**AI-Based Travel Planner**

**1. Introduction**

* 1. **Problem Overview**

Travel planning is a complex and time-consuming task, requiring travelers to research destinations, accommodations, and activities while adhering to budget and time constraints. Many existing tools lack personalization, rely on outdated information, or fail to integrate real-time data, resulting in inefficient planning and suboptimal travel experiences. The AI Trip Advisor addresses these challenges by providing a web-based platform that leverages AI and Google APIs to generate personalized travel itineraries and budget-friendly hotel recommendations, all while operating within a $200/month API cost limit.

* 1. **Objectives**

The AI Trip Advisor aims to:

- Create customized travel itineraries based on user inputs (destination, budget, travel dates, preferences).

- Offer real-time, budget-conscious hotel recommendations.

- Utilize Google Maps APIs and Google Gemini AI for dynamic, location-based planning.

- Provide a responsive, user-friendly interface using modern web technologies.

- Ensure cost-efficiency and scalability in API usage.

**2. Methodology**

**2.1 Algorithm Description**

The core algorithm processes user inputs to generate a personalized itinerary:

1. Input Collection: Users provide destination, budget, travel dates, and preferences (e.g., cultural, adventure) through a web form.

2. AI Processing: Google Gemini AI analyzes preferences to prioritize activities and attractions.

3. Data Retrieval: Google Maps Places, Photos, and Geolocation APIs fetch location details and hotel options.

4. Itinerary Construction: The system organizes activities and hotels into a daily schedule, ensuring compliance with the budget.

5. Output Rendering: The itinerary is displayed in a responsive UI with interactive elements such as maps and hotel details.

**2.2 Pseudocode**

FUNCTION GenerateItinerary(userInput)

SET destination = userInput.destination

SET budget = userInput.budget

SET dates = userInput.dates

SET preferences = userInput.preferences

// Fetch location data

locations = Call GoogleMapsAPI.GetPlaces(destination, preferences)

// Prioritize activities using AI

activities = Call GeminiAI.PrioritizeActivities(locations, budget, dates)

// Fetch hotels within budget

hotels = Call GoogleMapsAPI.GetHotels(destination, budget)

// Structure itinerary

itinerary = StructureItinerary(activities, hotels, dates)

RETURN itinerary

END FUNCTION

**2.3 Use Case Diagram Flowchart**

The use case diagram illustrates interactions between the user and the system. Below is a textual description of the flowchart, which can be recreated in Microsoft Word using SmartArt or a diagramming tool (e.g., Lucidchart, Draw.io).

**Actors:**

- User: A traveler planning a trip.

- System: The AI Trip Advisor web application.

- Google APIs: External services providing location data and AI processing.

- Firebase: External service for authentication and data storage.

**Use Cases:**

1. Login: User authenticates via Google OAuth to access personalized features.

2. Create Trip: User inputs preferences to generate a new itinerary.

3. View Trip: User views details of a specific trip.

4. View My Trips: User accesses a list of all saved trips.

5. Logout: User ends the session.

**Relationships:**

- User -> Login: Initiates authentication, extending to Google OAuth.

- User -> Create Trip: Submits preferences, system queries Google APIs.

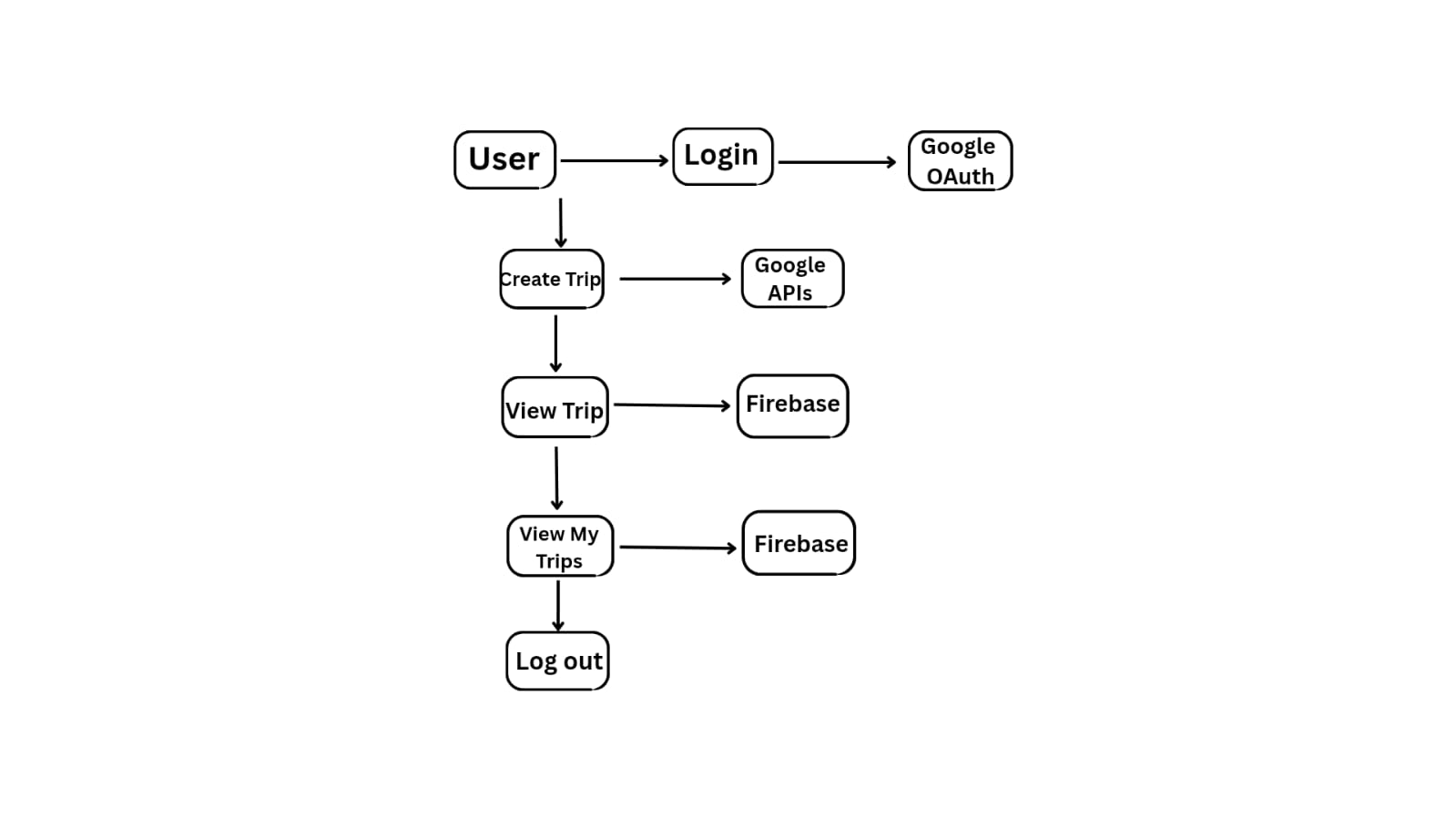
- User -> View Trip: Selects a trip, system retrieves data from Firebase.

- User -> View My Trips: Requests saved trips, system queries Firebase.

- System -> Google APIs: Fetches real-time location and AI data.

- System -> Firebase: Stores and retrieves user and trip data.

**Flowchart Description:**



**Fig. flow chart**

**Instructions for Creation in Word:**

1. Open Microsoft Word and go to Insert -> SmartArt -> Process or Hierarchy.

2. Add an oval for "User" at the top.

3. Create rectangles for each use case: "Login", "Create Trip", "View Trip", "View My Trips", "Logout".

4. Connect "User" to each use case with arrows.

5. Add rectangles for "Google OAuth" (connected to "Login"), "Google APIs" (connected to "Create Trip"), and "Firebase" (connected to "View Trip" and "View My Trips").

6. Label arrows with actions (e.g., "Authenticates", "Fetches data", "Stores trip").

7. Alternatively, use a diagramming tool like Lucidchart or Draw.io, export the diagram as a PNG, and insert it into the Word document via Insert -> Pictures.

**3. Implementation Details**

**3.1 Programming Language and Libraries**

- Language: JavaScript (ES2020).

- Frontend:

- React 18.3.1: Component-based UI framework.

- Vite 6.3.5: Fast build and development tool.

- Tailwind CSS 3.4.6: Utility-first CSS framework.

- Radix UI: Accessible UI components (e.g., Accordion, Dialog).

- APIs:

- Google Maps Places, Photos, Geolocation: For location and hotel data.

- Google Gemini AI: For activity prioritization and itinerary generation.

- Backend:

- Firebase 11.7.1: Authentication and data storage.

- Libraries:

- Axios 1.7.2: HTTP requests for API calls.

- React Router DOM 6.25.1: Client-side routing.

- GSAP 3.12.5: Animations for UI elements.

- React Scroll Parallax 3.4.5: Parallax scrolling effects.

- Tools:

- ESLint 8.57.0: Code linting.

- PostCSS 8.4.39 and Autoprefixer 10.4.19: CSS processing.

- Vite Plugin React 4.3.1: React integration with Vite.

**3.2 Code Structure**

The project is organized as follows:

- src/:

- App.jsx: Main component rendering the Hero section.

- main.jsx: Configures routing, Google OAuth, and Parallax providers.

- components/custom/: Custom components like Header.jsx and Hero.jsx.

- create-trip/index.jsx: Logic for trip creation form and API calls.

- view-trip/[tripId]/index.jsx: Displays trip details.

- my-trips/index.jsx: Lists saved trips.

- index.css: Tailwind CSS styles with custom variables.

- public/: Static assets (bg.jpg, vite.svg).

- Configuration Files:

- .env: Stores API keys.

- .eslintrc.cjs: ESLint configuration.

- tailwind.config.js: Tailwind CSS settings.

- vite.config.js: Vite build configuration.

**3.3 How to Run the Code**

1. Clone the Repository:

git clone https://github.com/your-username/ai-trip-advisor.git

cd ai-trip-advisor

2. Set Up Environment Variables:

Create a .env.local file:

VITE\_GOOGLE\_PLACE\_API\_KEY=your\_google\_place\_api\_key

VITE\_GOOGLE\_GEMINI\_AI\_API\_KEY=your\_google\_gemini\_ai\_api\_key

VITE\_GOOGLE\_AUTH\_CLIENT\_ID=your\_google\_auth\_client\_id

FIREBASE\_API\_KEY=your\_firebase\_api\_key

3. Install Dependencies:

npm install

4. Run the Development Server:

npm run dev

Access the app at http://localhost:5173.

5. Build for Production:

npm run build

npm run preview

**4. Dataset Handling**

The AI Trip Advisor does not use a static dataset. Instead, it relies on:

- Real-Time Data:

- Google Maps APIs provide location, hotel, and photo data based on user inputs.

- Google Gemini AI processes preferences to generate activity recommendations.

- User Inputs: Collected via forms (destination, budget, dates, preferences).

- Data Storage: Firebase stores user profiles and trip itineraries.

- Optimization: API responses are cached locally to minimize costs, ensuring compliance with the $200/month limit.

**5. Output Format**

The system outputs a structured itinerary, stored in Firebase and displayed in the UI. Example JSON-like structure:

{

"tripId": "12345",

"destination": "Paris",

"dates": ["2025-06-01", "2025-06-03"],

"itinerary": [

{

"day": 1,

"activities": ["Eiffel Tower", "Louvre Museum"],

"hotel": {"name": "Hotel Paris", "price": 100}

},

{

"day": 2,

"activities": ["Notre-Dame", "Seine River Cruise"],

"hotel": {"name": "Hotel Paris", "price": 100}

}

],

"totalCost": 450

}

The UI renders this as a visually appealing timeline or list using Tailwind CSS, with interactive elements like hotel details, map links, and activity descriptions.

**6. Challenges and Solutions**

- Challenge: Staying within the $200/month API cost limit.

- Solution: Implemented caching of API responses and optimized queries using Axios.

- Challenge: Real-time data processing causing UI delays.

- Solution: Used async/await with React state management for seamless updates.

- Challenge: Ensuring secure user authentication.

- Solution: Integrated Google OAuth with Firebase for robust security.

- Challenge: Creating a responsive UI for all devices.

- Solution: Leveraged Tailwind CSS’s mobile-first approach and tested across screen sizes.

**7. Test Case Results**

- Test Case 1:

- Input: 3-day trip to Paris, $500 budget, cultural preferences.

- Output: Itinerary with Eiffel Tower, Louvre Museum, and a budget hotel ($100/night).

- Result: Successfully generated, total cost $450, displayed correctly in UI.

- Test Case 2:

- Input: 5-day trip to Tokyo, $1000 budget, focus on food and culture.

- Output: Itinerary with Tsukiji Market, Shibuya Crossing, and mid-range hotels ($150/night).

- Result: Successfully generated, total cost $950, responsive UI.

- Note: Results are hypothetical based on the app’s functionality, as no specific test data was provided.

**8. Future Improvements**

- Offline Access: Enable itinerary viewing without internet connectivity.

- Multi-Language Support: Add internationalization for global accessibility.

- Additional APIs: Integrate weather, event, or flight APIs for comprehensive planning.

- Mobile App: Develop native iOS/Android apps using React Native.

**9. References**

- React Documentation: https://react.dev/

- Google Maps APIs: https://developers.google.com/maps

- Firebase: https://firebase.google.com/

- Tailwind CSS: https://tailwindcss.com/

- Axios: https://axios-http.com/

- React Router: https://reactrouter.com/