ATMIYA UNIVERSITY

RAJKOT

A

Report On

AI-Base Travel Itinerary Planner

Under subject of

MINI PROJECT

B.TECH, Semester – VII

(Computer Engineering)

Submitted by:

Shingarakhiya Kishan Maldebhai (220002070)

Prof. Ashwini Vaidya

(Faculty Guide)

Prof. Tosal Bhalodia

(Head of the Department)

Academic Year

(2025-26)

CANDIDATE'S DECLARATION

We hereby declare that the work presented in this project entitled "**AI-Base Travel Itinerary Planner"** submitted towards completion of project in **7th Semester** of B. Tech. (Computer Engineering) is an authentic record of our original work carried out under the guidance of "**Ashvini Vaidya".**

I further declare that this project has not been submitted by me for the award of any other degree, diploma, or certificate in any other institution. The results, features, and implementation described in this project are based on my own effort and research.

Semester: 7th
Place: Rajkot

Signature:
Kishan Shingarakhiya (220002070)

ATMIYA UNIVERSITY

RAJKOT

CERTIFICATE

Date:

This is to certify that the "**AI-Base Travel Itinerary Planner**" has been carried out by Kishan Shingarakhiya under my guidance in fulfillment of the subject Mini Project in COMPUTER ENGINEERING (7th Semester) of Atmiya University, Rajkot during the academic year 2025-26.

Prof. Ashwini Vaidya
(Project Guide)

Prof. Tosal M. Bhalodia
(Head of the Department)

ACKNOWLEDGEMENT

I am grateful to **Atmiya University**, **Department of Computer Science**, for providing me with the opportunity to work on my project titled **"AI-Driven Travel Itinerary System"**. This project has helped me gain practical knowledge and improve my technical skills in web and mobile application development.

I would like to express my sincere thanks to my guide, **Prof. Ashvini Vaidya**, for their constant guidance, valuable suggestions, and continuous support throughout the project. I am also thankful to our respected Head of Department, **Prof. Tosal M. Bhalodia**, for providing the necessary resources and encouragement to complete this work successfully.

This project focuses on developing a **Travel Itinerary System** with two parts: a **web admin panel** to manage places, hotels, virtual views, and history, and a **mobile app** for users to explore history, hotels, virtual tours, search routes, plan trips, and interact with an AI-powered chatbot. I am also thankful to my friends and family for their support during this journey.

Kishan Shingarakhiya (220002070)

ABSTRACT

The **AI-Driven Travel Itinerary System** is designed to simplify and enhance the process of planning trips by providing users with personalized travel assistance. The system consists of two main components: a **web-based admin panel** and a **mobile application**. The admin panel enables administrators to **add, edit, and update** information about tourist places, hotels, virtual views, and historical details, ensuring that users always have access to accurate and up-to-date data.

The mobile application is designed for travelers, allowing them to **explore historical information, view virtual tours, check hotel details, search routes between two locations, calculate distances, and plan their trips effectively**. Additionally, the app integrates an **AI-powered chatbot** to assist users with travel-related queries and provide personalized recommendations, enhancing the overall user experience.

This project aims to deliver an efficient, user-friendly, and intelligent platform that connects travelers with essential information and smart planning tools. By integrating modern technologies, the system ensures better decision-making, optimized travel plans, and a seamless experience for both administrators and users.

INDEX

LIST OF FIGURES

LIST OF TABLES

CHAPTER – 1

INTRODUCTION

1.1. Purpose

- The purpose of the project entitled "AI-Driven Travel Itinerary System" is to create a smart, secure, and efficient platform for managing and planning trips. The system is divided into two modules:
  - A web-based admin panel for administrators to manage tourist-related data
  - A mobile application for travelers to explore and plan their trips

- The admin panel allows administrators to add, edit, and update details about tourist places, hotels, virtual views, and historical information. On the other hand, the mobile application provides users with features like viewing history, virtual tours, and hotel details, searching routes between two places, calculating distances, planning trips, and accessing an AI-powered chatbot for travel assistance.

- The system also implements a secure OTP-based login in the mobile app to ensure user authentication and data privacy, making the overall platform safe and reliable.

1.2 Scope

- The AI-Driven Travel Itinerary System is designed to simplify travel planning for users and streamline data management for administrators. The scope of the system is divided into two parts.

- **Admin Panel (Web Application)**
  - Accessible only by administrators
  - Add, edit, and manage tourist places
  - Maintain and update hotel details
  - Upload and manage virtual views
  - Store and update historical information of tourist places

- **User App (Mobile Application)**
  - OTP-based secure login for travelers
  - Explore history, hotels, and virtual views of tourist destinations
  - Search routes and distances between two locations

- Plan personalized tours effectively
- Get travel assistance via an AI-powered chatbot

- This system eliminates the hassle of manually searching for travel information on different platforms. By combining AI-driven planning, virtual views, secure authentication, and optimized data management, the project delivers a comprehensive and user-friendly solution for travelers and administrators.

1.3 Technology and Tools

- **Languages**
  - JavaScript – For developing the web admin panel (React.js) and mobile app frontend (React Native)
  - Python – For backend server-side scripting and API development

- **Frameworks & Platform**
  - React.js → Web-based admin panel for managing tourist places, hotels, historical data, etc.
  - React Native → Cross-platform mobile app for Android & iOS
  - Flask → Python-based lightweight framework for creating RESTful APIs
  - Firebase → For authentication, real-time database, and media storage
  - Android & iOS → Target mobile platforms

- **Backend APIs & Services**
  - OpenRouteService API
    → Route optimization
    → Travel distance & time calculation
    → Generating route geometry for map visualization
  - Twilio API
    → For OTP-based authentication via SMS
  - Leaflet.js
    → For interactive map rendering inside the React Native WebView
  - Firebase Realtime Database
    → Tourist places
    → Hotels
    → History paragraphs & images

→ Virtual tours

- **Features Implemented in Backend (Flask)**
  - OTP Authentication
    → Secure login using Twilio for SMS-based verification
  - Route Optimization API
    → Accepts a list of tourist locations
    → Uses OpenRouteService + Haversine formula (fallback)
    → Calculates optimized travel routes
    → Generates route coordinates for Leaflet map display
  - Travel Time & Distance Calculation
    → Based on real-time routing data
  - Media Serving
    → Exposes /media/<filename> endpoint for accessing uploaded images & videos

- **Tools & Libraries**
  - React Native WebView → For embedding the interactive Leaflet map
  - React Navigation → For smooth screen transitions in the mobile app
  - Axios / Fetch API → For API communication between frontend and backend
  - OpenRouteService Python Client → For interacting with the route optimization API
  - Twilio Python SDK → For sending OTPs to users
  - Math & Haversine → For fallback distance calculations when OpenRouteService fails
- **Editor & Development Environment**
  - Visual Studio Code → For frontend & backend development
  - Android Studio → For running & testing the mobile app
  - Postman → For API testing and debugging
  - Firebase Console → For managing authentication, database, and storage
  - Python Environment (Flask) → For running the backend server locally
  - Android Emulator / Physical Device → For testing the mobile app

CHAPTER – 2

PROJECT MANAGEMENT

2.1 Project Planning

- Project planning is the second stage of the software project management life cycle, following initiation. For my project "AI-Driven Travel Itinerary Planner", I carefully planned each development phase since I am the sole developer.
- The project consists of two main modules:
  1. **Web Admin Panel (React.js)** → For adding, editing, and managing tourist places, hotels, historical data, and 360° virtual views.
  2. **Mobile Application (React Native)** → For users to explore destinations, view history, hotels, virtual tours, optimized travel paths, and interact with an AI-powered travel assistant.
- During planning, I finalized the following technology stack:
  - **Frontend**: React.js (web), React Native (mobile app)
  - **Backend**: Python Flask for API development and server-side scripting
  - **Database & Storage**: Firebase Realtime Database for authentication, data storage, and image management
  - **APIs & Integrations**:
    → OpenRouteService API → For distance calculations and optimized travel routes
    → Twilio API → For OTP-based login authentication
    → OpenAI ChatGPT API → For AI-powered itinerary suggestions, travel advice, and chatbot integration
  - **Testing & Deployment**: Android Emulator, physical Android devices, Postman API testing, Firebase hosting

2.2 Project Scheduling

- Since I developed the entire system alone, I divided the project into phases to manage time and ensure proper integration between the web app, mobile app, and backend APIs.
- My Project Timeline & Milestones:
  - **Week 1-2** → Requirement gathering, project planning, and finalizing architecture
  - **Week 3-4** → Flask backend development: authentication, OTP verification, API endpoints
  - **Week 5-6** → Building React.js web admin panel for managing tourist places, hotels, and 360° virtual views

- **Week 7-8** → Developing React Native **mobile app** UI and integrating Firebase authentication
- **Week 9** → Integrating OpenRouteService API for optimized travel paths and distance calculations
- **Week 10** → Implementing Twilio OTP login and user authentication flow
- **Week 11** → Integrating OpenAI ChatGPT API for AI-powered chatbot and itinerary planning
- **Week 12** → Testing app features, APIs, and admin panel on multiple emulators and real devices
- **Week 13** → Final deployment and preparation of project documentation

2.3 Risk Management

- Risk management consists of a series of steps that help a software development team to understood and manage uncertain problems that may arise during software development and can plague a software project.

2.3.1 Risk Identification :

- Technology Risks :

- ChatGPT API downtime → May cause delays in generating AI-based travel suggestions.
- OpenRouteService API quota limitations → Limited free usage may affect distance and path calculations.
- Firebase data synchronization issues → Possible delays in updating real-time data for places, hotels, and virtual views.
- Flask server overload → Multiple API requests could cause the server to crash or slow down.

- Integration Risks :

- React Native & Firebase compatibility issues → Problems may occur during mobile app authentication and data fetching.
- AI response delays → ChatGPT API responses may sometimes take longer, affecting chatbot performance.
- OTP delivery failures → Twilio OTP authentication may fail due to network issues or service downtime.

- **Personal Risks :**
  - Managing frontend, backend, database, AI integration, and testing alone increases the workload.
  - Handling multiple APIs simultaneously could lead to errors and delays.
  - Time management challenges due to solo development.

2.3.2 Risk Analysis :

"Risk analysis = risk assessment + risk management + risk communication."

Risk assessment :

- Involves identifying sources of potential harm, assessing the likelihood that harm will occur, and the consequences if harm does occur.
- For this project, the possible risks include:
  - System Crash due to server overload.
  - API Downtime for ChatGPT, OpenRouteService, or Firebase.
  - OTP Delivery Failure using Twilio service.
  - AI Response Delays due to high traffic or network issues.

Risk management :

- Evaluates which risks identified in the risk assessment process require management and selects and implements the actions needed to control those risks.
- Precautions taken to minimize risks are as follows:
  - Periodic database backups to avoid data loss.
  - Implemented error handling for server and API failures.
  - Added retry mechanisms for API calls to handle temporary downtime.
  - Used local caching for frequently requested AI responses.
  - Tested OTP functionality with both Twilio live numbers and demo numbers to ensure reliability.

Risk communication :

- Involves a systematic process of identifying risks and ensuring they are properly addressed during development. Since this project is developed individually, communication is primarily between the developer and client/stakeholder.
- Steps taken for risk communication:
  - All possible risks are analyzed before development.

- The probability of certain risks, such as API downtime or OTP delivery issues, is documented.
- The project is designed while keeping these risks in mind to ensure smooth user experience and reliable performance.

CHAPTER – 3

SYSTEM REQUIREMENTS STUDY

3.1 Hardware and Software Requirement

- This shows minimum requirements to carry on to run this system efficiently.

3.1.1 Server-side Hardware Requirement :

Devices	Description
Processor	Intel Core i3 or More
RAM	4GB or More
Hard Disk	10 GB or More

Table 3.1.1 Server-side Hardware Requirement

3.1.2 Software Requirements :

For Which	Software*
Operating System	Windows 7/8/10/11, Linux
Frontend (web)	React.js, HTML5, CSS3, JavaScript
Frontend (Mobile)	React Native (Android & ISO)
Back End	Flask (Python) REST API
Database	Firebase Realtime DB
AI Integration	ChatGPT API (OpenAI)
Maps & Routing	OpenRouteService API
OTP Service	Twilio API
IDE / Editor	Visual Studio Code
Testing Tools	Postman, Android Emulator

Table 3.1.2 Software Requirement

3.1.3 Client-side Requirements :

For Which	Requirement*
Android	Greater Then 4.1
Desktop	Any Version of Chrome

Table 3.1.3 Client-side Requirement

3.2 Constraints

3.2.1 Hardware Limitations :

- The major hardware limitations faced by the system are:
  - If appropriate hardware like processor, RAM, and storage is not available, server performance will degrade.
  - Limited storage may cause database issues since the project involves storing places data, images, user profiles, and AI responses.
  - Without a stable internet connection, ChatGPT API, OpenRouteService, and Twilio OTP will fail.

3.2.2 Reliability Requirements :

- Since multiple users can access the server simultaneously, the backend should handle high traffic and API requests efficiently. To ensure reliability:
  - High-configuration servers are required for better performance.
  - Regular database backups must be taken to prevent data loss.
  - The system must handle huge data, including routes, locations, user profiles, and chat responses.
  - Proper input validation is implemented to prevent wrong or unauthorized data entry.
  - If incorrect information is entered, users are alerted with error messages and prompted to correct it before saving.

3.2.3 Safety and Security Consideration :

- User Data Privacy – Encrypt personal, location, and travel data.
- Authentication – Secure login (Firebase Auth, OTP, Google sign-in).
- Secure APIs – Use HTTPS, JWT tokens, and secure API keys.
- Data Integrity – Regular backups, prevent unauthorized changes.
- Attack Prevention – Protect from SQL injection, XSS, brute-force.
- Route Safety – Avoid unsafe/restricted areas, provide real-time alerts.
- Offline Support – Cached itineraries, emergency sharing option.
- Compliance – Follow GDPR/IT laws and ethical AI use.

CHAPTER – 4

SYSTEM ANALYSIS

4.1 Study Current System

- The implementation phase involves turning the theoretical design of the AI-driven personalized travel itinerary planner into a fully functional system. The goal is to provide a smart travel assistant that uses AI and user preferences to create optimized travel plans.
- Currently, most users depend on manual travel planning or generic travel websites that:
  - Don't provide personalized itineraries.
  - Lack AI-based recommendations.
  - Fail to optimize travel time between destinations.
- Our proposed system integrates AI, geolocation, and real-time travel data to create personalized, time-efficient itineraries based on user preferences and travel constraints.

4.2 Problem and weakness of current system

- Users manually plan trips without AI assistance
- No optimized travel route planning
- Existing apps fail to consider travel time + buffer time
- Lack of integration between tourist spots, user preferences, and timings
- No personalized suggestions for food, activities, or hidden spots
- No real-time updates for traffic, weather, and location changes

4.3 Requirements of New System

4.3.1 User Requirements :

- The system should:
  - Provide AI-powered personalized itineraries.
  - Suggest tourist places, restaurants, and activities.
  - Show optimized routes using Dijkstra's Algorithm and OpenRouteService.
  - Be fast, mobile-friendly, and easy to use.
  - Allow users to save, edit, and share itineraries.

4.3.2 System Requirements :

- **Functional System Requirement :**
  - AI-based itinerary planning using ChatGPT API.

- Optimized route calculation via Dijkstra's algorithm.
- Real-time traffic and weather updates.
- Firebase integration for storing:
  → User data
  → Saved itineraries
  → Travel history
- Map visualization using Leaflet in React Native WebView.
- Search & filter tourist spots.
- Admin panel for managing tourist places.

- Non-Functional System Requirements :

i. EFFICIENCY REQUIREMENT :

- The AI response time should be under 3 seconds.
- Route optimization should handle up to 50 destinations seamlessly.

ii. RELIABILITY REQUIREMENT :

- The system should work online and offline (cached data).
- Firebase ensures real-time synchronization between users and devices.

iii. USABILITY REQUIREMENT :

- Simple React Native UI for mobile users.
- Clean admin dashboard for managing data.

iv. IMPLEMENTATION REQUIREMENT :

- Frontend: React Native
- Backend: Flask (Python)
- Database: Firebase
- AI Integration: OpenAI ChatGPT API
- Routing Engine: OpenRouteService + Dijkstra's Algorithm
- Map Visualization: Leaflet JS in WebView

v. DELIVERY REQUIREMENT :

- The system will be delivered within 3 months.
- Weekly evaluations will be conducted by the project guide.

4.4 Feasibility Study

- The feasibility study determines whether the proposed AI-driven travel itinerary system is practical and sustainable.

Technical Feasibility :

- Frontend: React Native ensures cross-platform compatibility.
- Backend: Flask handles API integration, AI processing, and routing.
- Firebase: Provides secure authentication and real-time database.
- AI: ChatGPT API delivers personalized recommendations.
- Mapping: OpenRouteService + Leaflet ensure real-time route visualization.

4.5 Feature Of New System

1. **AI-Powered Personalized Itinerary Generation**
   - → Generates custom travel plans based on user preferences such as budget, available time, interests, and preferred destinations.
   - → Uses machine learning and intelligent algorithms to recommend optimal schedules.
2. **Smart Route Optimization**
   - → Uses Dijkstra's algorithm and OpenRouteService API to calculate shortest and fastest travel paths.
   - → Minimizes total travel time by considering distance, traffic conditions, and buffer times.
3. **Interactive Map Integration**
   - → Integrated with Google Maps and OpenRouteService to visualize travel routes.
   - → Allows users to view locations, nearby attractions, and route suggestions on an interactive map.
4. **Real-Time Weather & Travel Insights**
   - → **Displays live weather forecasts for selected destinations.**
   - → **Suggests best visiting times and adjusts the itinerary accordingly.**
5. **User-Friendly Interface**
   - → Built using React Native for mobile and React.js for web, ensuring a seamless and responsive experience.
   - → Allows users to add, edit, and customize destinations, travel timings, and activities easily.
6. **Offline Access & Data Caching *(Future Scope)***
   - → Saves generated itineraries for offline access during travel.
   - → Uses local storage or SQLite database for offline data handling.

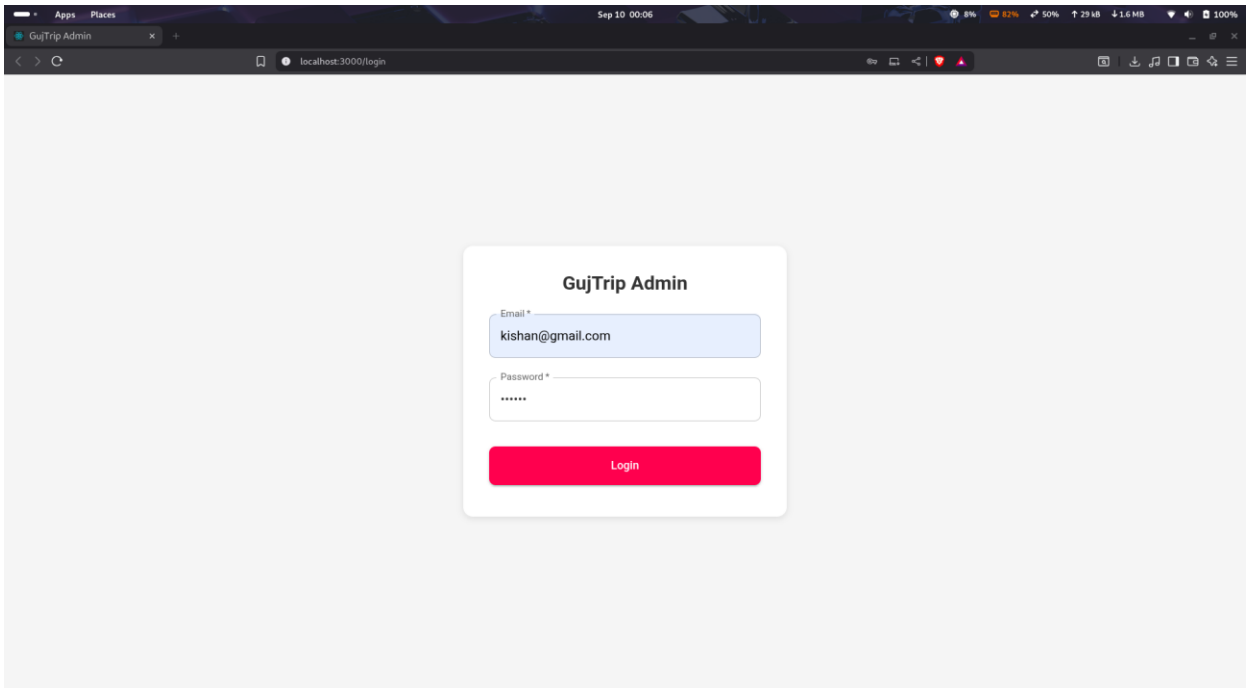
CHAPTER – 5

System Design

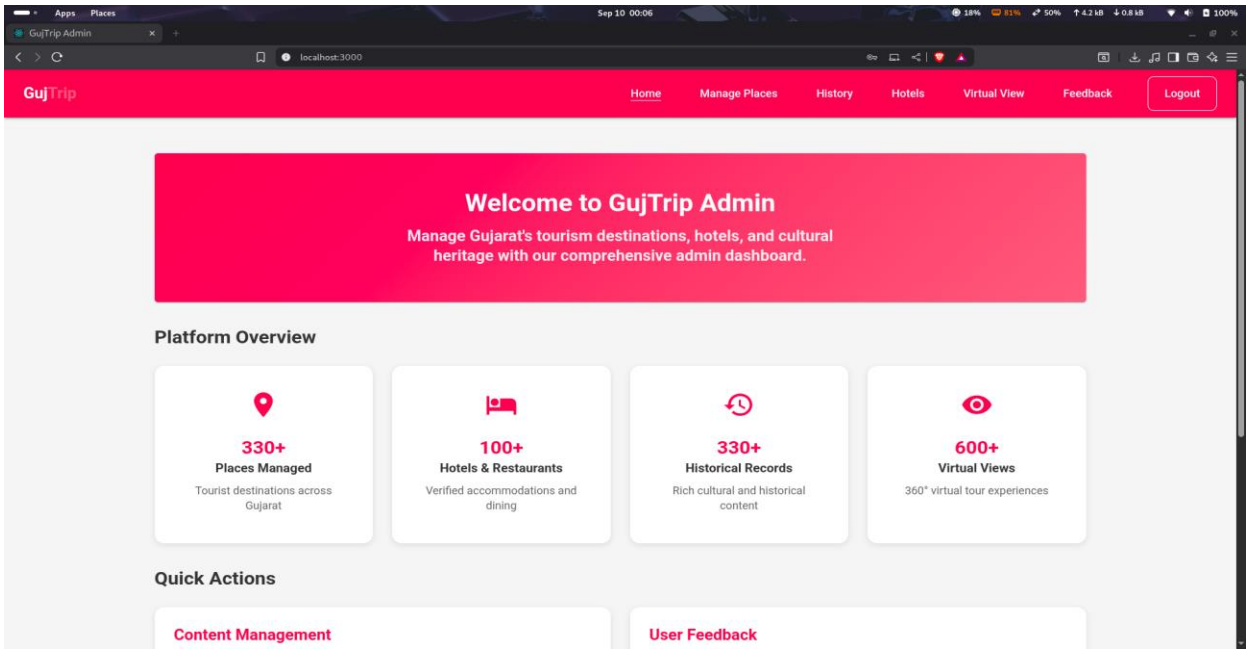
5.1 System Design Screenshot

5.1.1 Web Images

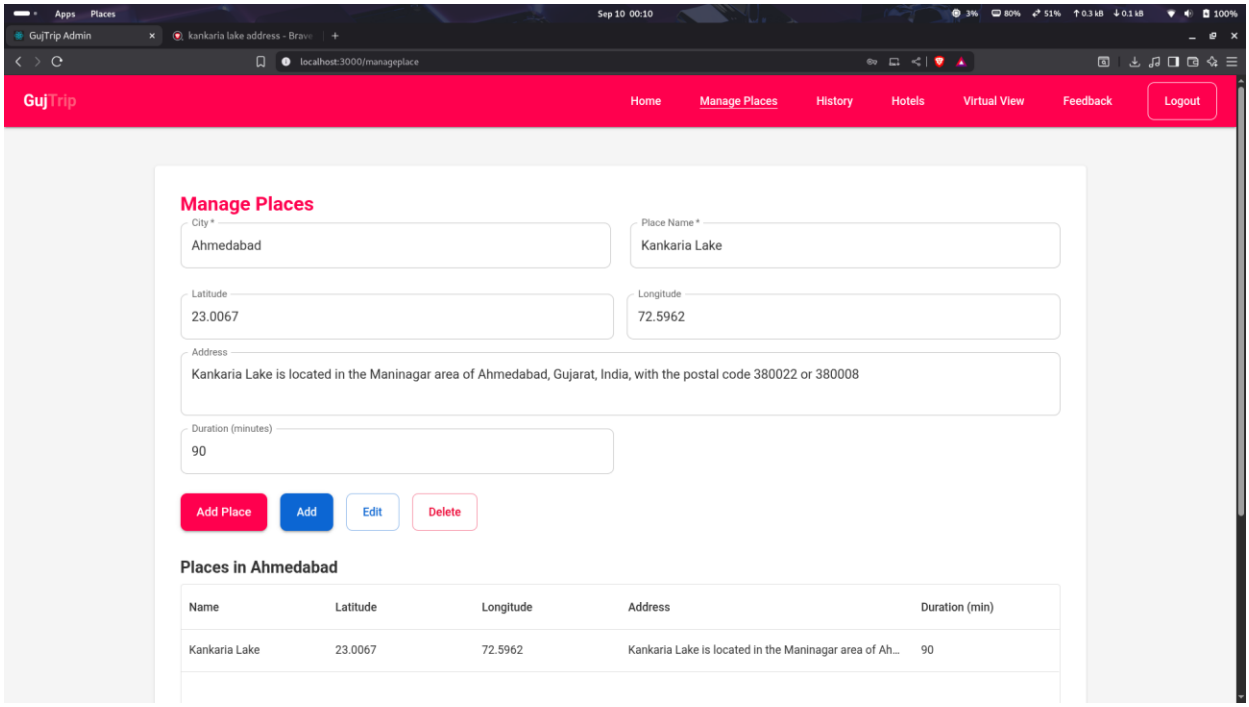
- **Image 1.**



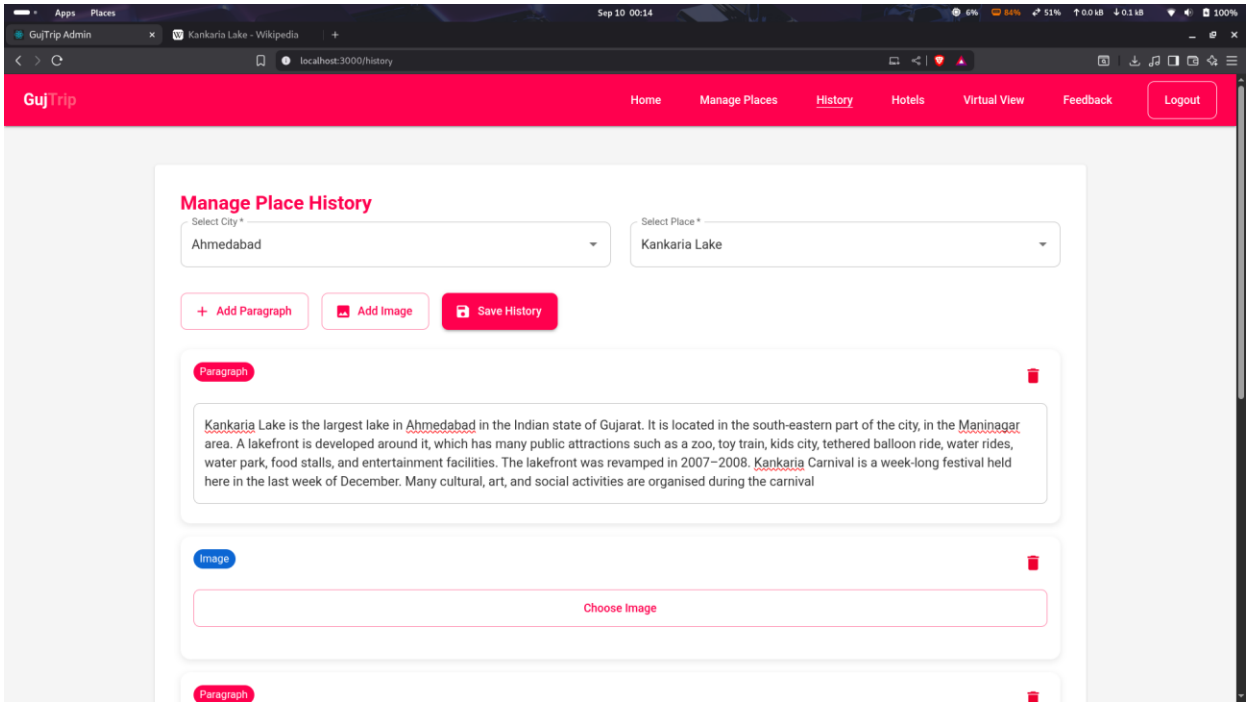
- **Image 2.**



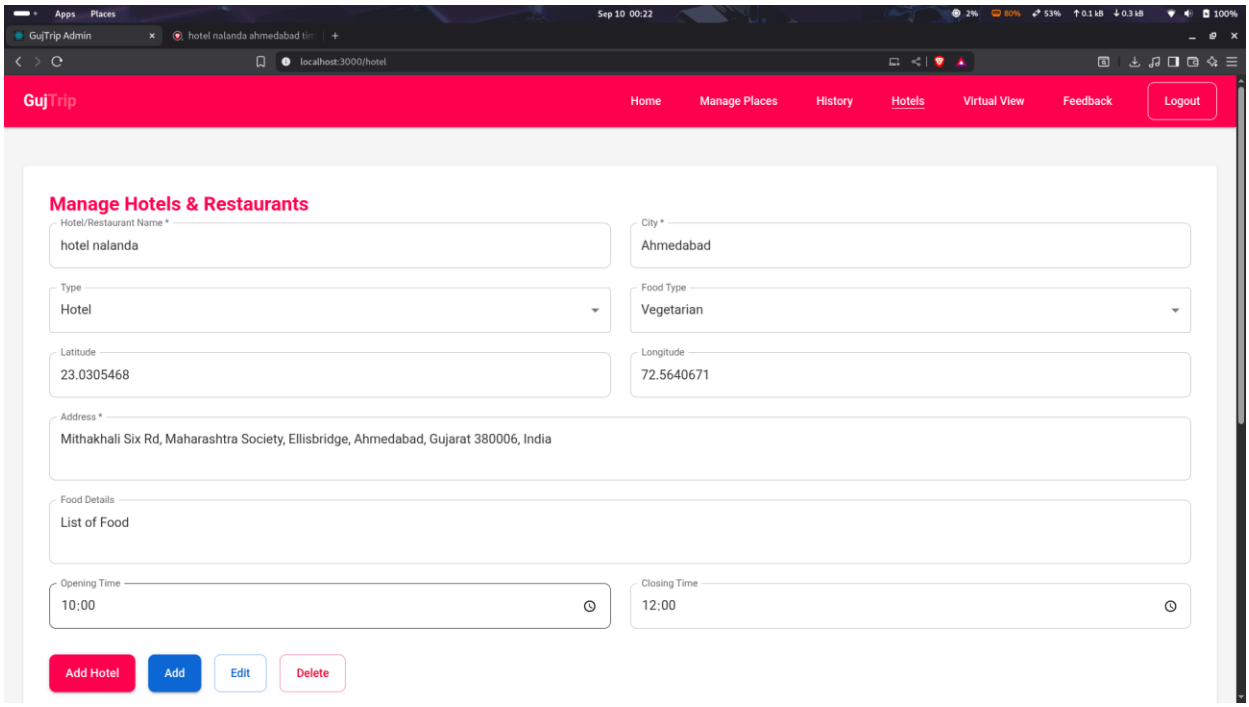
- **Image 3.**



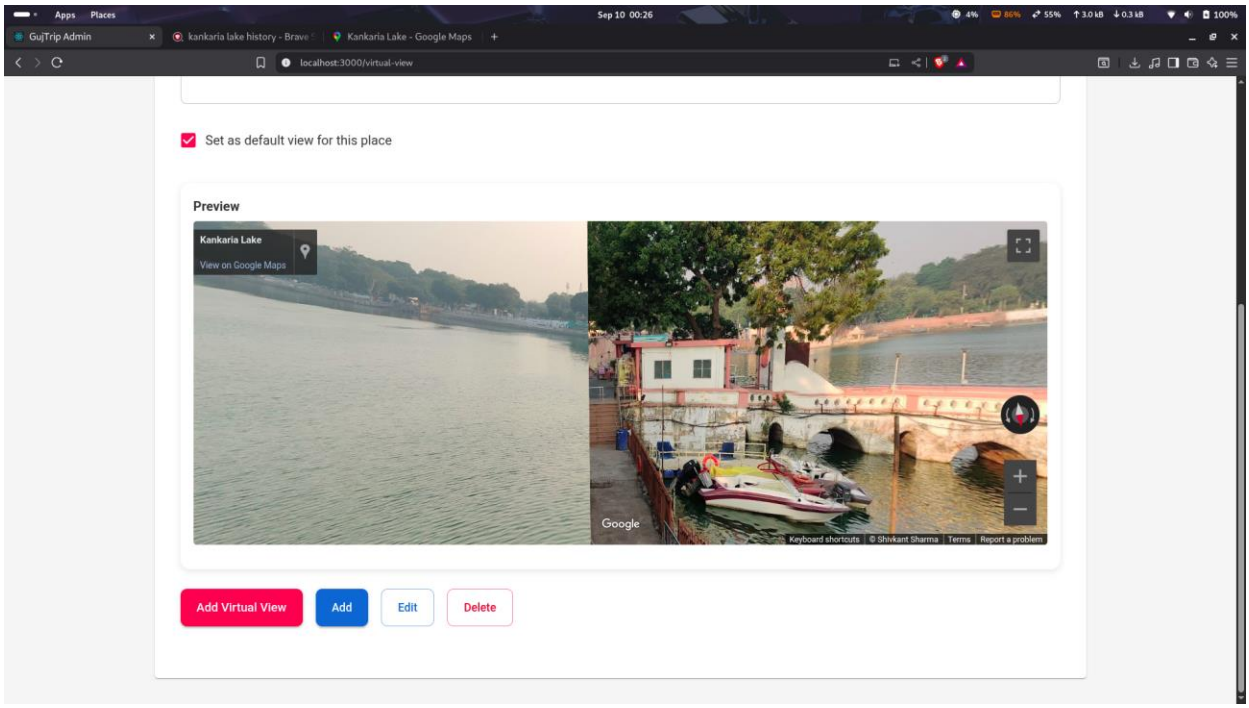
- **Image 4.**



- **Image 5.**

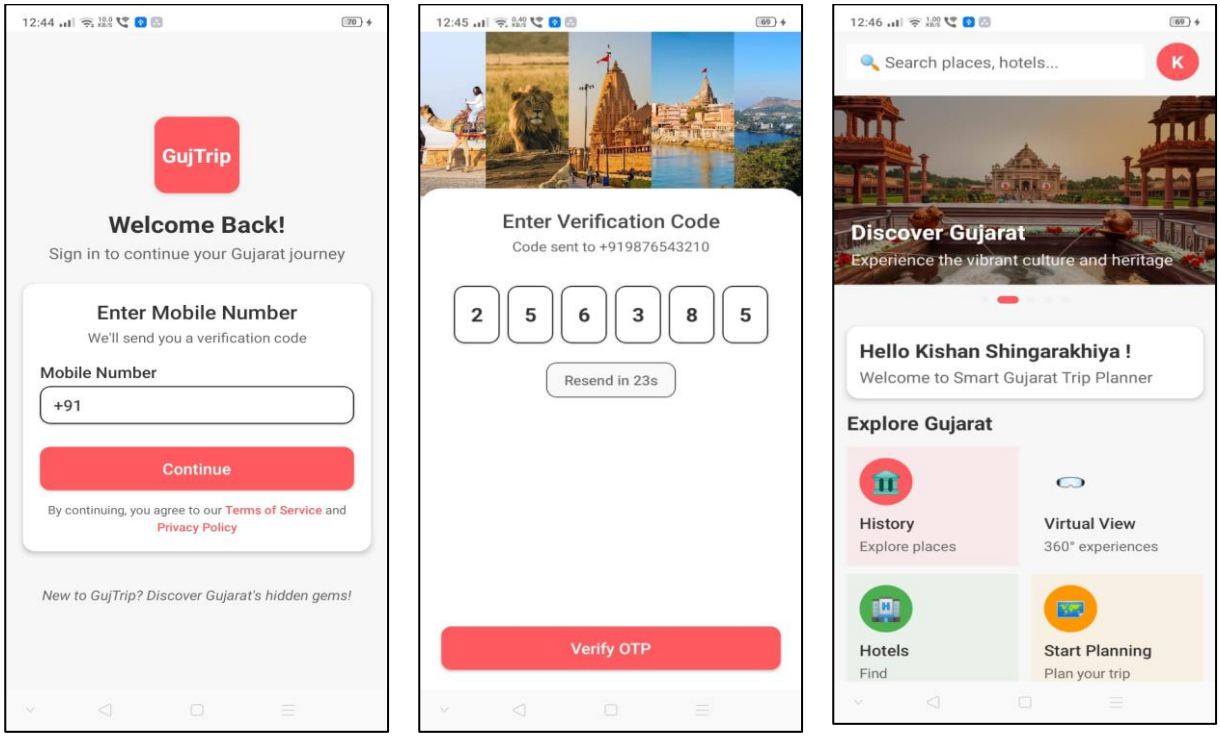


- **Image 6.**

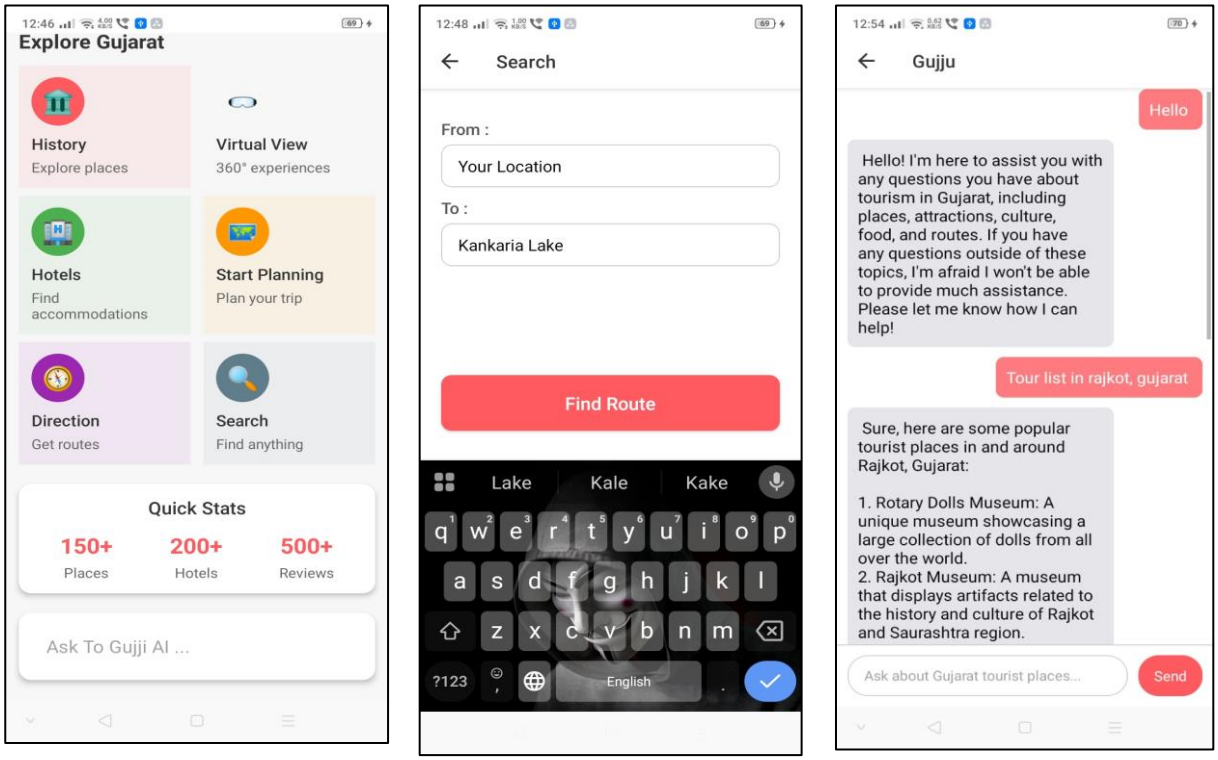


5.1.2 App Images

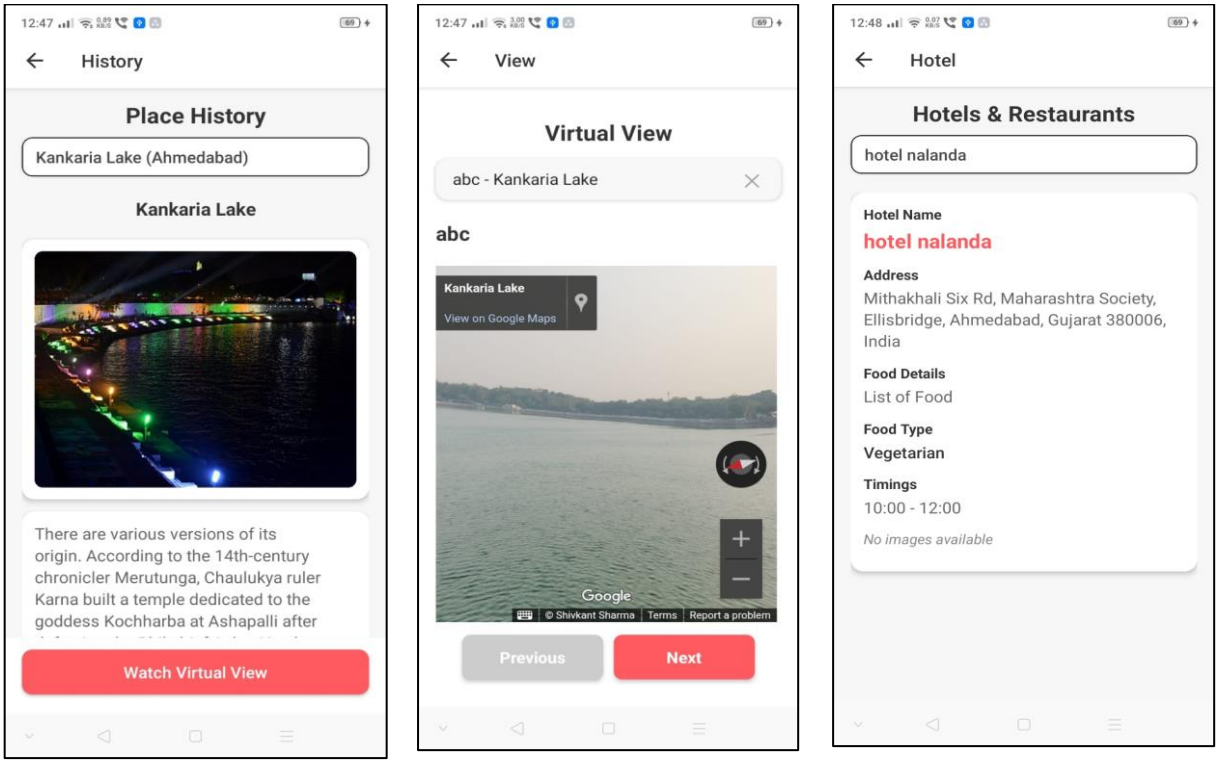
- **Image 1.**



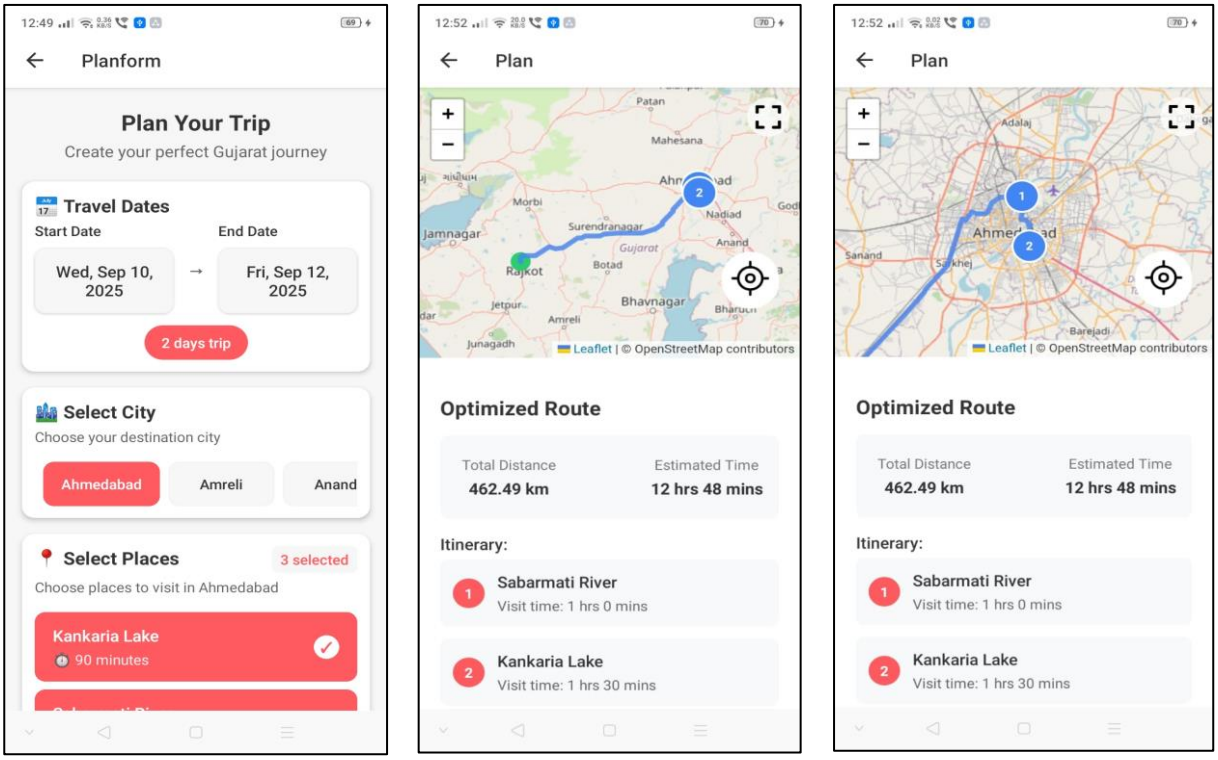
- **Image 2.**



- **Image 3.**

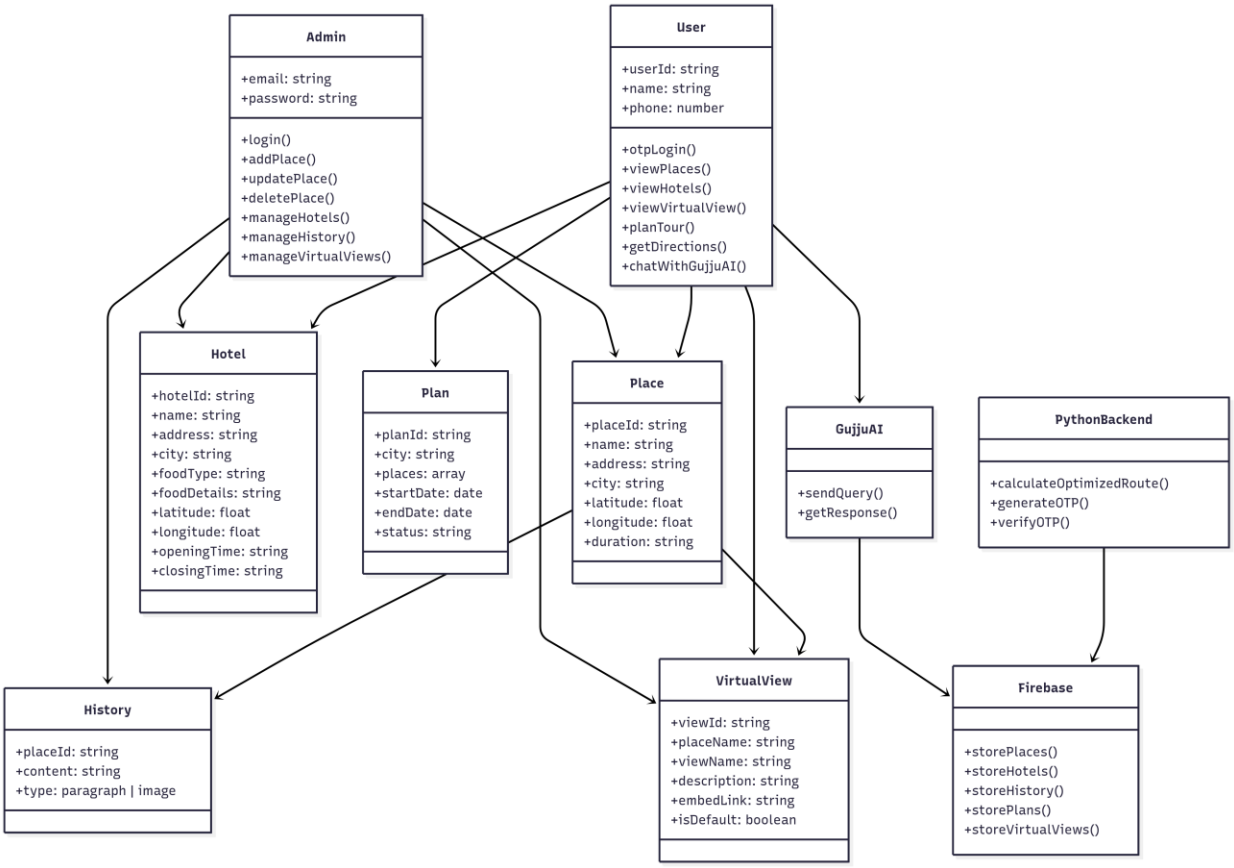

History
Place History
Kankaria Lake (Ahmedabad)
Kankaria Lake
There are various versions of its origin. According to the 14th-century chronicler Merutunga, Chaulukya ruler Karna built a temple dedicated to the goddess Kochharba at Ashapalli after
Watch Virtual View
View
Virtual View
abc - Kankaria Lake
abc
Kankaria Lake
View on Google Maps
Google
Previous
Next
Hotel
Hotels & Restaurants
hotel nalanda
Hotel Name
hotel nalanda
Address
Mithakhali Six Rd, Maharashtra Society, Ellisbridge, Ahmedabad, Gujarat 380006, India
Food Details
List of Food
Food Type
Vegetarian
Timings
10:00 - 12:00
No images available

- **Image 4.**

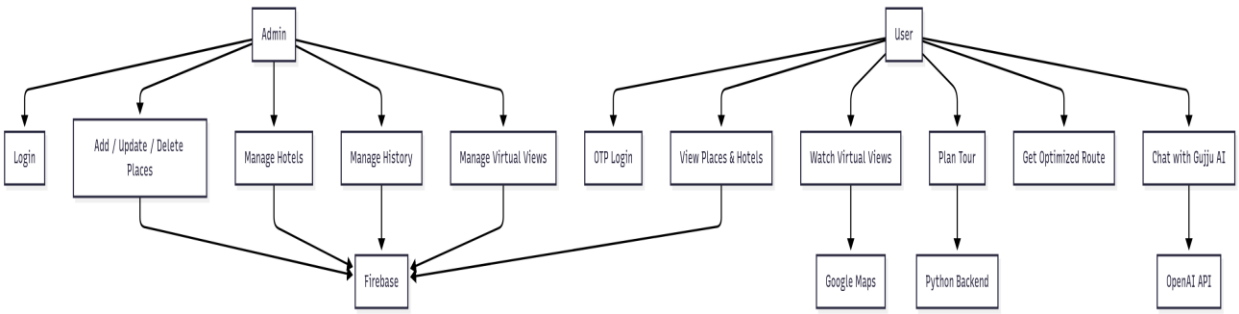

Planform
Plan Your Trip
Create your perfect Gujarat journey
Travel Dates
Start Date
End Date
Wed, Sep 10, 2025
Fri, Sep 12, 2025
2 days trip
Select City
Choose your destination city
Ahmedabad
Amreli
Anand
Select Places
3 selected
Choose places to visit in Ahmedabad
Kankaria Lake
90 minutes
Plan
Leaflet | © OpenStreetMap contributors
Optimized Route
Total Distance
462.49 km
Estimated Time
12 hrs 48 mins
Itinerary:
1
Sabarmati River
Visit time: 1 hrs 0 mins
2
Kankaria Lake
Visit time: 1 hrs 30 mins

5.2 Interface Design

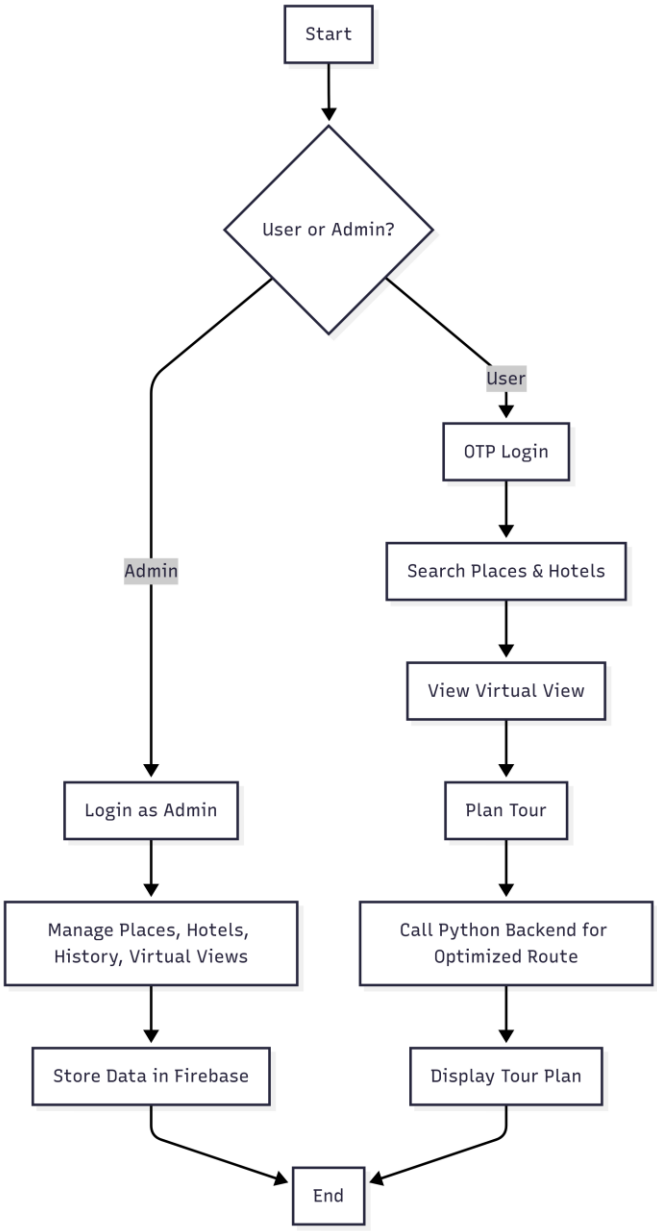
5.2.1 Class Diagram :



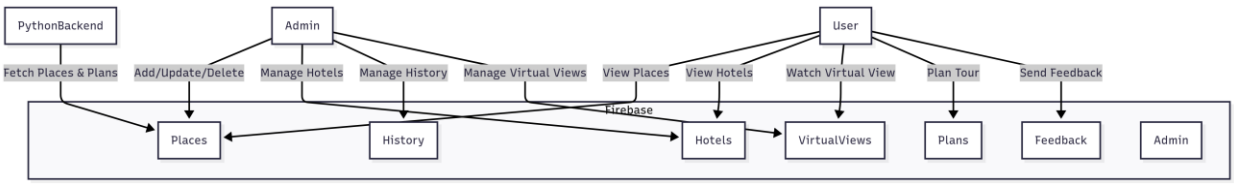
5.2.2 Use Case Diagram :



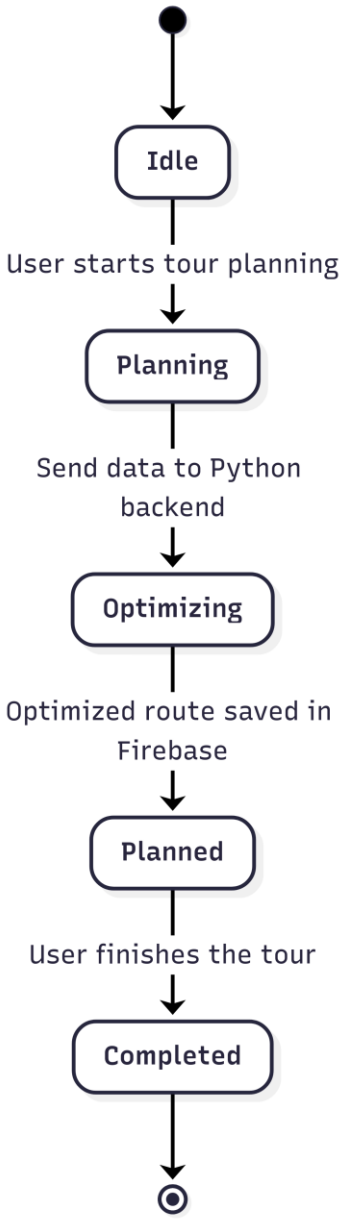
5.2.3 Activity Diagram :



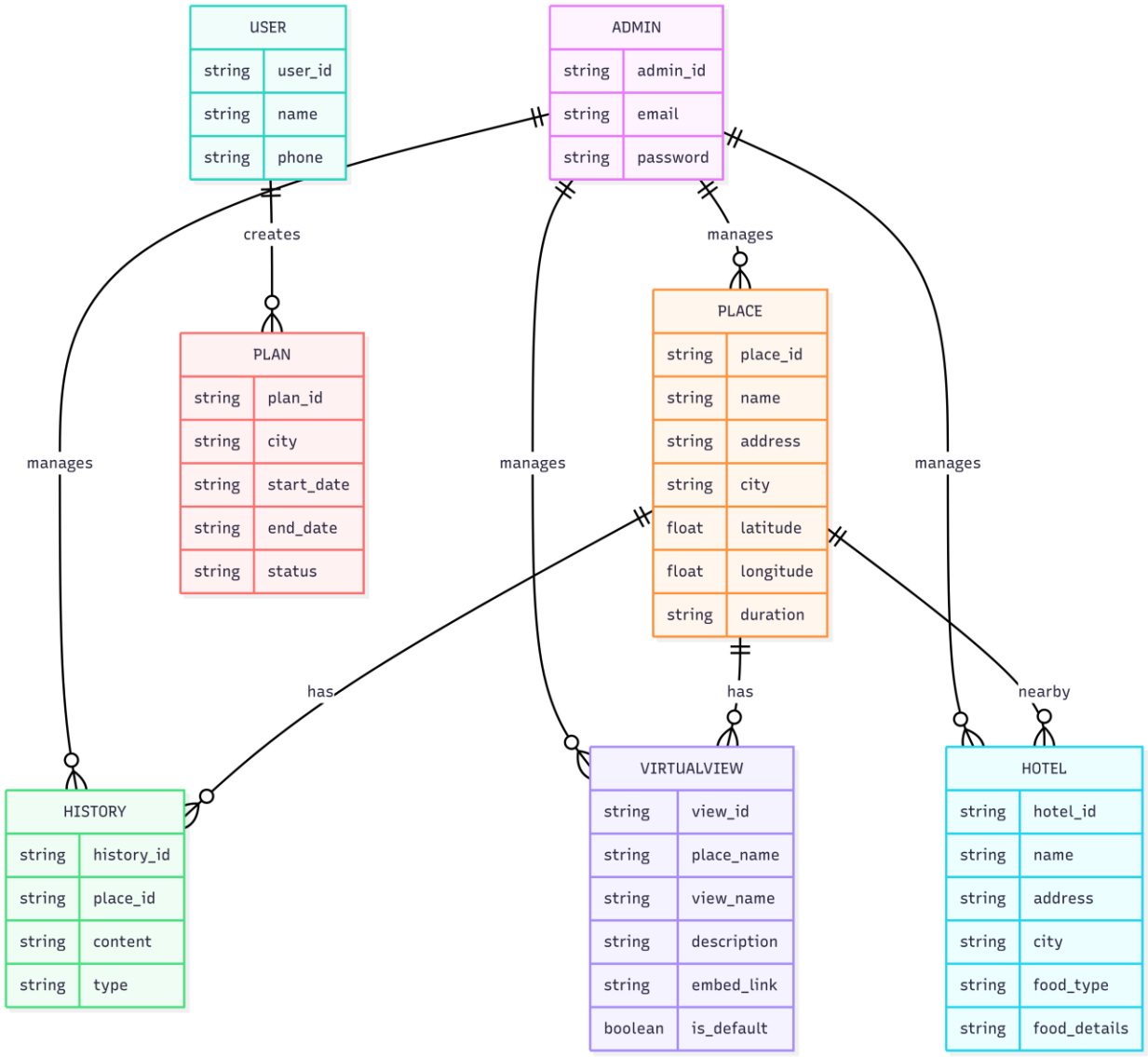
5.2.4 Data Flow Diagram :



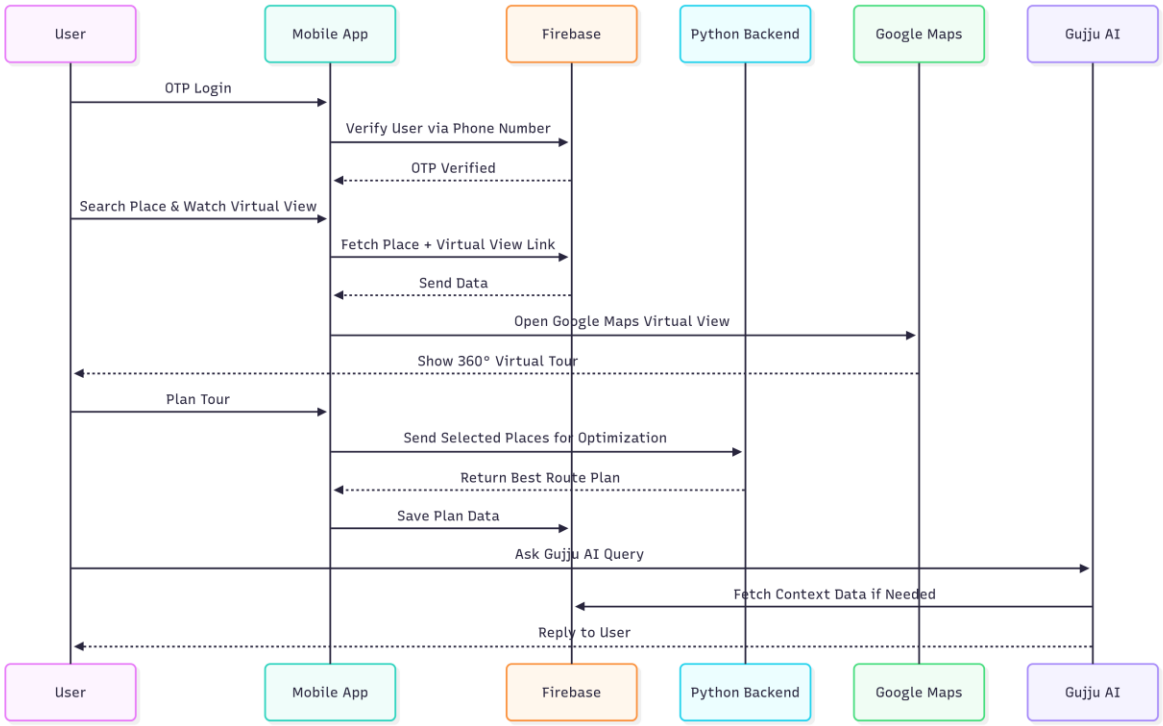
5.2.5 State Diagram :



5.2.6 E-R Diagram :



5.2.7 Sequence Diagram :



CHAPTER – 6

Code Implementation

6.1 Implementation Environment

- Development Machine (Host) :

- Operating System: Windows 10 / Windows 11 / Linux (Kali or Ubuntu)
- Processor: Intel Core i5 / i7 or equivalent (recommended)
- RAM: 8 GB (recommended; 4 GB minimum)
- Storage: 100 GB free disk space for SDKs, builds, and database

- Software & Tools :

- Frontend (Web): React.js (for Admin Panel)
- Frontend (Mobile App): React Native (Cross-Platform App)
- Backend: Flask (Python) for API & server-side scripting
- Database: Firebase Realtime Database + Firebase Authentication
- API Integration:
  → OpenRouteService API → Route Optimization & Travel Time Calculation
  → Twilio API → OTP Verification for Login/Signup
- IDE / Editors:
  → Visual Studio Code → React.js & React Native development
  → Android Studio → Emulator & APK builds
- Testing Devices:
  → Physical Android Device → Android 13+ (4 GB RAM, 128 GB)
  → Android Emulator → For quick testing & debugging

- Important NPM / Python Dependencies :

- **Frontend Dependencies (Web)**
  → react — Core React library for building the UI.
  → react-dom — Renders React components into the browser DOM.
  → react-router-dom — For navigation and routing between pages.
  → axios — For making HTTP requests to the Flask backend.
  → firebase — Firebase authentication and database integration.
  → chart.js & react-chartjs-2 — For displaying travel analytics and reports.
  → react-icons — To add modern, scalable icons.
  → tailwindcss — Utility-first CSS framework for styling.

→ sweetalert2 — For interactive alerts and confirmation dialogs.

- **Frontend Dependencies (Mobile App)**
  → react-native — Core framework for mobile app development.
  → react-navigation — For screen navigation and tab management.
  → axios — To send and receive data from the Flask backend.
  → firebase — User authentication, OTP, and data storage.
  → react-native-maps — To display maps and tourist locations.
  → react-native-geolocation-service — To fetch the user's current location.
  → react-native-permissions — For handling runtime permissions.
  → react-native-vector-icons — For modern and attractive icons.
  → react-native-paper — Pre-built UI components for a clean design.
  → react-native-toast-message — To display success/error notifications.

- **Backend Dependencies (Flask + Python)**
  → flask — Lightweight Python web framework for building APIs.
  → flask-cors — To allow secure communication between React/React Native and Flask.
  → firebase-admin — To manage Firebase authentication and database from the backend.
  → requests — For handling API requests (e.g., OpenRouteService, Twilio).
  → openrouteservice — For calculating optimized travel routes and distances.
  → twilio — For sending OTPs via SMS during user authentication.
  → geopy — For geolocation and distance calculations.
  → uuid — For generating unique user IDs and session tokens.
  → python-dotenv — For managing environment variables securely.
  → gunicorn — For deploying the Flask app in production.

6.2 Program/Module Specification

6.2.1 High-Level Modules

1. **Web Admin Panel (React.js)**
   - Home → Displays statistics & analytics
   - Places Management → Add, edit, delete tourist places
   - Hotels Management → Manage hotel details

- Virtual Views & History → Upload images & descriptions
- Login → Login as Admin

2. **Mobile App (React Native)**
   - Home Screen → Displays featured destinations & categories
   - Places Screen → Search & explore tourist spots
   - Hotel Screen → Search & explore Hotels
   - Virtual View → Shows panoramic images & information
   - History → Shows images & information
   - Route Finder → Search two places & display shortest path + distance
   - Tour Planner → Generates personalized itineraries
   - AI Chatbot → Uses ChatGPT API for travel-related queries
   - OTP Login → Firebase-based secure authentication
   - Profile Screen → Manage User Profile

3. **Backend (Flask Server)**
   - /upload → Upload images & documents
   - /send-otp → Send OTP via Twilio
   - /verify-otp → Verify user OTP
   - /optimize-route → Calculate the best possible travel route using OpenRouteService API

4. **Database (Firebase)**
   - User Collection → Stores user profiles & preferences
   - Places Collection → Stores tourist place details & virtual views
   - Hotels Collection → Stores hotel information
   - Itinerary Collection → Saves planned itineraries

6.2.2 Interaction Flow (Example – Plan a Trip)

- User logs into mobile app using Twilio OTP.
- User selects places to visit and trip duration.
- Mobile app sends request to Flask Itinerary API.
- Flask use OpenRouteService to create optimized itinerary.

- Itinerary is stored in Firebase for later access.
- User can view trip plan, hotel details, routes, and virtual views in the app.

6.2.3 Project File Structure (Simplified)

```
GujTrip/
├── backend/
│   ├── backend.py
│   └── media/
├── GujTrip/
│   └── components/
│       ├── HistoryScreen.jsx
│       ├── HomeScreen.jsx
│       ├── HotelScreen.jsx
│       ├── LoginScreen.jsx
│       ├── MapScreen.jsx
│       ├── OtpScreen.jsx
│       ├── PlaceDetailsScreen.jsx
│       ├── PlanDisplayScreen.jsx
│       ├── PlanningScreen.jsx
│       ├── ProfileScreen.jsx
│       ├── Search.jsx
│       ├── SocialScreen.jsx
│       ├── TripPlannerScreen.jsx
│       ├── VirtualViewScreen.jsx
│       ├── firebase.js
│       ├── logout.jsx
│       ├── planFormScreen.jsx
│       └── ui/
│           ├── LoadingSpinner.jsx
│           ├── ThemedButton.jsx
│           ├── ThemedCard.jsx
│           └── ThemedTextInput.jsx
└── web/
    └── src/
        ├── App.jsx
        ├── components/
        │   ├── Feedback.css
        │   ├── Feedback.jsx
        │   ├── History.css
        │   ├── History.jsx
        │   ├── Home.css
        │   ├── Home.jsx
        │   ├── Hotel.css
        │   ├── Hotel.jsx
        │   ├── Login.css
        │   ├── Login.jsx
        │   ├── ManagePlace.css
        │   ├── ManagePlace.jsx
        │   ├── Navbar.jsx
        │   ├── View.css
        │   └── View.jsx
        ├── firebase.js
        ├── index.css
        ├── index.js
        └── theme.js
```

6.3 Coding Standards

6.3.1 Naming Conventions

- Files & Folders: UpperCamelCase
- Classes: UpperCamelCase
- Variables & Functions: lowerCamelCase
- Constants: lowerCamelCase

6.3.2 Code Structure & Formatting

- Separate frontend and backend layers.
- Keep components modular and reusable.
- Use async/await for all API calls.
- Avoid deeply nested callbacks.

6.3.2 Error Handling

- Try/Catch used for all API requests.
- Display friendly error messages like:
  → "Network error, please try again."
  → "No route found between selected places."

6.3.4 UI/UX Standards

- Consistent color theme using React Native Stylesheet.
- Loading indicators during API calls.
- Virtual views and maps are responsive on all screen sizes.

6.3.5 Comments & Documentation

- Use // for single-line comments in JS, # in Python.
- Use /** ... */ for multi-line comments in JS.
- API endpoints and third-party integrations are properly documented.

CHAPTER – 7

Testing

7.1 Testing Strategy

- The testing strategy for the AI-Powered Travel Itinerary Planner was designed to ensure that every module works as expected, and when combined, the entire system functions smoothly and reliably.
- We followed a bottom-up testing approach:
  1. Unit Testing → Testing each individual module (e.g., authentication, itinerary generation, route optimization, AI recommendations).
  2. Integration Testing → Testing interactions between modules (e.g., places API + itinerary planner, AI suggestions + itinerary results).
  3. Validation Testing → Testing the entire system against the functional requirements to confirm expected behavior.
- We performed manual testing on physical Android devices, desktop browsers.

7.2 Testing Method

7.2.1 Unit Testing :

- **Objective:**
  - Verify that each **module** works independently without relying on other components.

- **Scope:**
  - **Authentication Module (Firebase & Twilio):**
    → Check user signup, login, OTP verification, and secure session management.
  - **Itinerary Planner Module (Flask + OpenRouteService):**
    → Check if the shortest route is generated with accurate travel times and buffer durations.
  - **AI Suggestion Module (ChatGPT API):**
    → Validate if the chatbot suggests relevant tourist places and personalized plans.
  - **Places Management Module (React.js Admin Panel):**
    → Check if adding, editing, and deleting places works correctly.
  - **Map Integration Module (React Native Maps):**
    → Verify location display, zoom, and route markers.

  - **Example:**
    - Input: Start: "Sabarmati River", End: "Kankaria Lake"

- Expected Output: Optimized travel route with accurate travel time and distance.

7.2.2 Integration Testing :

- **Objective:**
  - Verify that multiple modules work **together seamlessly**.
- **Scope:**
  - **Firebase Authentication + Itinerary Planner:**
    → Ensure only logged-in users can create travel plans.
  - **Places API + OpenRouteService API:**
    → Fetch coordinates, calculate optimized routes, and return travel time.
  - **AI Suggestions + User Preferences:**
    → Suggest personalized places based on user's available time, interests, and location.
  - **Frontend + Backend Integration:**
    → Ensure React Native mobile app and React.js dashboard correctly consume Flask APIs.
- **Example:**
  - Input: User selects Rajkot for a 1-day trip.
  - Flow:
    → User enters preferences → AI suggests 8 must-visit places → Flask backend calculates optimized travel route using OpenRouteService → Mobile app displays complete itinerary.
  - Expected Result: Optimized itinerary is displayed with time slots, distance, and estimated cost.

7.2.3 Validation Testing :

- **Objective:**
  - Ensure the complete system meets **user requirements** and functions properly under real usage scenarios.
- **Scope:**
  - Users can log in, and manage their profiles.
  - Users can select a city and get AI-based itinerary recommendations.
  - Optimized routes are generated using OpenRouteService API.
  - Users can view maps with markers for all planned locations.

- System should work offline for saved itineraries and provide error handling for no-internet cases.

- **Example Validation:**
  - Requirement: The system should generate an optimized one-day travel itinerary for Rajkot.
  - Test: Select Rajkot→ Choose 8 places → Plan trip.
  - Expected Result:
    → Suggested travel plan shows places in the best visiting order, estimated travel time, costs, and distance.

7.3 Test Cases :

7.3.1 Test Suite :

Test Case ID	Description	Input	Expected Output	Status
TC01	Verify OTP login with valid number	Phone: 9876543210	OTP sent successfully	Pass
TC02	Verify OTP login with invalid number	Phone: 123	Error: "Invalid number"	Pass
TC03	Verify login with incorrect OTP	OTP: 000000	Error: "Incorrect OTP"	Pass
TC04	Fetch list of tourist places	City: Rakot	Returns list of places with details	Pass
TC05	AI-based place suggestions	"Best places in Rajkot"	Returns top 5 places	Pass
TC06	Generate optimized itinerary	8 selected places	Returns optimized plan with time & distance	Pass
TC07	Display route on map	Start & End location	Map shows correct route with markers	Pass

TC08	Offline mode for saved itineraries	Internet off	Loads saved itinerary successfully	Pass
TC09	OpenRouteService API response	Start + End location	Returns distance, duration, and waypoints	Pass
TC10	ChatGPT AI response	"Plan my Rajkot trip"	Returns AI-generated itinerary	Pass
TC11	Admin adds a new place	Place data	Place saved successfully	Pass
TC12	Admin deletes a place	Place ID	Place removed from database	Pass

CHAPTER – 8

Limitations and Future Enhancement

8.1 Limitations

- Although the AI-Driven Personalized Travel Itinerary Planner successfully generates optimized travel plans, provides personalized recommendations, and integrates AI-based suggestions, there are some limitations:

1. **Internet Dependency**
   - The system relies heavily on a stable internet connection because itinerary generation, route optimization, and AI-powered recommendations depend on Firebase, ChatGPT API, and OpenRouteService API.
2. **Platform Limitation**
   - Currently, the system is developed only for Android mobile (React Native app) and web (React.js). There is no iOS application implemented yet.
3. **Limited Offline Functionality**
   - Without an active internet connection, users cannot fetch tourist places, get AI recommendations, or generate travel itineraries.
4. **Accuracy Constraints**
   i. AI-generated recommendations may sometimes provide non-optimal or less relevant results if place data is insufficient.
   ii. Travel time and route predictions depend on third-party APIs, which may not always be accurate in real-time traffic conditions.
5. **Battery & Performance Usage**
   - Continuous location tracking, map rendering, and API calls may cause higher battery drain on low-end mobile devices.
6. **Data Storage Limitation**
   - While Firebase stores user preferences and itineraries, translation history, travel patterns, and AI conversations are not stored for long-term analysis.
7. **Third-Party API Limitations**
   - The system relies on external services like ChatGPT API, OpenRouteService, and Firebase. If any of these services face downtime or API rate limits, itinerary generation and AI responses may fail.

8.2 Future Enhancement

- To improve the application and make it more powerful and user-friendly, the following enhancements are planned for future versions:
- **Offline Mode**
  - Integrate offline maps and cached itineraries so that users can access saved routes and recommendations without an internet connection.
- **iOS & Cross-Platform Support**
  - Extend the app to iOS devices and possibly create a desktop version for better accessibility.
- **Real-Time Traffic & Weather Integration**
  - Include live traffic updates and weather forecasts to make itineraries more accurate and dynamic.
- **Smart Budget-Based Planning**
  - Enhance AI capabilities to generate itineraries based on user budgets, including ticket prices, transportation costs, and hotel stays.
- **Multi-Language Support**
  - Integrate real-time translations for tourist places, recommendations, and AI responses to help international travelers.
- **AI-Powered Dynamic Itineraries**
  - Allow the AI to auto-update travel plans based on delays, closed attractions, or unexpected conditions.
- **User Preference Learning**
  - Implement machine learning models to analyze user behavior and provide personalized suggestions over time.
- **Social Travel Planning**
  - Add a feature to plan group trips where multiple users can collaborate on a shared itinerary.
- **Cloud Sync & Multi-Device Support**
  - Enable cloud synchronization so users can seamlessly switch between devices and retain itineraries, preferences, and history.
- **Integration with Booking Platforms**
  - Connect with hotel, transport, and ticket booking APIs to allow end-to-end trip planning and reservations.

CHAPTER – 9

Conclusion

The **AI-Driven Personalized Travel Itinerary Planner** successfully achieves its objective of providing users with a **smart, efficient, and personalized travel planning experience**. By integrating **React.js** for the web frontend, **React Native** for the mobile app, **Python Flask** for the backend, **Firebase** for secure data storage, and **ChatGPT API** for AI-powered recommendations, the system delivers an **intuitive**, **interactive**, and **intelligent** platform for travelers.

The system simplifies the entire **trip planning process** by automating **tourist place suggestions**, **optimized route planning**, **travel time estimation**, and **personalized recommendations** based on user preferences. Through **OpenRouteService API**, the application generates optimized paths and accurate travel times, while **Firebase** ensures smooth authentication and data management.

The project demonstrates how **AI integration** can enhance user experience by suggesting **best visiting times**, **must-see attractions**, and **optimized schedules**. This eliminates the need for manual research, saving travelers both **time** and **effort**.

Although there are some limitations, such as **internet dependency**, **lack of offline support**, and **API restrictions**, the system provides a strong foundation for future enhancements like **real-time traffic integration**, **offline functionality**, **multi-language support**, and **cross-platform availability**.

In conclusion, this project delivers a **scalable**, **user-friendly**, and **AI-powered travel planning solution** that improves decision-making for travelers and sets a solid base for further advancements in **smart tourism applications**.

CHAPTER – 10

Reference

Websites :

- React Js Documentaion: https://react.dev/learn
- React Native Documentation: https://reactnative.dev/docs
- Firebase Documentation: https://firebase.google.com/docs
- OpenAI API Documentation: https://platform.openai.com/docs
- Twilio OTP API: https://www.twilio.com/docs/verify
- Google Maps Platform: https://developers.google.com/maps
- OpenRouteService API (for route optimization): https://openrouteservice.org/dev/
- JavaScript MDN Documentation: https://developer.mozilla.org
- Python Official Docs: https://docs.python.org/3/