A

PROJECT REPORT ON

VAYU VIHARA -- AN ONLINE BOOKING APPLICATION

Submitted in partial fulfillment of
requirements for the award of the degree

BACHELOR OF COMPUTER APPLICATIONS

of

ಬೆಂಗಳೂರು ನಗರ ವಿಶ್ವವಿದ್ಯಾಲಯ
BENGALURU CITY UNIVERSITY
ಆಗು ನೀ ಅನಿಕೇತನ
BE BOUNDLESS

By

K.R SAI VARUN - U18EP22S0051

Under the guidance of

Prof. KARTIK BADARINARAYANA
Department of BCA

MEWA
VANGUARD
BUSINESS SCHOOL

MEWA VANGUARD BUSINESS SCHOOL
NO.128, 38th Cross, East End Main Road,
Jayanagar 9th Block, Bengaluru – 560069.

MEWA VANGUARD BUSINESS SCHOOL
JAYANAGAR, Bengaluru-560069

Department of Computer Applications

This is to certify that **K.R SAI VARUN (U18EP22S0051),** is a Bonafide student of **MEWA VANGUARD BUSINESS SCHOOL** and has carried out a project entitled "**Vayu Vihara**" under our guidance. This project has been submitted during the academic year 2024-2025 in partial fulfilment of BCA degree as prescribed by the Bengaluru City University

---	---
Internal Guide	**H.O.D of BCA Department**
---	---
Internal Examiner	**External Examiner**

Date:

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success.

We convey our sincere gratitude to **Dr. BHARTISH RAO**, Principal, **MEWA VANGUARD BUSINESS SCHOOL**, for providing all the facilities needed for the successful completion of the project.

We are grateful to our respected Head of Department of BCA, **Prof. NEERAJA LAKSHMI K**, whose support and permission were instrumental in enabling us to embark on this dissertation journey and guiding us through its completion.

We are extremely grateful to our guide, **Assistant Prof. KARTIK BADARINARAYANA**, Department of Computer Applications, for their support and guidance provided for the completion and preparation of this report.

We are thankful to all the teaching and non-teaching staff of Dept. of BCA, **MEWA VANGUARD BUSINESS School**, who have shown their keen interest and support for the successful completion of the project.

Above all, we would like to thank our friends for their valuable help and support through- out the project. Last but not least, we thank all those who have helped us directly and indirectly in bringing out this project with bright colors.

Submitted By –

K.R SAI VARUN

ABSTRACT

This project introduces **Vayu Vihara**, a web application designed to provide a premium and efficient helicopter booking service. Its primary objective is to facilitate rapid transportation from **Bengaluru Airport (BLR)** to various locations within the city, offering customers a luxurious and time-saving alternative to traditional ground travel. The platform features a central hub at BLR, an interactive map displaying approved helipad locations, and an intuitive route selection system with estimated travel times.

The key functionalities of Vayu Vihara include secure booking and payment systems, real-time flight status updates, robust user account management, and a comprehensive administrative panel for service oversight. The application is developed using modern web technologies such as **HTML, CSS, and JavaScript**, ensuring a responsive and dynamic user interface. To further enrich the user experience, the platform incorporates features like a rotating hero image slider and engaging scroll-triggered animations.

While Vayu Vihara presents an innovative solution for urban air travel, the current implementation has identified several areas for improvement. These include broken image paths, limited accessibility features, browser compatibility concerns, and issues related to mobile optimization. Future development will focus on resolving these challenges to enhance the overall user experience, improve performance, and expand the platform's functionality, ultimately ensuring a seamless and reliable service.

Date:

Place: Bengaluru

K.R SAI VARUN - U18EP22S0051

LIST OF CONTENTS

Chapter 1: INTRODUCTION

Vayu Vihara is a cutting-edge helicopter booking web application crafted to meet the growing need for luxurious, fast, and reliable air travel. It is designed for users who want to escape the delays and limitations of traditional transportation, especially in areas where road or rail access is difficult.

The application provides an integrated solution, bringing together helipad data, flight routes, booking processes, and payment methods into a single stream- lined platform. By offering real-time information and responsive design, it caters to a tech-savvy user base that expects immediacy and reliability.

Vayu Vihara supports both individuals and businesses looking for quick intercity travel or emergency access to remote regions. It leverages cloud technology to deliver a scalable, secure, and user-friendly platform. Built using the latest frontend and backend frameworks, the app ensures compatibility across devices and browsers.

Its intuitive interface allows users to view routes, book flights, and manage their travel plans with ease. This project reflects the fusion of modern web development with the dynamic requirements of the aviation industry.

Ultimately, Vayu Vihara redefines the helicopter booking experience with a focus on simplicity, speed, and service quality.

1.1 Problem Statements

Helicopter booking today is plagued with inefficiencies stemming from the lack of an integrated and user-friendly digital solution. Current systems are either manual or use disconnected mobile applications that fail to provide seamless user experiences. Real-time flight availability is hard to access, and there's minimal coordination between helipads, pilots, and booking portals. Furthermore, payment systems used in legacy applications are outdated, putting user data and transactions at risk. There's also a lack of transparency in booking status updates and poor management of user data. User interfaces in existing systems are unintuitive and discourage potential users. Inconsistent communication among stakeholders results in delays and poor customer satisfaction. User account features are usually basic or missing, which limits the ability to track history or rebook easily. Administrative tools are often inaccessible or underdeveloped, restricting management efficiency. Vayu Vihara addresses these limitations by providing a modern, fully integrated, and secure web-based system.

- Existing helicopter booking systems are disjointed and manually operated or isolated apps.
- Users face outdated interfaces, poor UX, and a lack of real-time route availability updates.
- Payment security is often weak, risking data exposure.
- No unified platform consolidates flights, routes, helipads, and bookings.
- Admin tools for managing bookings and schedules are inefficient or missing.
- Communication between providers and users is slow due to a lack of automation.
- User accounts lack features like booking history and management.
- Fragmentation causes user frustration and reduces trust.
- A unified, secure, and efficient system is needed.
- Vayu Vihara fills these gaps with a comprehensive smart booking solution.

1.2 Existing System

The current landscape of helicopter booking systems is fragmented and in- efficient. Most of the existing solutions are either manual or built as standalone applications with limited scope and integration. These systems lack a centralized approach, making it difficult for users and service providers to manage bookings, routes, and helipads effectively.

Key limitations of the existing system include:

- Disjointed Platforms: Helicopter booking services are often available through isolated applications or offline processes. There is no unified platform that combines flight schedules, helipad information, payment systems, and user accounts into a single interface.
- Outdated User Interfaces: Many systems still operate with old-fashioned interfaces that are not user-friendly. This results in a frustrating user experience, especially for those who are accustomed to modern digital services.
- Absence of Real-time Information: Existing systems do not provide real-time updates on flight availability, schedule changes, or route conditions. Users are left to rely on outdated or manually updated information, which can lead to missed opportunities or scheduling conflicts.
- Lack of Security in Transactions: Many booking platforms do not employ robust security

protocols. Payment processing lacks end-to-end encryption.

- Inefficient Administrative Tools: For service providers and administrators, the tools

  available to manage helipads, routes, and bookings are either missing or rudimentary. This leads to operational delays and increased workload.
- Limited User Account Functionality: Users often have access to only basic features like booking tickets. Advanced features such as viewing booking history, managing account settings, or receiving alerts are usually unavailable.
- Poor Communication and Delays: Due to the lack of automation and centralized data handling, communication between users and service providers is slow. This results in frequent miscommunication or delays in confirmations and support.
- Fragmentation of Service Components: Flight booking, route selection, payment processing, and customer support are handled separately, which reduces system efficiency and increases the likelihood of errors or data inconsistency.
- Low Trust and Satisfaction: Due to the issues mentioned above, users lose confidence in the system, leading to reduced usage and poor customer satisfaction.

1.2.1 Physical Storage

The existing helicopter booking systems rely on paper-based records, registers, and manual filing methods. These require significant physical space and are prone to data loss, misplacement, and human error. There is no centralized repository, making information retrieval slow and inefficient. Security and backup mechanisms are minimal, posing risks to data integrity.

1.2.2 Proposed System

The proposed system, Vayu Vihara, is a web-based helicopter booking plat- form designed to overcome the limitations of the existing manual and fragmented systems. It provides a centralized, secure, and user-friendly inter- face that integrates all booking-related functions such as flight scheduling, route selection, helipad management, and payment processing into a single digital platform. The system caters to passengers, administrators, and helipad operators, offering real-time updates, robust security, and cross-device compatibility.

Key Features of Proposed System:

- Integrates all services—bookings, routes, payments, and user accounts— into one cohesive system.

- Real-Time Flight and Route Updates: Provides live status of flight schedules, availability, and route conditions to avoid delays and confusion.
- Secure Payment Gateway: Supports multiple payment methods (credit/debit cards, UPI) with end-to-end encryption and compliance protocols.
- User Dashboard: Allows users to view booking history, manage profiles, track flight statuses, and download receipts.
- Admin Control Panel: Enables administrators to manage helipads, up- date routes, monitor transactions, and approve bookings.
- Mobile and Cross-Device Support: Fully responsive design ensures us- ability on desktops, tablets, and smartphones.
- Authentication and Access Control: Uses JWT and OAuth for secure login and session handling for users and admins.
- Cloud Deployment & Scalability: Hosted on cloud platforms like AWS/Heroku to ensure performance and accommodate growing user demands.
- Modular Architecture: Allows independent development and updates of features like booking, payment, or admin tools without system-wide disruption.
- Data Security and Compliance: Implements HTTPS, TLS, and secure database practices to protect sensitive information.

1.3 Structure of the project:

```
DatabaseManager/
└── DatabaseManager (1)/
    └── DatabaseManager/
        ├── .env
        ├── .gitignore
        ├── components.json
        ├── drizzle.config.ts
        ├── generated-icon.png
        ├── package-lock.json
        ├── package.json
        ├── postcss.config.js
        ├── tailwind.config.ts
        ├── tsconfig.json
        ├── vite.config.ts
        ├── .config/          ← npm configuration
        ├──. git/             ← git metadata
        ├── attached_assets/  ← custom folder (contents TBD)
        ├── client/           ← likely frontend
        ├── dist/             ← build output
        ├── node_modules/     ← dependencies
        ├── server/           ← likely backend logic
        │   └── shared/       ← possibly shared interfaces, utilities
```

Chapter 02: SYSTEM REQUIREMENT SPECIFICATION

The system requirements for Vayu Vihara were defined to support high performance, security, and scalability. On the software side, the application requires an operating system such as Windows or Linux, and uses HTML5, CSS3, JavaScript, and React (Next.js) for the frontend.

The backend is developed using Node.js with Express, or alternatively, Python frameworks like Django or Flask. For data storage, PostgreSQL or MySQL serves as the relational database system.

2.1 Software Requirements

2.1.1 Operating System

- Development Environment: Windows 10/11, macOS, or any Linux distribution (Ubuntu preferred).
- Deployment Environment: Ubuntu Server (for AWS/Heroku hosting).
- The operating system should be compatible with Node.js, PostgreSQL, and web servers like Nginx or Apache.

2.1.1 Frontend Technologies

- HTML5 & CSS3: For structuring and styling web pages.
- JavaScript (ES6+): Core scripting language for frontend logic.
- React.js with Next.js: For creating dynamic, responsive, and SEO- optimized user interfaces.
- Tailwind CSS or Bootstrap: For consistent UI design and responsive layouts.
- The frontend provides an intuitive and mobile-responsive interface for users and admins.

2.1.2 Backend Technologies

- Node.js: JavaScript runtime used to build the server-side logic.
- Express.js: Web framework for handling routing, middleware, and APIs.
- JWT (JSON Web Tokens): For secure authentication and authorization.
- The backend is responsible for handling business logic, data processing and secure communications.

2.1.3 Database Management System

- PostgreSQL (Preferred): Relational databases to store user data, bookings, routes, helipads, and payments.
- pgAdmin or DBeaver: Tools for database administration and query execution.
- These databases ensure structured storage, data integrity, and support for complex queries.

2.1.4 Web Server

- Nginx or Apache: Used as a reverse proxy and for serving static assets in production.
- Helps manage load balancing, security, and performance optimizations.

2.1.5 Version Control

- Git: A Version control system to manage codebase changes.
- GitHub: Hosting platform for collaboration and CI/CD integration.
- Enables team collaboration, branching, and backup of the source code.

2.1.6 Development Tools & IDEs

- Visual Studio Code: Primary code editor.
- Postman: For testing RESTful APIs.
- Browser Dev Tools: For frontend debugging.
- These tools improve development efficiency and debugging.

2.1.7 Testing Tools

- Jest or Mocha: For unit and integration testing.
- Super test: For API testing.
- OWASP ZAP: For security testing.
- Ensures code quality, security, and bug-free deployment.

2.1.8 Deployment & Cloud Services

- AWS EC2 / Heroku: For cloud hosting and application deployment.
- Docker (Optional): For containerization and consistent deployment environments.

- GitHub Actions: For CI/CD pipelines to automate testing and deployment.
- These services ensure scalability, uptime, and easier deployment workflows.

2.1.9 Security & Protocols

- HTTPS with TLS/SSL: For encrypted communication.
- OAuth 2.0 (Optional): For third-party authentication (e.g., Google Sign-In).
- Helmet.js: Secures HTTP headers on Express.js.
- Security layers are vital for protecting sensitive user data and transactions.

2.2 Hardware Requirements

2.2.1 Development Environment

- Processor
  - Minimum: Intel Core i5 (Quad-core).
  - Recommended: Intel Core i7 / AMD Ryzen 5 or higher.
  - Required for compiling code, running local servers, and multitasking efficiently.
- Ram
  - Minimum: 8 GB.
  - Recommended: 16 GB or more.
  - Sufficient memory is needed for running IDEs, local databases, and testing environments simultaneously.
- Storage
  - Minimum: 100 GB SSD.
  - Recommended: 250 GB SSD or higher.
  - SSD storage ensures faster file access, boot times, and system responsiveness during development.
- Internet Connection:
  - Minimum: 10 Mbps.

- Recommended: 50+ Mbps.
- A reliable internet connection is required for package installations, cloud syncing, and real-time API testing.

2.2.2 Production (Deployment) Server Requirements

➢ Processor (vCPU):

- Minimum: 2 vCPUs.
- Recommended: 4 vCPUs or more (for high traffic).

➢ RAM:

- Minimum: 4 GB.
- Recommended: 8 GB or more.
- Required for running the web server, database, and backend logic smoothly.

➢ Storage:

- Minimum: 100 GB SSD.
- Adequate space for logs, backups, and persistent database storage.

➢ Bandwidth:

- Unmetered / Scalable with Load.
- Sufficient to handle multiple users accessing and transacting on the platform simultaneously.

➢ Backup and Redundancy:

- Optional: RAID storage or cloud backup systems.
- Provides data safety and recovery options in case of server failure.

2.2.3 Devices for Testing

- Desktops/Laptops: With different OS (Windows, macOS, Linux).
- Smartphones/Tablets: Android and iOS devices.
- For ensuring cross-platform and responsive compatibility.

2.3 Functional Requirements

2.3.1User Module (Passenger Functionality)

- User Registration and Login:

  - Users must be able to create an account and log in securely.
  - Passwords should be stored securely using encryption (e.g., hash- ing).
  - Support for" Forgot Password" functionality.

- View Flight Schedules:

  - Users should be able to view available helicopter routes and timings in real-time.

- Search and Filter Flights:

  - Filter by source, destination, date, or availability.

- Book a Flight:

  - Select a route, date, and number of passengers.
  - Confirm booking after reviewing the fare and schedule.

- Reschedule or Cancel Booking:

  - Users should be able to modify or cancel bookings under policy constraints.

- Payment Gateway Integration:

  - Secure online payments via Stripe/PayPal using cards, UPI, etc.
  - View payment confirmation and transaction history.

- View Booking History:

  - Display of all past and upcoming reservations with status.

- Profile Management:

  - Edit personal details like name, phone, and email.

2.3.2Admin Module

- Admin Login:

  - Authorized admin users should log in to manage the backend operations.

- Manage Helipads:
  - Add, update, or delete helipad locations with coordinates and descriptions.
- Manage Routes:
  - Create and edit routes between helipads.
  - Set flight duration and availability.
- Monitor Bookings:
  - View all user bookings with filters for date, route, and status.
- Manage Payments:
  - Track completed, pending, and failed payments.
  - Export transaction reports for finance use.
- User Management:
  - View and manage registered users.
  - Suspend or flag accounts if required.

2.3.3 Helipad Operator Module (Optional Role)

- Update Helipad Status:
  - Operators can mark helipads as active/inactive due to weather or technical reasons.
- View Scheduled Flights:
  - Check upcoming flights assigned to their helipad.

2.3.4 System-Level Functionalities

- Real-time Data Updates:
  - Automatically refresh schedules, bookings, and availability without full page reload.
- Email/SMS Notifications:
  - Send booking confirmations, reschedule notices, and payment receipts to users.

2.3.5 Authentication & Authorization:

- Use JWT/OAuth to control access for different user roles (User, Admin, Operator).
- Error Handling & Alerts:

- Display meaningful error messages for invalid actions or server is- sues.

2.4Non-Functional Requirements

➢ Performance Requirements:

- The system should handle at least 100 concurrent users without performance degradation.
- API responses should be delivered in under 2 seconds under normal load conditions.
- The booking process from flight selection to payment should complete in under 1 minute.

➢ Scalability:

- The application should be able to scale horizontally by adding more servers as the user base grows.
- Database and server infrastructure should support increased data volumes (e.g., user data, bookings, transactions) without performance loss.

➢ Security Requirements:

- Use HTTPS (SSL/TLS) for all communication to protect data in transit.
- User passwords must be hashed and salted using strong encryption (e.g., bcrypt).
- Implement JWT-based authentication and OAuth for secure access control.
- The system must be resistant to SQL injection, XSS, and CSRF attacks.
- Payment data must comply with PCI-DSS standards when processed through payment gateways.

➢ Availability:

- The system should ensure 99.9% uptime, especially during high- demand periods (e.g., festivals or holidays).
- Implement auto-recovery mechanisms or failover support in case of server downtime.
- **https://www.fairtradecertified.org/** (InformUsability:
- The user interface should be intuitive and responsive, requiring minimal training or guidance.
- Ensure consistent UX across desktop, tablet, and mobile devices.
- Users should be able to complete tasks like booking, payment, and cancellations with minimal clicks.

➢ Maintainability:

- The codebase should follow modular and clean coding practices to simplify updates and debugging.
- Use version control (Git/GitHub) and maintain detailed documentation for future enhancements.

➢ Portability:

- The system should be deployable on multiple platforms, including AWS, Heroku, or other cloud services.
- The application should run smoothly across major browsers: Chrome, Firefox, Safari, Edge.

➢ Backup and Recovery:

- Daily automated backups of the database and application data must be maintained.
- A recovery system should ensure data can be restored within 2 hours of a major failure.

➢ Logging and Monitoring:

- Implement logging for user activities, payment attempts, and error events.
- Monitor server performance, API latency, and uptime using tools like New Relic, Datadog, or AWS CloudWatch.

Chapter 03: MODULE DESCRIPTION

3.1 User Management Module

- Purpose: Handles user registration, login, profile updates, and account management.

  Key Features:

  - User registration with email/phone verification.
  - Secure login with JWT authentication.
  - Profile management (name, email, phone).
  - View and manage booking history.

3.2 Booking Management Module

- Purpose: Enables users to browse routes, check availability, and book helicopter rides.

  Key Features:

  - Real-time search of routes and schedules.
  - Booking confirmation with route, date, and fare details.
  - Rescheduling and cancellation support.
  - Booking status updates via dashboard and notifications.

3.3 Payment Gateway Module

- Purpose: Facilitates secure and flexible payment transactions.

  Key Features:

  - Encrypted transaction processing.
  - Payment status tracking (success, pending, failed).
  - Generation of digital receipts and invoices.

3.4 Admin Management Module

- Purpose: Provides system administrators tools to control all backend data and operations.

Key Features:

- Admin login and dashboard access.
- Management of users, bookings, and financial records.
- Monitoring of system metrics and activity logs.
- Role-based access control for enhanced security.

3.5 Helipad & Route Management Module

- Purpose: Allows admins to manage helipad locations and flight routes between them.

Key Features:

- Add/edit/delete helipads with geolocation data.
- Define routes with source and destination helipads.
- Set availability, pricing, and flight durations.
- View routes on a map interface using Google Maps API.

3.6 Notification & Communication Module

- Purpose: Keeps users and admins informed about key events via messages and alerts.

Key Features:

- Email/SMS notifications for booking confirmations, cancellations, and reminders.
- System alerts for admins (e.g., failed payments, low availability).
- Optional push notification integration.

3.7 Reporting & Analytics Module

- Purpose: Offers insights into bookings, payments, and user activity.

Key Features:

- Generate reports on daily/monthly bookings.
- Revenue and payment summaries.
- Export options in PDF/CSV.

- Dashboard analytics for admin.

3.8 Security & Authentication Module

- Purpose: Ensures secure access and protects sensitive user data.

Key Features:

- Implementation of JWT-based secure authentication
- Password encryption using industry-standard hashing algorithms (e.g., bcrypt)
- Two-factor authentication (2FA) for users and admins
- Protection against common threats (e.g., SQL injection, XSS, CSRF)
- Regular session timeout and logout mechanisms

Chapter 04: OVER OF THE TECHNICAL AREA

Vayu Vihara is a sophisticated web-based helicopter booking system de- signed using a modern and scalable technology stack. The front end is developed using Next.js, a React-based framework that supports server- side rendering, enabling fast page loads, SEO optimization, and efficient routing. HTML5, CSS3, JavaScript (ES6+), and Tailwind CSS are used to create a clean, responsive, and mobile-friendly user interface.

The application supports cross-browser compatibility and is accessible on desktops, tablets, and smartphones. Client-side validation and dynamic routing ensure an interactive and error-free user experience. The backend is powered by Node.js with the Express.js framework, which offers robust RESTful API development and middleware capabilities for handling HTTP requests, routing, error handling, and authentication.

The system follows an MVC (Model-View-Controller) architecture to separate concerns and maintain clean code. For data persistence, PostgreSQL is used as a relational database management system. The database contains normalized tables such as Users, Bookings, Routes, Helipads, and Payments, ensuring referential integrity using primary and foreign keys. Sequalae or Prisma ORM is employed for interacting with the database using high-level JavaScript APIs.

Security is a core feature user passwords are securely hashed using crypt, and authentication is managed using JWT (JSON Web Tokens), along with optional OAuth for third-party logins. All data transfers are encrypted with HTTPS using TLS/SSL, and APIs are protected against CSRF, XSS, and SQL injections. The platform integrates Stripe or PayPal for secure online payments, supporting various methods like cards and UPI, and payment confirmation is sent via email or SMS using webhooks. The project is deployed on cloud platforms such as AWS, Heroku, or Vencel for scalability and uptime.

CI/CD pipelines using GitHub Actions automate testing, builds, and deployment processes. Application monitoring and logging are managed through tools like Sentry, Loggly, or AWS CloudWatch, offering real-time tracking of errors and performance. Google Maps API is integrated to display helipad locations and routes visually. The backend services are designed to scale horizontally, with asynchronous API calls and efficient caching mechanisms using Redis or similar tools.

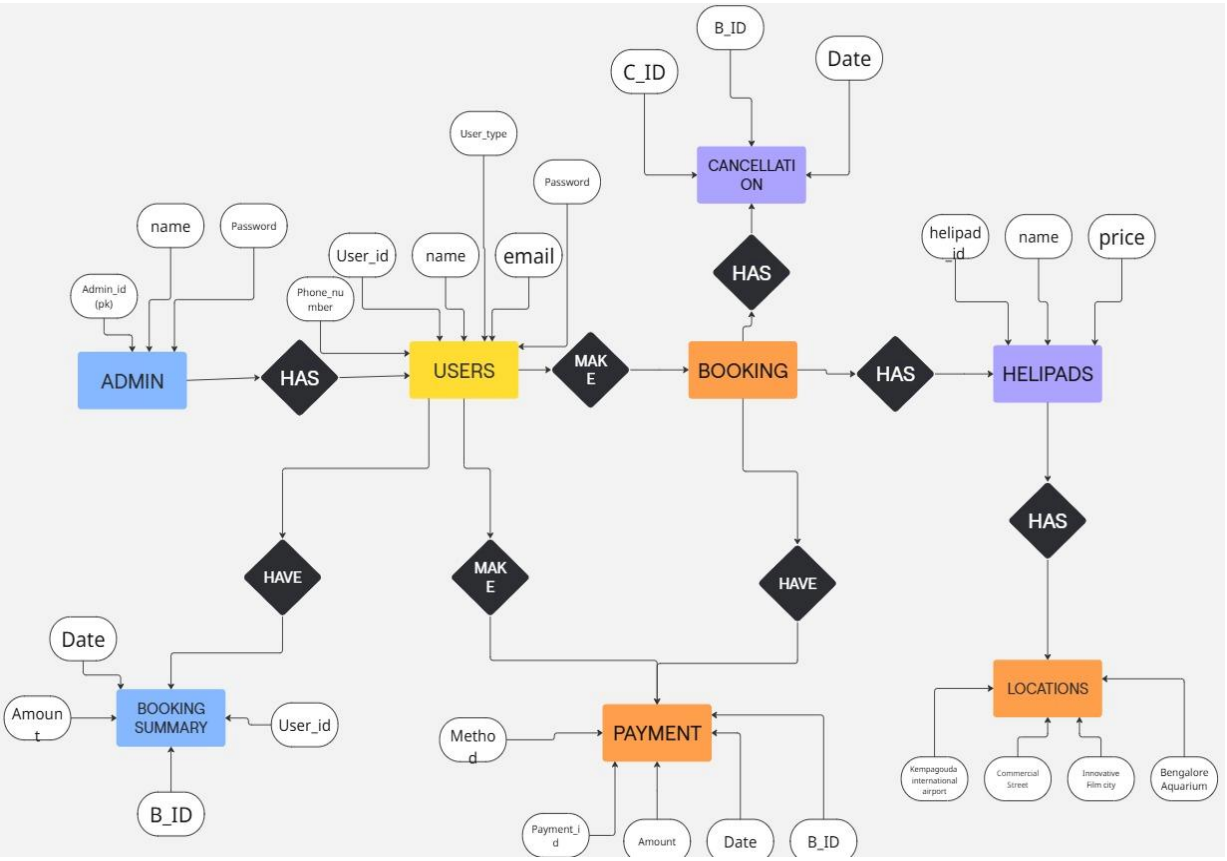
The APIs are versioned to ensure back- ward compatibility during upgrades. The system is modular, allowing for the easy addition of features and microservices in future expansions. Static assets and media are served via CDNs for reduced latency. The application also implements role-based access control (RBAC), ensuring different permissions for users, admins, and helipad operators.

Service reliability is enhanced with retry mechanisms, health checks, and load balancing. In the testing phase, unit, integration, usability, and security tests are conducted to ensure robustness and user satisfaction. Deployment logs and server metrics help in maintaining uptime and quick troubleshooting.

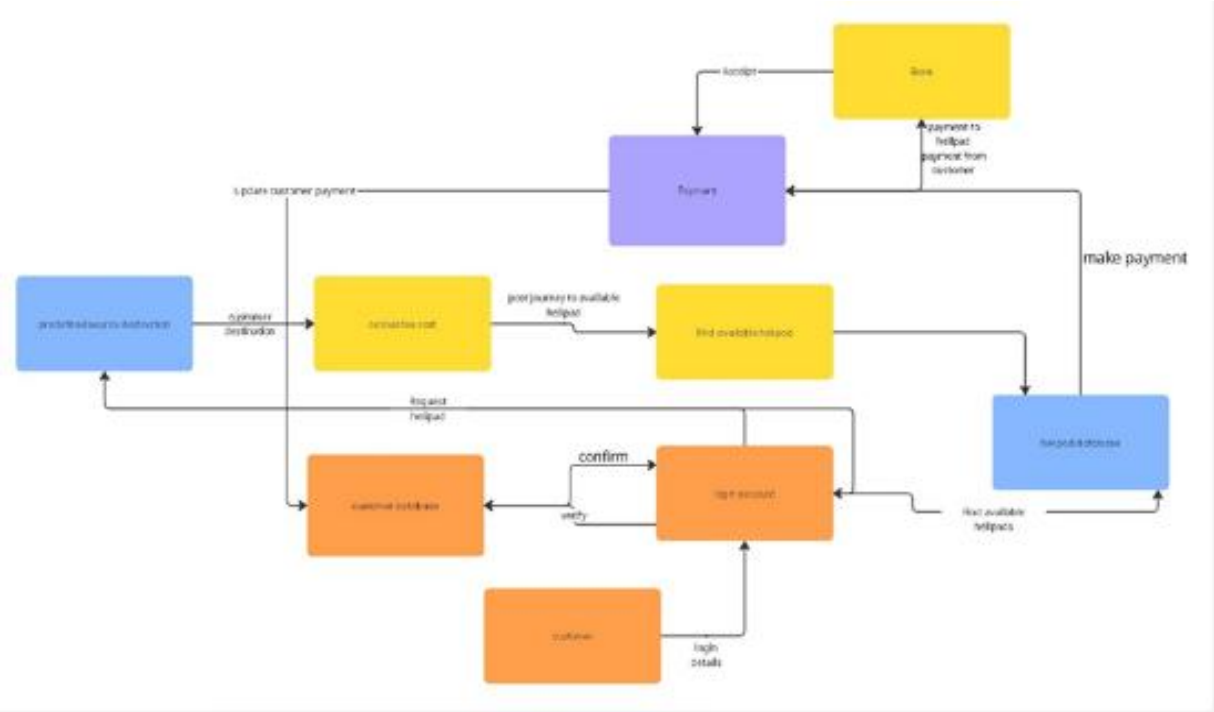
With this combination of cutting-edge technologies, Vayu Vihara delivers a secure, scalable, user-centric solution tailored for modern air travel booking needs. The technical foundation ensures high performance under concurrent usage, smooth API communication, real-time updates, and flexibility for evolving requirements, making it a future-ready aviation solution.

Chapter 05: PROGRAM REPRESENTATION

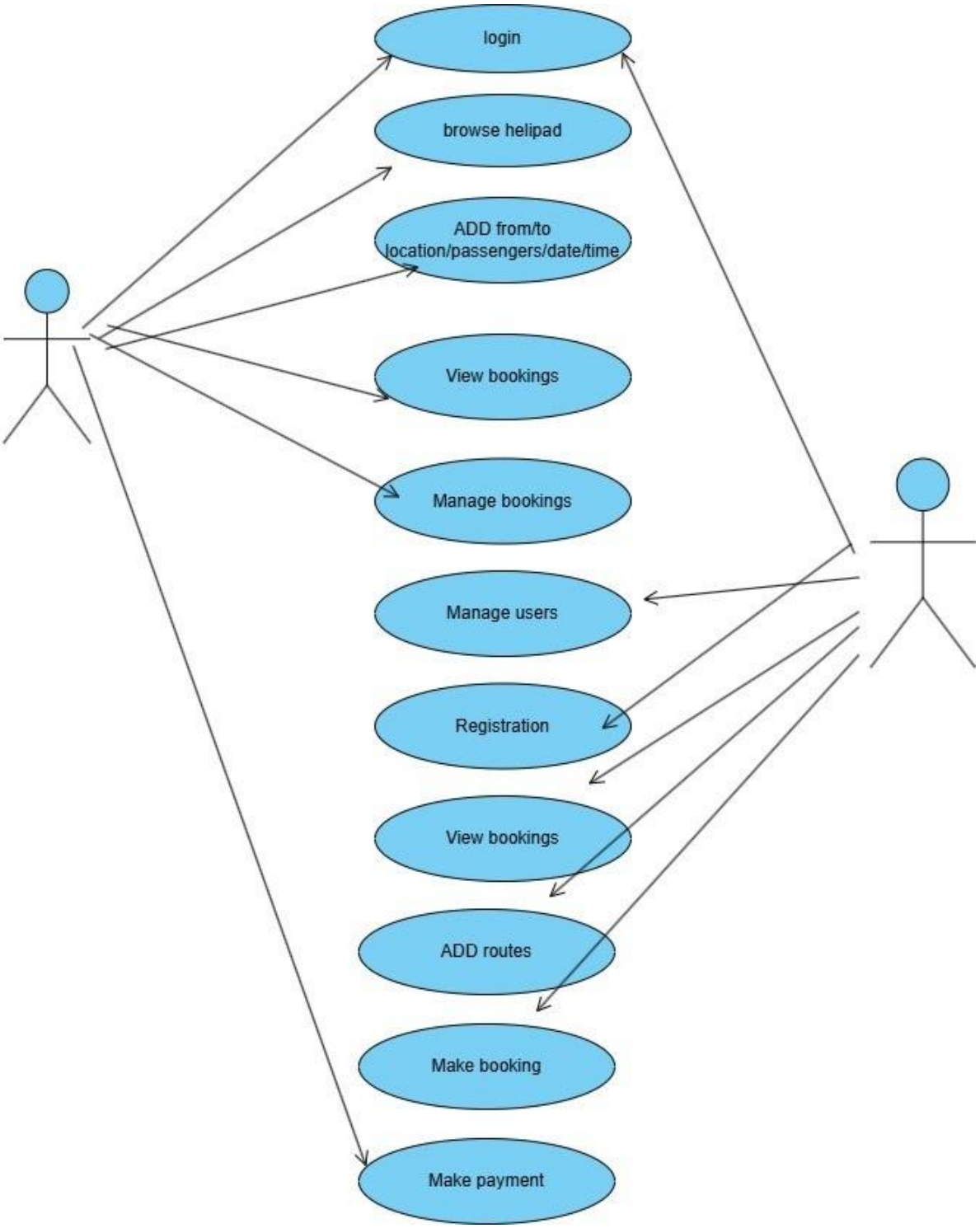
5.1 ER Diagram



5.2 Data Flow Diagram



Chapter 06: USE CASE DIAGRAM



Chapter 07: IMPLEMENTATION

The implementation phase of Vayu Vihara focused on building a scalable, secure, and responsive web application using agile methodology. The user interface was developed using React with Next.js, ensuring fast rendering, SEO optimization, and cross-platform compatibility. For the backend, Node.js was employed to handle API requests, business logic, and server communication, using JWT for authentication and session management. The PostgreSQL database schema was implemented with relational integrity, enabling efficient storage of user, booking, and payment data.

Stripe's secure API was used to handle online payments, allowing users to pay using a variety of methods, including credit/debit cards and UPI. The project was deployed to cloud platforms like AWS and Heroku,with CI/CD pipelines via GitHub Actions for automatic deployment, testing, and version control. Rigorous cross-browser and mobile testing ensured the application performs well on all major plat forms, supporting real - world use cases with consistency and efficiency.

Chapter 08: CODE

File: .env

```
# environment variables
```

File: .gitignore

```
node_modules
dist
.env
.DS_Store
*.log
```

File: components.json

```
{
 "compiler": {
 "name": "@webcomponents/webcomponentsjs"
 }
}
```

File: drizzle.config.ts

```
import type { Config } from "drizzle-kit";
import * as dotenv from "dotenv";
dotenv.config();
export default {
 schema: "./server/schema.ts",
 out: "./drizzle",
 driver: "turso",
 dbCredentials: {
 url: process.env.DATABASE_URL!,
 },
```

```
} satisfies Config;
```

File: package.json

```
{
 "name": "starter",
 "version": "0.0.0",
 "scripts": {
 "dev": "vite",
 "build": "tsc && vite build",
 "preview": "vite preview",
 "lint": "eslint . --ext ts --fix"
 },
 "dependencies": {
 "@repo/ui": "*",
 "lucide-react": "^0.324.0",
 "react": "^18.2.0",
 "react-dom": "^18.2.0"
 },
 "devDependencies": {
 "@typescript-eslint/eslint-plugin": "^6.22.0",
 "@typescript-eslint/parser": "^6.22.0",
 "autoprefixer": "^10.4.19",
 "eslint": "^8.56.0",
 "eslint-plugin-react": "^7.33.2",
 "postcss": "^8.4.38",
 "tailwindcss": "^3.4.3",
 "typescript": "^5.4.5",
 "vite": "^5.2.10"
```

```
 }
}
-
File: postcss.config.js
export default {
 plugins: {
 tailwindcss: {},
 autoprefixer: {},
 },
};
File: tailwind.config.ts
import type { Config } from "tailwindcss";
const config: Config = {
 content: ["./client/**/*.{ts,tsx}"],
 theme: {
 extend: {},
 },
 plugins: [],
};
export default config;
File: tsconfig.json
{
 "compilerOptions": {
 "target": "ESNext",
 "useDefineForClassFields": true,
 "module": "ESNext",
 "moduleResolution": "Bundler",
```

```
 "strict": true,
 "jsx": "react-jsx",
 "resolveJsonModule": true,
 "isolatedModules": true,
 "esModuleInterop": true,
 "allowImportingTsExtensions": true,
 "forceConsistentCasingInFileNames": true,
 "skipLibCheck": true,
 "baseUrl": ".",
 "paths": {
 "~/*": ["./*"]
 },
 "types": ["vite/client"]
 },
 "include": ["client", "server", "shared"]
}
```

File: vite.config.ts

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
export default defineConfig({
 plugins: [react()],
 server: {
 port: 3000,
 },
});
```

-

File: .replit.txt

run = "npm run dev"

language = "nodejs"

Client File: DatabaseManager (1)/DatabaseManager/client/index.html

```
<!DOCTYPE html>

<html lang="en">

 <head>

 <meta charset="UTF-8" />

 <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1" />

 <title>Vayu Vihar - Helipad Booking Platform</title>

 <link rel="preconnect" href="https://fonts.googleapis.com">

 <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

 <link

href="https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;600;700&family=Open+Sans:wght@400;500;60

0&display=swap" rel="stylesheet">

 <link rel="icon" type="image/svg+xml" href="data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg'

width='24' height='24' viewBox='0 0 24 24' fill='none' stroke='%230056b3' stroke-width='2'

stroke-linecap='round' stroke-linejoin='round'%3E%3Cpath d='m12 4 1 2 5 1-5 1-1 2-1-2-5-1 5-1 1-2Z'/%3E%3Cpath

d='M7.5 12.5, L6 15H4l1.5-5 2-1 1.5 3.5-1.5-1z'/%3E%3Cpath d='M16.5 12.5, L18 15h2l-1.5-5-2-1-1.5 3.5

1.5-1z'/%3E%3Cpath d='M12 10v10'/%3E%3Cpath d='m8 16 4 4 4-4'/%3E%3C/svg%3E" />

 </head>

 <body>

 <div id="root"></div>

 <script type="module" src="/src/main.tsx"></script>

 <!-- This is a replit script which adds a banner on the top of the page when opened in development mode
```

```
outside the replit environment -->
<script type="text/javascript" src="https://replit.com/public/js/replit-dev-banner.js"></script>
</body>
</html>
```

Client File: DatabaseManager (1)/DatabaseManager/client/client/eslint.config.js

```
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
export default [
 { ignores: ['dist'] },
 {
 files: ['**/*.{js,jsx}'],
 languageOptions: {
 ecmaVersion: 2020,
 globals: globals.browser,
 parserOptions: {
 ecmaVersion: 'latest',
 ecmaFeatures: { jsx: true },
 sourceType: 'module',
 },
 },
 plugins: {
 'react-hooks': reactHooks,
 'react-refresh': reactRefresh,
 },
 rules: {
```

```
 ...js.configs.recommended.rules,
 ...reactHooks.configs.recommended.rules,
 'no-unused-vars': ['error', { varsIgnorePattern: '^[A-Z_]' }],
 'react-refresh/only-export-components': [
 'warn',
 { allowConstantExport: true },
 ],
 },
 },
]
```

-

Client File: DatabaseManager (1)/DatabaseManager/client/client/index.html

```
<!doctype html>
<html lang="en">
 <head>
 <meta charset="UTF-8" />
 <link rel="icon" type="image/svg+xml" href="/vite.svg" />
 <meta name="viewport" content="width=device-width, initial-scale=1.0" />
 <title>Vite + React</title>
 </head>
 <body>
 <div id="root"></div>
 <script type="module" src="/src/main.jsx"></script>
 </body>
</html>
```

Client File: DatabaseManager (1)/DatabaseManager/client/client/package-lock.json

```
{
```

"name": "client",

"version": "0.0.0",

"lockfileVersion": 3,

"requires": true,

"packages": {

"": {

"name": "client",

"version": "0.0.0",

"dependencies": {

"react": "^19.1.0",

"react-dom": "^19.1.0"

},

"devDependencies": {

"@eslint/js": "^9.25.0",

"@types/react": "^19.1.2",

"@types/react-dom": "^19.1.2",

"@vitejs/plugin-react": "^4.4.1",

"eslint": "^9.25.0",

"eslint-plugin-react-hooks": "^5.2.0",

"eslint-plugin-react-refresh": "^0.4.19",

"globals": "^16.0.0",

"vite": "^6.3.5"

}

},

"node_modules/@ampproject/remapping": {

"version": "2.3.0",

"resolved": "https://registry.npmjs.org/@ampproject/remapping/-/remapping-2.3.0.tgz",

"integrity":

"sha512-
30iZtAPgz+LTIYoeivqYo853f02jBYSd5uGnGpkFV0M3xOt9aN73erkgYAmZU43x4V
fqcnLxW9Kpg3R5LC4YYw==",

"dev": true,

"license": "Apache-2.0",

"dependencies": {

"@jridgewell/gen-mapping": "^0.3.5",

"@jridgewell/trace-mapping": "^0.3.24"

},

"engines": {

"node": ">=6.0.0"

}

},

"node_modules/@babel/code-frame": {

"version": "7.27.1",

"resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.27.1.tgz",

"integrity":

"sha512-
cjQ7ZlQ0Mv3b47hABuTevyTuYN4i+loJKGeV9flcCgIK37cCXRh+L1bd3iBHlynerhQ
7BhCkn2BPbQUL+rGqFg==",

"dev": true,

"license": "MIT",

"dependencies": {

"@babel/helper-validator-identifier": "^7.27.1",

"jstokens":"^400"

-

Client File: DatabaseManager (1)/DatabaseManager/client/client/package.json

{

```
"name": "client",
"private": true,
"version": "0.0.0",
"type": "module",
"scripts": {
"dev": "vite",
"build": "vite build",
"lint": "eslint .",
"preview": "vite preview"
},
"dependencies": {
"react": "^19.1.0",
"react-dom": "^19.1.0"
},
"devDependencies": {
"@eslint/js": "^9.25.0",
"@types/react": "^19.1.2",
"@types/react-dom": "^19.1.2",
"@vitejs/plugin-react": "^4.4.1",
"eslint": "^9.25.0",
"eslint-plugin-react-hooks": "^5.2.0",
"eslint-plugin-react-refresh": "^0.4.19",
"globals": "^16.0.0",
"vite": "^6.3.5"
}
}
```

Client File: DatabaseManager (1)/DatabaseManager/client/client/vite.config.js

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
// https://vite.dev/config/
export default defineConfig({
 plugins: [react()],
})
```

Client File: DatabaseManager (1)/DatabaseManager/client/client/src/App.css

```
#root {
 max-width: 1280px;
 margin: 0 auto;
 padding: 2rem;
 text-align: center;
}
.logo {
 height: 6em;
 padding: 1.5em;
 will-change: filter;
 transition: filter 300ms;
}
.logo:hover {
 filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
 filter: drop-shadow(0 0 2em #61dafbaa);
}
@keyframes logo-spin {
 from {
```

```
 transform: rotate(0deg);
 }
 to {
transform:rotate(360deg);
-
Client File: DatabaseManager (1)/DatabaseManager/client/client/src/index.css
:root {
 font-family: system-ui, Avenir, Helvetica, Arial, sans-serif;
 line-height: 1.5;
 font-weight: 400;
 color-scheme: light dark;
 color: rgba(255, 255, 255, 0.87);
 background-color: #242424;
 font-synthesis: none;
 text-rendering: optimizeLegibility;
 -webkit-font-smoothing: antialiased;
 -moz-osx-font-smoothing: grayscale;
}
a {
 font-weight: 500;
 color: #646cff;
 text-decoration: inherit;
}
a:hover {
 color: #535bf2;
}
body {
```

```
 margin: 0;
 display: flex;
 place-items: center;
 min-width: 320px;
 min-height: 100vh;
}
h1 {
 font-size: 3.2em;
 line-height: 1.1;
}
button {
 border-radius: 8px;
 border: 1px solid transparent;
 padding: 0.6em 1.2em;
 font-size: 1em;
 font-weight: 500;
 font-family: inherit;
 background-color: #1a1a1a;
 cursor: pointer;
 transition: border-color 0.25s;
}
button:hover {
 border-color: #646cff;
}
button:focus,
button:focus-visible {
 outline: 4px auto -webkit-focus-ring-color;
```

```
}
@media (prefers-color-scheme: light) {
 :root {
 color: #213547;
 background-color: #ffffff;
 }
 a:hover {
 color: #747bff;
 }
 button {
```

-

Client File: DatabaseManager (1)/DatabaseManager/client/src/App.tsx

```
import { Switch, Route } from "wouter";
import { QueryClientProvider } from "@tanstack/react-query";
import { queryClient } from "./lib/queryClient";
import { Toaster } from "@/components/ui/toaster";
import { TooltipProvider } from "@/components/ui/tooltip";
import { AuthProvider } from "@/context/AuthContext";
// Pages
import HomePage from "@/pages/HomePage";
import BookingPage from "@/pages/BookingPage";
import MyBookingsPage from "@/pages/MyBookingsPage";
import AboutPage from "@/pages/AboutPage";
import HelpCenterPage from "@/pages/HelpCenterPage";
import ProfilePage from "@/pages/ProfilePage";
import ContactPage from "@/pages/ContactPage";
```

```
import PrivacyPage from "@/pages/PrivacyPage";
import TermsPage from "@/pages/TermsPage";
import AdminLogin from "@/pages/admin/AdminLogin";
import AdminDashboard from "@/pages/admin/AdminDashboard";
import AdminHelipads from "@/pages/admin/AdminHelipads";
import AdminBookings from "@/pages/admin/AdminBookings";
import AdminUsers from "@/pages/admin/AdminUsers";
import NotFound from "@/pages/not-found";
function Router() {
 return (
 <Switch>
 {/* Public routes */}
 <Route path="/" component={HomePage} />
 <Route path="/booking" component={BookingPage} />
 <Route path="/qr-payment" component={BookingPage} />
 <Route path="/my-bookings" component={MyBookingsPage} />
 <Route path="/profile" component={ProfilePage} />
 <Route path="/settings" component={ProfilePage} />
 <Route path="/about" component={AboutPage} />
 <Route path="/help" component={HelpCenterPage} />
 <Route path="/contact" component={ContactPage} />
 <Route path="/privacy" component={PrivacyPage} />
 <Route path="/terms" component={TermsPage} />

 {/* Admin routes */}
 <Route path="/admin/login" component={AdminLogin} />
 <Route path="/admin" component={AdminDashboard} />
```

```
<Route path="/admin/helipads" component={AdminHelipads} />
<Route path="/admin/bookings" component={AdminBookings} />
<Route path="/admin/users" component={AdminUsers} />

{/* Fallback to 404 */}
<Route component={NotFound} />
</Switch>
);
}
function App() {
return (
<QueryClientProvider client={queryClient}>
<AuthProvider>
<TooltipProvider>
<Toaster />
<Router />
</TooltipProvider>
</AuthProvider>
</QueryClientProvider>
);
}
```

-

Client File: DatabaseManager (1)/DatabaseManager/client/src/index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;
```

```
:root {
 --background: 0 0% 100%;
--foreground: 20 14.3% 4.1%;
--muted: 60 4.8% 95.9%;
--muted-foreground: 25 5.3% 44.7%;
--popover: 0 0% 100%;
--popover-foreground: 20 14.3% 4.1%;
--card: 0 0% 100%;
--card-foreground: 20 14.3% 4.1%;
--border: 20 5.9% 90%;
--input: 20 5.9% 90%;
--primary: 207 90% 54%;
--primary-foreground: 211 100% 99%;
--secondary: 60 4.8% 95.9%;
--secondary-foreground: 24 9.8% 10%;
--accent: 60 4.8% 95.9%;
--accent-foreground: 24 9.8% 10%;
--destructive: 0 84.2% 60.2%;
--destructive-foreground: 60 9.1% 97.8%;
--ring: 20 14.3% 4.1%;
--radius: 0.5rem;
 }
 .dark {
 --background: 240 10% 3.9%;
--foreground: 0 0% 98%;
--muted: 240 3.7% 15.9%;
--muted-foreground: 240 5% 64.9%;
```

```
--popover: 240 10% 3.9%;
--popover-foreground: 0 0% 98%;
--card: 240 10% 3.9%;
--card-foreground: 0 0% 98%;
--border: 240 3.7% 15.9%;
--input: 240 3.7% 15.9%;
--primary: 207 90% 54%;
--primary-foreground: 211 100% 99%;
--secondary: 240 3.7% 15.9%;
--secondary-foreground: 0 0% 98%;
--accent: 240 3.7% 15.9%;
--accent-foreground: 0 0% 98%;
--destructive: 0 62.8% 30.6%;
--destructive-foreground: 0 0% 98%;
--ring: 240 4.9% 83.9%;
--radius: 0.5rem;
 }
@layer base {
 * {
 @apply border-border;
 }
 body {
 @apply font-sans antialiased bg-background text-foreground;
 }
}
```

-

Client File: DatabaseManager (1)/DatabaseManager/client/src/main.tsx

```
import { createRoot } from "react-dom/client";
import App from "./App";
import "./index.css";
const root = document.getElementById("root");
if (!root) {
 throw new Error("Root element not found");
}
// Add Google and Facebook SDK for OAuth login
const loadScript = (id: string, src: string) => {
 if (document.getElementById(id)) return;

 const script = document.createElement("script");
 script.id = id;
 script.src = src;
 script.async = true;
 script.defer = true;
 document.head.appendChild(script);
};
// Load Google API script
loadScript(
 "google-api",
 "https://accounts.google.com/gsi/client"
);
// Load Facebook SDK
loadScript(
 "facebook-sdk",
 "https://connect.facebook.net/en_US/sdk.js"
```

```
);
// Declare FB on the window object to satisfy TypeScript
declare global {
 interface Window {
 FB: any;
 fbAsyncInit: () => void;
 }
}
// Initialize Facebook SDK
window.fbAsyncInit = function() {
 window.FB.init({
 appId: import.meta.env.VITE_FACEBOOK_APP_ID || '123456789',
 cookie: true,
 xfbml: true,
 version: 'v18.0'
 });
};
createRoot(root).render(<App />);
```

Client File: DatabaseManager (1)/DatabaseManager/client/src/components/admin/AdminSidebar.tsx

```
import { useState } from "react";
import { Link, useLocation } from "wouter";
import { useAuth } from "@/context/AuthContext";
import { Button } from "@/components/ui/button";
import { useDarkMode } from "@/hooks/use-dark-mode";
import {
 BarChart3,
 Plane,
```

```
 Users,
Calendar
-
Client File: DatabaseManager
(1)/DatabaseManager/client/src/components/admin/BookingsManager.tsx
import { useState } from "react";
import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
import { Booking, bookingStatusEnum } from "@shared/schema";
import { apiRequest } from "@/lib/queryClient";
import { useToast } from "@/hooks/use-toast";
import { formatCurrency, formatDate } from "@/lib/utils";
import {
 Card,
 CardContent,
 CardDescription,
 CardHeader,
 CardTitle,
} from "@/components/ui/card";
import {
 Table,
 TableBody,
 TableCell,
 TableHead,
 TableHeader,
 TableRow,
} from "@/components/ui/table";
import {
```

```
 Dialog,
 DialogContent,
 DialogDescription,
 DialogFooter,
 DialogHeader,
 DialogTitle,
} from "@/components/ui/dialog";
import {
 AlertDialog,
 AlertDialogAction,
 AlertDialogCancel,
 AlertDialogContent,
 AlertDialogDescription,
 AlertDialogFooter,
 AlertDialogHeader,
 AlertDialogTitle,
} from "@/components/ui/alert-dialog";
import { Button } from "@/components/ui/button";
import { Skeleton } from "@/components/ui/skeleton";
import { Input } from "@/components/ui/input";
import {
 Select,
 SelectContent,
 SelectItem,
 SelectTrigger,
 SelectValue,
} from "@/components/ui/select";
```

```
import {
 Eye,
 Ban,
 CheckCircle,
 Search,
 ChevronLeft,
 ChevronRight,
 MessageSquare,
 Calendar,
 User,
 MapPin,
 Clock,
CreditCard,
-
```

Client File: DatabaseManager (1)/DatabaseManager/client/src/components/admin/Dashboard.tsx

```
import { useQuery } from "@tanstack/react-query";
import {
 Card,
 CardContent,
 CardDescription,
 CardHeader,
 CardTitle
} from "@/components/ui/card";
import {
 Tabs,
 TabsContent,
 TabsList,
```

```
 TabsTrigger
} from "@/components/ui/tabs";
import { Button } from "@/components/ui/button";
import { formatCurrency } from "@/lib/utils";
import {
 AreaChart,
 BarChart,
 ResponsiveContainer,
 Area,
 Bar,
 XAxis,
 YAxis,
 CartesianGrid,
 Tooltip,
 Legend
} from "recharts";
import {
 Calendar,
 Plane,
 Users,
 DollarSign,
 Clock
} from "lucide-react";
// Types for the dashboard data
interface DashboardStats {
 totalBookings: number;
 totalRevenue: number;
```

```
 totalUsers: number;
 totalHelipads: number;
 pendingBookings: number;
}
interface BookingData {
 name: string;
 bookings: number;
 revenue: number;
}
export default function Dashboard() {
 // Fetch dashboard statistics
 const { data: stats, isLoading: isStatsLoading } = useQuery<DashboardStats>({
 queryKey: ['/api/admin/statistics'],
 });
 // Fetch booking analytics
 const { data: dailyData, isLoading: isDailyLoading } = useQuery<BookingData[]>({
 queryKey: ['/api/admin/analytics/daily'],
 });
 const { data: weeklyData, isLoading: isWeeklyLoading } = useQuery<BookingData[]>({
 queryKey: ['/api/admin/analytics/weekly'],
 });
```

-

Client File: DatabaseManager

(1)/DatabaseManager/client/src/components/admin/HelipadsManager.tsx

```
import { useState } from "react";
import { useQuery, useMutation, useQueryClient } from "@tanstack/react-query";
```

```
import { Helipad, insertHelipadSchema } from "@shared/schema";
import { zodResolver } from "@hookform/resolvers/zod";
import { useForm } from "react-hook-form";
import { apiRequest } from "@/lib/queryClient";
import { z } from "zod";
import { useToast } from "@/hooks/use-toast";
import { formatCurrency } from "@/lib/utils";
import {
 Card,
 CardContent,
 CardDescription,
 CardHeader,
 CardTitle,
} from "@/components/ui/card";
import {
 Table,
 TableBody,
 TableCell,
 TableHead,
 TableHeader,
 TableRow,
} from "@/components/ui/table";
import {
 Dialog,
 DialogContent,
 DialogDescription,
 DialogFooter,
```

```
 DialogHeader,
 DialogTitle,
} from "@/components/ui/dialog";
import {
 AlertDialog,
 AlertDialogAction,
 AlertDialogCancel,
 AlertDialogContent,
 AlertDialogDescription,
 AlertDialogFooter,
 AlertDialogHeader,
 AlertDialogTitle,
} from "@/components/ui/alert-dialog";
import {
 Form,
 FormControl,
 FormField,
 FormItem,
 FormLabel,
 FormMessage,
} from "@/components/ui/form";
import { Input } from "@/components/ui/input";
import { Textarea } from "@/components/ui/textarea";
import { Button } from "@/components/ui/button";
import { Checkbox } from "@/components/ui/checkbox";
import { Skeleton } from "@/components/ui/skeleton";
import {
```

```
 PlusCircle,
 Pencil,
 Trash2,
 MapPin,
 Search,
ChevronLeft,
-
```

Client File: DatabaseManager (1)/DatabaseManager/client/src/components/admin/UsersManager.tsx

```
import { useState } from "react";
import { useQuery, useMutation } from "@tanstack/react-query";
import { queryClient, apiRequest } from "@/lib/queryClient";
import { useToast } from "@/hooks/use-toast";
import { formatDate } from "@/lib/utils";
import jsPDF from "jspdf";
import autoTable from "jspdf-autotable";
import {
 Card,
 CardContent,
 CardDescription,
 CardHeader,
 CardTitle,
} from "@/components/ui/card";
import {
 Table,
 TableBody,
 TableCell,
 TableHead,
```

```
 TableHeader,
 TableRow,
} from "@/components/ui/table";
import { Button } from "@/components/ui/button";
import { Badge } from "@/components/ui/badge";
import {
 Dialog,
 DialogContent,
 DialogDescription,
 DialogHeader,
 DialogTitle,
} from "@/components/ui/dialog";
import { Eye, Shield, User, Phone, Mail, Calendar } from "lucide-react";
import { Select, SelectContent, SelectItem, SelectTrigger, SelectValue } from
   "@/components/ui/select";
interface User {
 id: number;
 name: string;
 email: string;
 phone?: string;
 role: 'user' | 'admin';
 authProvider?: string;
 createdAt: string;
}
interface RouteType {
 id: number;
 name: string;
 sourceLocation: string;
```

```
 destinationLocation: string;
 basePrice: number;
 duration: string;
 distance: number;
}
interface BookingType {
 bookingReference: string;
 bookingDate: string;
 userId: number;
 passengers: number;
 totalAmount: number;
 booking_status: string;
 payment_status: string;
}
```

-

Client File: DatabaseManager (1)/DatabaseManager/client/src/components/auth/AuthModal.tsx

```
import { useState } from "react";
import { X, Eye, EyeOff } from "lucide-react";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { z } from "zod";
import {
 Dialog,
 DialogContent,
 DialogHeader,
 DialogTitle,
```

```
} from "@/components/ui/dialog";
import {
 Form,
 FormControl,
 FormField,
 FormItem,
 FormLabel,
 FormMessage,
} from "@/components/ui/form";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import { Separator } from "@/components/ui/separator";
import { useAuth } from "@/context/AuthContext";
// Form schemas with enhanced validation
const loginSchema = z.object({
 email: z.string()
 .email("Invalid email address")
 .regex(/^[^\s@]+@[^\s@]+\.[^\s@]+$/, "Please enter a valid email format"),
 password: z.string()
 .min(1, "Password is required")
 .max(100, "Password is too long"),
});
const registerSchema = z.object({
 name: z.string()
 .min(2, "Name must be at least 2 characters")
 .max(50, "Name must be less than 50 characters")
```

```
.regex(/^[a-zA-Z\s]+$/, "Name can only contain letters and spaces"),
email: z.string()
.email("Invalid email address")
.regex(/^[^\s@]+@[^\s@]+\.[^\s@]+$/, "Please enter a valid email format"),
phone: z.string()
.regex(/^[6-9]\d{9}$/, "Please enter a valid 10-digit Indian mobile number")
.optional()
.or(z.literal("")),
password: z.string()
.min(8, "Password must be at least 8 characters")
.max(50, "Password must be less than 50 characters")
.regex(/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/, "Password must contain at least one uppercase letter, one
lowercase letter, and one number"),
confirmPassword: z.string(),
}).refine(data => data.password === data.confirmPassword, {
message: "Passwords don't match",
path: ["confirmPassword"]
});
const adminLoginSchema = z.object({
email: z.string()
.email("Invalid email address")
.regex(/^[^\s@]+@[^\s@]+\.[^\s@]+$/, "Please enter a valid email format"),
password: z.string()
.min(6, "Password must be at least 6 characters")
```

-

Client File: DatabaseManager

(1)/DatabaseManager/client/src/components/booking/BookingSection.tsx

```
import { useState } from 'react';
import { Tabs, TabsContent, TabsList, TabsTrigger } from "@/components/ui/tabs";
import PredefinedBookingForm from './PredefinedBookingForm';
import CustomBookingForm from './CustomBookingForm';
export default function BookingSection() {
 const [bookingType, setBookingType] = useState<'predefined' | 'custom'>('predefined');
 return (
 <section id="booking" className="py-16 bg-white dark:bg-neutral-800">
 <div className="container mx-auto px-4">
 <div className="text-center mb-12">
 <h2 className="text-3xl font-bold font-heading mb-2">Book Your Helicopter</h2>
 <p className="text-neutral-600 dark:text-neutral-300">Select your preferred booking
   mode</p>
 </div>

 <Tabs
 defaultValue="predefined"
 value={bookingType}
 onValueChange={(value) => setBookingType(value as 'predefined' | 'custom')}
 className="max-w-4xl mx-auto"
 >
 <TabsList className="grid w-full grid-cols-2 mb-8">
 <TabsTrigger value="predefined">Predefined Routes</TabsTrigger>
 <TabsTrigger value="custom">Custom Route</TabsTrigger>
 </TabsList>

 <TabsContent value="predefined">
```

```
<PredefinedBookingForm />
</TabsContent>

<TabsContent value="custom">
<CustomBookingForm />
</TabsContent>
</Tabs>
</div>
</section>
);
}
```

Client File: DatabaseManager

(1)/DatabaseManager/client/src/components/booking/CustomBookingForm.tsx

```
import { useState } from 'react';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { useQuery } from '@tanstack/react-query';
import { z } from 'zod';
import { useMutation } from '@tanstack/react-query';
import { Helicopter } from '@shared/schema';
import { apiRequest } from '@/lib/queryClient';
import { useToast } from '@/hooks/use-toast';
import { useAuth } from '@/context/AuthContext';
import AuthModal from '@/components/auth/AuthModal';
import {
 Card,
 CardContent,
```

```
} from '@/components/ui/card';
import {
 Form,
FormControl
-
Client File: DatabaseManager
(1)/DatabaseManager/client/src/components/booking/PredefinedBookingForm.tsx
import { useState } from 'react';
import { useForm } from 'react-hook-form';
import { zodResolver } from '@hookform/resolvers/zod';
import { useQuery } from '@tanstack/react-query';
import { z } from 'zod';
import { useMutation } from '@tanstack/react-query';
import { Route } from '@shared/schema';
import { apiRequest } from '@/lib/queryClient';
import { useToast } from '@/hooks/use-toast';
import { useAuth } from '@/context/AuthContext';
import AuthModal from '@/components/auth/AuthModal';
import {
 Card,
 CardContent,
} from '@/components/ui/card';
import {
 Form,
 FormControl,
 FormField,
 FormItem,
```

```
 FormLabel,
 FormMessage,
} from '@/components/ui/form';
import { Button } from '@/components/ui/button';
import {
 Select,
 SelectContent,
 SelectItem,
 SelectTrigger,
 SelectValue,
} from '@/components/ui/select';
import { Input } from '@/components/ui/input';
import { formatCurrency, calculateGST, calculateTotalWithGST } from '@/lib/utils';
// Form schema with enhanced validation and passenger details
const formSchema = z.object({
 routeId: z.string().min(1, 'Please select a route'),
 bookingDate: z.string().min(1, 'Please select a date'),
 bookingTime: z.string().min(1, 'Please select a time'),
 passengers: z.string().min(1, 'Please select number of passengers'),
 contactName: z.string()
 .min(2, "Name must be at least 2 characters")
 .regex(/^[a-zA-Z\s]+$/, "Name can only contain letters and spaces"),
 contactEmail: z.string()
 .email("Invalid email address")
 .regex(/^[^\s@]+@[^\s@]+\.[^\s@]+$/, "Please enter a valid email format"),
 contactPhone: z.string()
 .regex(/^[6-9]\d{9}$/, "Please enter a valid 10-digit Indian mobile number"),
```

```
 passengerDetails: z.array(z.object({
 name: z.string()
 .min(2, "Name must be at least 2 characters")
 .regex(/^[a-zA-Z\s]+$/, "Name can only contain letters and spaces"),
 email: z.string()
 .email("Invalid email address")
 .regex(/^[^\s@]+@[^\s@]+\.[^\s@]+$/, "Please enter a valid email format"),
 })).optional(),
});
type FormData = z.infer<typeof formSchema>;
// Sample time slots
consttimeSlots=[
-
Client File: DatabaseManager (1)/DatabaseManager/client/src/components/help/Chatbot.tsx
import { useState, useRef, useEffect } from "react";
import { X, Send, MessageCircle, User } from "lucide-react";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { Badge } from "@/components/ui/badge";
import { Separator } from "@/components/ui/separator";
interface Message {
 id: string;
 text: string;
 isUser: boolean;
 timestamp: Date;
}
// Responses for demonstration purposes
```

```
const botResponses: Record<string, string> = {

 "hello": "Hello! How can I help you today with Vayu Vihar's helicopter booking
    services?",

 "hi": "Hi there! How can I assist you with your helicopter booking needs?",

 "book": "To book a helicopter, go to our booking page and choose between a predefined
    route or a custom

journey. You can access the booking page from the navigation menu or through the 'Book
    Now' button on the

homepage.",

 "cancel": "You can cancel a booking up to 24 hours before the scheduled time for a full
    refund. To cancel, go

to 'My Bookings', select the booking you wish to cancel, and click the 'Cancel Booking'
    button.",

 "refund": "Refunds are processed automatically to your original payment method. For
    cancellations made at

least 24 hours in advance, you'll receive a full refund. Cancellations within 24 hours are
    subject to a 50%

cancellation fee.",

 "payment": "We accept all major credit/debit cards, UPI, and net banking through our secure
    Razorpay payment

gateway. All transactions are encrypted for your security.",

 "contact": "You can reach our customer support team at +91 80 4567 8900 or email us at
    support@vayuvihar.com.

Our office hours are Monday to Saturday, 8:00 AM to 8:00 PM, and Sunday, 10:00 AM to
    6:00 PM.",

 "luggage": "Each passenger is allowed up to 5kg of hand baggage. For specific requirements
    or additional

baggage, please contact our support team in advance.",

 "children": "Children of all ages are welcome. Children under 2 years can sit on an adult's
    lap, while those

2 and above require their own seat. Please inform us about children during booking.",

 "weather": "If weather conditions make flying unsafe, we'll contact you to reschedule your
    booking at no
```

```
additional cost. Alternatively, you can request a full refund if rescheduling doesn't
    work for you.",

 "time": "Please arrive at least 15 minutes before your scheduled departure time for check-in
    and safety

briefing. Arriving late may result in missing your flight with no refund.",

};

// Helper function to find the best response

function findBestResponse(query: string): string {

 query = query.toLowerCase().trim();


 // Direct match

 if (botResponses[query]) {

 return botResponses[query];

 }


 // Keyword match

 for (const [key, response] of Object.entries(botResponses)) {

 if (query.includes(key)) {

 return response;

 }

 }


 // Default response

 return "I'm not sure how to answer that. For specific assistance, please contact our support
    team at +91 80

4567 8900 or email us at support@vayuvihar.com.";

}

interface ChatbotProps {

 onClose: () => void;
```

```
}
```

-

Client File: DatabaseManager (1)/DatabaseManager/client/src/components/help/ContactForm.tsx

```
import { useState } from "react";
import { useForm } from "react-hook-form";
import { zodResolver } from "@hookform/resolvers/zod";
import { z } from "zod";
import { useToast } from "@/hooks/use-toast";
import { apiRequest } from "@/lib/queryClient";
import {
 Form,
 FormControl,
 FormField,
 FormItem,
 FormLabel,
 FormMessage,
} from "@/components/ui/form";
import { Input } from "@/components/ui/input";
import { Textarea } from "@/components/ui/textarea";
import { Button } from "@/components/ui/button";
import { Loader2 } from "lucide-react";
// Form schema
const formSchema = z.object({
 name: z.string().min(2, "Name must be at least 2 characters"),
 email: z.string().email("Please enter a valid email"),
 subject: z.string().min(5, "Subject must be at least 5 characters"),
 message: z.string().min(10, "Message must be at least 10 characters"),
```

```
});
type FormData = z.infer<typeof formSchema>;
export default function ContactForm() {
 const [isSubmitting, setIsSubmitting] = useState(false);
 const { toast } = useToast();

 // Form definition
 const form = useForm<FormData>({
 resolver: zodResolver(formSchema),
 defaultValues: {
 name: "",
 email: "",
 subject: "",
 message: "",
 },
 });

 const onSubmit = async (data: FormData) => {
 setIsSubmitting(true);

 try {
 // In a real app, you would send this data to your backend
 // For now, we'll simulate a successful API call
 await new Promise(resolve => setTimeout(resolve, 1000));


 // Display success message
```

```
toast({
title: "Message Sent",
description: "Thank you for your message. We'll get back to you shortly.",
});

// Reset form
form.reset();
} catch (error) {
toast({
title: "Error",
description: "Failed to send message. Please try again later.",
```

-

Client File: DatabaseManager (1)/DatabaseManager/client/src/components/help/FAQ.tsx

```
import {
 Accordion,
 AccordionContent,
 AccordionItem,
 AccordionTrigger,
} from "@/components/ui/accordion";
import { Card, CardContent } from "@/components/ui/card";
import { Input } from "@/components/ui/input";
import { Search } from "lucide-react";
import { useState } from "react";
// FAQ items data
const faqItems = [
 {
```

question: "How do I book a helicopter?",

answer: "You can book a helicopter through our website by selecting either a predefined route or creating a

custom route. Simply choose your date, time, and the number of passengers, then proceed to payment. Once your

booking is confirmed, you'll receive all the details via email and WhatsApp."

},

{

question: "What is the difference between predefined and custom routes?",

answer: "Predefined routes are fixed routes between established helipads with set pricing. Custom routes

allow you to specify your own pickup and drop-off locations within Bangalore, with pricing calculated based on

distance, duration, and helicopter type."

},

{

question: "How far in advance should I book?",

answer: "We recommend booking at least 48 hours in advance to ensure availability, especially during

weekends and holidays. However, we do accommodate last-minute bookings subject to availability."

},

{

question: "What happens if there's bad weather on my booking date?",

answer: "Safety is our priority. If weather conditions make flying unsafe, we'll contact you to reschedule

your booking at no additional cost. Alternatively, you can request a full refund if rescheduling doesn't work

for you."

},

{

question: "What is the cancellation policy?",

answer: "You can cancel your booking up to 24 hours before the scheduled time for a full refund.

Cancellations within 24 hours are subject to a 50% cancellation fee. No refunds are provided for no-shows or

cancellations after the scheduled time."

},

{

question: "How many passengers can a helicopter accommodate?",

answer: "Our helicopters can accommodate between 4-7 passengers depending on the model. The exact capacity

will be shown during the booking process based on your selected helicopter type."

},

{

question: "Is there a baggage limit?",

answer: "Yes, there is a weight restriction for safety reasons. Each passenger is allowed up to 5kg of hand

baggage. For specific requirements or additional baggage, please contact our support team in advance."

},

{

question: "Are children allowed on helicopter flights?",

answer: "Yes, children of all ages are allowed, but children under 2 years must sit on an adult's lap.

Children aged 2 and above require their own seat. Please inform us about children during booking for proper

arrangements."

},

{

question: "How do I pay for my booking?",

answer: "We accept payments through credit/debit cards, UPI, and net banking through our

secure Razorpay

payment gateway. All transactions are encrypted for your security."

},

{

-

Server File: DatabaseManager (1)/DatabaseManager/server/db.ts

```
import ws from "ws";
import pg from "pg";
const { Pool } = pg;
import { neonConfig } from '@neondatabase/serverless';
import { drizzle } from 'drizzle-orm/node-postgres';
neonConfig.webSocketConstructor = ws;
if (!process.env.DATABASE_URL) {
 throw new Error(
 "DATABASE_URL must be set. Did you forget to provision a database?",
 );
}
export const pool = new Pool({ connectionString: process.env.DATABASE_URL });
// Import or define your schema here
// import * as schema from './schema'; // <-- comment out or create schema.ts if needed
export const db = drizzle(pool); // <-- pass only pool if not using schema
// Example schema definition; replace with your actual schema
export const exampleTable = {
 id: 'number',
 name: 'string',
};
```

Server File: DatabaseManager (1)/DatabaseManager/server/index.ts

```
import 'dotenv/config';
import express, { type Request, Response, NextFunction } from "express";
import { registerRoutes } from "./routes";
import { setupVite, serveStatic, log } from "./vite";
import path from "path";
const app = express();
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use((req, res, next) => {
 const start = Date.now();
 const path = req.path;
 let capturedJsonResponse: Record<string, any> | undefined = undefined;
 const originalResJson = res.json;
 res.json = function (bodyJson, ...args) {
 capturedJsonResponse = bodyJson;
 return originalResJson.apply(res, [bodyJson, ...args]);
 };
 res.on("finish", () => {
 const duration = Date.now() - start;
 if (path.startsWith("/api")) {
 let logLine = `${req.method} ${path} ${res.statusCode} in ${duration}ms`;
 if (capturedJsonResponse) {
 logLine += ` :: ${JSON.stringify(capturedJsonResponse)}`;
 }
 if (logLine.length > 80) {
 logLine = logLine.slice(0, 79) + "?";
 }
```

```
 log(logLine);
 }
 });
next();
-
Server File: DatabaseManager (1)/DatabaseManager/server/routes.ts
import type { Express, Request, Response } from "express";
import { createServer, type Server } from "http";
import { storage } from "./storage";
import { db } from "./db";
import { users, bookings, helipads, payments, routes } from "@shared/schema";
import { sql, eq, inArray } from "drizzle-orm";
import { z } from "zod";
import {
 userRegistrationSchema,
 userLoginSchema,
 adminLoginSchema,
 predefinedBookingSchema,
 customBookingSchema,
 insertTestimonialSchema
} from "@shared/schema";
import { createHash } from "crypto";
import { generateBookingReference } from "../client/src/lib/utils";
import { requireAuth, requireAdmin } from "./middleware/auth";
import { processPayment } from "./utils/payment";
import { calculateTotal, createPredefinedBooking, createCustomBooking } from
   "./utils/booking";
import bcrypt from 'bcrypt';
```

```
import session from 'express-session';
import MemoryStore from 'memorystore';
import jsPDF from "jspdf";
import autoTable from "jspdf-autotable";
export async function registerRoutes(app: Express): Promise<Server> {
 const httpServer = createServer(app);
 // Setup session middleware
 const MemoryStoreInstance = MemoryStore(session);
 app.use(session({
 secret: process.env.SESSION_SECRET || 'vayu-vihar-secret-key',
 resave: false,
 saveUninitialized: false,
 cookie: {
 secure: process.env.NODE_ENV === 'production',
 maxAge: 24 * 60 * 60 * 1000 // 24 hours
 },
 store: new MemoryStoreInstance({
 checkPeriod: 86400000 // prune expired entries every 24h
 })
 }));
 // Health check endpoint
 app.get('/api/health', (req: Request, res: Response) => {
 res.json({ status: 'ok' });
 });
 // ===== AUTH ROUTES =====

 // Register new user
```

```
app.post('/api/auth/register', async (req: Request, res: Response) => {
try {
const validation = userRegistrationSchema.safeParse(req.body);
if (!validation.success) {
return res.status(400).json({ message: validation.error.errors[0].message });
}

const { email, password, confirmPassword, ...userData } = validation.data;

// Check if user already exists
const existingUser = await storage.getUserByEmail(email);
if (existingUser) {
return res.status(400).json({ message: 'User with this email already exists' });
```

-

Server File: DatabaseManager (1)/DatabaseManager/server/storage.ts

```
import { users, helipads, bookings, helicopters, routes, payments, testimonials, type User,
    type InsertUser,
type Helipad, type InsertHelipad, type Booking, type InsertBooking, type Helicopter, type
    Route, type
InsertRoute, type Payment, type InsertPayment, type Testimonial, type InsertTestimonial }
    from
"@shared/schema";
import { db } from "./db";
import { eq } from "drizzle-orm";
export interface IStorage {
// User methods
getUser(id: number): Promise<User | undefined>;
getUserByEmail(email: string): Promise<User | undefined>;
```

```
  createUser(user: InsertUser): Promise<User>;
  updateUser(id: number, userData: Partial<User>): Promise<User | undefined>;

  // Helipad methods
  getAllHelipads(): Promise<Helipad[]>;
  getHelipad(id: number): Promise<Helipad | undefined>;
  getFeaturedHelipads(limit?: number): Promise<Helipad[]>;
  createHelipad(helipad: InsertHelipad): Promise<Helipad>;
  updateHelipad(id: number, helipad: Partial<Helipad>): Promise<Helipad | undefined>;
  deleteHelipad(id: number): Promise<boolean>;

  // Booking methods
  getAllBookings(): Promise<Booking[]>;
  getBooking(id: number): Promise<Booking | undefined>;
  getBookingByReference(reference: string): Promise<Booking | undefined>;
  getUserBookings(userId: number): Promise<Booking[]>;
  createBooking(booking: InsertBooking): Promise<Booking>;
  updateBookingStatus(id: number, status: string): Promise<Booking | undefined>;

  // Helicopter methods
  getAllHelicopters(): Promise<Helicopter[]>;
  getHelicopter(id: number): Promise<Helicopter | undefined>;

  // Route methods
  getAllRoutes(): Promise<Route[]>;
  getRoute(id: number): Promise<Route | undefined>;
  createRoute(route: InsertRoute): Promise<Route>;
```

```
// Payment methods
createPayment(payment: InsertPayment): Promise<Payment>;
getPaymentByBookingId(bookingId: number): Promise<Payment | undefined>;

// Testimonial methods
getApprovedTestimonials(): Promise<Testimonial[]>;
createTestimonial(testimonial: InsertTestimonial): Promise<Testimonial>;
approveTestimonial(id: number): Promise<Testimonial | undefined>;
}
export class DatabaseStorage implements IStorage {
async getUser(id: number): Promise<User | undefined> {
const [user] = await db.select().from(users).where(eq(users.id, id));
return user || undefined;
}
async getUserByEmail(email: string): Promise<User | undefined> {
const [user] = await db.select().from(users).where(eq(users.email, email));
return user || undefined;
}
async createUser(insertUser: InsertUser): Promise<User> {
const [user] = await db
.insert(users)
.values(insertUser)
.returning();
```

-

Server File: DatabaseManager (1)/DatabaseManager/server/vite.ts

```
import express, { type Express } from "express";
import fs from "fs";
import path from "path";
import { createServer as createViteServer, createLogger } from "vite";
import { type Server } from "http";
import viteConfig from "../vite.config";
import { nanoid } from "nanoid";
const viteLogger = createLogger();
export function log(message: string, source = "express") {
 const formattedTime = new Date().toLocaleTimeString("en-US", {
 hour: "numeric",
 minute: "2-digit",
 second: "2-digit",
 hour12: true,
 });
 console.log(`${formattedTime} [${source}] ${message}`);
}
export async function setupVite(app: Express, server: Server) {
 const serverOptions = {
 middlewareMode: true,
 hmr: { server },
 allowedHosts: true as true,
 };
 const vite = await createViteServer({
 ...viteConfig,
 configFile: false,
 customLogger: {
```

```
...viteLogger,
error: (msg, options) => {
viteLogger.error(msg, options);
process.exit(1);
},
},
server: serverOptions,
appType: "custom",
});
app.use(vite.middlewares);
app.use("*", async (req, res, next) => {
const url = req.originalUrl;
try {
const clientTemplate = path.resolve(
import.meta.dirname,
"..",
"client",
"index.html",
);
// always reload the index.html file from disk incase it changes
let template = await fs.promises.readFile(clientTemplate, "utf-8");
template = template.replace(
`src="/src/main.tsx"`,
`src="/src/main.tsx?v=${nanoid()}"`,
);
const page = await vite.transformIndexHtml(url, template);
res.status(200).set({ "Content-Type": "text/html" }).end(page);
```

```
 } catch (e) {
 vite.ssrFixStacktrace(e as Error);
 next(e);
```

-

Server File: DatabaseManager (1)/DatabaseManager/server/@types/session.d.ts

```
import { User } from "@shared/schema";
declare module "express-session" {
 interface SessionData {
 user?: Omit<User, 'password'>;
 }
}
```

Server File: DatabaseManager (1)/DatabaseManager/server/middleware/auth.ts

```
import { Request, Response, NextFunction } from 'express';
// Middleware to require authentication
export function requireAuth(req: Request, res: Response, next: NextFunction) {
 if (!req.session.user) {
 return res.status(401).json({ message: 'Authentication required' });
 }
 next();
}
// Middleware to require admin role
export function requireAdmin(req: Request, res: Response, next: NextFunction) {
 if (!req.session.user) {
 return res.status(401).json({ message: 'Authentication required' });
 }
```

```
 if (req.session.user.role !== 'admin') {
 return res.status(403).json({ message: 'Admin access required' });
 }

 next();
}
```

Server File: DatabaseManager (1)/DatabaseManager/server/utils/booking.ts

```
import { storage } from '../storage';
import { InsertBooking } from '@shared/schema';
import { generateBookingReference } from '../../client/src/lib/utils';
// Calculate total amount for a booking including GST
export function calculateTotal(baseAmount: number): number {
 // Add 18% GST
 const gst = Math.round(baseAmount * 0.18);
 return baseAmount + gst;
}
// Create a predefined route booking
export async function createPredefinedBooking(userId: number, bookingData: any) {
 const { routeId, bookingDate, bookingTime, passengers } = bookingData;

 // Get the route details
 const route = await storage.getRoute(routeId);
 if (!route) {
 throw new Error('Route not found');
 }

 // For predefined routes, we'll use the route locations directly
```

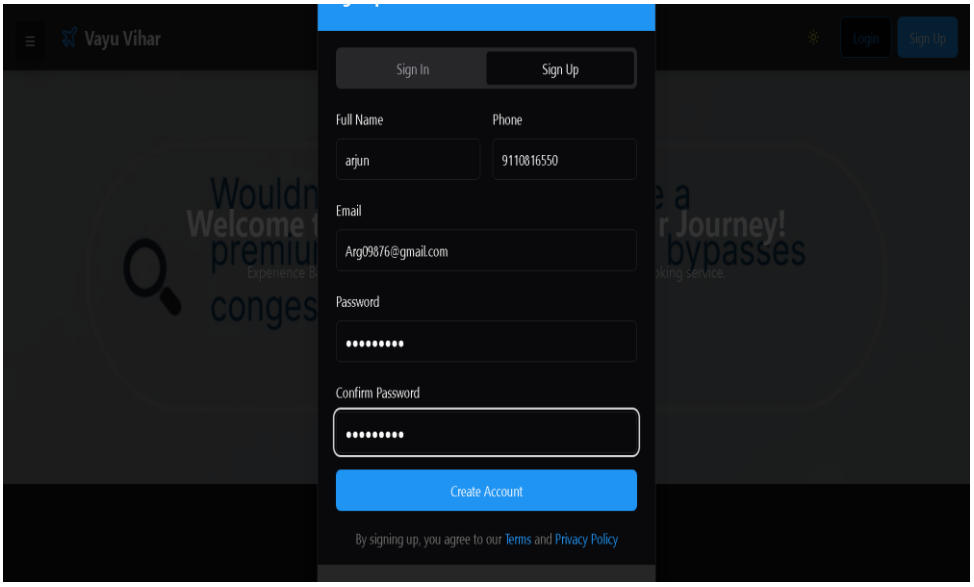
```
// Note: This route might not have specific helipad IDs but has source/destination
   locations

if (!route.sourceLocation || !route.destinationLocation) {

throw new Error('Route locations not properly defined');

}


// Calculate booking date (combine date and time)

let bookingDateTime;

try {

//Handledifferentdate/timeformats

-

Server File: DatabaseManager (1)/DatabaseManager/server/utils/payment.ts

import { storage } from '../storage';

import { InsertPayment } from '@shared/schema';

import { createHash } from 'crypto';

interface PaymentRequest {

bookingId: number;

amount: number;

paymentMethod: string;

}

interface PaymentResponse {

success: boolean;

paymentId?: string;

error?: string;

transactionDetails?: any;

}

// Process payment through Razorpay or other payment gateway

export async function processPayment(paymentRequest: PaymentRequest):
```

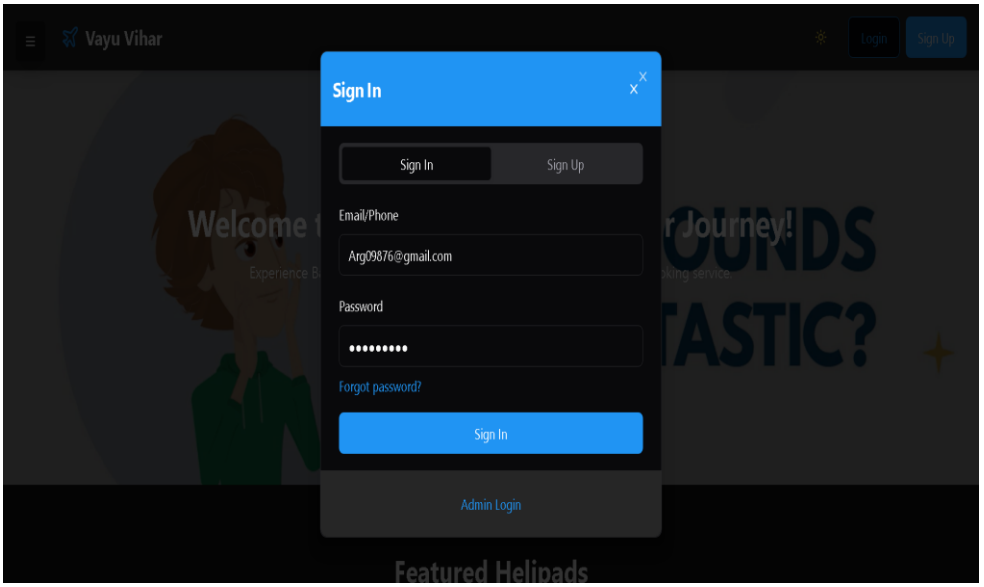
```
Promise<PaymentResponse> {
const { bookingId, amount, paymentMethod } = paymentRequest;

try {
// In a real application, this would integrate with Razorpay API
// For now, we'll simulate a successful payment

// Generate a unique payment reference
const paymentReference =
`PAY-${createHash('md5').update(`${bookingId}-
${Date.now()}`).digest('hex').substring(0, 8)}`;

// Create payment record
const payment: InsertPayment = {
bookingId,
amount,
paymentReference,
paymentMethod,
paymentStatus: 'completed',
};

const savedPayment = await storage.createPayment(payment);

// Update booking payment status
const booking = await storage.getBooking(bookingId);
if (booking) {
await storage.updateBookingStatus(bookingId, 'confirmed');
}
```

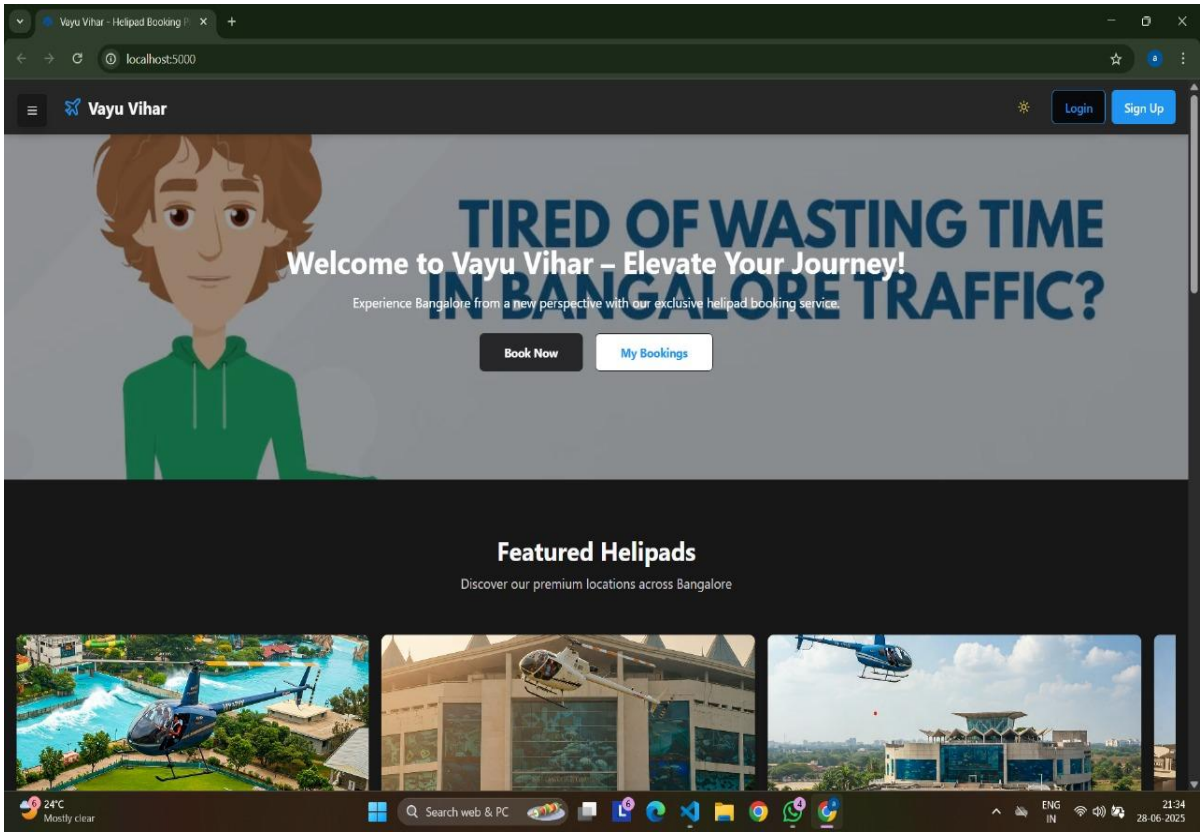
```
return {
success: true,
paymentId: savedPayment.id.toString(),
transactionDetails: {
id: savedPayment.id,
reference: paymentReference,
method: paymentMethod,
amount,
status: 'completed',
timestamp: new Date().toISOString(),
}
};
} catch (error) {
console.error('Payment processing error:', error);
return {
success: false,
error: 'Payment processing failed',
};
}
```

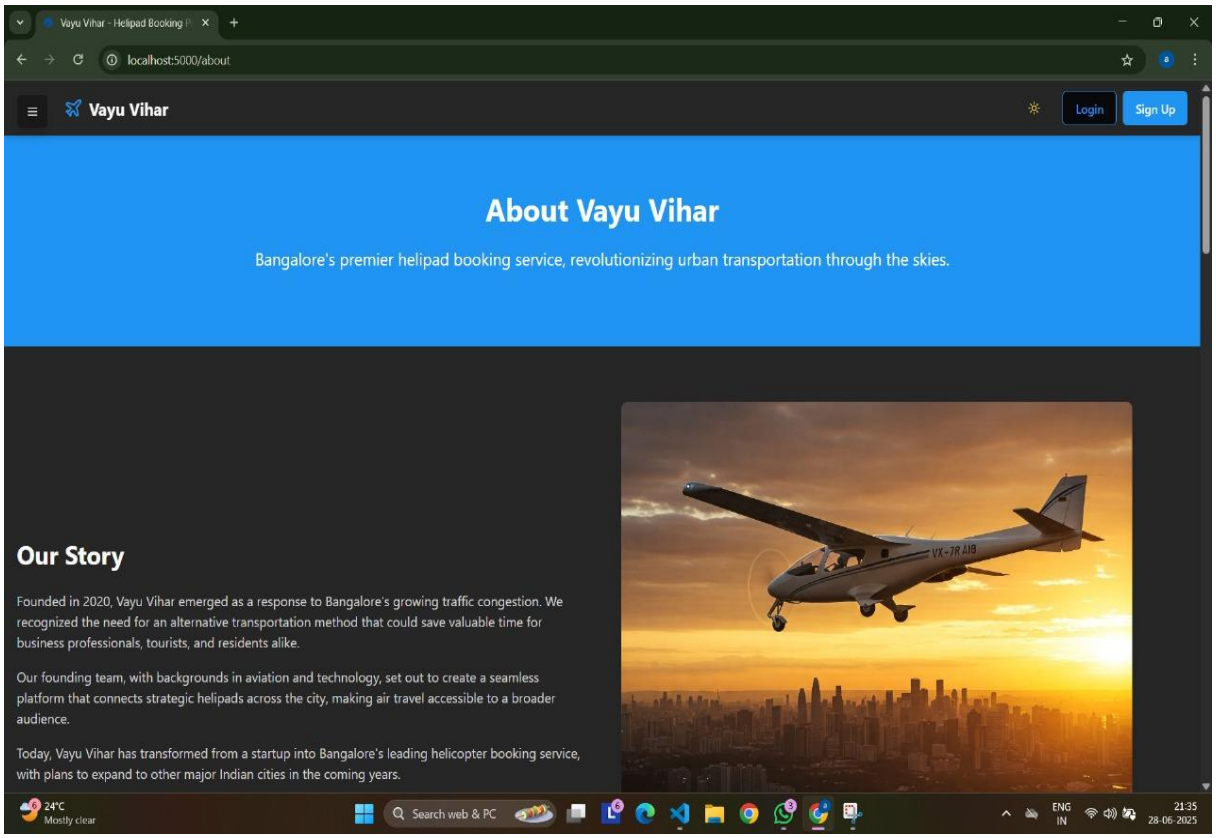
-

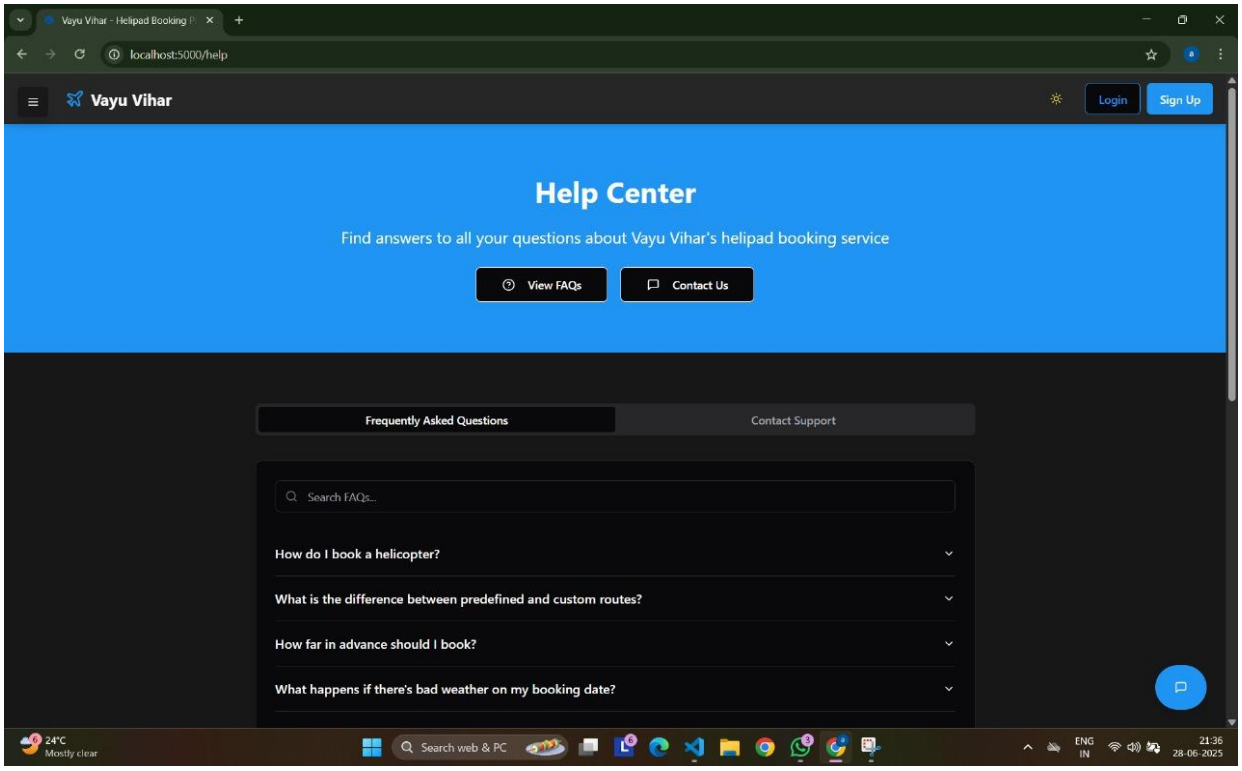
Chapter 09: OUTPUT

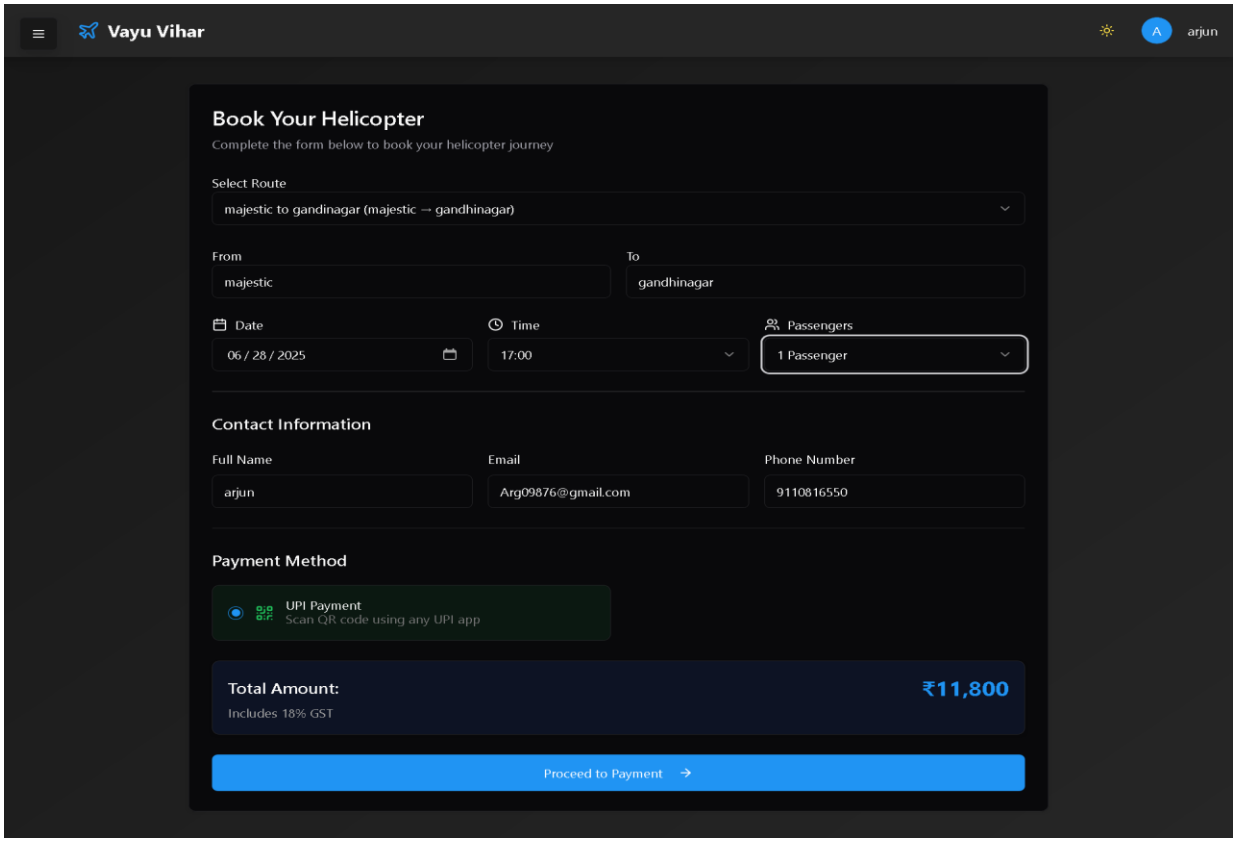


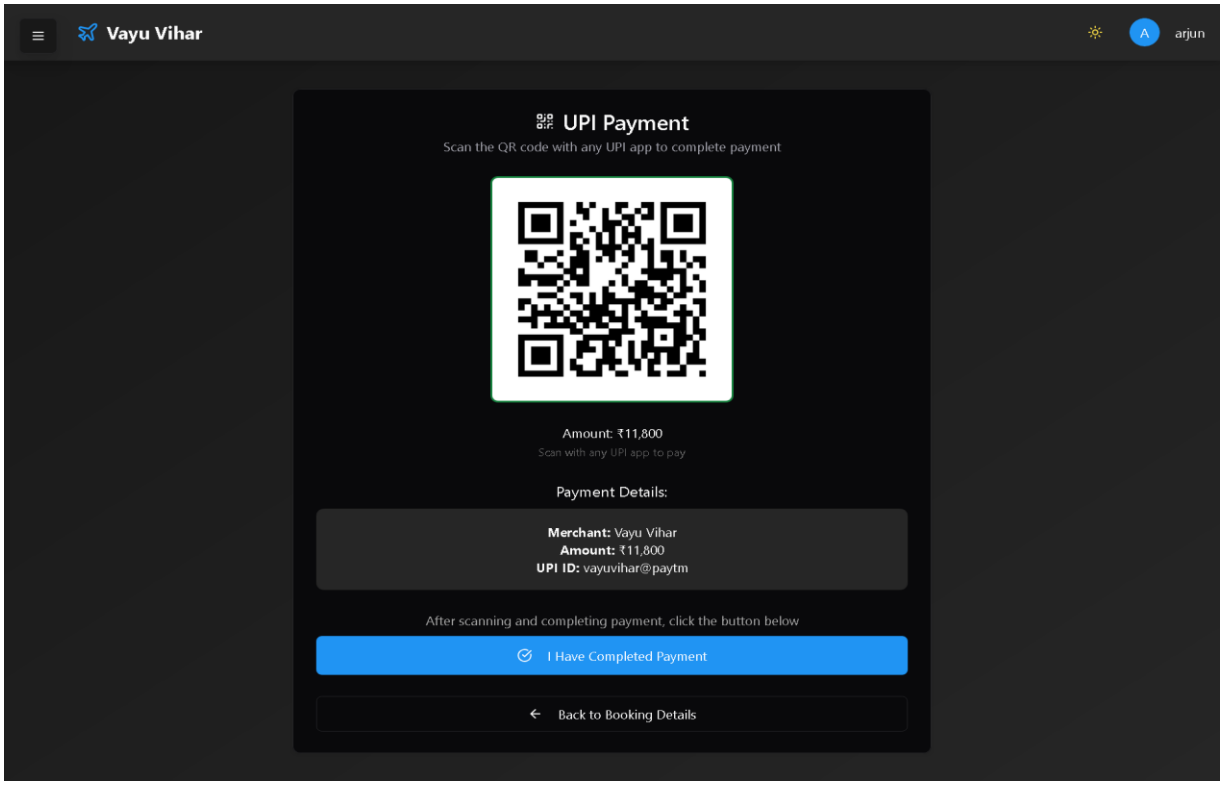


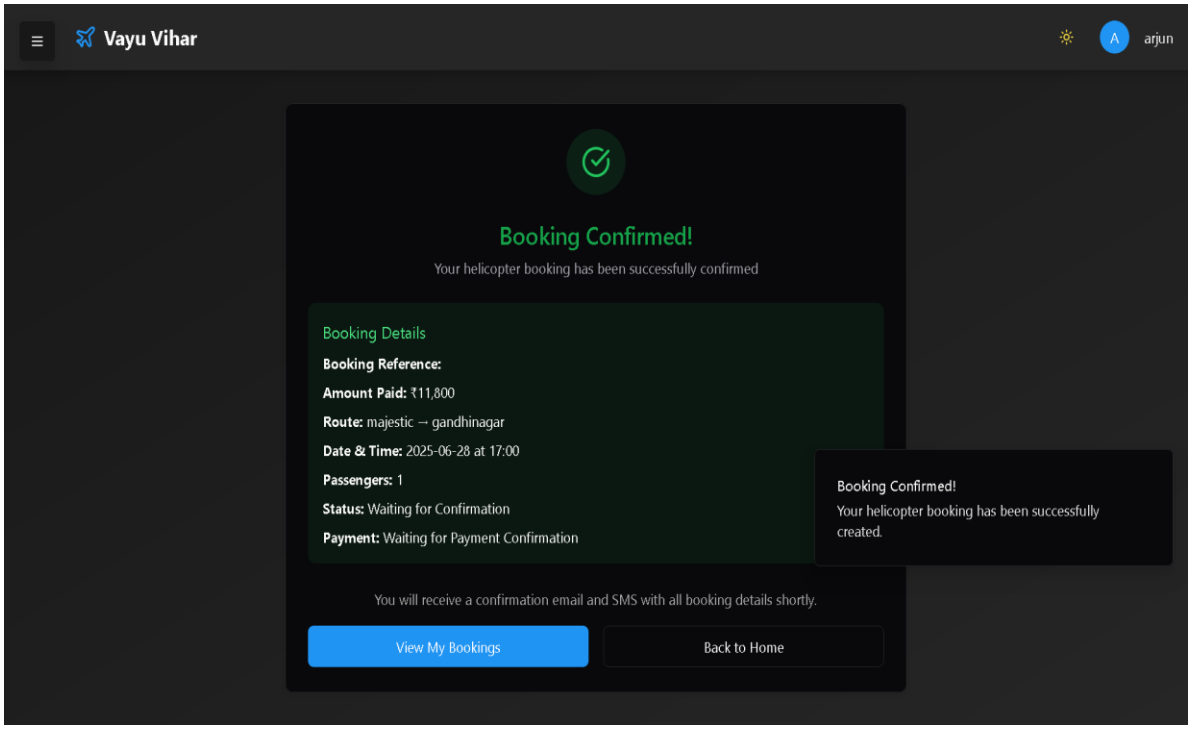

Vayu Vihar - Helipad Booking P
localhost:5000
Vayu Vihar
Login
Sign Up
TIRED OF WASTING TIME IN BANGALORE TRAFFIC?
Welcome to Vayu Vihar – Elevate Your Journey!
Experience Bangalore from a new perspective with our exclusive helipad booking service.
Book Now
My Bookings
Featured Helipads
Discover our premium locations across Bangalore
24°C
Mostly clear
Search web & PC
ENG
IN
21:34
28-06-2025

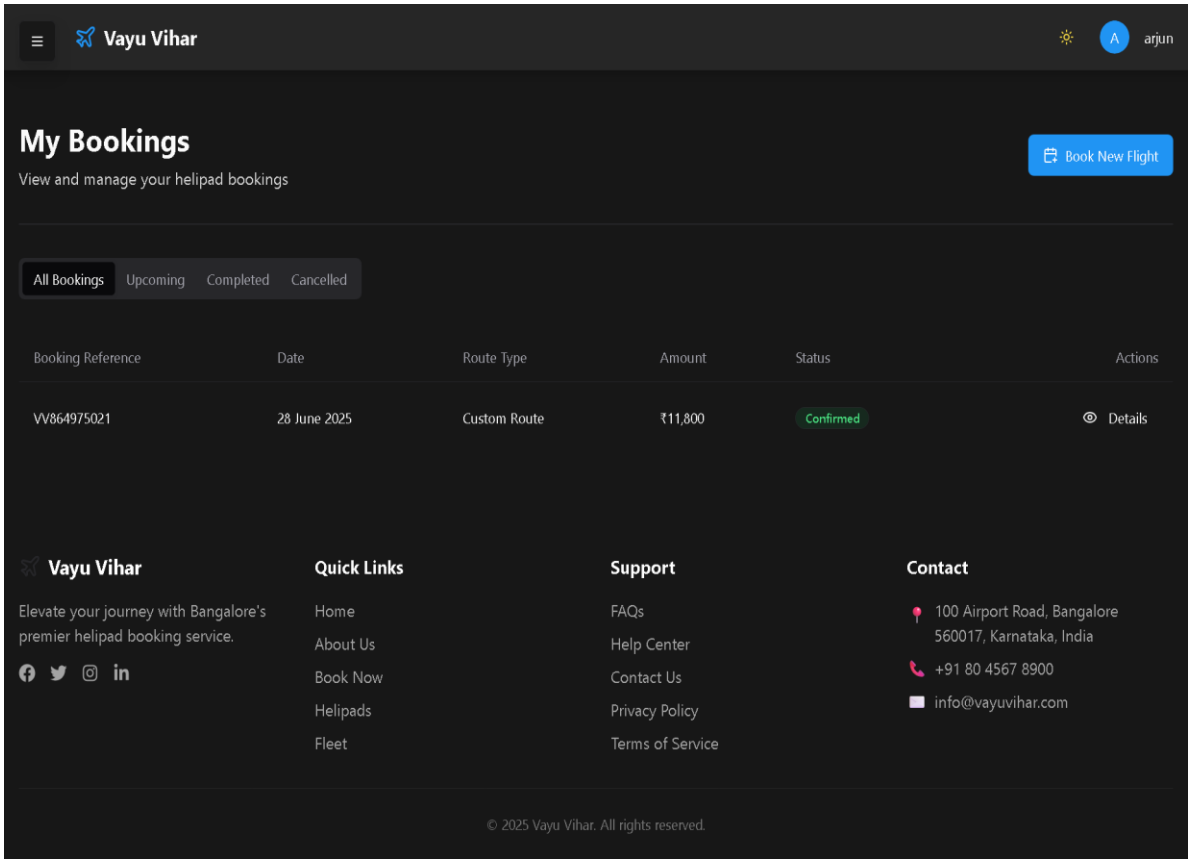

Vayu Vihar - Helipad Booking P
localhost:5000/about
Vayu Vihar
Login
Sign Up
About Vayu Vihar
Bangalore's premier helipad booking service, revolutionizing urban transportation through the skies.
Our Story
Founded in 2020, Vayu Vihar emerged as a response to Bangalore's growing traffic congestion. We recognized the need for an alternative transportation method that could save valuable time for business professionals, tourists, and residents alike.
Our founding team, with backgrounds in aviation and technology, set out to create a seamless platform that connects strategic helipads across the city, making air travel accessible to a broader audience.
Today, Vayu Vihar has transformed from a startup into Bangalore's leading helicopter booking service, with plans to expand to other major Indian cities in the coming years.
24°C
Mostly clear
Search web & PC
ENG
IN
21:35
28-06-2025

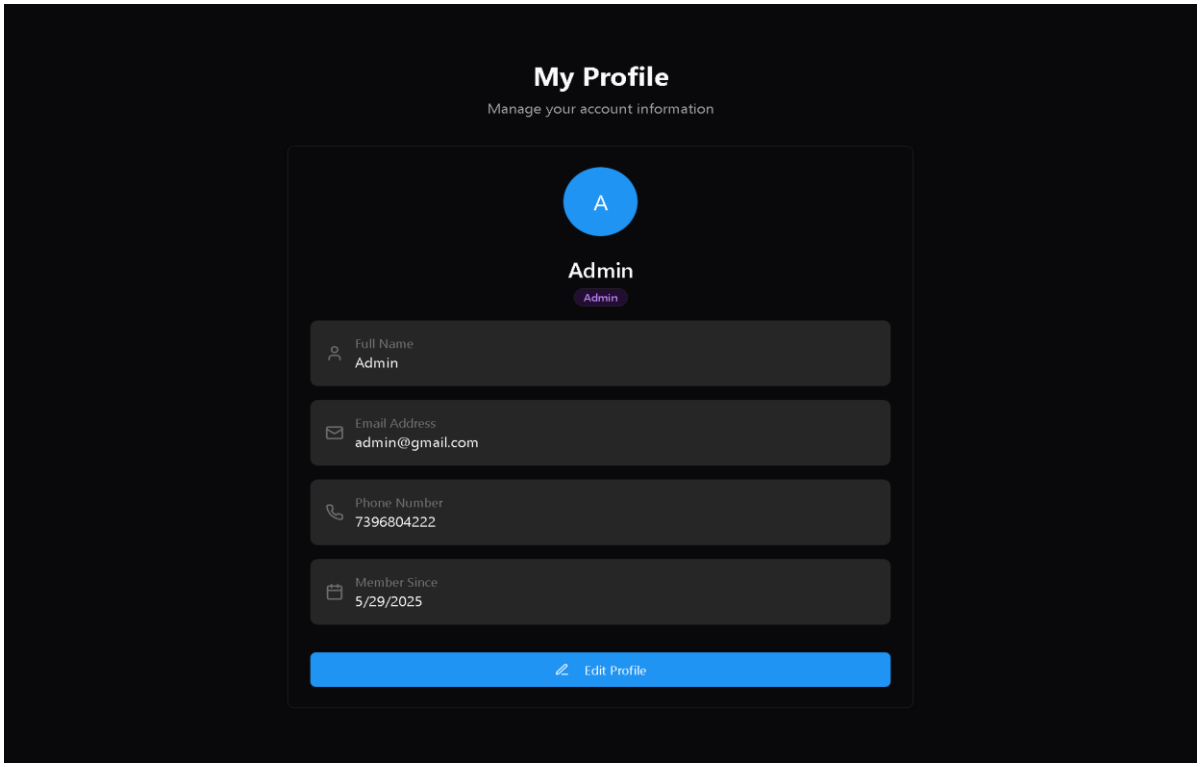

Vayu Vihar - Helipad Booking
localhost:5000/help
Vayu Vihar
Login
Sign Up
Help Center
Find answers to all your questions about Vayu Vihar's helipad booking service
View FAQs
Contact Us
Frequently Asked Questions
Contact Support
Search FAQs...
How do I book a helicopter?
What is the difference between predefined and custom routes?
How far in advance should I book?
What happens if there's bad weather on my booking date?
24°C
Mostly clear
Search web & PC
ENG
IN
21:36
28-06-2025

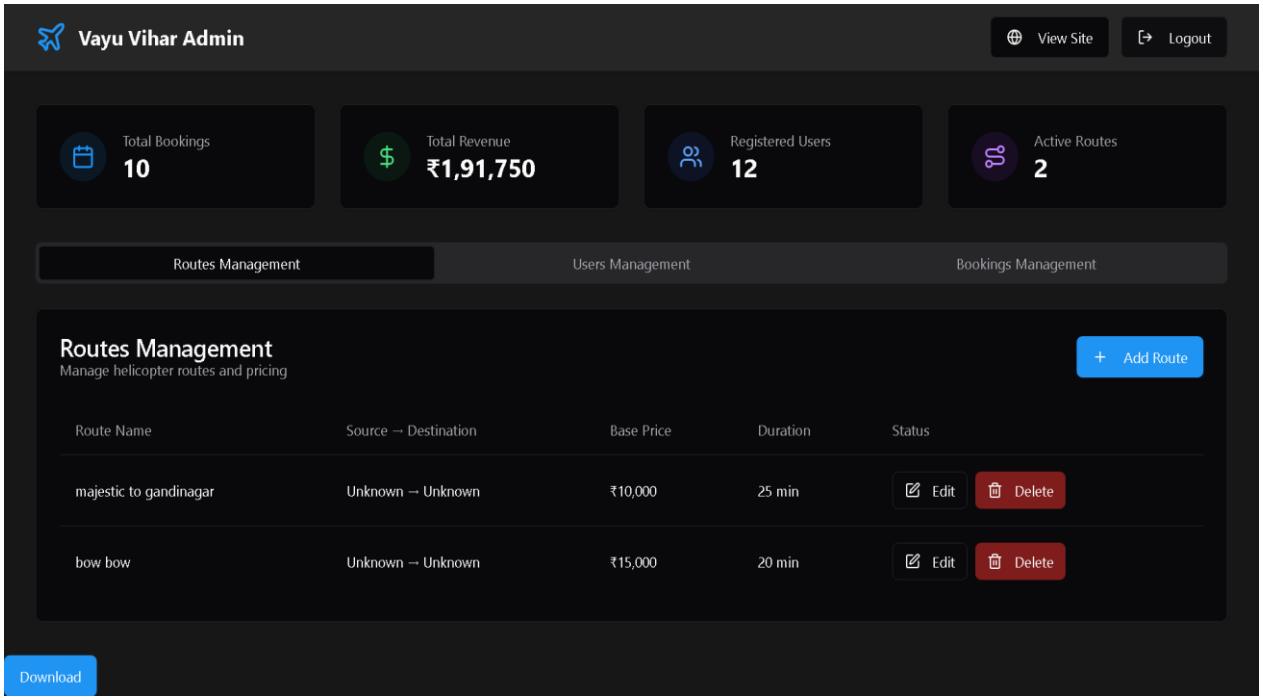

Vayu Vihar
A
arjun
Book Your Helicopter
Complete the form below to book your helicopter journey
Select Route
majestic to gandinagar (majestic → gandhinagar)
From
majestic
To
gandhinagar
Date
06 / 28 / 2025
Time
17:00
Passengers
1 Passenger
Contact Information
Full Name
arjun
Email
Arg09876@gmail.com
Phone Number
9110816550
Payment Method
UPI Payment
Scan QR code using any UPI app
Total Amount:
Includes 18% GST
₹11,800
Proceed to Payment

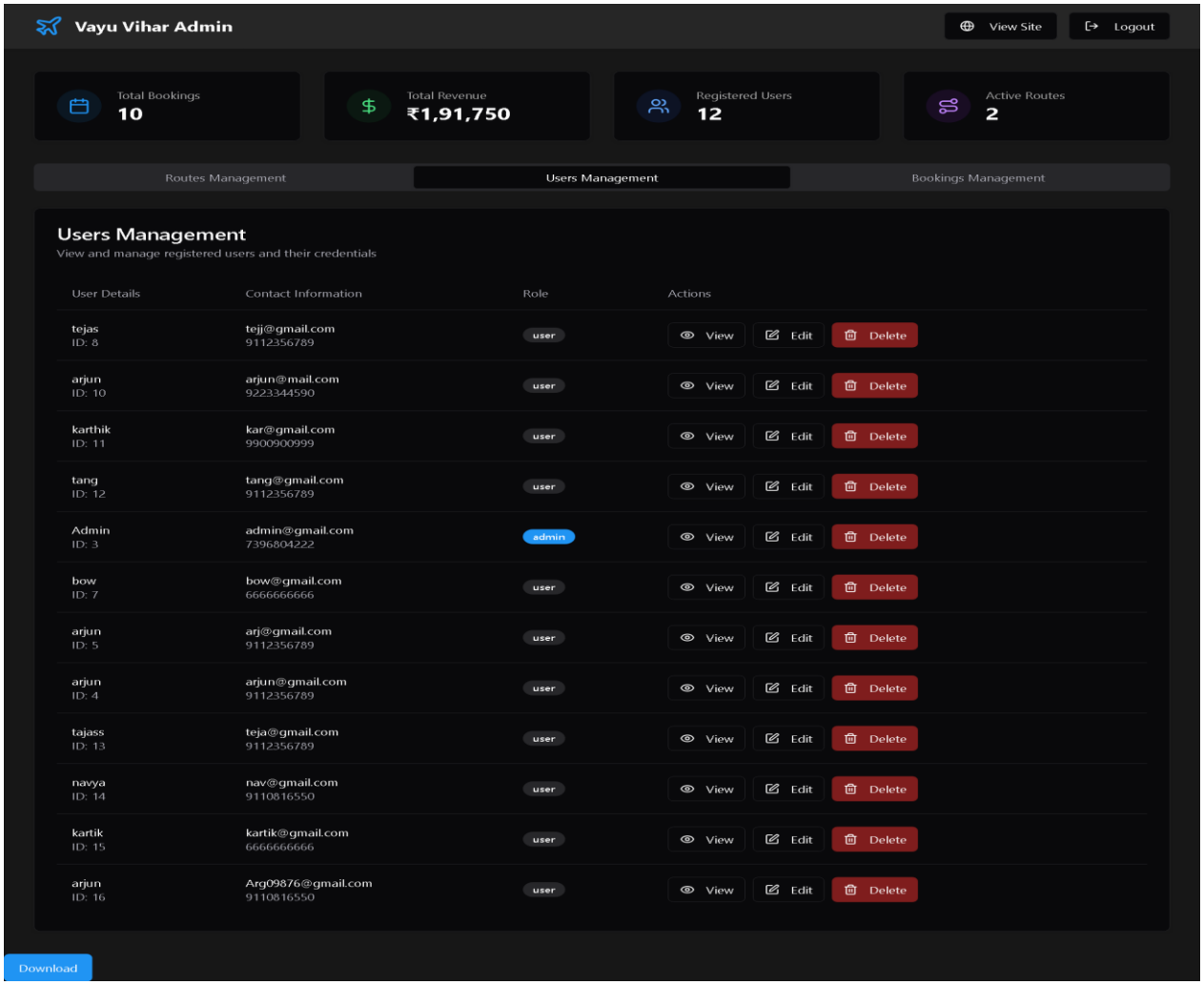

Vayu Vihar
arjun
UPI Payment
Scan the QR code with any UPI app to complete payment
Amount: ₹11,800
Scan with any UPI app to pay
Payment Details:
Merchant: Vayu Vihar
Amount: ₹11,800
UPI ID: vayuvihar@paytm
After scanning and completing payment, click the button below
I Have Completed Payment
Back to Booking Details

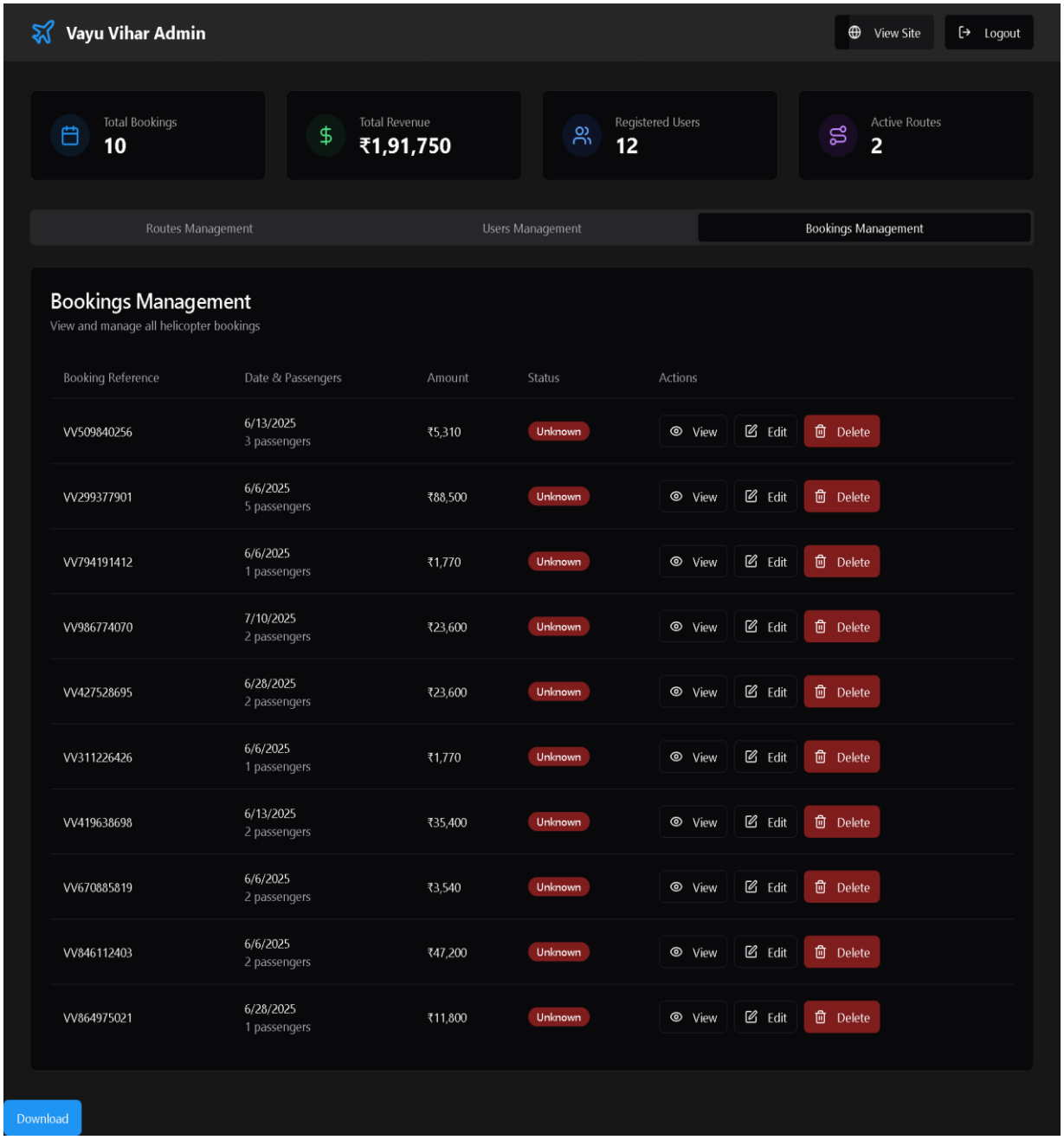

Vayu Vihar
arjun
Booking Confirmed!
Your helicopter booking has been successfully confirmed
Booking Details
Booking Reference:
Amount Paid: ₹11,800
Route: majestic → gandhinagar
Date & Time: 2025-06-28 at 17:00
Passengers: 1
Status: Waiting for Confirmation
Payment: Waiting for Payment Confirmation
Booking Confirmed!
Your helicopter booking has been successfully created.
You will receive a confirmation email and SMS with all booking details shortly.
View My Bookings
Back to Home


Vayu Vihar
A
arjun
My Bookings
View and manage your helipad bookings
Book New Flight
All Bookings
Upcoming
Completed
Cancelled
Booking Reference
Date
Route Type
Amount
Status
Actions
VV864975021
28 June 2025
Custom Route
₹11,800
Confirmed
Details
Vayu Vihar
Elevate your journey with Bangalore's premier helipad booking service.
Quick Links
Home
About Us
Book Now
Helipads
Fleet
Support
FAQs
Help Center
Contact Us
Privacy Policy
Terms of Service
Contact
100 Airport Road, Bangalore 560017, Karnataka, India
+91 80 4567 8900
info@vayuvihar.com
© 2025 Vayu Vihar. All rights reserved.


My Profile
Manage your account information
A
Admin
Admin
Full Name
Admin
Email Address
admin@gmail.com
Phone Number
7396804222
Member Since
5/29/2025
Edit Profile


Vayu Vihar Admin
View Site
Logout
Total Bookings
10
Total Revenue
₹1,91,750
Registered Users
12
Active Routes
2
Routes Management
Users Management
Bookings Management
Routes Management
Manage helicopter routes and pricing
Add Route
Route Name
Source → Destination
Base Price
Duration
Status
majestic to gandinagar
Unknown → Unknown
₹10,000
25 min
Edit
Delete
bow bow
Unknown → Unknown
₹15,000
20 min
Edit
Delete
Download


Vayu Vihar Admin
View Site
Logout
Total Bookings
10
Total Revenue
₹1,91,750
Registered Users
12
Active Routes
2
Routes Management
Users Management
Bookings Management
Users Management
View and manage registered users and their credentials
User Details
Contact Information
Role
Actions
tejas ID: 8 teji@gmail.com 9112356789 user
arjun ID: 10 arjun@mail.com 9223344590 user
karthik ID: 11 kar@gmail.com 9900900999 user
tang ID: 12 tang@gmail.com 9112356789 user
Admin ID: 3 admin@gmail.com 7396804222 admin
bow ID: 7 bow@gmail.com 6666666666 user
arjun ID: 5 arj@gmail.com 9112356789 user
arjun ID: 4 arjun@gmail.com 9112356789 user
tajass ID: 13 teja@gmail.com 9112356789 user
navya ID: 14 nav@gmail.com 9110816550 user
kartik ID: 15 kartik@gmail.com 6666666666 user
arjun ID: 16 Arg09876@gmail.com 9110816550 user
View
Edit
Delete
Download


Vayu Vihar Admin
View Site
Logout
Total Bookings
10
Total Revenue
₹1,91,750
Registered Users
12
Active Routes
2
Routes Management
Users Management
Bookings Management
Bookings Management
View and manage all helicopter bookings
Booking Reference
Date & Passengers
Amount
Status
Actions
VV509840256
6/13/2025
3 passengers
₹5,310
Unknown
View
Edit
Delete
VV299377901
6/6/2025
5 passengers
₹88,500
Unknown
View
Edit
Delete
VV794191412
6/6/2025
1 passengers
₹1,770
Unknown
View
Edit
Delete
VV986774070
7/10/2025
2 passengers
₹23,600
Unknown
View
Edit
Delete
VV427528695
6/28/2025
2 passengers
₹23,600
Unknown
View
Edit
Delete
VV311226426
6/6/2025
1 passengers
₹1,770
Unknown
View
Edit
Delete
VV419638698
6/13/2025
2 passengers
₹35,400
Unknown
View
Edit
Delete
VV670885819
6/6/2025
2 passengers
₹3,540
Unknown
View
Edit
Delete
VV846112403
6/6/2025
2 passengers
₹47,200
Unknown
View
Edit
Delete
VV864975021
6/28/2025
1 passengers
₹11,800
Unknown
View
Edit
Delete
Download

Chapter 10: FUTURE ENHANCEMENT

- Mobile Application Development:

  A dedicated Android and iOS mobile app can be developed to enhance accessibility, provide push notifications, and offer offline features for users in remote areas.

- AI-Powered Dynamic Pricing:

  Integrating AI algorithms to adjust flight prices based on demand, season, and availability could increase revenue and optimize seat utilization.

- Multilingual Support:

  Adding support for regional and international languages will make the platform accessible to a broader user base, improving user inclusivity and satisfaction.

- Integration with GPS and IoT Devices:

  Real-time tracking of helicopters using GPS and IoT sensors can improve transparency, safety, and flight monitoring capabilities for both users and administrators.

- Voice Command Booking:

  Enable voice-based booking using smart assistants like Google Assistant, Alexa, or Siri for quicker, hands-free user interaction.

- Emergency/Medical Booking Module:

  Introduce special emergency booking options for medical evacuations or natural disaster scenarios, allowing for priority-based handling.

- In-App Messaging and Notifications:

  Provide real-time communication between users, pilots, and admins through in-app messaging and instant notifications for updates and changes.

- Blockchain for Transparent Transactions:

  Integrate blockchain technology for secure, traceable, and trans- parent financial transactions, especially useful for high-value corporate or international bookings.

- Loyalty and Reward Programs:

  Introduce frequent flyer programs, discounts, and reward points to improve

customer retention and satisfaction.

- Drone Integration (Future Mobility):

  As the drone travel industry evolves, the system can be extended to include drone ride bookings for short-distance or lastmile connectivity.

- Third-Party Integrations:

  Integration with travel agencies, tourism portals, or corporate ERPs for B2B use cases and bulk or scheduled bookings.

- Advanced Reporting Dashboard:

  Add graphical analytics and real-time dashboards for administrators to monitor operations, revenue, and user activity efficiently.

- AR/VR Integration:

  Augmented and virtual reality features for virtual tours of helipads, routes, or helicopters to help users choose flights more confidently.

- Carbon Footprint Tracking:

  Display environmental impact for each flight and offer options for carbon offset contributions during checkout to support sustainability.

Chapter 11: CONCLUSION

Vayu Vihara successfully addresses the limitations of traditional helicopter booking systems by providing a digital solution that is modern, secure, and scalable. It empowers users with the ability to book helicopter rides in just a few clicks, view route maps, and manage their accounts and payments efficiently. From its robust back infrastructure to its elegant and intuitive user interface, every aspect of the application has been designed with both performance and user experience in mind.

The implementation of real-time scheduling, encrypted payment processing, and mobile responsiveness makes the platform suitable for real-world deployment. As travel needs continue to evolve, especially in areas with limited ground transportation, Vayu Vihara stands out as a smart, forward-looking solution. It paves the way for the digital trans- formation of aviation services, ensuring accessibility, transparency, and convenience for all users involved.

Chapter 12: BIBLIOGRAPHY

REFERENCES

- Online Resources

1. Stack Overflow
   - **https://stackoverflow.com/questions**(For technical Q&A and discussions)
2. GitHub:
   - **https://github.com/** (For open-source projects and code repositories
3. Towards Data Science:
   - **https://web.gisma.com/** (Articles on AI, machine learning, and data science)

- Industry Insights

1. Forbes:
   - **https://www.forbes.com/** (For articles on various topics, including AI in retail and e-commerce)

- Sustainability and Ethics

1. Sustainable Brands:
   - **https://sustainablebrands.com/** (Insights on sustainable business practices)
2. Fair Trade Certified:
   - **https://www.fairtradecertified.org/** (Information on fair trade practices)