# Vellore Institute of Technology
# School of Computer Science and Engineering

**Software Engineering**

**Course Code:** BCSE301L

**Class Number:**

**Slot: G2 + TG2**

**PROJECT ON**

# "VIHARA AI TOUR PLAN"


**TEAM MEMBERS:**

1. **Aditya Shriwal**
2. **Aman Chauhan**
3. **Archit Yadav**

# ACKNOWLEDGEMENT

I would like to express my deep appreciation and heartfelt gratitude to my supervisor, Prof.Yoganand S., from VIT University. Without his unwavering motivation and continuous encouragement, this research would not have been completed successfully.

I am sincerely grateful to the Chancellor of VIT University, Dr. G Viswanathan, the Vice Presidents, and the Vice Chancellor for their inspiring guidance and support. Their encouragement and provision of essential infrastructure and resources were instrumental in the progress of my research.

I would also like to extend my sincere thanks to Dr. N. Jaisankar, the Dean of the School of Computer Science and Engineering (SCOPE) at VIT University, for his kind words of support and encouragement throughout my research journey. Additionally, I am grateful to my 'HOD's' and classmates for their support in various ways throughout my research work.

Furthermore, I wish to express my profound gratitude to my parents for their enduring sacrifices during my research. Their unwavering support and encouragement have been invaluable to me.

I am truly grateful to all those mentioned above for their contributions to my research, and I feel incredibly fortunate to have had their guidance and support throughout this endeavor.

Place – Vellore
Date- 24-03-25

**Team Members**

**1. Aditya**

**Shriwal**

**2. Aman Chauhan**

**3. Archit Yadav**

Signature of Supervisor

**Table of Contents:**

## INTRODUCTION:

In today's digital era, where travel planning has become increasingly complex due to the vast array of options available, the need for a comprehensive and user-friendly tool has never been more apparent. VIHARA - TOUR PLANNER emerges as a solution to this challenge, offering an all-in-one platform designed to simplify and enhance the travel planning experience. This document outlines the detailed requirements for VIHARA, ensuring that the development team has a clear understanding of the project's scope, functionalities, and technical specifications.
Purpose
The primary purpose of VIHARA is to provide travelers with a seamless, efficient, and personalized travel planning experience. It aims to:

- Streamline Travel Planning: By integrating various travel-related services into one platform, VIHARA reduces the time and effort required to plan a trip.
- Enhance User Experience: Through intuitive interfaces, real-time data integration, and AI-driven recommendations, VIHARA ensures that users can plan their trips with ease and confidence.
- Support Future Development: This document serves as a foundational guide for future developers and maintenance teams, providing insights into the initial design and requirements, facilitating modifications, and ensuring the project's scalability.

To illustrate how VIHARA might be used in real-life situations, consider the following scenarios:

1. The Solo Adventurer:
    a. Scenario: Alex, a solo traveler, wants to explore the less-traveled paths of Southeast Asia. Using VIHARA, Alex inputs preferences for budget travel, outdoor activities, and cultural experiences. VIHARA suggests offbeat destinations like Luang Prabang in Laos, providing detailed information on local treks, accommodations, and weather forecasts. Alex can book a budget-friendly guesthouse and plan a route that avoids tourist-heavy areas, ensuring a unique travel experience.
2. The Family Vacation Planner:
    a. Scenario: The Smith family is planning a summer vacation to Europe. They need a family-friendly itinerary that includes educational visits, amusement parks, and safe accommodations. VIHARA tailors an itinerary that includes visits to the Louvre in Paris, EuroDisney, and a stay in a hotel with family suites. It also provides real-time weather updates, suggesting indoor activities during rainy days, and integrates transport options like family-friendly train routes.

## ABSTRACT:

VIHARA - TOUR PLANNER is an innovative web-based solution designed to revolutionize the travel planning experience. This comprehensive platform integrates various travel-related services to provide users with an all-in-one tool for effortlessly planning their holidays and tours. Leveraging advanced technologies such as Angular for the frontend, Go for backend services, and Firebase for authentication and real-time database management, VIHARA ensures a robust and scalable architecture.

The system offers a wide range of features to enhance the user experience. These include detailed information about popular and offbeat tourist destinations, AI-driven route optimization based on factors like distance, time, cost, and weather conditions, and personalized hotel and accommodation recommendations with seamless booking options. VIHARA also integrates various modes of transport, provides real-time cost estimations for entire trips, and offers weather-based travel recommendations to ensure smooth and enjoyable journeys.

Key functionalities of VIHARA include user registration and login, destination and hotel search capabilities, weather-based travel recommendations, booking services, and user profile management. The platform prioritizes user safety and security, implementing measures such as data encryption, secure authentication methods, and reliable emergency assistance features.

Non-functional requirements focus on ensuring high performance, with quick load times and responsive APIs, as well as scalability to handle a growing user base and expanding feature set. The system is designed to be available 24/7, accommodating users across different time zones and ensuring reliable access to critical services.

VIHARA caters to a diverse user base, including solo travelers, families, and business professionals, offering tailored experiences to meet various travel needs and preferences. By simplifying the travel planning process and providing comprehensive, real-time information and services, VIHARA aims to become an indispensable tool for modern travelers seeking efficient, safe, and enjoyable travel experiences

# SRS

Table of Contents



## 1. INTRODUCTION

VIHARA is a tool for planning your holidays and tour through online by the Customer. It provides the proper management tools and easy access to the customer information.

1.1 Purpose

The purpose of "Vihara" is to provide an all-in-one web-based solution for travellers to plan their trips effortlessly. It will include tourist destination information, optimized travel routes, budget friendly hotel recommendations, booking options, and personalized itineraries.

This SRS for Tour Planner can also be used for future as basis for detailed understanding on how project was started. It provides a blueprint to upcoming new developers and maintenance teams to assist in maintaining and modifying this project as per required changeability.

1.2 Scope of the Project

"Vihara" focuses on providing:

1. Information about popular and offbeat tourist places.
2. Route optimization based on distance, time, and cost.
3. Hotel and accommodation recommendations with booking options.
4. Integration of travel modes (flights, trains, cabs).
5. Cost estimation for the entire trip.
6. Weather based plan change

VIHARA-TOUR PLANNER

7. Suggestions for nearby restaurants and shopping areas.
8. Personalized itineraries based on user preferences.
9. Emergency SOS help provinces.

1.3 Definitions, Acronyms and abbreviations

SRS Software Requirement Specifications
TP Tour Planner
DBMS Database Management System
Blueprint A design technical plan
AI Artificial Intelligence
HTTP/HTTPS Hyper Text Transfer Protocol/Secure
API Application Programming Interface
UI/UX User Interface/ User Experience
FR Functional Requirement
NFR Non-Functional Requirement
VIHARA-TOUR PLANNER

1.5 Overview

The remaining sections of this documentations describes the overall descriptions which includes product perspective and functions, characteristics of users. It also consists of Assumptions, and Constraints. Overall description is listed in section 2. Section 3 includes Specific Requirements which consists of Functional and Non-functional requirements, External Interface Requirements, Software System Attributes, Performance Requirements, Capacity Requirements, Availability Requirements, Safety Requirements and Requirement Traceability Matrix.

## 2 OVERALL DESCRIPTION

2.1 Product Perspective

Vihara is an innovative travel planning solution, leveraging AI/ML algorithms for optimized routes and cost
efficient planning. The product integrates multiple APIs for real-time data retrieval, ensuring seamless user
experience.

2.2 Product Functions

Our Product General functions are:
1. Tourist Place Discovery: Displays a curated list of destinations for selected cities or regions.
2. Route Optimization: Finds the most time and cost-efficient travel route.
3. Hotel Booking: Recommends accommodations based on user preferences (budget, ratings, location).
4. Itinerary Planner: Generates personalized travel plans.
5. Cost Estimation: Provides an estimated budget for the trip.

2.3 User Characteristics
1. Travelers of all ages and interests.
2. Friendly with Solo, Family, Friend.
3. Individuals seeking budget-friendly and customized travel experiences.

2.4 Constraints
1. Internet connectivity is required for full functionality.
2. Integration with external APIs (e.g., travel and hotel booking services).
3. Budget limitations for API usage and hosting services.

2.5 Assumption and Dependencies
1. Users provide accurate input regarding preferences.
2. APIs for hotel and travel services are reliable and provide updated data.
3. The application is used on modern browsers with JavaScript enabled.
If incase of any difficulties, SRS should be flexible enough to change accordingly.

**3 Specific Requirement**

3.1. External Interface Requirements

These define how the system will interact with external components such as APIs, users, and

third-party services.
3.1.1. User Interface (UI) Requirements:
• The website must be intuitive and user-friendly, allowing users to perform key functions such as
searching, booking, and payments easily.
• Responsive Design: The UI must adapt seamlessly to various screen sizes (mobile, tablet, desktop).
• Accessibility: The website must comply with WCAG 2.0 standards, ensuring that users with disabilities
can access the website. This includes screen reader support, high-contrast modes, and keyboard navigation.
• Login/Register Pages: Secure login and registration forms with clear input validation for email, password strength, etc.
3.1.2. Third-Party API Integration Requirements:
• Weather API (e.g., Gemini API): The system must fetch real-time weather data for specific destinations
and modify travel plans accordingly.
• Transport Services API: Integration with flight, train, or car rental services for real-time transport
information.
• Map Integration (Google Maps API): To show the user their current location, hotel locations, nearby
emergency services, and other points of interest.
3.2. User Requirements
These specify how different types of users will interact with the system, what roles they will have, and what
functionalities are required.
3.2.1. User Roles and Access:
• Customer (Family, Solo, Friend, Couple):
o Can search for destinations and hotels based on preferences.
o Can book hotels and transport options.
o Receives weather-based travel recommendations.
o Can use emergency SOS feature for assistance.
3.3. Software Requirements
This outlines the software technologies and services necessary to build and run the travel tour website.
3.3.1. Frontend Software Requirements:
• Framework: Angular (for UI/UX, responsive components).
• HTML/CSS/JavaScript: For creating responsive and interactive web pages.
• FontAwesome: For adding icons and user-friendly design elements.
• API Integration Libraries: Axios or Fetch API for interacting with third-party APIs like weather,
payment, and transport.

3.3.2. Backend Software Requirements:
• API Development: Go (for API services to handle user data, bookings, and communication with third

party APIs).
• Database: MongoDB or PostgreSQL (to store user information, bookings, hotels, destinations, etc.).
• Authentication & Authorization: JWT for secure user authentication and role-based access control.
• Weather-Based Travel Plan Logic: Python/TensorFlow (optional) for recommendation systems based
on weather, budget, and user preferences.
3.4. Communication Requirements
This section outlines how data is transmitted and secured between components and external entities.
3.4.1. Client-Server Communication:
• Protocol: HTTPS (Secure HTTP) for all communications to ensure encryption of user data, especially
during login, booking, and payment.
• Real-Time Data Transfer: Use of WebSockets or RESTful APIs to fetch real-time weather updates and
transport information.
3.4.2. Third-Party API Communication:
• Weather API: Secure API key-based authentication for accessing weather data (e.g., using the Gemini
API).

3.4.3 Internal System Communication:
• Backend to Database: Secure connection between backend (Go services) and the database (MongoDB/PostgreSQL) using SSL/TLS to prevent data breaches.
• Frontend to Backend: All communications between the frontend (Angular) and backend (APIs) should
be over secure channels using HTTPS.

**4. Functional Requirements**
These define the primary operations and processes that your system will perform. Each functional requirement
corresponds to user tasks, features, and system behaviors.
4.1. User Registration and Login:
• FR1.1: The system must allow users to register by providing their email, password, name, and other
optional details (e.g., phone number).
• FR1.2: The system must send a confirmation email for user account verification.
• FR1.3: Users must be able to log in using their registered email and password.
• FR1.4: The system must provide password recovery functionality for users who forget their passwords.
• FR1.5: User authentication must be secure using JWT tokens to allow authorized access.
4.2. Search for Destinations and Hotels:
• FR2.1: Users must be able to search for destinations by providing input such as location, travel dates, and
preferences.
• FR2.2: The system should display available hotels and tour packages based on user preferences and real
time data.
• FR2.3: The system should allow filtering based on price range, type of accommodation, and amenities.

• FR2.4: Weather data (integrated through an API) should be factored into search results, recommending
alternate travel plans if needed due to adverse weather conditions.
4.3. Weather-Based Travel Recommendations:
• FR3.1: The system should fetch real-time weather data from the weather API (e.g., Gemini API) based on
the destination.
• FR3.2: If severe weather conditions are detected, the system should recommend alternative travel dates or
nearby destinations.
• FR3.3: Users must be able to access detailed weather forecasts as part of their travel plan.
4.4. Booking:
• FR4.1: Users must be able to book hotels, transport, and tour packages through the website.
4.5. User Profile Management:
• FR5.1: Users should be able to update their personal information, including name, email, phone number,
and password.
• FR5.2: Users must have access to their booking history and the ability to cancel or modify future
bookings.
5. Non-Functional Requirements
These specify the quality attributes of the system and how the functional requirements will be achieved in terms
of safety, security, performance, capacity, availability, and more.
5.1. Safety Requirements:
• NFR1.1: The system must ensure that user data and transactions are handled securely, protecting sensitive
information (e.g., payment details) from leaks or breaches.
• NFR1.2: The SOS feature must function reliably in emergency situations, ensuring that user location and
distress signals are transmitted accurately. We are giving only recommendation.
5.2. Security Requirements:
• NFR2.1: All user data must be encrypted during transmission using HTTPS/SSL, especially during login,
booking, and payment processes.
• NFR2.2: Sensitive user data (e.g., passwords, payment details) must be securely stored using encryption
algorithms (e.g., AES-256).
• NFR2.3: The system must prevent unauthorized access through the use of secure authentication methods,
such as JWT-based tokens.

5.3. Performance Requirements:
• NFR3.1: The website should load within 3 seconds for users on a standard broadband connection.

• NFR3.2: The backend API should respond to user requests (e.g., searching for hotels or processing
payments) within 2 seconds on average.
• NFR3.3: The system must support real-time weather updates and fetch weather data in less than 1
second for dynamic travel recommendations.
5.4. Capacity Requirements:
• NFR4.1: The database must support up to 1 thousand user profiles and their associated booking history.
• NFR4.2: The system should handle up to 1,000 hotel and destination listings across various regions.

5.5. Software Attributes Requirements:
• NFR5.1: The system must be scalable, allowing it to expand to accommodate more users, destinations, or
features as the business grows.
• NFR5.2: The system should be built on a modular architecture, ensuring that new features (e.g., additional APIs or services) can be added without major disruptions.
• NFR5.3: The platform should be compatible across browsers (e.g., Chrome, Firefox, Safari) and
devices (e.g., desktops, tablets, smartphones).
• NFR5.4: The codebase must be maintainable, following best practices for clean, documented code, to
allow future developers to easily update or troubleshoot the system.
5.6. Availability Requirements:
• NFR6.1: The system must be available 24/7 to accommodate users across different time zones.

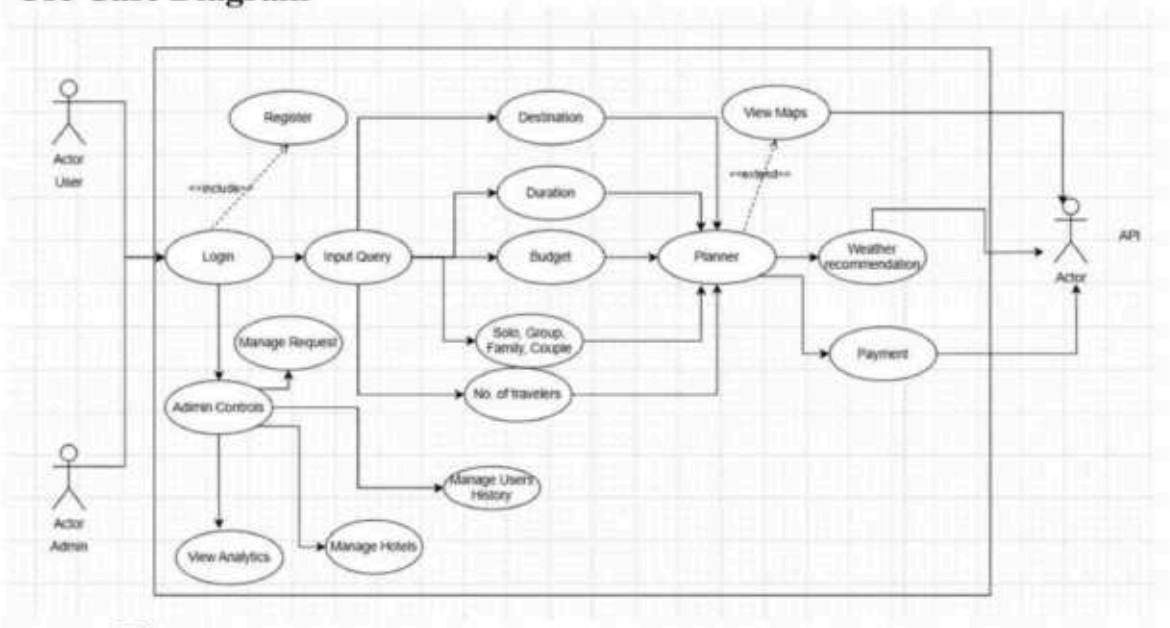
**6.Requirement Traceability Matrix**

The Requirement Traceability Matrix (RTM) reflects the correlation between Non-Functional Requirements (NFR) and Functional Requirements (FR). The RTM is a documentation that associates the requirements entirely throughout the validation process. Traceability is regarded to be one of the most important considerations for tracing the requirements.
In the table below we will be tracing the relation between Functional Requirements and Non Functional Requirements.

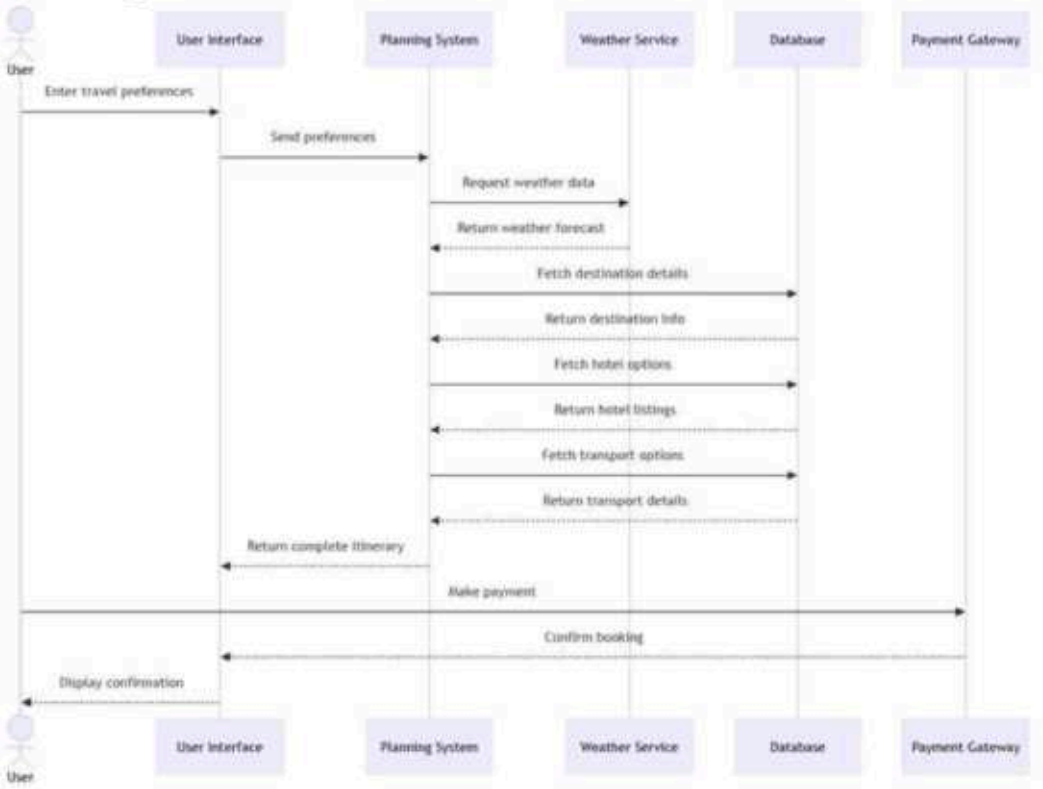
# Design

## a. Low level

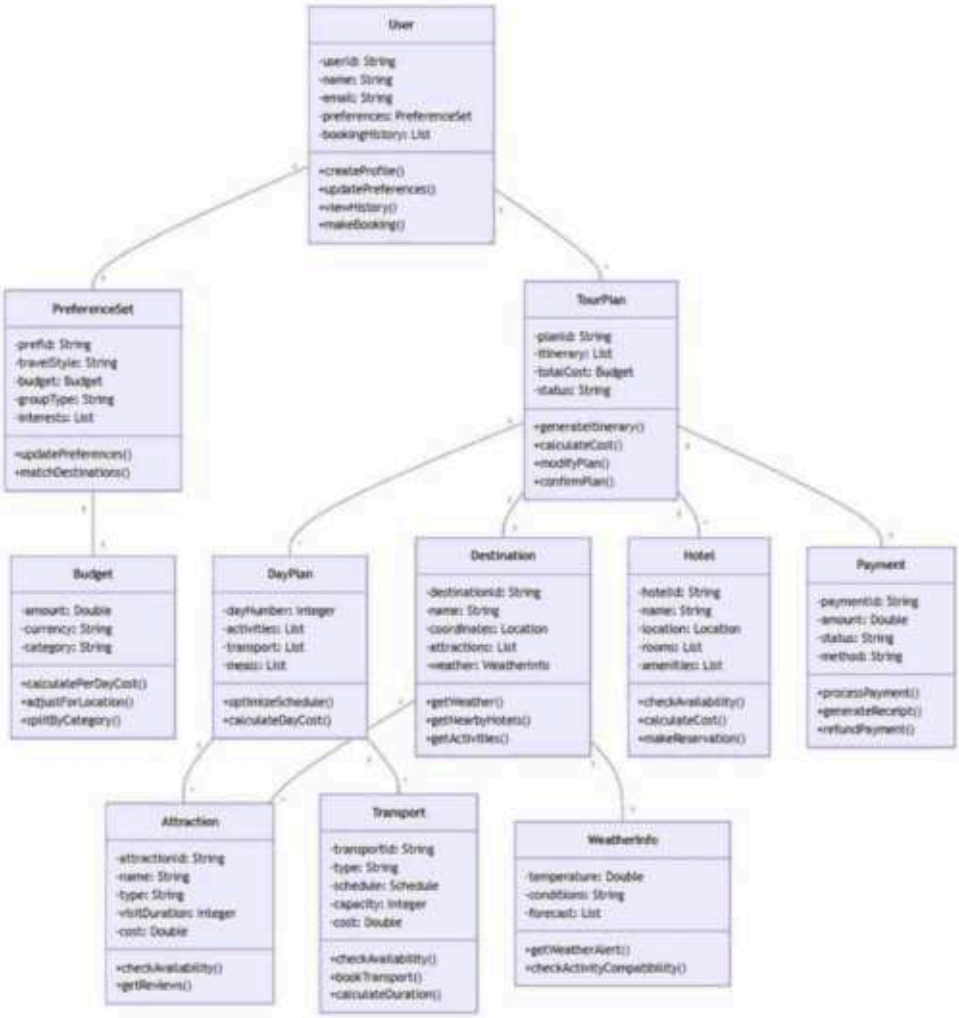
### Use Case Diagram



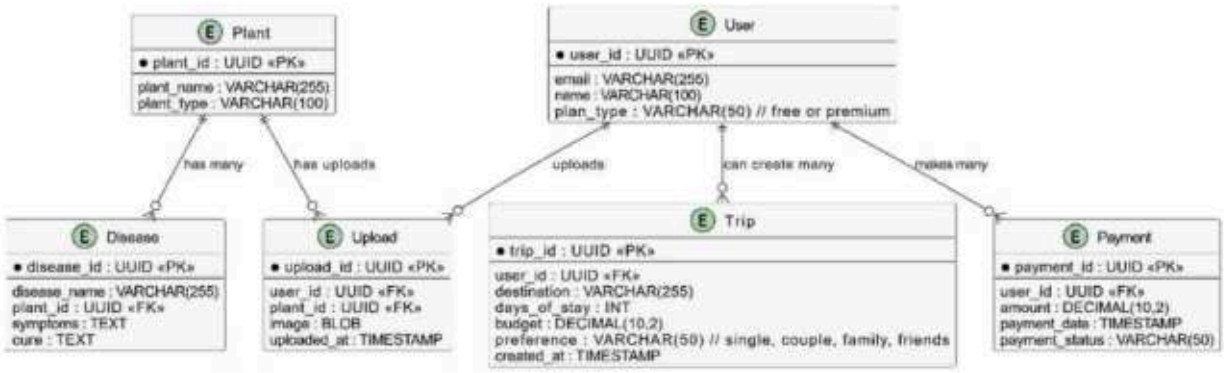

## Sequence Diagram



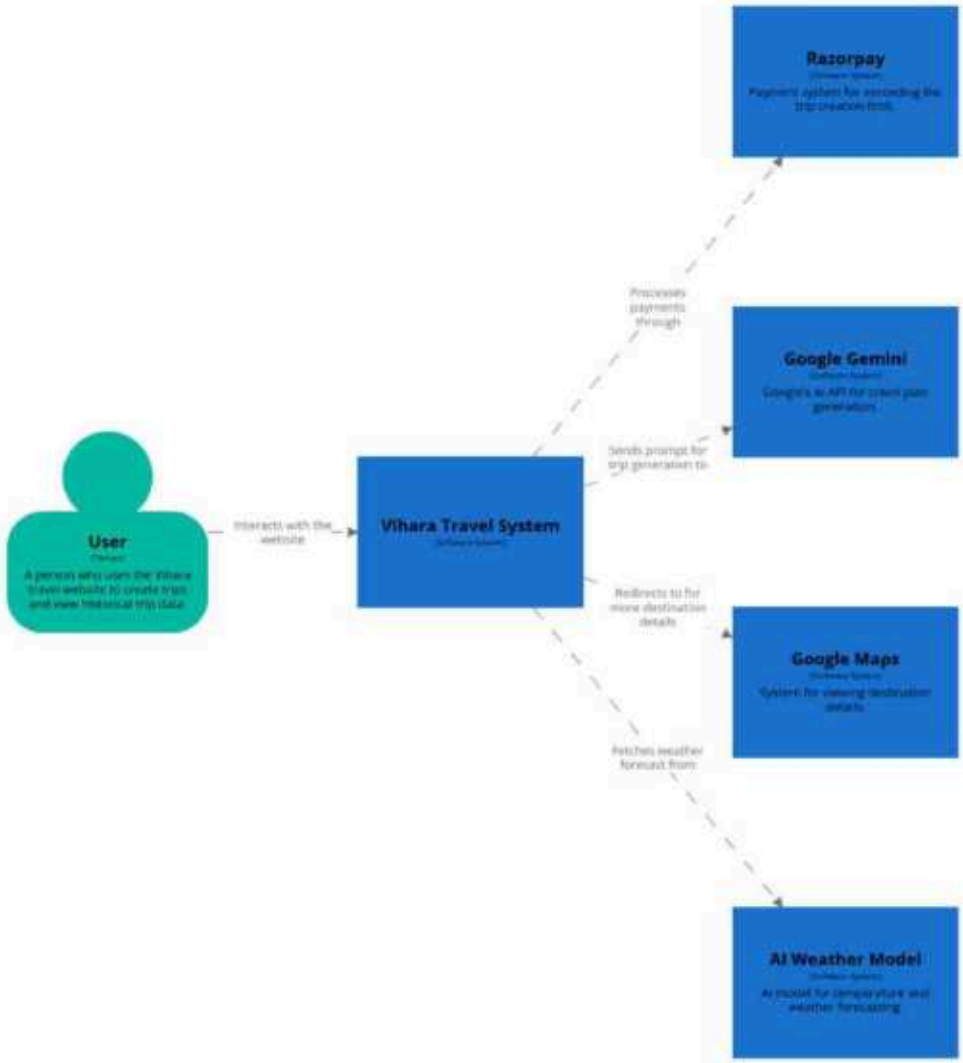

Class Diagram




## b. High Level Database Design:

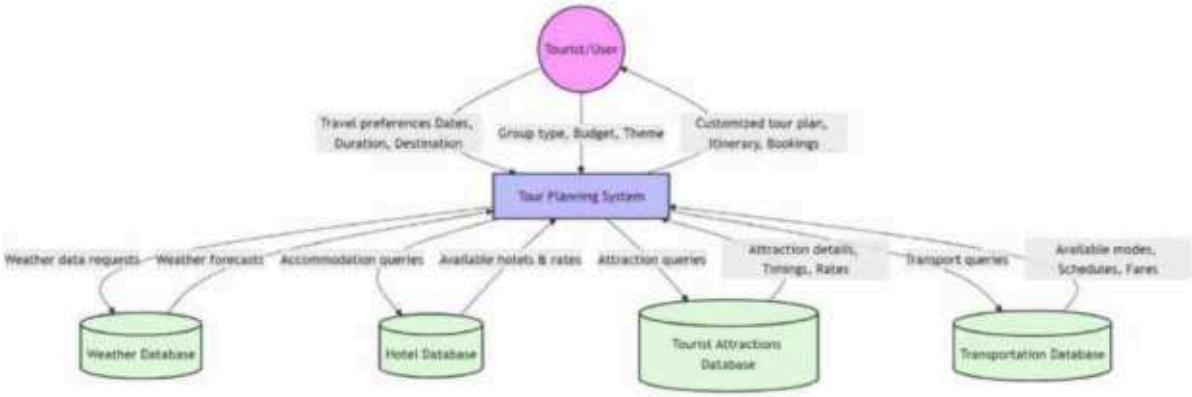



## System Overview:




[System Context] Vihara Travel System

Data Flow Diagram:

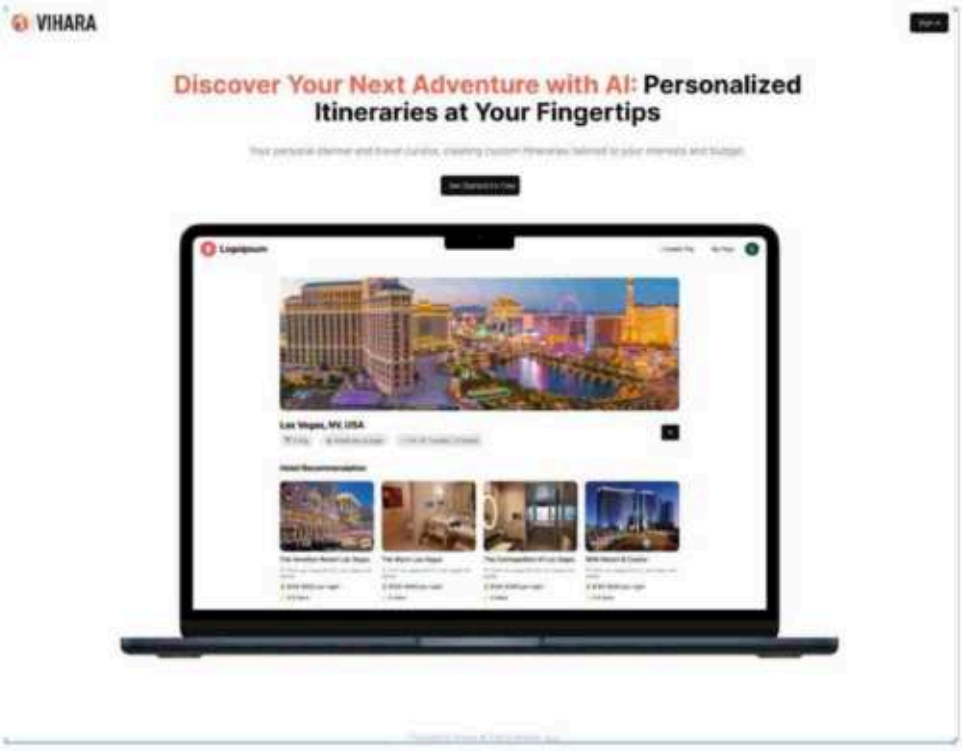



## c. ER and UI/UX

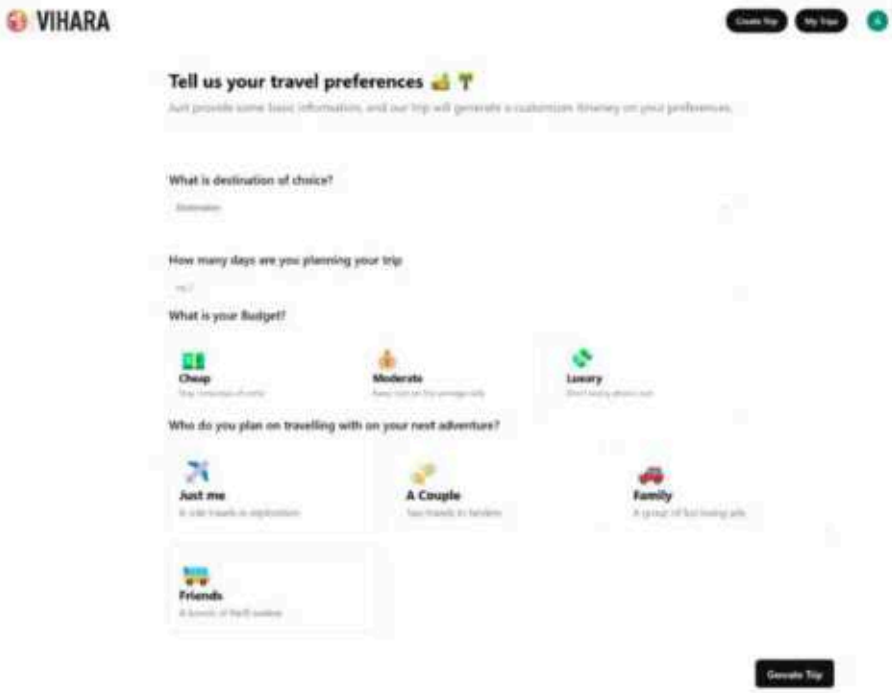
## ER Diagram


## UI AND UX DIAGRAM

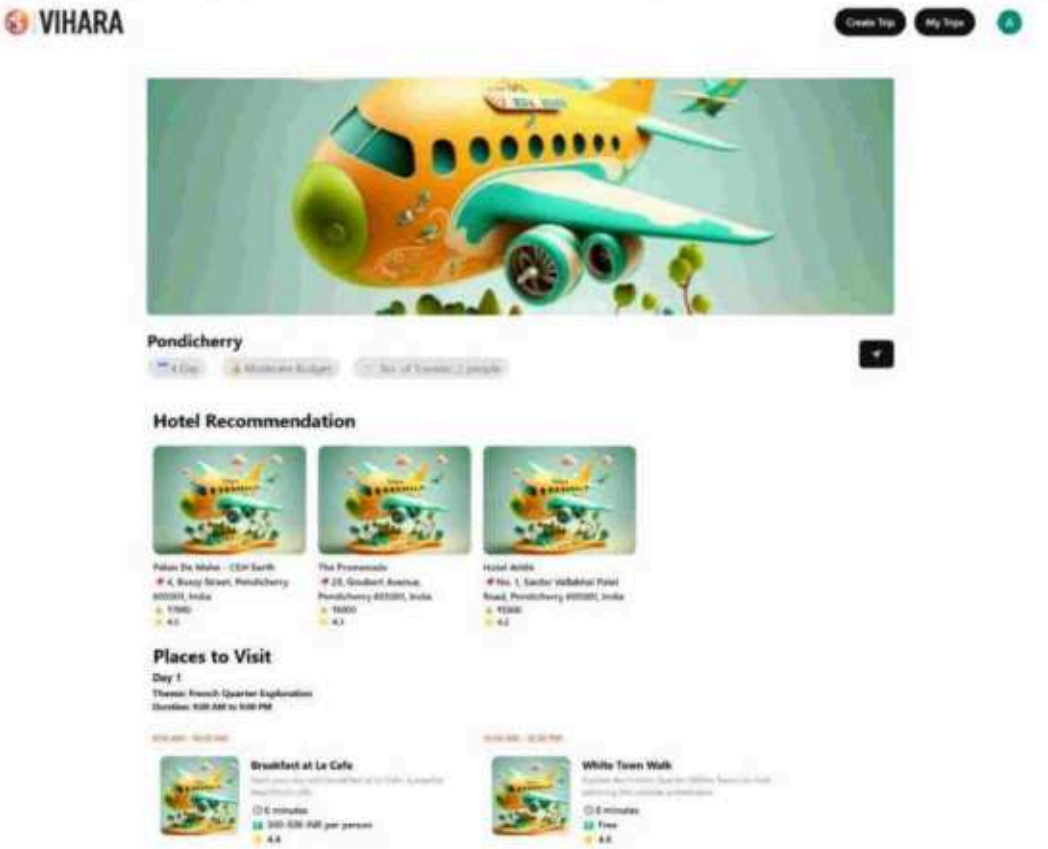
### 1. Login UI



## 2. Creating plan UI




## 3. Trip Plan UI



## ALGORITHM – CODE:

### 1. Login Authentication

- **User Clicks Login**: The user clicks a "Login with Google" button on your website.

- **Redirect to Google**: The website sends the user to Google's login page, where they can give permission to the app to access their Google data (like their email or profile).

- **User Grants Permission**: The user logs into their Google account and grants permission to the app to access their information.

- **Authorization Code**: If the user approves, Google sends the website a special code (called an authorization code) as a way to confirm that permission has been granted.

- **Exchange Code for Token**: The website sends this code to Google, which in return gives the website an access token (like a digital key) to access the user's data.

- **Access User Data**: The website uses the access token to get the user's profile information (like their name and email) from Google.

- **Log the User In**: The user is logged into the website with their Google account information.

```
const login = useGoogleLogin({
  onSuccess: (codeRes) => GetUserProfile(codeRes),
  onError: (error) => console.log(error),
});

//User profile after authorization
const GetUserProfile = (tokeninfo) => {
  axios
    .get(
      `https://www.googleapis.com/oauth2/v1/userinfo?access_token=${tokeninfo?.access_token}`,
      {
        headers: {
          Authorization: `Bearer ${tokeninfo?.access_token}`,
          Accept: "Application/json",
        },
      }
    )
    .then((res) => {
      console.log(res);
      localStorage.setItem("user", JSON.stringify(res.data));
      setOpenDialog(false);
      onGenerateTrip();
    });
};
```

**2. User Creating Trip**

a. Destination name.
b. How many day user want to plan your trips.
c. What is budget in trip.
d. How do you plan travelling with on you next adventure.

**3. View Trip and AI Prompt**

**a. AI Prompt**

```
toast("Please wait we are generating your trip, Happy Travelling");
const FINAL_PROMPT = AI_Prompt.replace("{location}", formData?.Destination)
  .replace("{totalDays}", formData?.noOfDays)
  .replace("{traveler}", formData?.Traveler)
  .replace("{budget} ", formData?.Budget)
  .replace("{totalDays}", formData?.noOfDays);
console.log(FINAL_PROMPT);
const result = await chatSession.sendMessage(FINAL_PROMPT);
console.log("Creation: ", typeof result.response?.text());
setloading(false);
saveAITrip(result.response?.text());
};
```

**b. View Trip**

```
function PlaceCardItem(place, index) {
  // console.log("PlaceCard ", place);

  const [photoUrl, setPhotoUrl] = useState();
  useEffect(() => {
    place && GetplacePhoto();
  }, [place]);
  const GetplacePhoto = async () => {
    const data = {
      textQuery: place?.place?.placeName,
    };
    const result = await GetPlacesDetails(data).then((resp) => {
      console.log(resp.data.places[0].photos[3].name);
      const PhotoUrl = PHOTO_REF_URL.replace(
        "{NAME}",
        resp.data.places[0].photos[3].name
      );
      console.log(PhotoUrl);
      setPhotoUrl(PhotoUrl);
    });
  };
```

### 4. View History

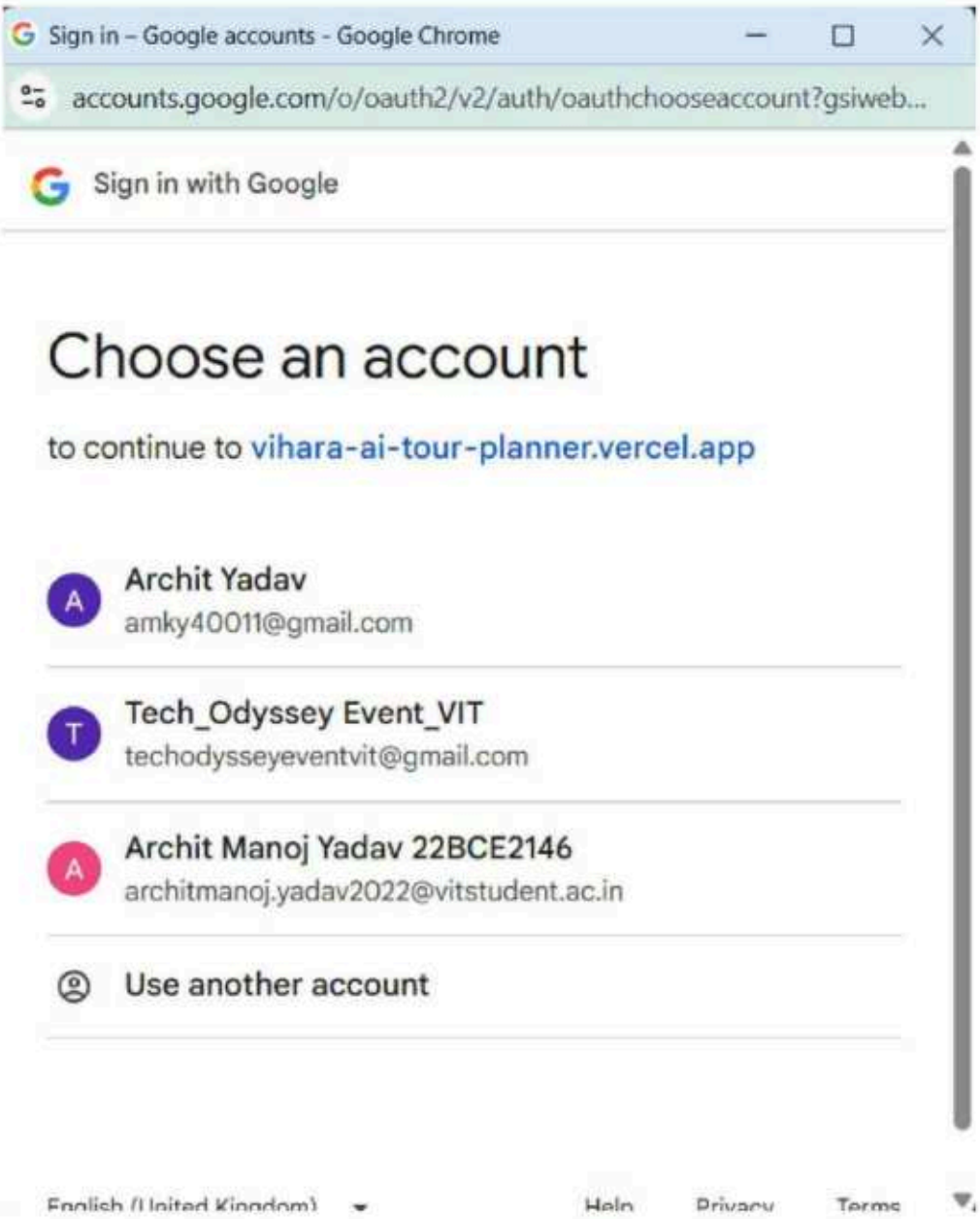
```
const GetUserTrips = async () => {
  const user = JSON.parse(localStorage.getItem("user"));

  if (!user) {
    navigation("/");
    return;
  }
  const q = query(
    collection(db, "AITrips"),
    where("userEmail", "==", user?.email)
  );
  const querySnapshot = await getDocs(q);
  setUserTrips([]);
  querySnapshot.forEach((doc) => {
    // doc.data() is never undefined for query doc snapshots
    //   console.log(doc.id, " => ", doc.data());
    setUserTrips((prev) => [...prev, doc.data()]);
  });
```

### 5. Payment Gateway

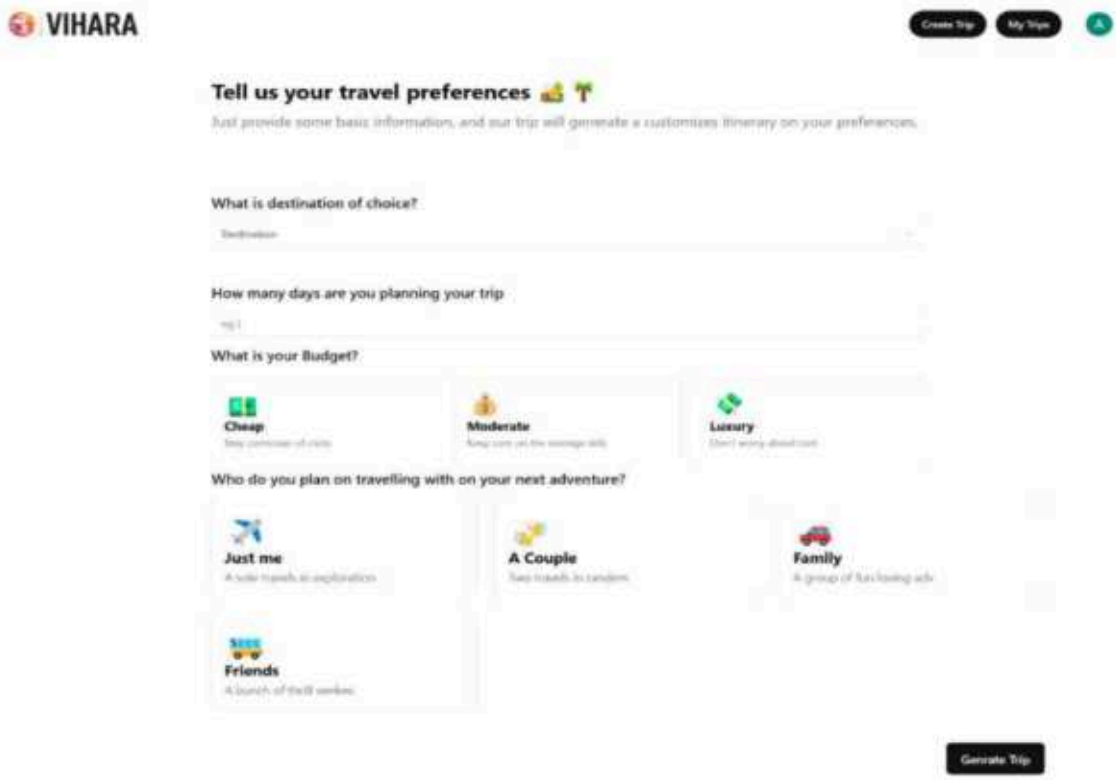
```
if (tripCount >= 2) {
  toast("Free trial finished. Purchase premium.");
  var options = {
    key: "rzp_test_vv1FCZvuDRF61Q",
    key_secret: "P4JAUwn4VdE6xDLJ6p2Zy8RQ",
    amount: 1,
    currency: "INR",
    name: "VIHARA",
    description: "for testing purpose",
    handler: function (response) {
      const paymentId = response.razorpay_payment_id;
      console.log("paymant id", paymentId, shipping_address);
    },
    theme: {
      color: "#07a291db",
    },
  };
  var pay = new window.Razorpay(options);
  pay.open();
  return;
}
setloading(true);
```

**IMPLEMENTATION:**

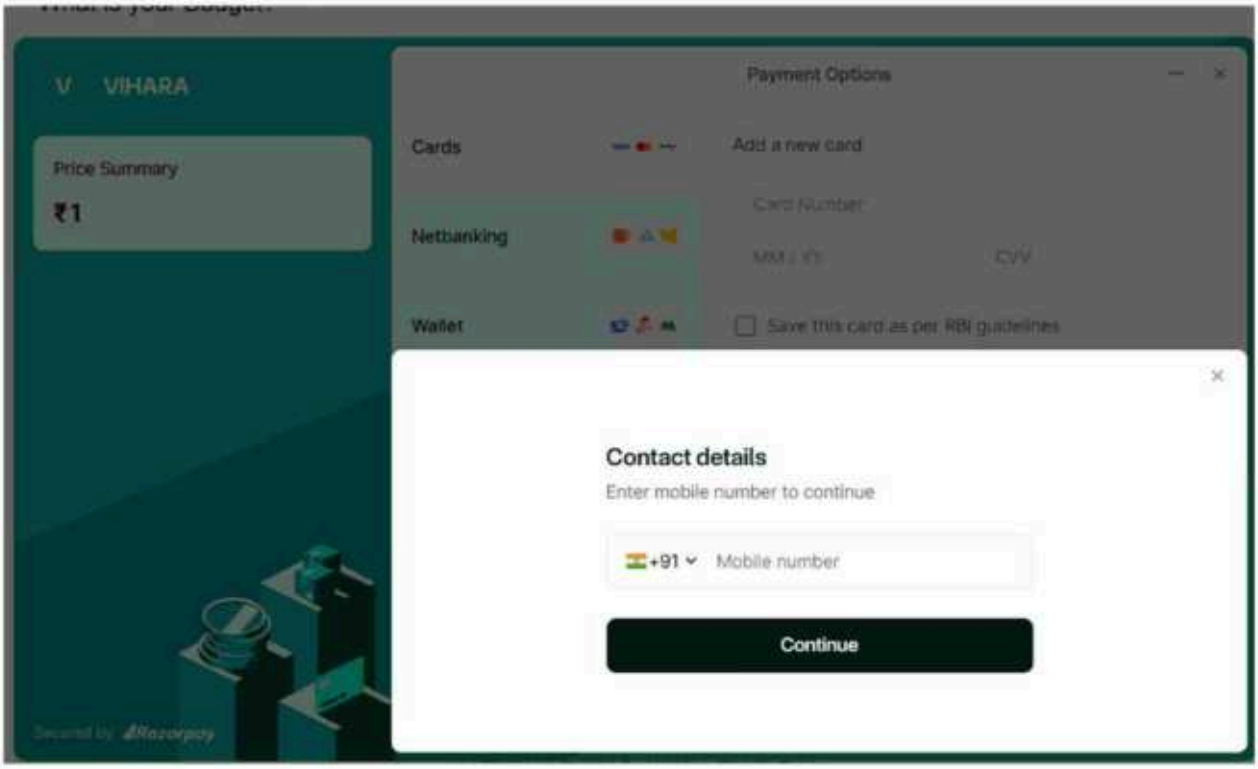
1. **Login**



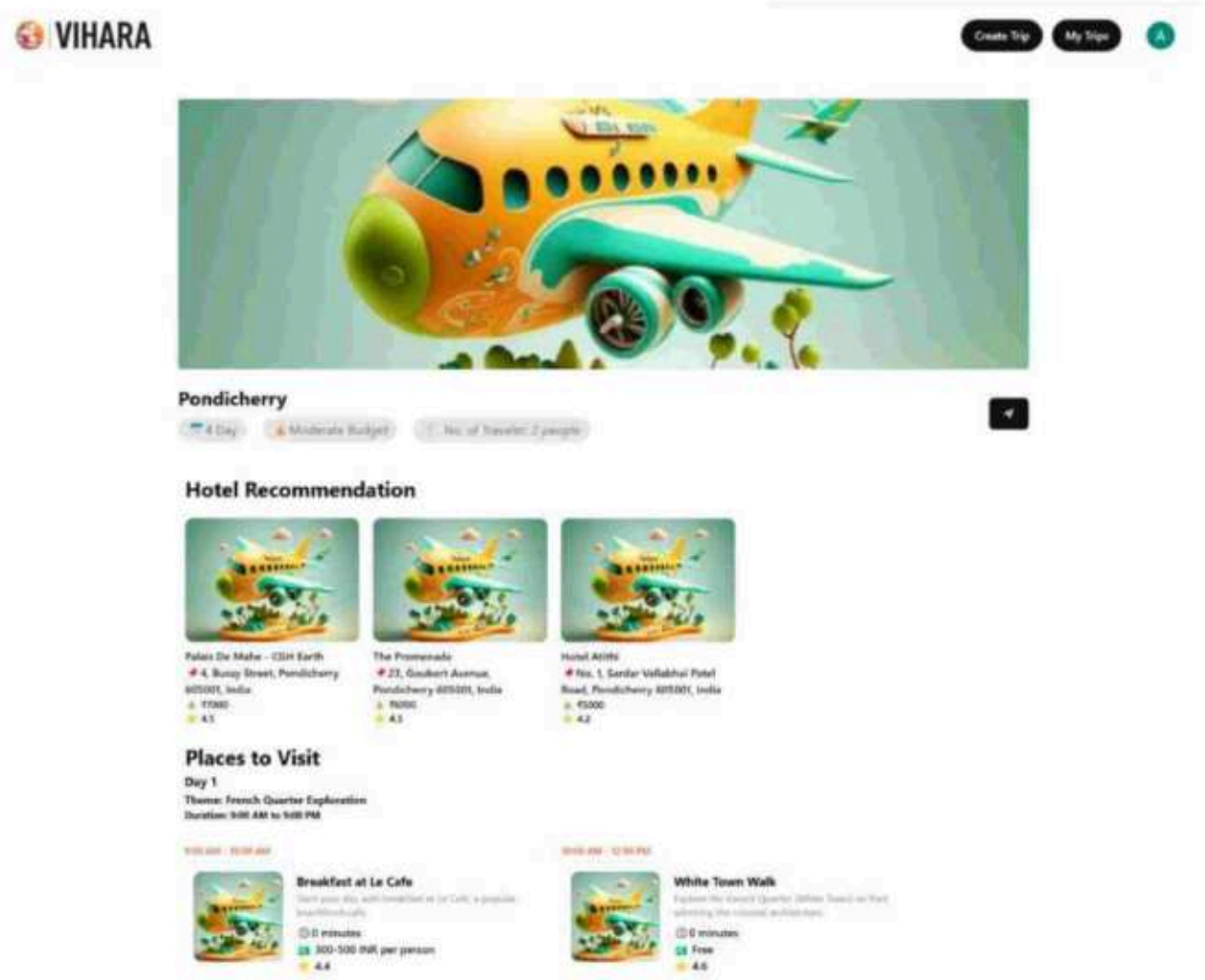

2. **Creating Trip**



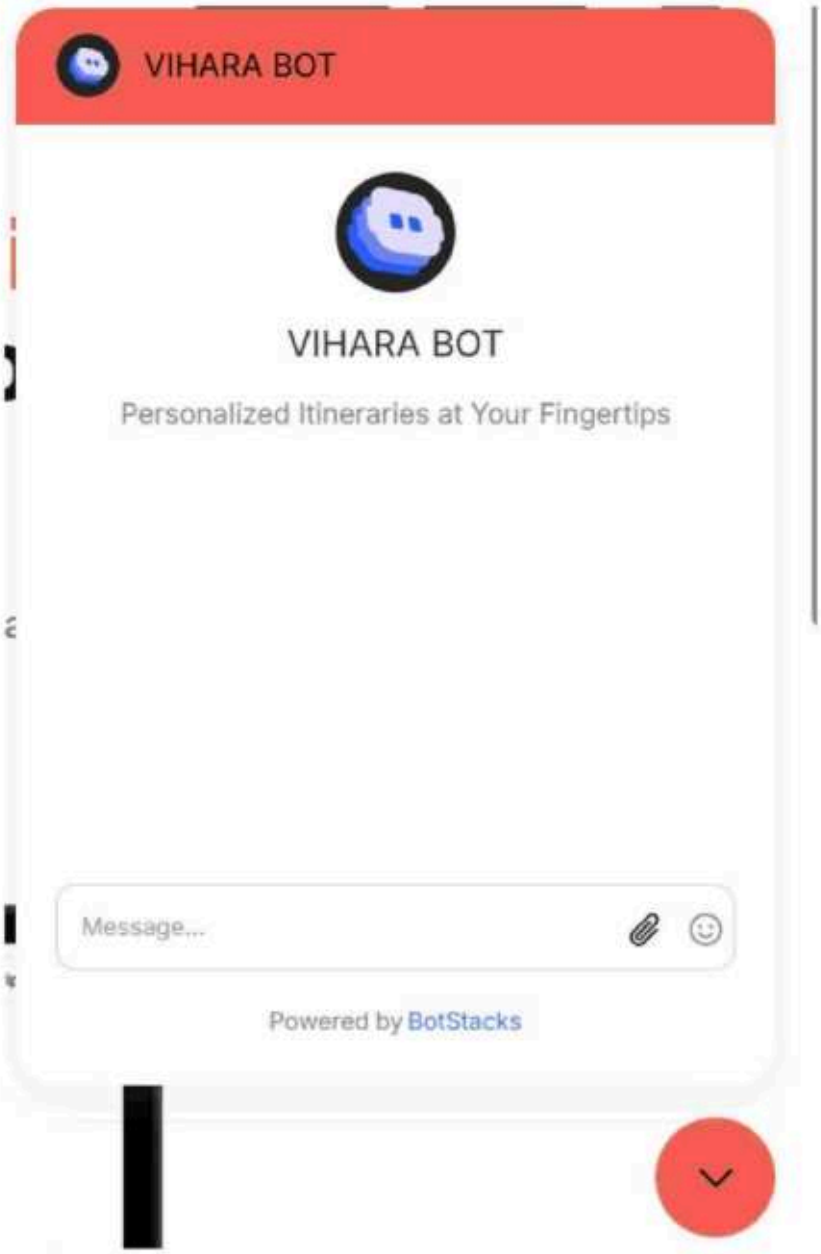

## 3. Payment Gateway



## 4. Trip Plan




## 5. Chat Bot




6. **Weather Forecast**

## Ahmedabad

2 Day | Moderate Budget | No. of Traveler: 1 person

### Weather in Ahmedabad

| Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep |
|---|---|---|---|---|---|---|---|---|
| 5/24°C | 3/26°C | 6/31°C | 6/35°C | 10/38°C | 15/41°C | 18/41°C | 19/42°C | 17/41°C |
| AQI 23 | AQI 19 | AQI 12 | AQI 11 | AQI 20 | AQI 20 | AQI 19 | AQI 13 | AQI 19 |

**TESTING (Test document as prepared in DA)**

# Software Engineering

## Theory DA - 1

Name : Aditya Shrival

Reg no. : 22BCE2132

Course Code : BCSE301L

Slot : G2+TG2

Faculty : Dr. S. Yoganand

Class : SJT 604

Date : 6/3/2025

1] Prepare the test document with the following requirements.

ⓐ Scope
ⓑ Objective
ⓒ Test environment
ⓓ Test tool
ⓔ Test Strategy
ⓕ Test Scenarios with Test cases (use the format given in class)

→ ⓐ Scope →

The scope of this test document covers the comprehensive testing of the Vihara-AI Tour planner web application (https://vihara-ai-tour-planner.vercel.app/). The testing will focus on functional, usability and performance aspects of the web application across different devices and browsers. The application leverages Artificial Intelligence to assist users in planning personalized travel itineraries. The scope encompases functional security and compatibility testing as well to ensure the application meets it requirements and delivers a seamless user experience.

ⓑ Objective →

The primary objectives of this testing are to:

1. Verify the functional integrity of all features in the Vihara AI tour planner.
2. Ensure a smooth and intuitive user experience.
3. Validate the responsiveness and performance of the application.
4. Identify and document any potential bugs or fixes or issues
5. Confirm the application meets the specified requirements and user experience.

The objective is to validate that the Vihara AI Tour planner functions as intended, providing accurate and personalized travel planning services. This includes verifying that AI-generated itineraries align with user preferences, the integration with external services like (google maps) operates correctly, and the application maintains performance standards under various conditions.

© Test environment →

1. closed Testing environment / local host for development and testing purpose.
2. Controlled environment for fewer external dependencies which are responsible for API (Application program Interface).
3. Testing in different Hardware configuration
   - ⓐ Desktop Computers
     - Windows 10/11 system
     - MacOS system
     - Linux System
   - ⓑ Mobile Devices
     - IOS smartphones and tablets.
     - Android smartphones.
4. Testing in different Software configuration
   - ⓐ Browsers
     - Google chrome (Latest version)
     - Mozilla Firefox (Latest version)
     - Safari (Latest version)
     - Microsoft Edge
     - Opera
   - ⓑ Operating System
     - Windows 10/11
     - mac OS
     - IOS
     - Android
     - Linux distribution
5. Network Condition
   - High-speed broadband
   - 4G/5G
   - For slow network condition.

## ⓓ Test tool

① Selenium: Automated function Testing [Selenium Webdriver]

② Jest : Javascript testing framework to verify code in JS is correct.

③ Chrome devtools: set of web developer tools to diagnose problems and check network, security issues.

④ React testing library - A light weight solution for testing react components.

⑤ Postman API - A tool that tests document API with regression for proper communication between client and server.

⑥ Google Lighthouse - For performing Testing with different scenarios.

⑦ JMeter - An application (open source software) to test load test functional behaviour and measure performance.

⑧ Jira Test case - Agile software testing management app that helps you to test, plan, track and relase software.

⑨ Gitissues - To track the bugs in the versions handled by version control (git).

⑩ OWASP ZAP - For security Testing that identifies security vulnerabilities.

## ⓔ Test Strategy

↳ ① Manual Testing - It is crucial for validating the functionality, usability, performance, and security of Vihara AI Tour planner web application.

This strategy outlines the step-by-step manual testing approach to ensure that the Application meets the requirements and deliver a smooth user experience.

ⓐ Functional Testing →

- Objective: Verify that all features work as expected and meet the functional requirements.
- Approach: (i) Execute test case manually by simulating user interactions.
  (ii) Validate input fields, buttons, links, and form submissions.
  (iii) Verify navigation between pages and consistency in data flow.
- Test coverage: (i) User Registeration & Authentication
  (ii) Itinerary generation.
  (iii) External Service Integration.

ⓑ Usability Testing →

- Objective: Ensure a seamless and intuitive user experience.
- Approach: (i) Conduct exploratory testing to check UI/UX elements.
  (ii) Evaluate design consistency, readability, and ease of navigation.
  (iii) Collect feedback from real users and adjust based on their experience.
- Test coverage: (i) Navigation and Accessibility
  (ii) Visual experience.

ⓒ Performance Testing →

- Objective: Measure application responsiveness under various conditions.
- Approach: (i) Manually test the response time of key features
  (ii) Test performance under normal and slow network conditions.
- Test coverage: (i) Page Load Time
  (ii) System Responsiveness.

ⓓ Security Testing →

- Objective: Identify potential vulnerabilities and ensure data protection.
- Approach: (i) Attempt SQL injection and XSS attacks using manual test inputs.
  (ii) Verify session handling
  (iii) Test password strength validation.
- Test coverage: Authentication Security.

## (e) Compatibility Testing

- Objective: Verify that the web application works on different devices, OS.
- Approach: (i) Manually test the application across multiple browsers.
  (ii) Verify the responsiveness on different screen sizes.
  (iii) Test different OS environments
- Test coverage: (i) Browser compatibility
  (ii) Device compatibility.

## (f) Test Scenarios with Test cases

Scenarios

(1) User Registeration and Authentication

| S.no | Scenarios | Test Case | Test Case Description | Actual Results | Expected Result | Status (Pass/Fail) |
|---|---|---|---|---|---|---|
| 1. | User Registeration and Authentications | 1.1 | Verify that user can register with valid information | Registeration Successful | User successfully registers. | Pass |
| | | 1.2 | Ensure that the login functionality works with correct credentials | Login successful | User logs in successfully | Pass |
| | | 1.3 | Check that appropriate error message display for invalid login attempts | Error message displayed: "Invalid credentials" | Error message appears for invalid login | Pass |
| 2. | Itineary Generation | 2.1 | Validate that users can input travel preferences and receive tailored itineraries | Itineary generated correctly | AI generates a suitable itineary. | Pass |

| S. no. | Scenarios | Test Case | Test case Description | Actual Results | Expected Results | Status (Pass/Fail) |
|---|---|---|---|---|---|---|
| | | 2.2 | Ensure that the AI provides alternative suggestions when initial options unavailable | No Alternative suggestions were displayed. | AI suggests alternative options | Pass |
| | | 2.3 | Verify that accuracy of information in the generated itineraries. | Correct locations details | Itinerary details are correct and relevant | Pass |
| 3 | Integration with External services | 3.1 | Confirm that the application integrates seamlessly with Google maps for location | Google Maps loaded correctly | Google maps display correctly | Pass |
| | | 3.2 | Ensure the external data is fetched and displayed correctly. | Weather data displayed, but local events were missing | Data is displayed correctly in the UI. | Pass |
| 4 | Performance Under Load | 4.1 | Assess application performance with multiple simultaneous user. | Slight delay observed but no crash | No crashes, smooth performance | Pass |
| | | 4.2 | Evaluate response times during peak usage. | Response time in range | Acceptable response time under high load | Pass |
| 5 | Security Measures | 5.1 | Verify that user data is encrypted during transmission and storage | Data encryption verified | Data is encrypted securely | Pass |
| | | 5.2 | Check for vulnerabilities to common security threats such as SQL injection | Application vulnerable to SQL injection | Application is secure against threats | Fail |
| 6 | User Interface and usability | 6.1 | Ensure that the application layout is responsive across different device and screen sizes. | UI adapted correctly on mobile and desktop | UI adapts well to various devices | Pass |
| 7 | Compatibility Across Platforms | 7.1 | Test functionality on various operating systems and browser combinations | Application worked well in all browsers | Application runs smoothly on all tested platforms | Pass |

2] Why Configuration management is necessary for Software Engineering. Justify and explore the necessary tools used for the Same.

→ Configuration Management (CM) is a critical aspect of Software engineering that ensures Software systems are developed, deployed, and maintained efficiently. It provides structured methodologies.

↳ Providing structured methodologies to manage changes systematically, control Software versions, and enhance collaboration among teams.

↳ CM ensures Software integrity, minimize risks, and enable seamless deployment in complex environments.

• Reasons why Configuration management is necessary are ↓

① Version Control → Software evolves over time as new features are added and bugs are fixed. Configuration management helps track changes and manage multiple version effectively.

Key benefits

- Maintains a history of changes: Helps track of who made changes, what modifical.
- Rollback capability: If a new update introduces a bug, the system can revert to a previous stable version
- Parallel development: Supports multiple branches for different features.

Example:

↳ A Software developer team working on a new feature can create a seperate branch in Git. Once the feature is complete and tested, it son be merged into the main branch without affecting the existing stable branch/version.

② Ensuring Consistency and Stability

↳ Software development involves multiple stages, including development, testing, staging and production. CM ensures that the software remains consistent across these stages by:

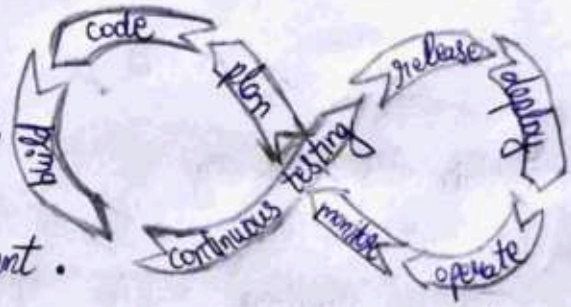
- Maintaining the same configurations across different environments.
- Preventing discrepancies between development and production system.
- Reducing errors due to mismatched configurations.

Example:

↳ A web application may work perfectly in a developer's local environment but fail in production due to missing dependencies. CM tool ensures that all environments have the same configurations.

③ Automated Deployment & Continuous Integration (CI/CD)

↳ Manual software deployment is error-prone and time-consuming. CM enables continuous integration and continuous deployment (CI/CD) to:




- Automate software build, test and deployment.
- Reduce human errors by using standardized deployment scripts.
- Ensure faster and more ~~de~~ reliable software releases.

Example:

↳ A CI/CD pipeline in Jenkins can automatically test and deploy new code changes to a staging environment before pushing to production. This reduces downtime and ensures a smooth release process.

④ Enhancing Collaboration Among Teams

↳ Modern software development involves multiple teams working on different aspects of a project. CM ensures smooth collaboration by:

- Allowing multiple developers to work on the same codebase without conflicts.
- Providing a shared repository where updates are tracked and synchronized.
- Managing dependencies and configuration across teams

Example:

In a large-scale enterprise software project, backend developers, frontend developers, and database administrator need to coordinate their work. Configuration management ensures that all teams work with synchronized versions of code and dependencies.

⑤ Disaster Recovery & Business Continuity

↳ Software failures, hardware crashes, or cyberattacks can lead to data loss and service disruptions. CM supports disaster recovery by:

- Maintaining backups of configurations and software versions.
- Enabling minimal downtime by restoring services efficiently.

Example:

If a cloud-based e-commerce platform experiences a failure after an update, configuration management allows rolling back to previous version.

⑥ Managing Infrastructure and Deployment Environments.

↳ With the rise of Infrastructure as Code (IaC), configuration management extends beyond software to IT infrastructure. Tools like Terraform, Ansible and Kubernetes help manage cloud resource, servers and containers effectively.

- Automate infrastructure provisioning and scaling.
- Ensures consistency in cloud-based deployments.
- Reduces manual configuration errors.

Example: A company using AWS can define infrastructure in Terraform scripts, ensuring that all deployments maintain consistent configurations across multiple environments.

⑦ Supporting Agile and DevOps practices.

↳ Modern software development methodologies like Agile and DevOps rely on fast iterations and continuous delivery. Configuration management :

- Reduces deployment times and improve efficiency.
- Support collaboration between development and operation teams.
- Enhances scalibility by automating infrastructure and configurations.

Example : A DevOps team using Docker and Kubernetes ensure that microservices based applications run consistently across development, testing, and production environments.

## Necessary Tools for Configuration Management

① Git - (Version Control System)

- Most widely used distributed version control system.
- Supports branching, merging, and rollback.
- Used in collaborative software development with remote repositories

↳ Performance metrices

- Near - instantaneous branching
- Directed acyclic graph (DAG) structure efficiently storage mechanism
- Minimum network overhead.
- Robust conflict resolution.

↳ Use case

- Tracking code changes.
- Managing different versions.
- Enabling team collaboration.

Git vs Subversion

↳

| Features | Git | Subversion |
|---|---|---|
| ① Architecture | Distributed | Centralized |
| ② Branching | Light weight | Heavy weight |
| ③ offline work | Full support | Limited. |
| ④ Performance | High | Moderate |
| ⑤ Storage efficiency | Optimized | Less efficient |

## ② Advanced CI/CD Ecosystem → Jenkin

- Automates building, testing, and deploying software.
- Ensures smooth integration of changes from multiple developers.

↳ Architectural Component

- Master-agent architecture
- Plugin-based extensibility
- Advanced pipeline configuration
- Dynamic resource allocation

↳ Enterprise Integration

- LDAP authentication
- Role-based access control
- Comprehensive logging

↳ Use case

- Automating software delivery pipelines.

## ③ Gitlab CI/CD

- Cloud-based platforms for managing Git repositories.
- Provides features like pull requests, issue tracking and CI/CD integration.

↳ Key feature

- Kubernetes native.
- Container based pipelines.
- Integrated security scanning.
- Comprehensive artifact management.

↳ Use case

- Storing and managing source code repositories securely.

## ④ Infrastructure as code (IaC) → Ansible

- Automate software provisioning, configuration, and deployment.
- Uses YAML-based playbooks to define system configuration
- Used for configuration orchestration.

↳ Technical Capabilities

- Agent architecture
- YAML-based declarative configuration
- Dynamic inventory management
- Parallel execution model

↳ Use case Scenarios

- Managing server configurations and automating software development.
- Complex multi-tier deployments.
- Hybrid cloud configuration.
- Compliance automation
- Security policy enforcement.

⑤ Terraform - Cloud - Agnostic Provisioning.

- Automates provisioning of cloud infrastructure.
- Ensures that infrastructure configuration are version-controlled
- Used for managing cloud resources efficiently.

↳ Advanced Provising feature

- State Management
- Modular infrastructure design
- Multi-cloud resource management
- Immutable Infrastructure principles

⑥ Docker (Containerization)

- Ensures software runs consistently across different environments.
- Helps in managing dependencies and configurations efficiently.
- Used in creating portable and consistent development environments.

⑦ Kubernetes (Container Orchestration)

- Manages containerized applications in a scalable and automated manner.
- Ensures high availability and load balancing for software application.
- Used to manage and deploy microservices in cloud environments.

↳ Configuration Management is vital for maintaining software reliability, security and consistency.

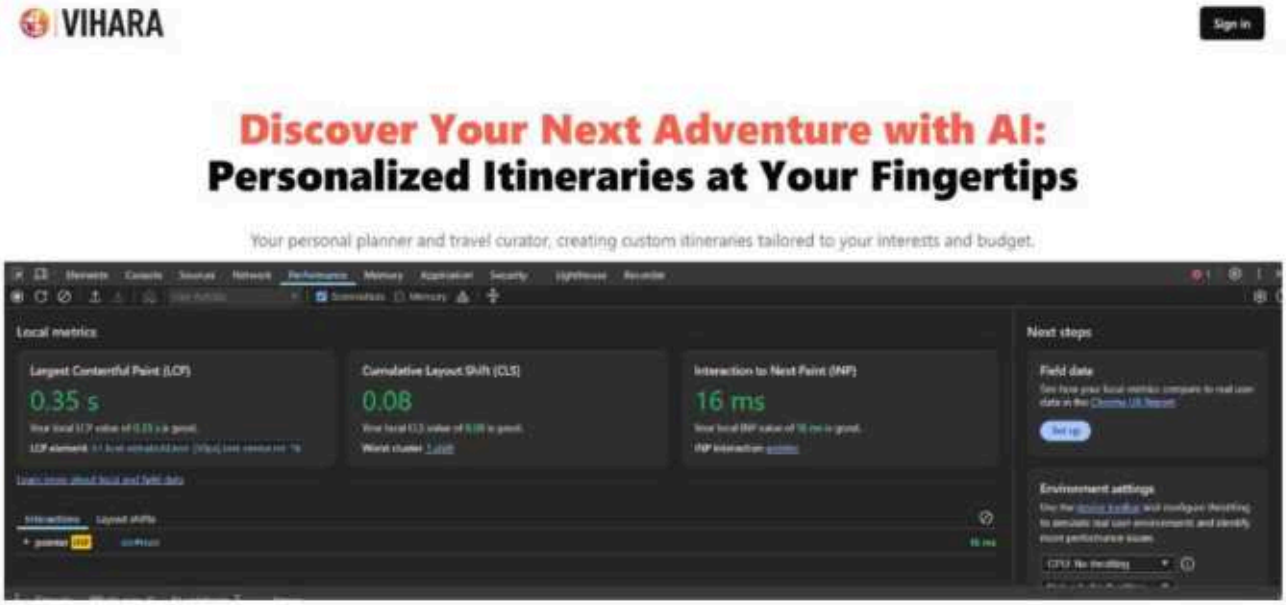
↳ Tools like Git, Ansible, Jenkin, Docker, Kubernetes help automate and streamline configuration processes, ensuring smooth software development and deployment.

↳ By integrating CM tools into the development workflow, organizations can enhance productivity, reduce errors, and improve software quality.

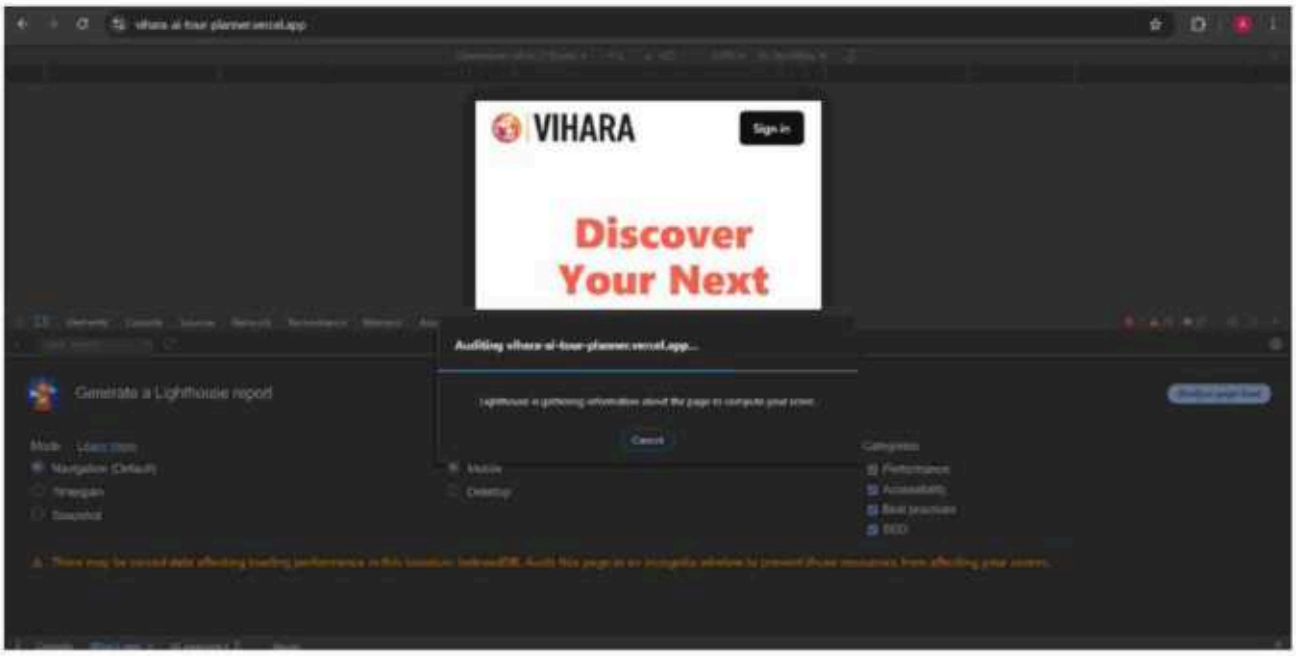
—x—

## TESTING (Test document as prepared in DA)

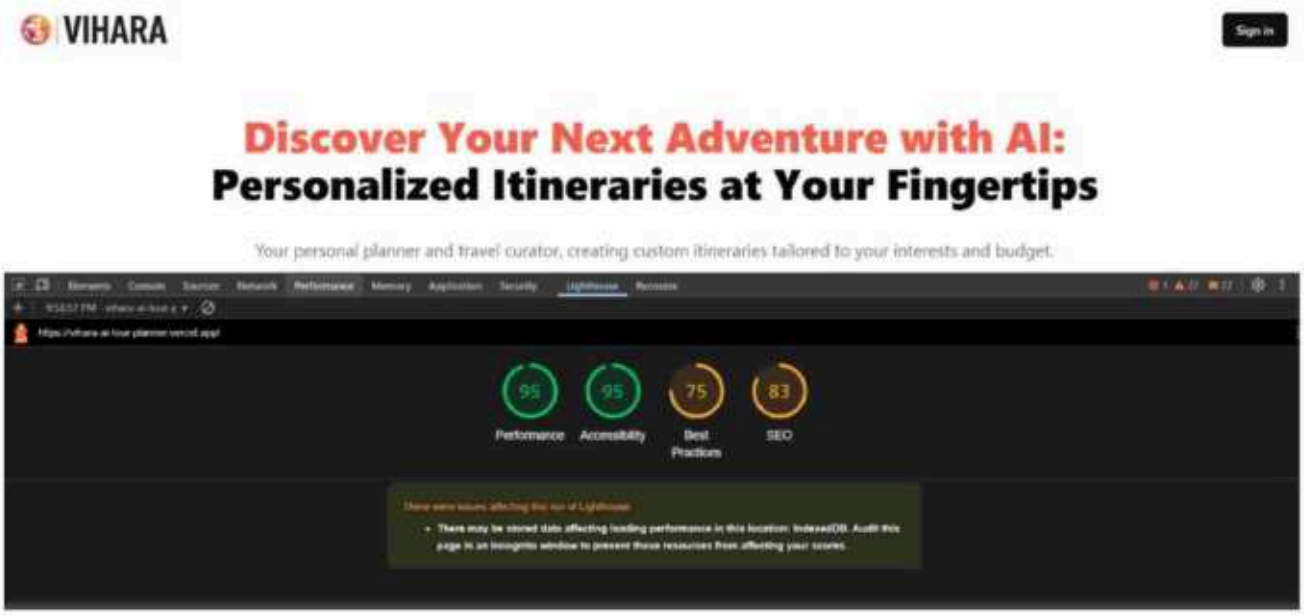
1. **Website metric score for mobile devices:**



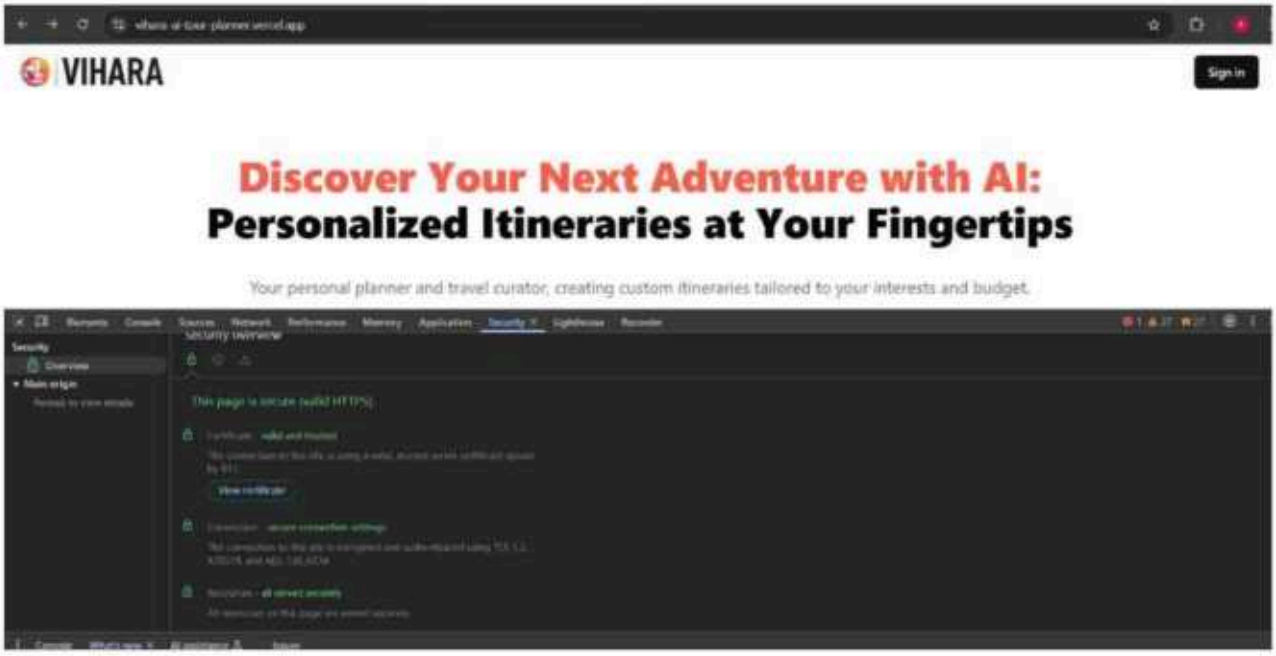

2. **Website auditing for Mobile devices:**



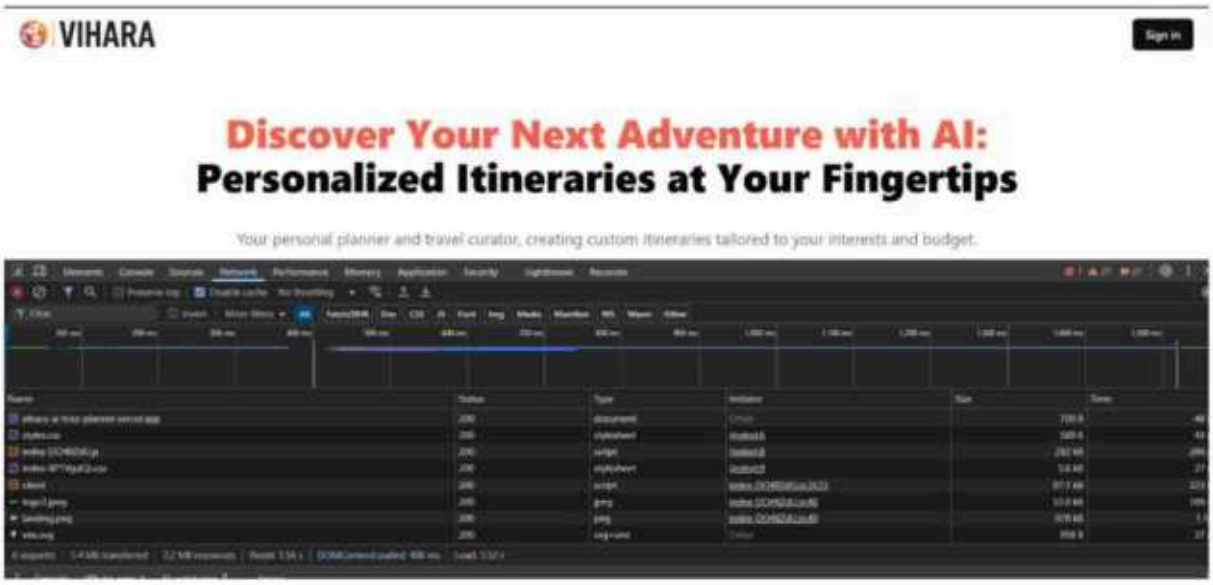

3. **Website test score for mobile devices:**



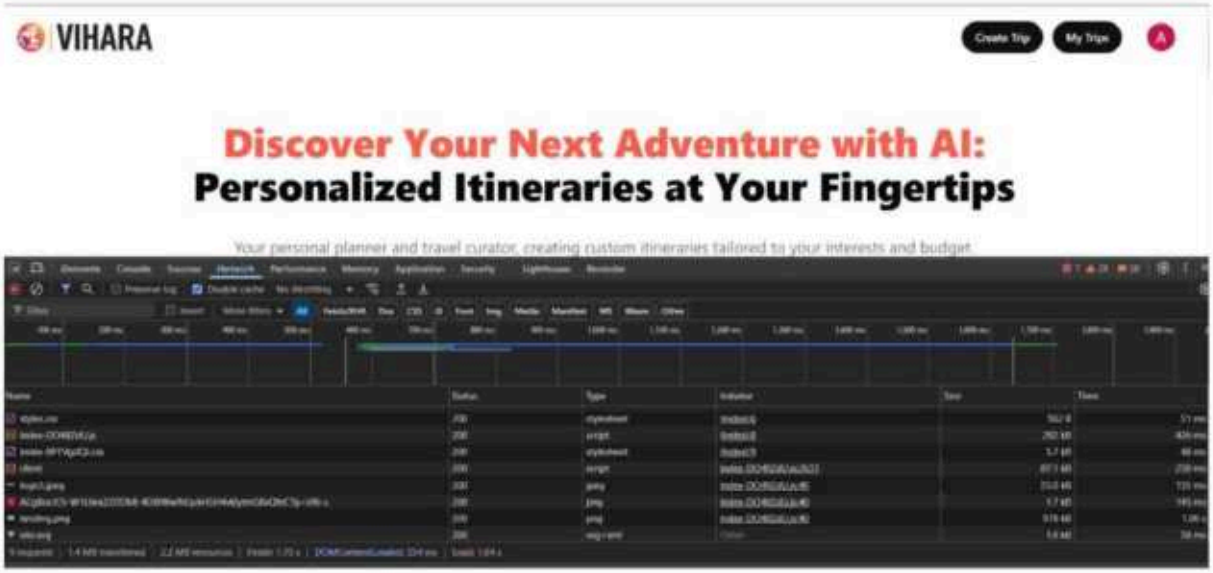

4. **Security Test:**



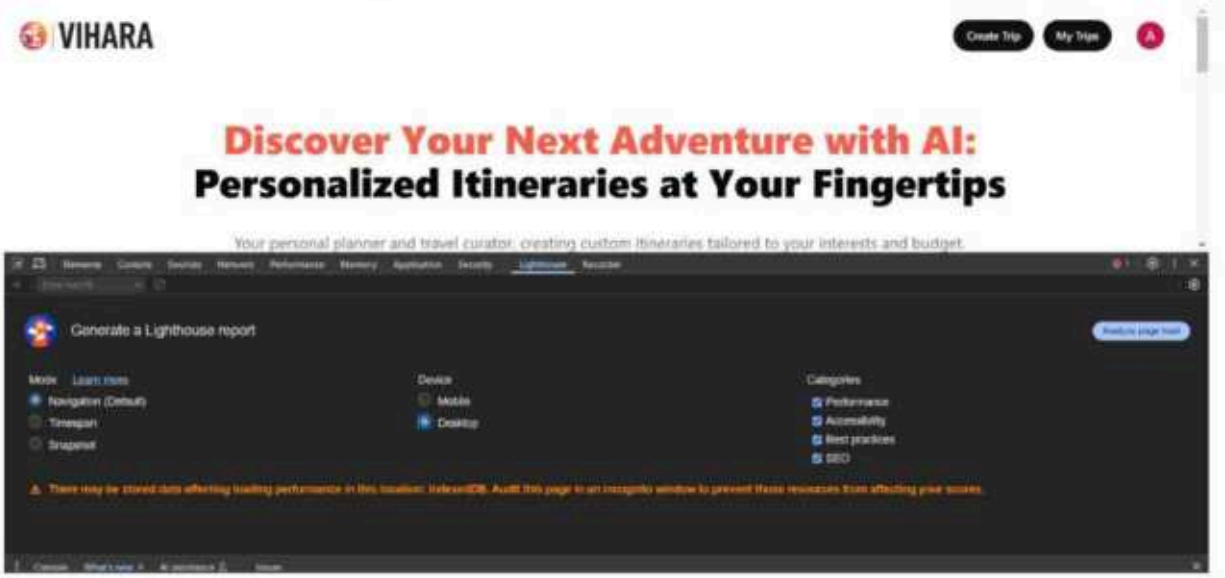
5. **Website Network test for mobile before sign in:**



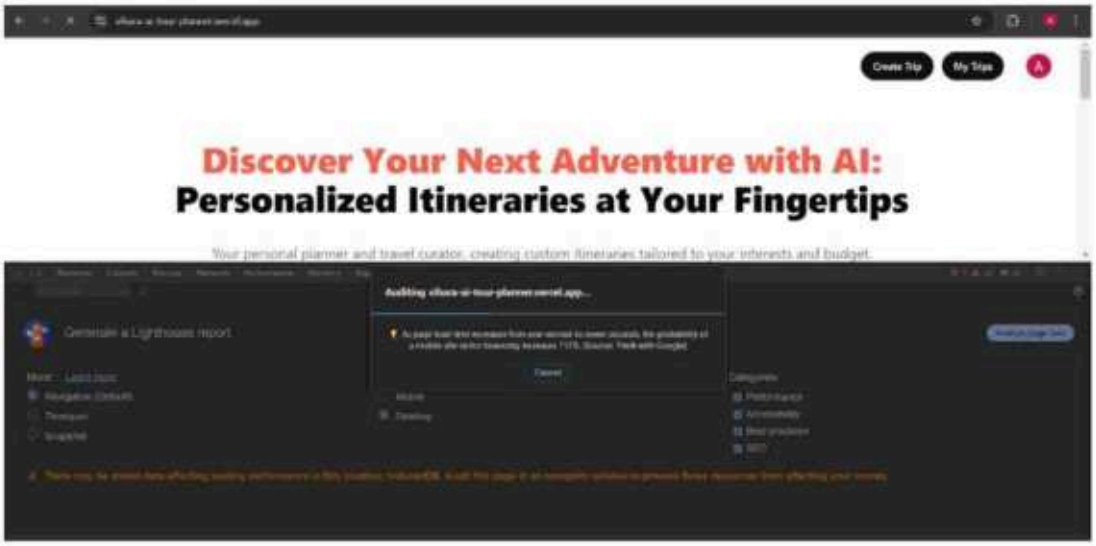

6. **Website Network test for mobile after sign in:**



7. **Website test for web platforms:**

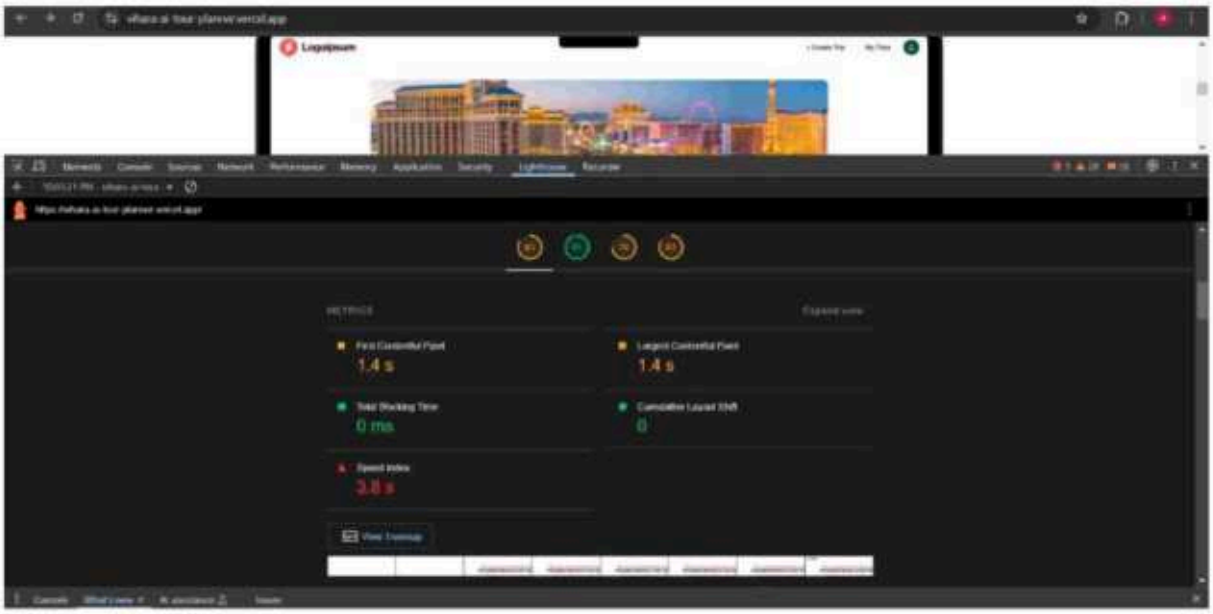



8. **Website auditing page for web platforms:**

9. **Website test score for web platforms:**


10. **Website test score for web platforms:**



## CONCLUSION:

The VIHARA - TOUR PLANNER represents a significant advancement in travel planning technology, offering a comprehensive solution to the challenges faced by modern travelers. Through the detailed Software Requirement Specification, we have outlined a robust framework for developing a user-centric platform that integrates multiple travel services into one cohesive system.

The project leverages cutting-edge technologies including Angular for frontend development, Go for backend services, and secure database management systems to ensure a seamless user experience. By incorporating AI-driven route optimization, real-time weather data integration, and personalized itinerary creation, VIHARA addresses the core needs of diverse travelers—from solo adventurers to families and business professionals.

The functional requirements detailed in this document establish clear guidelines for essential features such as user registration, destination search, booking capabilities, and profile management. These are complemented by comprehensive non-functional requirements focusing on security, performance, capacity, and availability, ensuring the system not only functions correctly but does so efficiently and securely.

VIHARA's architecture prioritizes scalability and modularity, allowing for future expansion and feature enhancement without significant disruption. The system's emphasis on security—implementing encryption, JWT-based authentication, and secure data transmission—reflects our commitment to protecting user information in an increasingly vulnerable digital landscape.

The integration of emergency assistance features demonstrates our dedication to user safety, while the weather-based recommendation system showcases our innovative approach to travel planning. By dynamically adjusting travel suggestions based on real-time weather conditions, VIHARA offers a level of personalization rarely seen in conventional travel platforms.

## REFERENCES:


[1] Travel APIs Documentation (e.g., Skyscanner API, Booking.com API).

[2] Google Reviews.

[3] ER Diagram Tutorial:
https://www.tutorialspoint.com/dbms/er_diagram_representation.htm

[4] Requirement Engineering:http://morse.inf.unideb.hu/valseg/gybitt/07/ch02.html

[5] Data Flow Diagram: http://myyee.tripod.com/cs457/dfd.htm

[6] Requirement Engineering: https://en.wikipedia.org/wiki/Requirements_engineering

[7] Google doc document

[8] Weather IMD

[9] CPCB AQI

[10] AI Model documentation