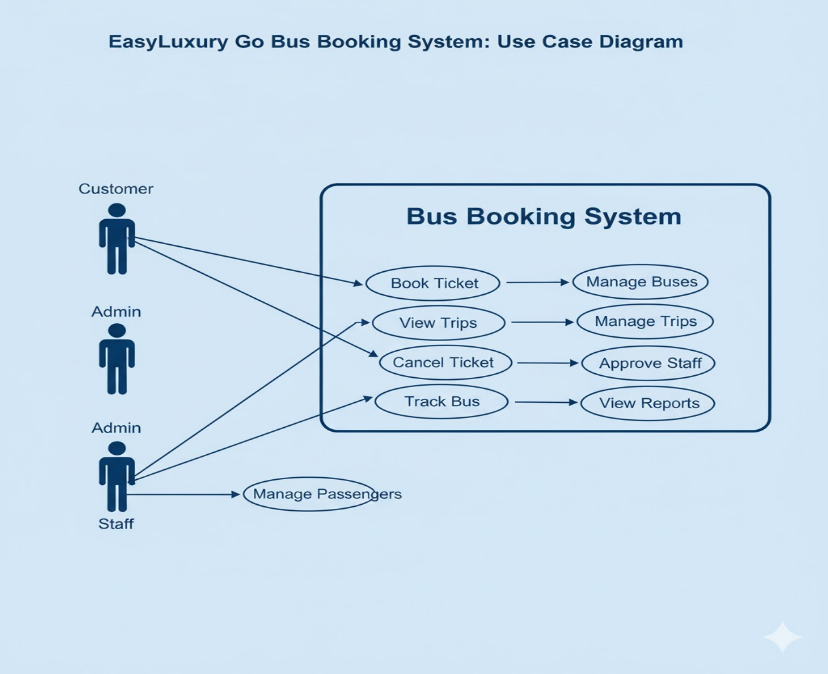
### **2.8.4 Class Diagram (Figure 5.3)**

Represents object-oriented architecture — classes (User, Bus, Booking, Trip) and their relationships.



### **2.8.5 Sequence Diagram (Figure 5.4)**

Shows message flow from user request → API → Database → Payment Gateway → Response.



### **2.8.6 Activity Diagram (Figure 5.5)**

Displays payment workflow, from selection to confirmation with alternate paths for failure handling.



## **3.1 Modularisation Details**

The **EasyLuxury Go Bus Booking System** follows a **modular, RESTful architecture** ensuring separation of concerns, scalability, and maintainability.

### **Backend (Node.js + Express)**

* Follows **MVC-like architecture** (Models, Controllers, Services, Routes).
* API endpoints categorized by resource:  
  | **Category** | **Base Path** | **Purpose** |  
  |---------------|---------------|--------------|  
  | Authentication | /auth | Login & registration (users/admin/staff) |  
  | Users | /users | Profile & wallet management |  
  | Trips | /trips | Trip search & info |  
  | Bookings | /bookings | Booking creation & cancellation |  
  | Buses | /buses | Fleet management (admin) |  
  | Routes | /routes | Route management |  
  | Admin | /admin | Admin operations |  
  | Staff | /staff | Staff dashboards |  
  | Analytics | /analytics | Reports & statistics |  
  | Payments | /payment | Razorpay integration |  
  | Location | /location | Live bus tracking |

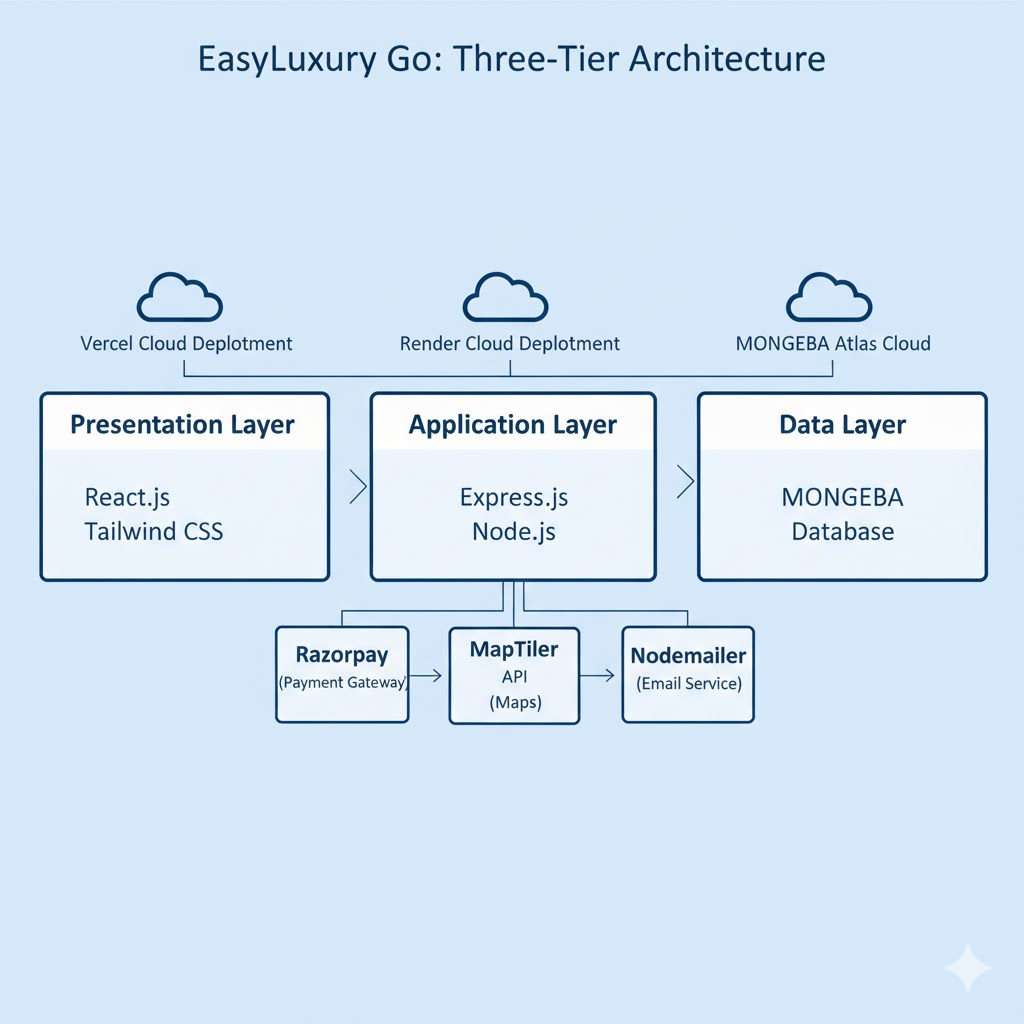
### **Frontend (React.js)**

* Built using **component-based design**, separating reusable UI elements (SeatLayout, TripCard) from page-level components (HomePage, BookingPage).

## **3.2 Data Integrity and Constraints**

Data validation and consistency are maintained at multiple levels:

* **Application Level:**
  + Uses express-validator to ensure input format and required fields.
  + Enforces business logic like “a user cannot book more than 6 seats”.
* **Database Level (MongoDB/Mongoose):**
  + Schema validation defines field types, required properties, and enums.
  + Unique constraints on email, busNumber, and pnrNumber.
  + Referential integrity maintained through Mongoose ref properties.
* **Transaction Level:**
  + Atomic operations ensure reliability during payments and seat locking.
  + Real-time **Socket.IO seat locking** prevents double booking (lock duration: 5 min).



## **3.1 Modularisation Details**

The **EasyLuxury Go Bus Booking System** follows a **modular, RESTful architecture** ensuring separation of concerns, scalability, and maintainability.

### **Backend (Node.js + Express)**

* Follows **MVC-like architecture** (Models, Controllers, Services, Routes).
* API endpoints categorized by resource:  
  | **Category** | **Base Path** | **Purpose** |  
  |---------------|---------------|--------------|  
  | Authentication | /auth | Login & registration (users/admin/staff) |  
  | Users | /users | Profile & wallet management |  
  | Trips | /trips | Trip search & info |  
  | Bookings | /bookings | Booking creation & cancellation |  
  | Buses | /buses | Fleet management (admin) |  
  | Routes | /routes | Route management |  
  | Admin | /admin | Admin operations |  
  | Staff | /staff | Staff dashboards |  
  | Analytics | /analytics | Reports & statistics |  
  | Payments | /payment | Razorpay integration |  
  | Location | /location | Live bus tracking |

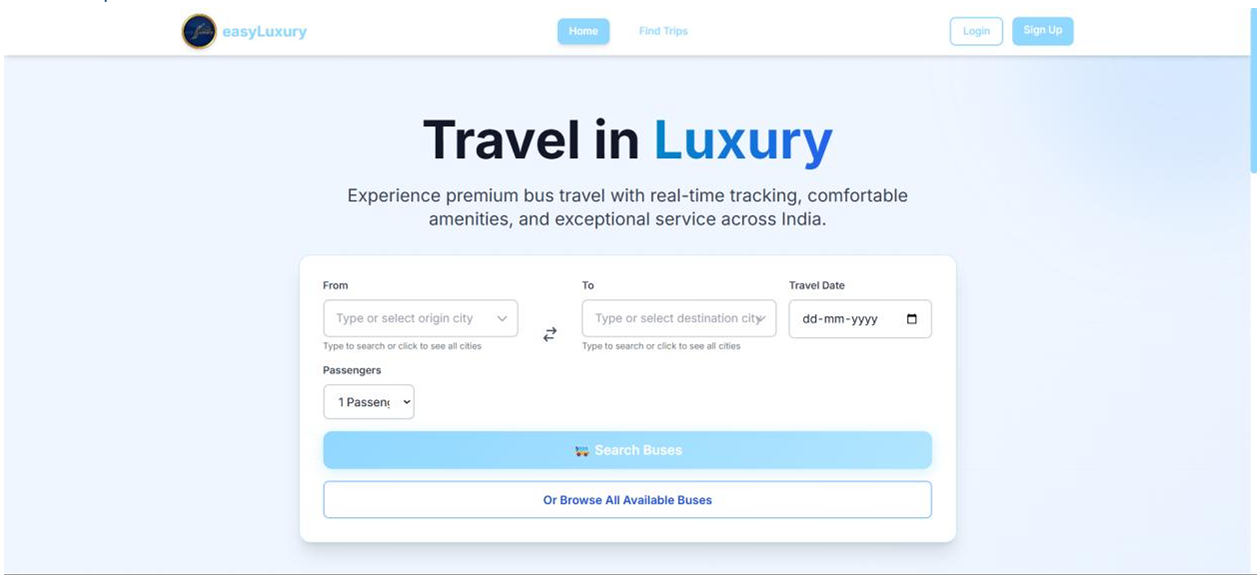
### **Frontend (React.js)**

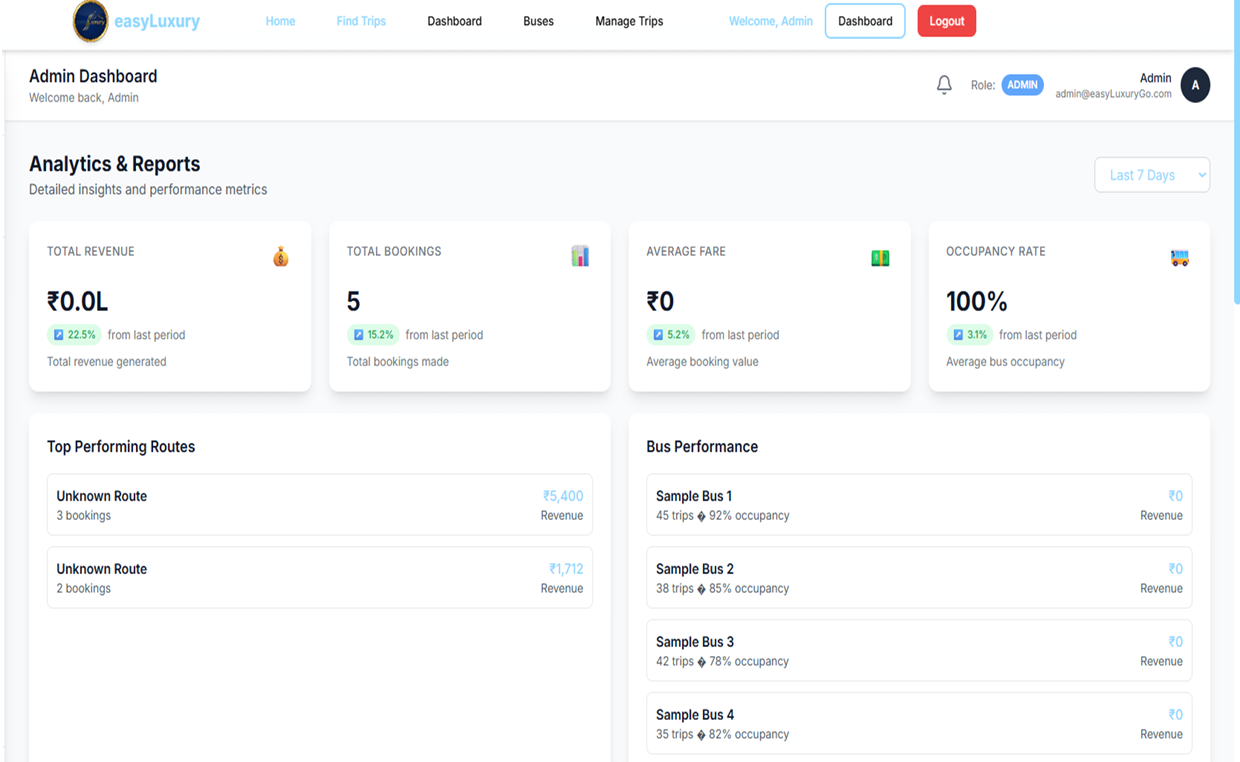
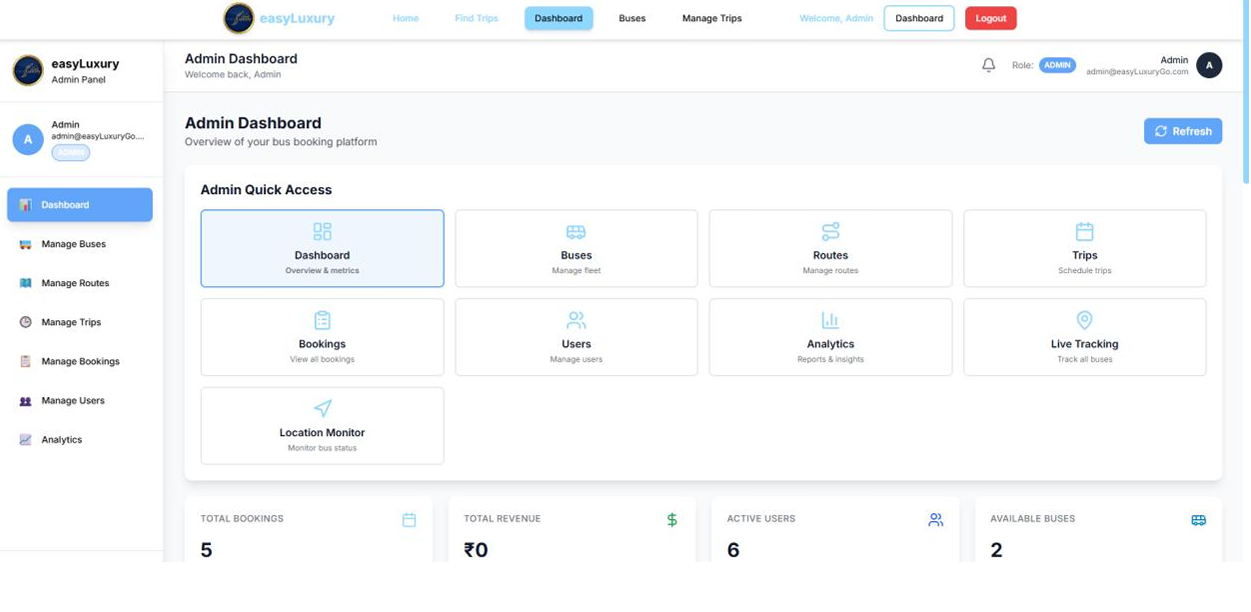
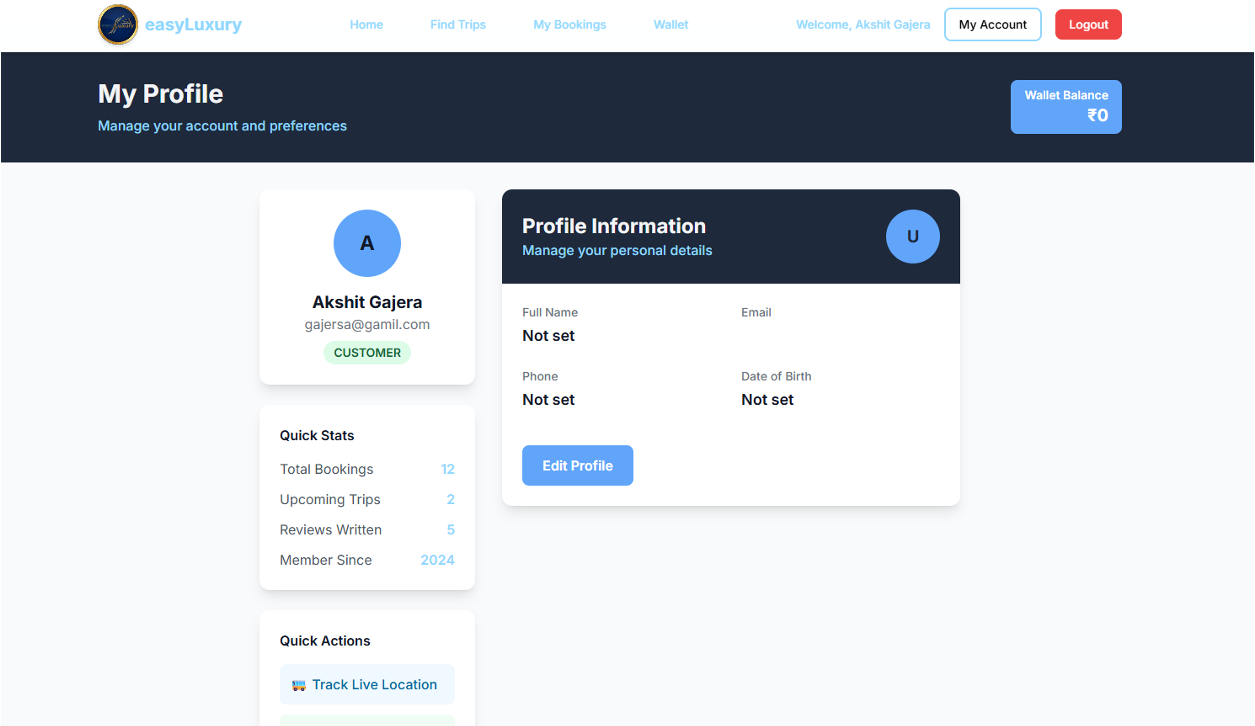
* Built using **component-based design**, separating reusable UI elements (SeatLayout, TripCard) from page-level components (HomePage, BookingPage).

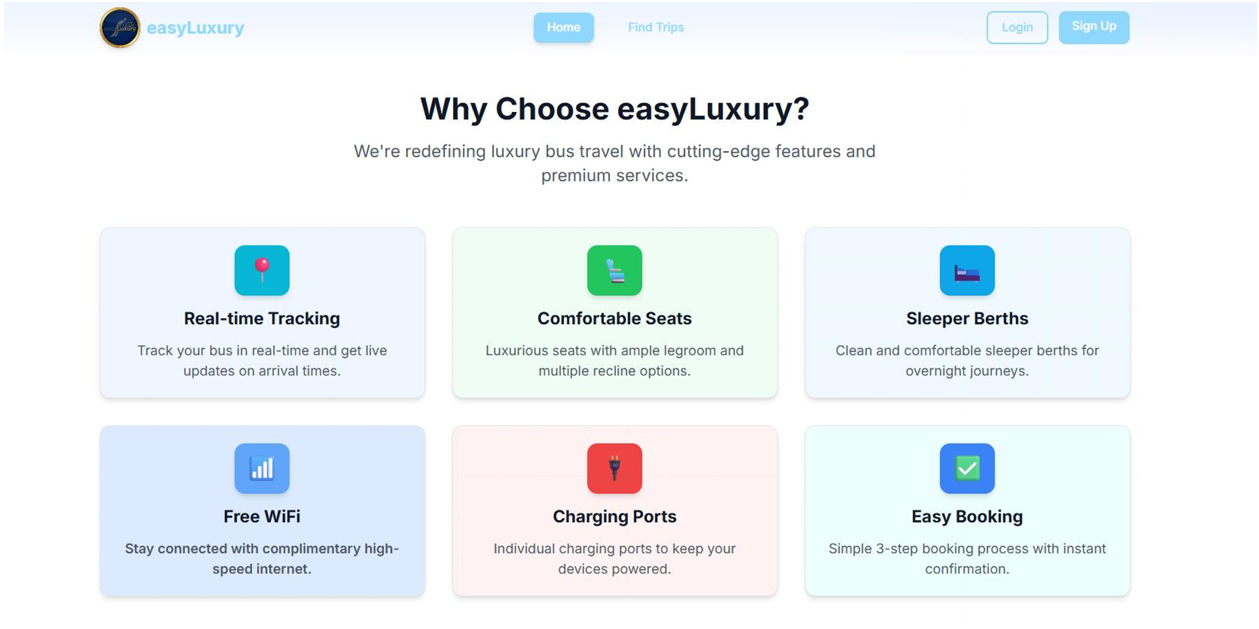
## **3.2 Data Integrity and Constraints**

Data validation and consistency are maintained at multiple levels:

* **Application Level:**
  + Uses express-validator to ensure input format and required fields.
  + Enforces business logic like “a user cannot book more than 6 seats”.
* **Database Level (MongoDB/Mongoose):**
  + Schema validation defines field types, required properties, and enums.
  + Unique constraints on email, busNumber, and pnrNumber.
  + Referential integrity maintained through Mongoose ref properties.
* **Transaction Level:**
  + Atomic operations ensure reliability during payments and seat locking.
  + Real-time **Socket.IO seat locking** prevents double booking (lock duration: 5 min).







## **3.5 Test Cases (Summary)**

### **3.5.1 Unit Test Cases (Sample)**

* **File:** auth.controller.js → **registerUser()**
  + Valid input creates a new user and returns JWT token (Status 201).
* **File:** booking.model.js → **bookingSchema.pre('save')**
  + Auto-generates unique pnrNumber before saving.

### **3.5.2 System Test Case (End-to-End Flow)**

**Scenario:** Customer books a bus ticket successfully.  
**Steps:**

1. Register → Login → Search → Select Trip → Select Seats → Add Passengers → Pay via Wallet → Download Ticket.  
   **Expected Results:**

* Booking created with status “Confirmed.”
* Wallet balance reduced accurately.
* Seats updated to “Booked.”
* Ticket PDF generated for download.

# **4. CODING (SUMMARY**

## **4.1 Database Schema and Access Rights**

The project uses **MongoDB** (NoSQL) with **Mongoose ODM** for schema definition.  
Access control is enforced via **JWT authentication** and role-based middleware rather than database-level permissions.

### **Example Schema (Trip.js)**

Defines relationships between Bus, Route, and Trip. Includes seat details, crew, pricing, and live GPS tracking.

* Indexed on route, departureDateTime, and status for faster queries.
* Supports atomic seat-locking and transactional updates for safe booking.

### **Access Rights (API Middleware)**

Middleware functions isAuthenticated and isAdmin ensure:

* Only logged-in users can access protected routes.
* Only admins can create, update, or delete sensitive data (e.g., buses, trips).  
  **Example:**  
  router.post('/buses', isAuthenticated, isAdmin, createBus);

## **4.2 Project Coding Overview**

The complete project (≈50,000 LOC) is divided into **backend**, **frontend**, and **shared utility layers**, hosted publicly at:  
🔗 [GitHub Repository – EasyLuxury Go](https://github.com/Akshitgajera96/easyLuxury-Go)

### **Backend Implementation**

* **Framework:** Node.js + Express
* **Database:** MongoDB via Mongoose
* **Authentication:** JWT + bcrypt for hashing
* **Real-time Communication:** Socket.IO for
  + Live bus tracking updates
  + Seat availability broadcasts
  + Real-time admin notifications
* **Key Services:**
  + PaymentService → Razorpay integration
  + LocationService → GPS-based alerts
  + EmailService → Booking confirmations via Nodemailer
* **Error Handling:** Centralized error middleware returns JSON responses for exceptions.

### **Frontend Implementation**

* **Framework:** React.js (Vite for builds)
* **Routing:** react-router-dom v6
* **State Management:** Context API + Reducers
* **Data Fetching:** Axios with global interceptors
* **Core Components:**
  + SeatLayout.jsx → Dynamic seat grid with locking logic
  + LiveBusMap.jsx → Real-time tracking via MapTiler
  + pdfGenerator.js → jsPDF for instant ticket generation

### **API Implementation**

All APIs follow **REST principles** with JSON responses.  
**Example:** POST /api/v1/bookings

* Validates trip and seat data
* Applies promo code or wallet deduction
* Confirms booking and sends email
* Returns JSON: { success: true, data: { bookingDetails } }

# **5. STANDARDIZATION OF CODING (SUMMARY)**

## **5.1 Code Efficiency & Error Handling**

* **Database:** Indexed queries and gzip compression optimize response times (~70% faster).
* **Frontend:** React lazy loading and Vite bundling improve load performance.
* **Backend:** Uses global errorHandler middleware for consistent error structure.
  + Example: { success: false, error: "Message", statusCode: 500 }
* **Frontend Errors:** Caught via ErrorBoundary + Axios interceptors → user-friendly toast notifications.

## **5.2 API Parameter Standards**

* **GET:** Query parameters (e.g., /trips?from=Surat&to=Mumbai)
* **POST/PUT:** JSON body requests
* **DELETE:** URL parameters (e.g., /buses/:id)
* **Authentication Header:** Authorization: Bearer <token>
* **Response Structure:**
  + Success → { success: true, data: {...}, message: "Success" }
  + Failure → { success: false, error: "Reason", statusCode: 4xx }

## **5.3 Validation Checks**

* **Client-Side:**
  + HTML5 and JavaScript validation prevent incorrect input.
  + Checks for invalid travel dates or empty fields.
* **Server-Side:**
  + Uses express-validator to sanitize and enforce field length, email format, and password rules.
  + Prevents NoSQL injection and malformed requests.

**Example:**

router.post('/register', [

body('name').isLength({ min: 3 }),

body('email').isEmail(),

body('password').isLength({ min: 6 })

], registerUser);

# **6. TESTING (SUMMARY)**

## **6.1 Testing Strategy**

A **multi-level testing approach** ensured robustness across modules:

1. **Unit Testing:** Individual functions (e.g., authentication, booking creation).
2. **Integration Testing:** Interaction between controllers, routes, and services.
3. **System Testing:** End-to-end flow validation (login → booking → payment).
4. **User Acceptance Testing (UAT):** Conducted by peers and faculty to ensure requirements alignment.

### **Sample Unit Test**

| **Module** | **Test ID** | **Function** | **Expected Output** |
| --- | --- | --- | --- |
| Auth | TC-U-01 | registerUser | Creates new user with JWT |
| Booking | TC-U-02 | bookingSchema.pre('save') | Auto-generates PNR |

### **System Test Example (Happy Path)**

* Register → Login → Search Trip → Select Seats → Add Passengers → Pay via Wallet.  
  **Expected Results:**  
  ✅ Booking confirmed, seats locked, wallet updated, ticket generated.

1. 

#### 6.1.2 Testing Techniques Used

* **Black Box Testing:** Used for System Testing and UAT. The tester interacts with the UI without knowledge of the internal code, testing features like booking, cancellation, and live tracking.
* **White Box Testing:** Used for Unit Testing. Developers wrote tests based on the internal logic of functions (e.g., testing if/else branches in the promo code validation logic).
* **API Testing:** Postman was used extensively for manual integration testing of all 100+ API endpoints, checking for correct responses, status codes, and error handling.
* **Manual Testing:** Due to project constraints, most testing was manual, following detailed test plans and test cases.
* **Cross-Browser Testing:** The final application was tested on Chrome, Firefox, and Safari (desktop and mobile) to ensure compatibility.

### 6.2 Test reports for Unit Test Cases and System Test Cases

#### 6.2.1 Unit Test Cases (Summary Report)

**Table 8.1: Unit Test Cases Summary**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Module** | **Total Cases** | **Passed** | **Failed** | **Pass Rate** | **Notes** |
| **Backend** |  |  |  |  |  |
| Auth Controller | 10 | 10 | 0 | 100% | Tested registration, login, JWT. |
| Booking Service | 15 | 14 | 1 | 93% | 1 fail related to refund edge case (fixed). |
| Trip Model | 5 | 5 | 0 | 100% | Tested PNR generation. |
| **Frontend** |  |  |  |  |  |
| useWallet Hook | 4 | 4 | 0 | 100% | Tested balance calculations. |
| PDF Generator | 3 | 3 | 0 | 100% | Verified PDF content. |
| **Total** | **37** | **36** | **1** | **97.3%** |  |

#### 6.2.2 System Test Cases (Summary Report)

**Table 8.2: System Test Cases Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Test Case ID** | **Description** | **Result** |
| **User Auth** | TC-S-01 | Register, Login, Logout | **PASS** |
| **Booking** | TC-S-02 | End-to-end booking (Wallet) | **PASS** |
|  | TC-S-03 | End-to-end booking (Razorpay) | **PASS** |
|  | TC-S-04 | Booking with Promo Code | **PASS** |
|  | TC-S-05 | Booking Failure (Insufficient Funds) | **PASS** |
|  | TC-S-06 | Booking Failure (Seat Taken) | **PASS** |
| **Cancellation** | TC-S-07 | Cancel booking (> 24hr) | **PASS** |
|  | TC-S-08 | Cancel booking (< 2hr) | **PASS** |
|  | TC-S-09 | Prevent cancellation after departure | **PASS** |
| **Live Tracking** | TC-S-10 | Staff shares location | **PASS** |
|  | TC-S-11 | Customer views location | **PASS** |
| **Admin** | TC-S-12 | Admin approves staff | **PASS** |
|  | TC-S-13 | Admin creates trip | **PASS** |
|  | TC-S-14 | Admin monitors bus location | **PASS** |

**Overall System Test Result:** **PASS** (All 14 critical path test cases passed).

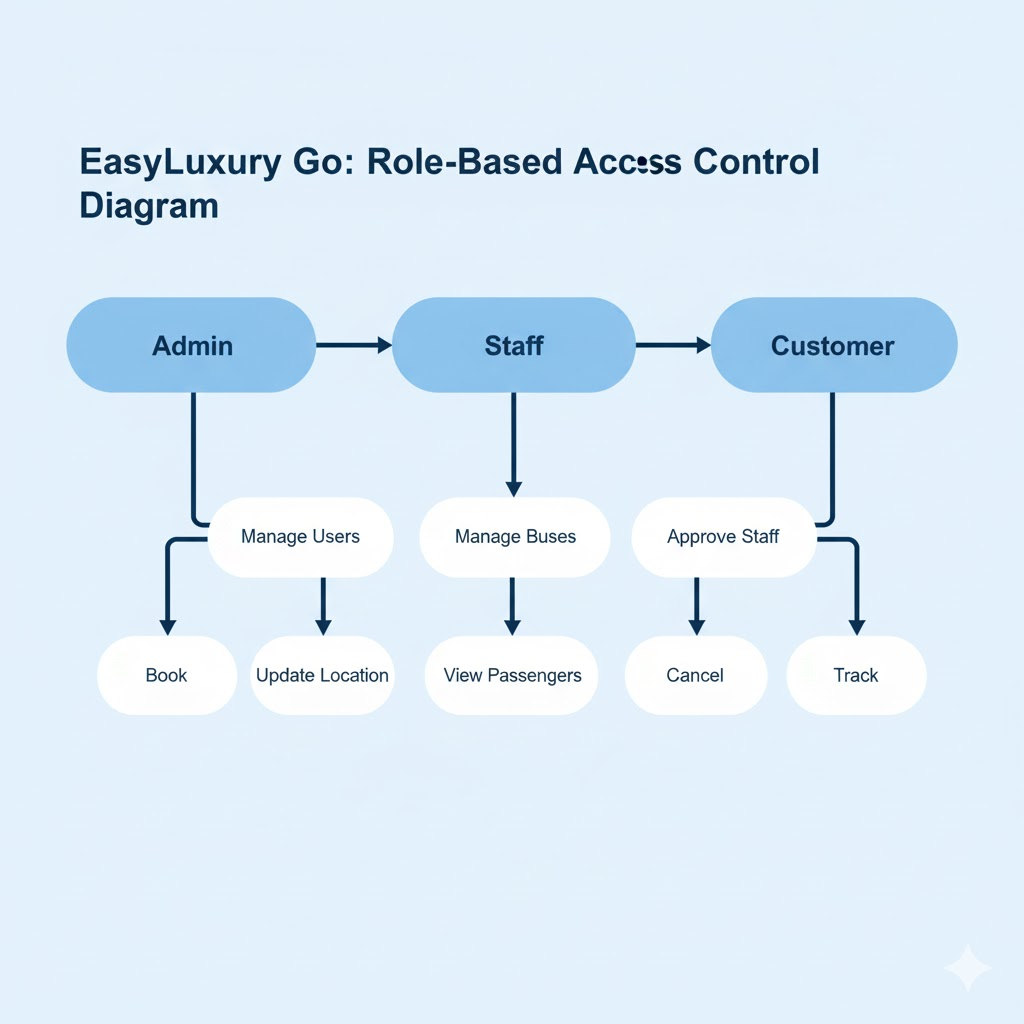
### 6.3 Debugging and Code improvement

* **Debugging:**
  + **Backend:** Used console.log for simple debugging and VS Code's built-in Node.js debugger for complex issues. Winston logger was crucial for tracking errors in the deployed (Render) environment.
  + **Frontend:** Used browser DevTools (Console, Network tab) and React DevTools extension to inspect component state and props.
* **Code Improvement (Example):**
  + **Initial Problem:** The admin's bus monitoring page was slow and made 100+ API calls on load.
  + **Improvement:** The backend was refactored to use Socket.IO. Now, the admin dashboard opens one WebSocket connection, and the server pushes real-time status updates (bus\_status\_update event) only when a bus's status changes. This reduced network traffic by 90% and provided a true real-time experience.

## 7. System Security measures

### 7.1 Database/data security

* **Database Access:** The MongoDB Atlas database is protected by a firewall. It is configured to only allow connections from specific IP addresses, including the Render backend servers. Direct connections from the public internet are blocked.
* **Password Hashing:** All user and staff passwords are **NOT** stored in plain text. They are hashed using the bcryptjs library with a salt of 12 rounds. This makes it computationally infeasible to reverse the password, even if the database is compromised.
* **Data Encryption:**
  + **In Transit:** All communication between the user's browser, the Vercel frontend, the Render backend, and the MongoDB Atlas database is encrypted using **HTTPS (TLS/SSL)**.
  + **At Rest:** MongoDB Atlas automatically encrypts all data stored on disk.
* **Environment Variables:** All sensitive information (database connection strings, API keys, JWT secrets) is stored in .env files and **NOT** committed to the Git repository. In production, these are set as secure environment variables in the Vercel and Render dashboards.



### 7.2 Creation of User profiles and access rights

The system implements robust Role-Based Access Control (RBAC).

* **User Roles:**
  1. **customer:** The default role for any new user. Can book and manage their own tickets.
  2. **staff:** Assigned by an admin. Can share location and manage passengers for their assigned trips.
  3. **admin:** Has full access to the entire system.
* **JWT (JSON Web Token):**
  1. When a user logs in, the server generates a JWT containing their userId and role.
  2. **Payload Example:** { "userId": "60c7...", "role": "customer", "iat": 1623..., "exp": 1624... }
  3. This token is sent to the frontend and stored.
* **Access Control Middleware:**
  1. **isAuthenticated:** This middleware runs on almost all API routes. It checks for a valid JWT. If no token or an invalid token is found, it returns a 401 Unauthorized error.
  2. **isAdmin / isStaff:** These middlewares run on protected routes. After isAuthenticated verifies the token, these check the req.user.role from the token payload.
  3. **Example:** A customer attempting to access /api/v1/admin/users would pass isAuthenticated but fail isAdmin, receiving a 403 Forbidden error.
* **User Classes & Characteristics:**
  1. **Customer:** (General public) Can search, book, pay, cancel, and track their own buses.
  2. **Administrator:** (Bus company owner/manager) Can do everything: manage fleet, routes, trips, staff, users, promo codes, and view all system analytics.
  3. **Staff:** (Driver/Conductor) Has a limited-access mobile-first dashboard. Can view their assigned trip, share their GPS location, and view the passenger list for check-in.

## 8. Cost Estimation of the Project

### 8.1 Development Cost

As this is an academic project, the primary development cost (human resources) is not calculated in monetary terms. The cost is based on material and tool expenses.

**Table 9.1: Development Cost Breakdown**

|  |  |  |
| --- | --- | --- |
| **Resource** | **Cost (₹)** | **Details** |
| **Software** |  |  |
| Development Team | ₹ 0 | Student project (academic) |
| Software Licenses | ₹ 0 | All technologies (MERN, Vite) are open-source. |
| Development Tools | ₹ 0 | VS Code, Postman, Git are free. |
| **Infrastructure** |  |  |
| Cloud Services (Dev) | ₹ 0 | Vercel, Render, and MongoDB Atlas all provide free tiers sufficient for development and testing. |
| Domain Name | ₹ 500 | (Optional) Cost for easyluxurygo.com (one-time). |
| **Total Development Cost** | **₹ 500** |  |
|  |  |  |

### 8.2 Operational Cost

This estimate assumes the project scales beyond the free tier.

**Table 9.2: Monthly Operational Costs (Estimated)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Service** | **Free Tier Limit** | **Paid Tier (Est.)** | **Cost (₹) / Month** |
| Vercel (Frontend) | Unlimited | Pro Plan | ₹ 1,500 |
| Render (Backend) | 750 hrs/month | Starter Plan | ₹ 2,000 |
| MongoDB Atlas | 512 MB Storage | M2 Cluster (5GB) | ₹ 3,500 |
| MapTiler API | 100K requests/month | Plus Plan | ₹ 2,500 |
| Razorpay Gateway | 2% per transaction | 2% per transaction | (Variable) |
| **Total Estimated Monthly Cost** |  |  | **₹ 9,500** |

### 8.3 Cost Estimation Model

A basic **Bottom-Up Estimation** model was used, where costs are estimated for each individual component (hosting, database, APIs) and then summed up.

### 8.4 Cost-Benefit Analysis

* **Costs:**
  + Initial Development: ₹ 500
  + Monthly Operational (Scaled): ~₹ 9,500
* **Benefits (Qualitative):**
  + **For Operator:** Reduced manual work, automated scheduling, error reduction, staff monitoring, valuable data analytics.
  + **For Customer:** 24/7 convenience, time saved (no queues), security (live tracking), digital payments.
* **Benefits (Quantitative - Projections):**
  + Assumed average booking: ₹ 800
  + Assumed commission/service fee per booking: ₹ 40 (5%)
  + **Break-even Point:** To cover ₹ 9,500 in monthly costs, the system would need:
    - ₹ 9,500 / ₹ 40 = 237.5 bookings per month
  + **Conclusion:** The system becomes profitable after approximately **238 bookings per month**.

## 9. Reports (Sample Layouts)

The primary "reports" in this system are generated in real-time on the Admin Dashboard, providing analytics and monitoring data.

### 9.1 Admin Dashboard Analytics

This is the main report layout for administrators.

**Layout Description:**

* **Top Bar:** Summary Cards
  + **Total Users:** (e.g., 1,200)
  + **Total Bookings:** (e.g., 2,500)
  + **Total Revenue:** (e.g., ₹20,00,000)
  + **Active Trips:** (e.g., 15)
* **Main Section (Left): Revenue Chart**
  + A bar chart showing revenue generated per month for the last 12 months.
  + [Sample Bar Chart Visual]
* **Main Section (Right): Booking Statistics**
  + A pie chart showing booking status distribution.
  + [Sample Pie Chart Visual: Confirmed (70%), Cancelled (20%), Pending (10%)]
* **Bottom Section: Tabbed View**
  + **Tab 1: Recent Bookings:** A table showing the last 10 bookings with PNR, User, Route, and Amount.
  + **Tab 2: Popular Routes:** A list of the top 5 most booked routes.
  + **Tab 3: System Health:** Status indicators for API, Database, and Payment Gateway.



## 10. Future scope and further enhancement

### 10.1 Phase 2 Enhancements

The following features are planned for future versions of the project:

* **Native Mobile Applications:** Develop dedicated iOS and Android apps using React Native for a better mobile experience.
* **Multi-language Support:** Add support for regional languages like Hindi, Gujarati, and Marathi.
* **Loyalty Program:** A points-based system to reward frequent travelers.
* **Dynamic Pricing:** An algorithm to adjust ticket prices based on demand, time of booking, and competitor pricing.
* **Social Login:** Allow users to register and log in using their Google or Facebook accounts.
* **Push Notifications:** Send real-time alerts to users' phones for booking confirmations, trip delays, and location alerts.

### 10.2 Scalability Roadmap

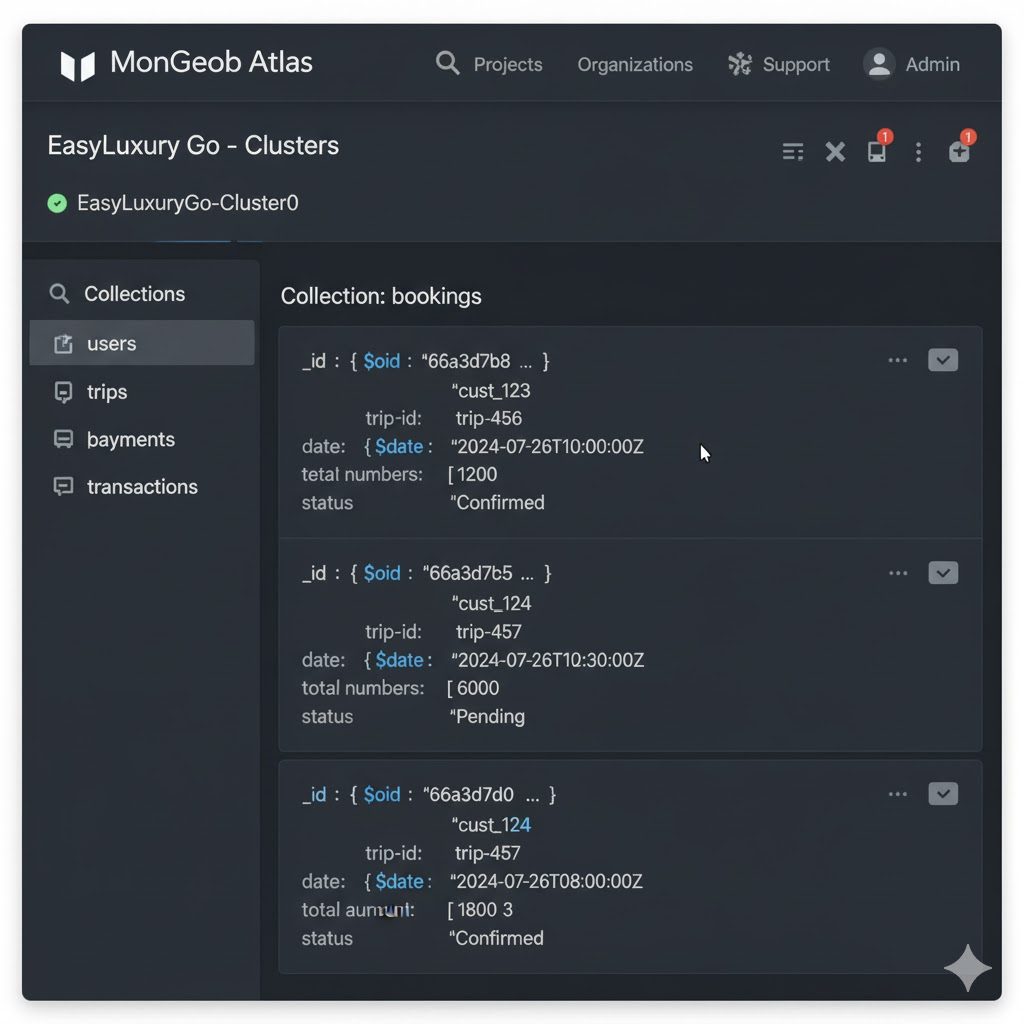
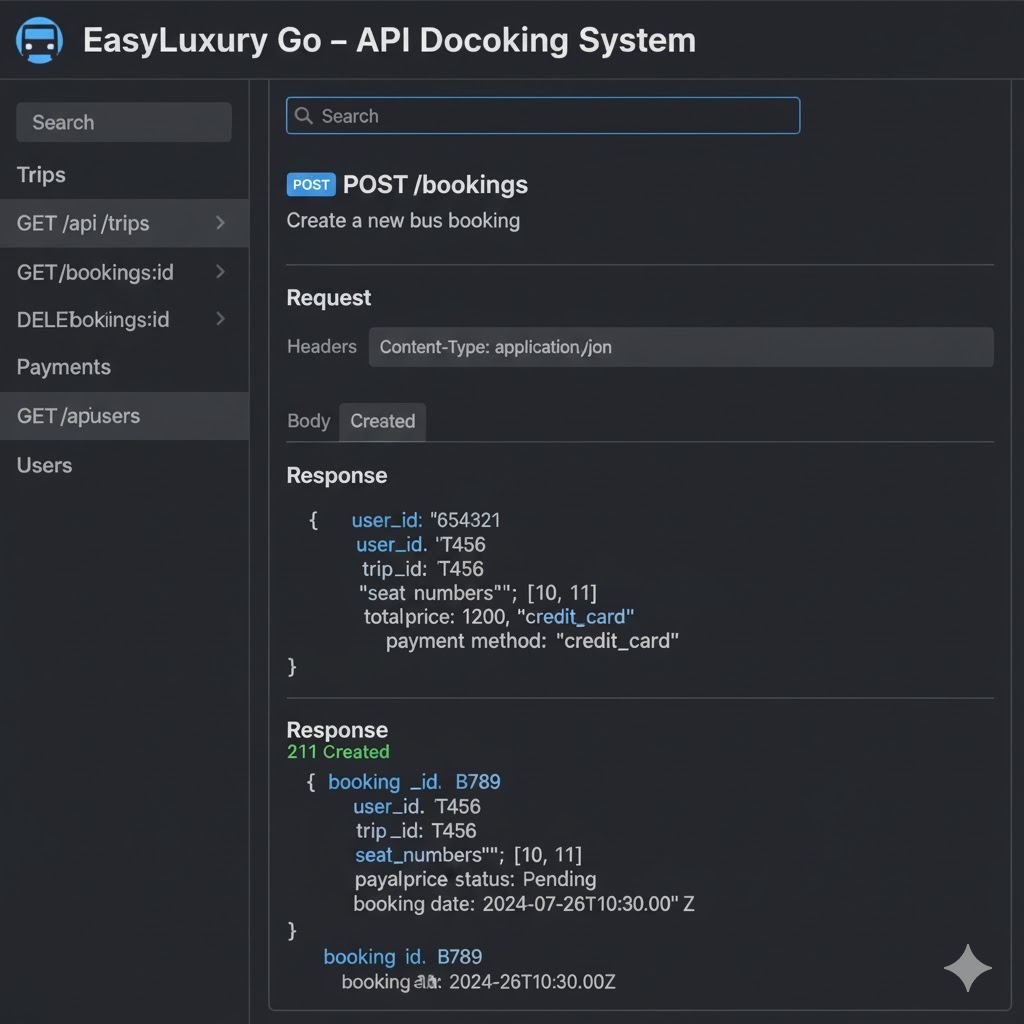
To handle growth, the following scalability steps are planned:

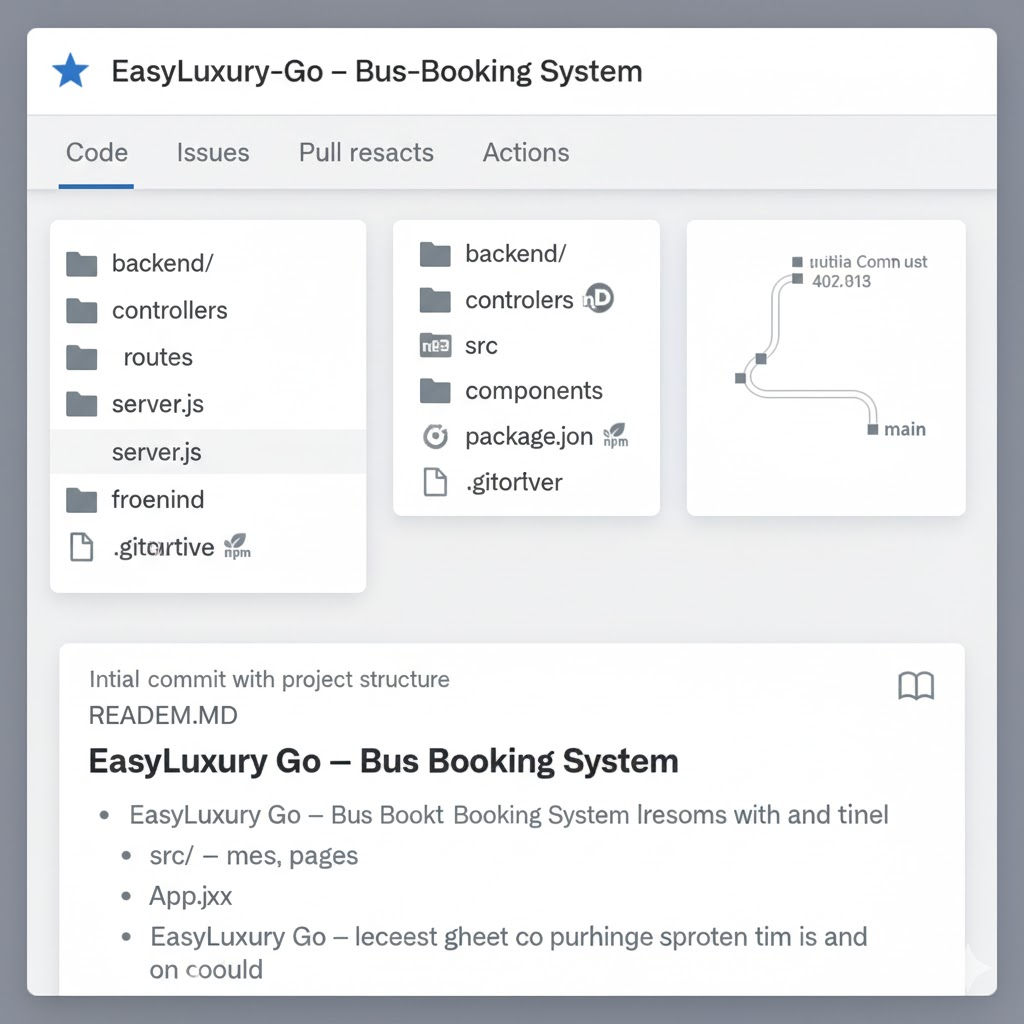
1. **Upgrade Cloud Tiers:** Move from free to paid plans on Render (Pro Plan) and MongoDB Atlas (M2+ Cluster) to get more RAM, CPU, and storage.
2. **Implement Caching:** Use a **Redis** in-memory cache to store frequently accessed data (like route lists or completed trip details), reducing database load.
3. **Optimize Database:** Implement database sharding in MongoDB Atlas to distribute the data load across multiple servers.
4. **Microservices:** (Long-term) Break down the monolithic backend into smaller microservices (e.g., Auth Service, Booking Service, Location Service) that can be scaled independently.

## 11. Bibliography

* **MERN Stack:**
  + MongoDB Documentation: https://docs.mongodb.com/
  + Express.js Documentation: https://expressjs.com/
  + React.js Documentation: https://www.google.com/search?q=https://reactjs.org/docs/
  + Node.js Documentation: https://nodejs.org/en/docs/
* **Libraries:**
  + Mongoose ODM: https://mongoosejs.com/docs/
  + Socket.IO: https://socket.io/docs/
  + Razorpay API: https://razorpay.com/docs/
  + jsPDF: https://www.google.com/search?q=https://rawgit.com/MrRio/jsPDF/master/docs/
  + MapTiler SDK: https://docs.maptiler.com/
* **Standards:**
  + IEEE 830-1998: Recommended Practice for Software Requirements Specifications.

## 12. Appendices





## 13. Glossary

**Table 16.1: Technical Acronyms Reference**

|  |  |  |
| --- | --- | --- |
| **Acronym** | **Full Form** | **Description** |
| **API** | Application Programming Interface | Interface for software communication |
| **CDN** | Content Delivery Network | Distributed server network |
| **CORS** | Cross-Origin Resource Sharing | Cross-domain request policy |
| **CRUD** | Create, Read, Update, Delete | Basic data operations |
| **DFD** | Data Flow Diagram | Diagram showing data movement |
| **ERD** | Entity Relationship Diagram | Diagram showing database structure |
| **JWT** | JSON Web Token | Authentication token |
| **MERN** | MongoDB Express React Node | Technology stack |
| **MVC** | Model View Controller | Architectural pattern |
| **NoSQL** | Not Only SQL | Non-relational database |
| **ODM** | Object Data Modeling | Database mapping technique (Mongoose) |
| **PDF** | Portable Document Format | Document file format |
| **PNR** | Passenger Name Record | Booking identifier |
| **RBAC** | Role-Based Access Control | Permission system |
| **REST** | Representational State Transfer | API architectural style |
| **SPA** | Single Page Application | Web app type |
| **SRS** | Software Requirement Specification | This requirements document |
| **SSL** | Secure Sockets Layer | Security protocol |
| **UI** | User Interface | Visual interface |
| **UX** | User Experience | User interaction quality |