Centre for Distance & Online Education

Manipal University Jaipur

Dehmi Kalan, Rajasthan – 303 007.

Project Final Report

On

AI Based Virtual Travel Guide

Name	JACOB BABY
Roll Number	2214100795
Program	BACHELOR OF COMPUTER APPLICATIONS (BCA)
Semester	VI
Session	MAR 2024

Contents

Introduction

The Trip Planner Application is designed to help users plan their trips by providing information about various destinations, attractions and itineraries. The application aims to enhance the travel planning experience by offering detailed information and personalized itineraries.

Objectives

- Provide users with detailed information about destinations, including attractions and hotels.

- Generate personalized itineraries based on user preferences.

- Create a user-friendly interface for easy navigation and trip planning.

Repository Details:

- **Repository Hosting Service**: GitHub

- **Repository URL**: https://github.com/jacobbaby-arch/smart_guide

- **Repository Name**: smart_guide

- **Repository Type**: Public

System Analysis

Identification Of Need

There is a growing need for a comprehensive trip planning tool that can provide detailed information about destinations and generate personalized itineraries to enhance the travel experience.

Preliminary Investigation

Initial research was conducted to identify the key features required for the application, including destination information, attractions, hotels, and itinerary generation.

Feasibility Study

A feasibility study was conducted to assess the technical, economic, and operational feasibility of the project. The study concluded that the project is feasible and can be implemented within the given constraints.

Project Planning

The project was planned in phases, including requirements gathering, design, development, testing, and deployment.

Project Scheduling

PERT Chart

Figure 1: PERT Chart

Gantt Chart

ID	ⓘ	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1	✓	📌	Requirements Gather	5 days	Tue 08-10-24	Mon 14-10-24	
2	✓	🖥	System Design	7 days	Tue 15-10-24	Wed 23-10-24	1
3	✓	🖥	Database Design	5 days	Tue 15-10-24	Mon 21-10-24	1
4	✓	🖥	FrontEnd Developme	10 days	Thu 24-10-24	Wed 06-11-24	2
5	✓	🖥	BackEnd Developmen	12 days	Thu 24-10-24	Fri 08-11-24	2,3
6	✓	🖥	Integration	5 days	Mon 11-11-24	Fri 15-11-24	4,5
7	✓	🖥	Testing	7 days	Mon 18-11-24	Tue 26-11-24	6
8	✓	🖥	Deployment	3 days	Wed 27-11-24	Fri 29-11-24	7

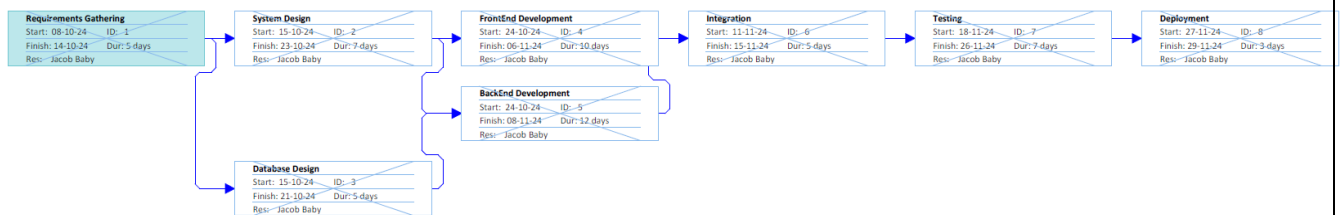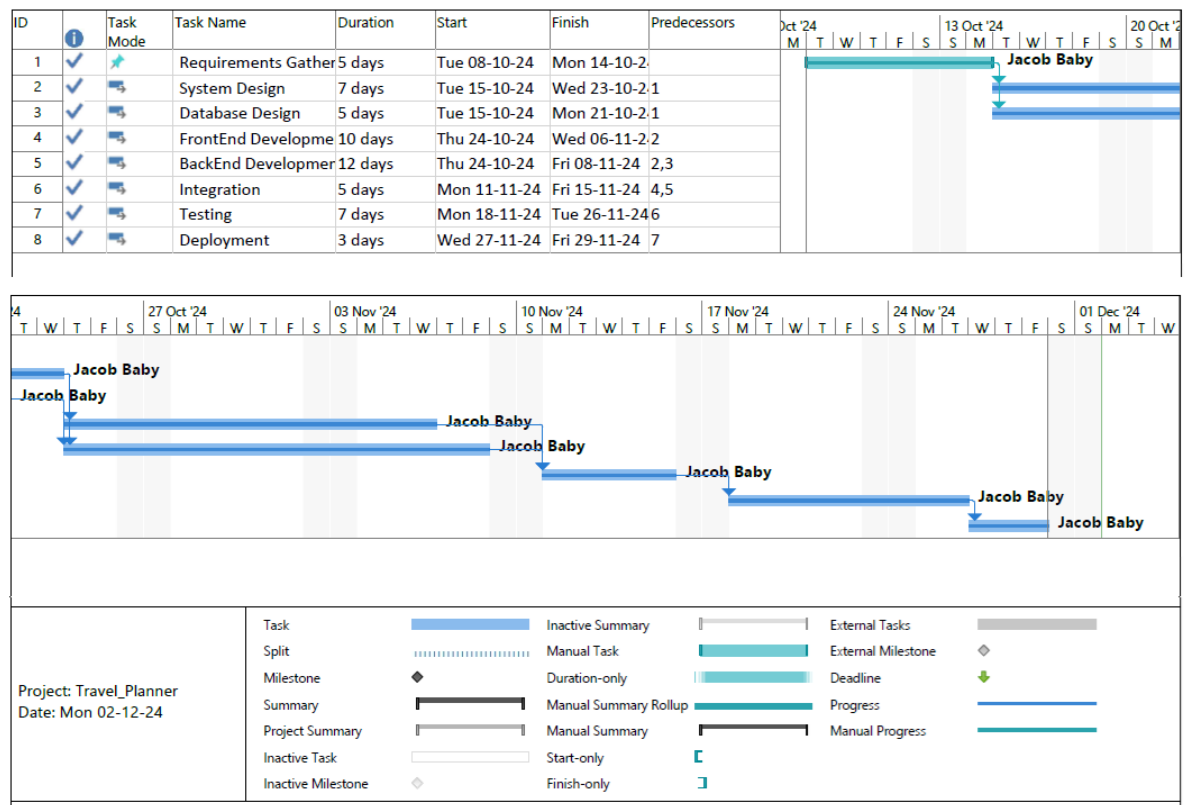|		
Project: Travel_Planner Date: Mon 02-12-24	Task	▬▬▬
	Split	....
	Milestone	◆
	Summary	▬▬▬
	Project Summary	▭▭▭
	Inactive Task	▭▭▭
	Inactive Milestone	◇
	Inactive Summary	▯▯
	Manual Task	▬▬▬
	Duration-only	▬▬▬
	Manual Summary Rollup	▬▬▬
	Manual Summary	▭▭▭
	Start-only	[
	Finish-only	]
	External Tasks	▬▬▬
	External Milestone	◇
	Deadline	⬇
	Progress	▬▬▬
	Manual Progress	▬▬▬

Figure 2: GANTT Chart

Software Requirement Specifications (SRS)

Agile Methodology is used for the project management.

Introduction to Agile Methodology

Agile methodology is a software development paradigm that emphasizes iterative development, collaboration, and flexibility. It is designed to accommodate changes and deliver functional software incrementally. Agile methodologies prioritize customer satisfaction, continuous feedback, and adaptive planning.

Key Principles of Agile Methodology

1. **Customer Collaboration**: Agile methodologies emphasize close collaboration with customers to ensure that the software meets their needs and expectations. Customers are involved throughout the development process, providing feedback and clarifying requirements.

2. **Iterative Development**: Agile projects are divided into small, manageable iterations or sprints, typically lasting 1-4 weeks. Each iteration results in a potentially shippable product increment, allowing for continuous delivery of value.

3. **Adaptive Planning**: Agile methodologies embrace change and allow for adaptive planning. Requirements and priorities are revisited and adjusted at the end of each iteration based on customer feedback and changing business needs.

4. **Cross-functional Teams**: Agile teams are composed of cross-functional members with diverse skills, including developers, testers, designers, and product owners. This promotes collaboration and ensures that all aspects of the project are considered.

5. **Continuous Improvement**: Agile methodologies encourage continuous improvement through regular retrospectives. Teams reflect on their processes and practices, identifying areas for improvement and implementing changes in subsequent iterations.

Agile Practices Applied in the Trip Planner Application

1. **User Stories and Backlog**: Requirements for the Trip Planner Application were captured as user stories and maintained in a product backlog. Each user story represents a small, valuable piece of functionality from the user's perspective.

2. **Sprint Planning**: The project was divided into multiple sprints, each lasting two weeks. During sprint planning meetings, the team selected user stories from the backlog to work on in the upcoming sprint, based on priority and team capacity.

3. **Daily Stand-ups**: Daily stand-up meetings were held to discuss progress, identify any blockers, and plan the day's work. This ensured that the team remained aligned and could address issues promptly.

4. **Incremental Delivery**: At the end of each sprint, a potentially shippable product increment was delivered. This allowed for continuous delivery of value and early detection of issues.

5. **Sprint Review and Retrospective**: Sprint review meetings were held at the end of each sprint to demonstrate the completed work to stakeholders and gather feedback. Retrospective meetings were conducted to reflect on the sprint, identify areas for improvement, and plan changes for the next sprint.

Benefits of Using Agile Methodology

1. **Flexibility and Adaptability**: Agile methodologies allow for changes in requirements and priorities, enabling the team to respond to evolving customer needs and market conditions.

2. **Customer Satisfaction**: Continuous collaboration with customers ensures that the software meets their expectations and delivers value incrementally.

3. **Improved Quality**: Regular testing and feedback loops help identify and address issues early, resulting in higher-quality software.

4. **Faster Time-to-Market**: Incremental delivery of functional software allows for faster release cycles and quicker realization of value.

5. **Enhanced Collaboration**: Cross-functional teams and regular communication foster collaboration and ensure that all aspects of the project are considered.

Data flow diagram(DFD)

A DFD typically includes processes, data stores, data flows, and external entities. Below is a DFD for the Trip Planner Application, covering key functionalities such as viewing destinations, attractions, and generating itineraries.
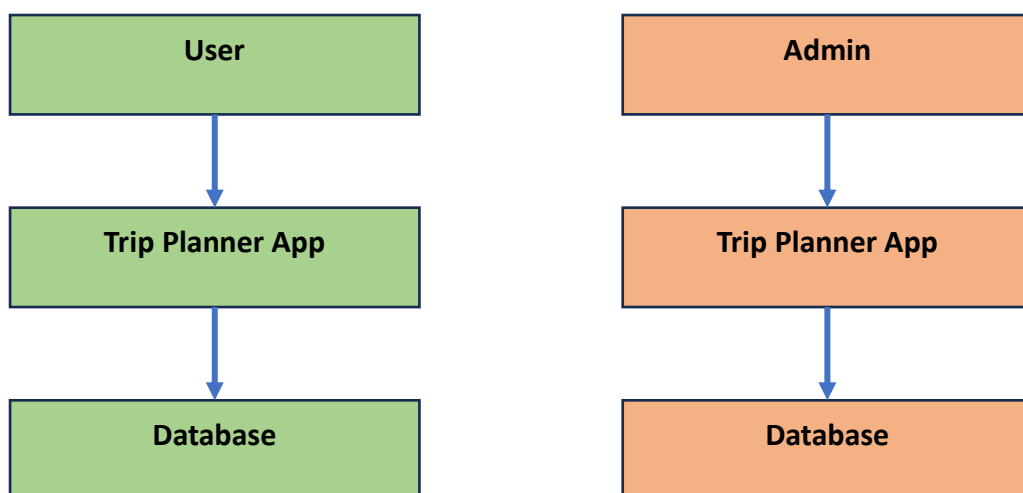
Level 0 DFD (Context Diagram)
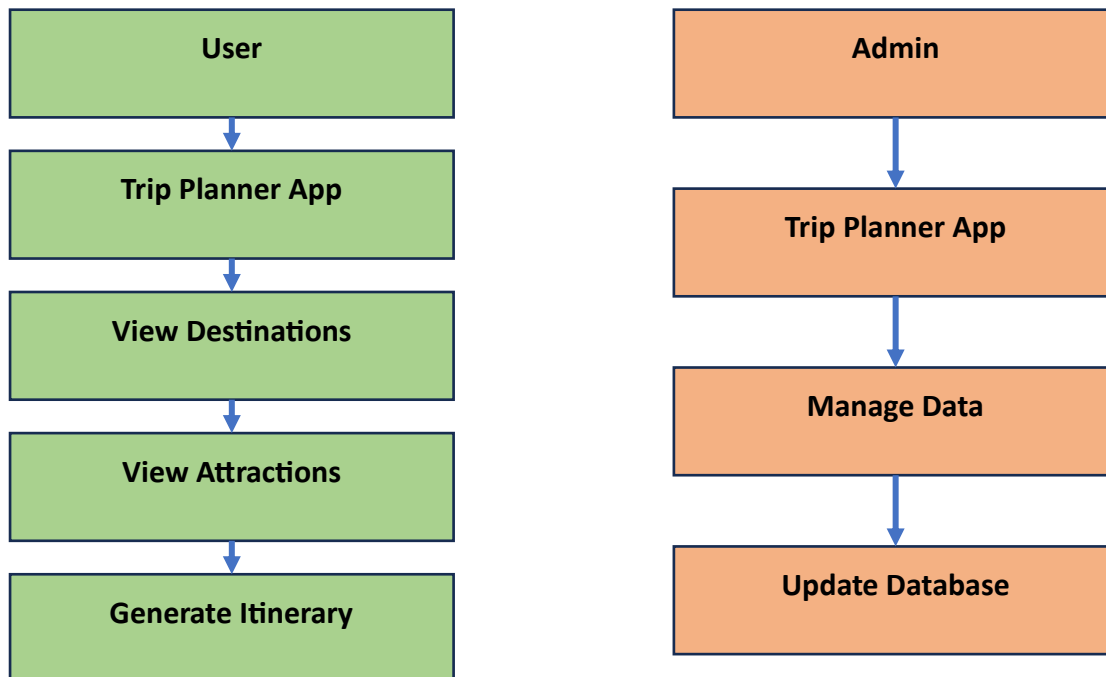

Figure 3: Level 0 DFD

Level 1 DFD

Figure 4: Level 1 DFD

1. **User**: The user interacts with the Trip Planner Application to view destinations, attractions, hotels, and generate itineraries.

2. **Admin**: The admin interacts with the Trip Planner Application to manage data, including adding, updating, and deleting records in the database.

3. **Trip Planner App**: The main application that handles user and admin interactions.

4. **View Destinations**: The process where the user views a list of destinations.

5. **View Attractions**: The process where the user views a list of attractions for a selected destination.

6. **View Hotels**: The process where the user views a list of hotels for a selected destination.

7. **Generate Itinerary**: The process where the user generates a personalized itinerary for a selected destination.

8. **Manage Data**: The process where the admin manages the application data.

9. **Update Database**: The process where the admin updates the database with new or modified data.

10. **Database**: The data store that contains information about continents, countries, destinations, attractions, hotels, and itineraries.
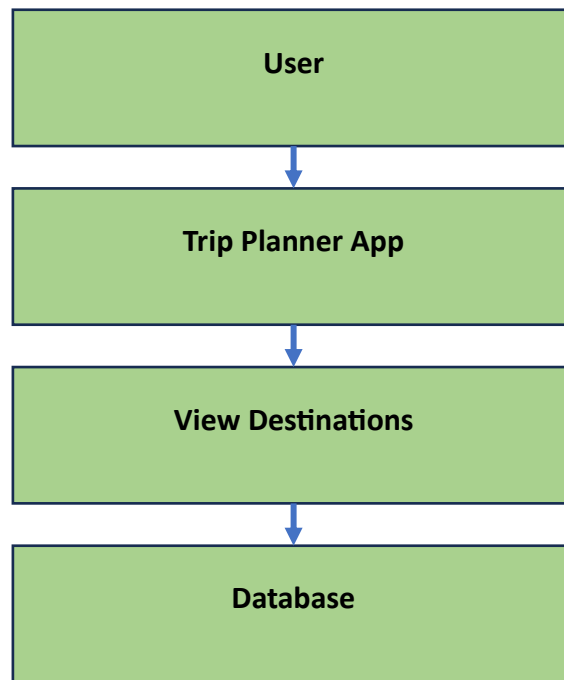
Level 2 DFD for Viewing Destinations

Figure 5: Level 2 DFD for Viewing Destinations
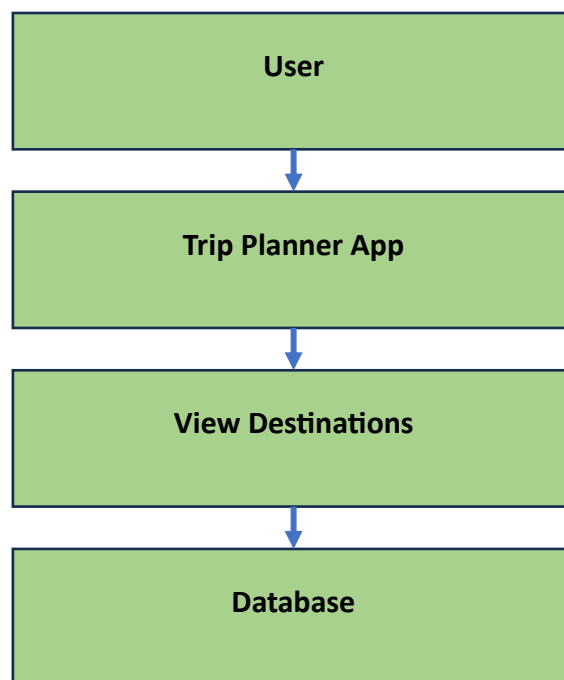
Level 2 DFD for Viewing Destinations

Figure 6: Level 2 DFD for Viewing Destinations

Level 2 DFD for Generating Itinerary
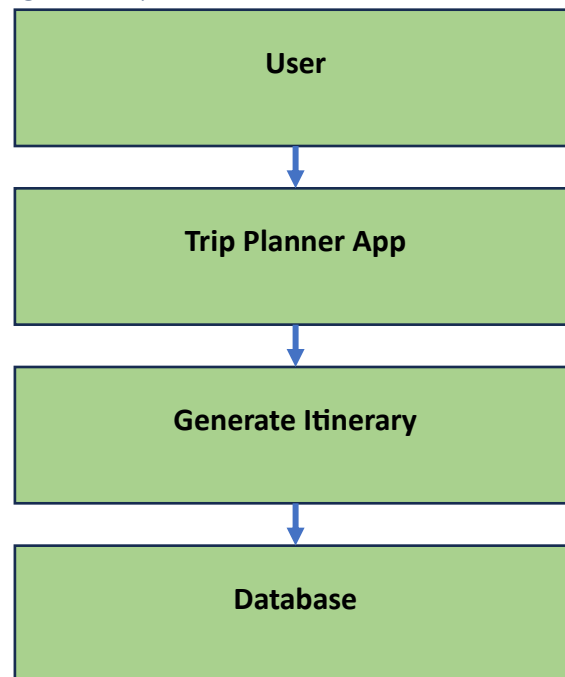
Figure 7: Level 2 DFD for Generating Itinerary

1. **View Destinations:** The user requests to view destinations, and the application fetches the list of destinations from the database.

2. **View Attractions:** The user requests to view attractions for a selected destination, and the application fetches the list of attractions from the database.

3. **Generate Itinerary:** The user requests to generate an itinerary for a selected destination, and the application fetches the itinerary from the database.

Control Flow Diagram (CFD):

Control Flow Diagrams (CFDs) involves illustrating the flow of control within the system. These diagrams help in understanding how different parts of the system interact with each other. Below are the Control Flow Diagrams for key functionalities of the Trip Planner Application.

Control Flow Diagram for Viewing Destinations

User Interface

↓

Select Continent

↓

Select Country

↓

Fetch Destinations

↓

Display Destinations
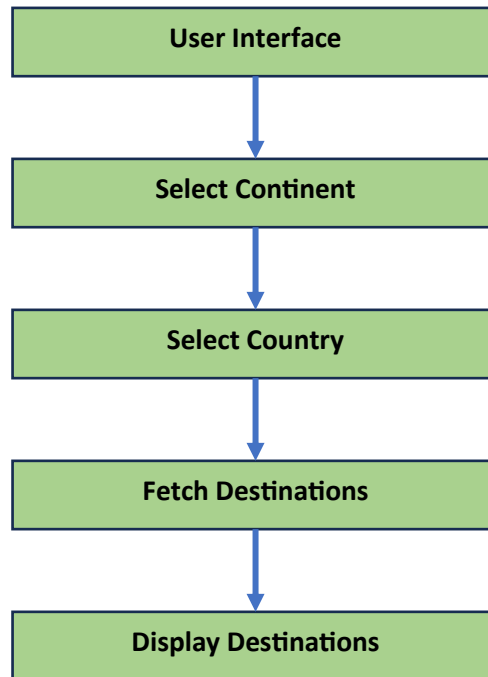
Figure 8: Control Flow Diagram for Viewing Destinations

Control Flow Diagram for Viewing Attractions

User Interface

↓

Select Destination

↓

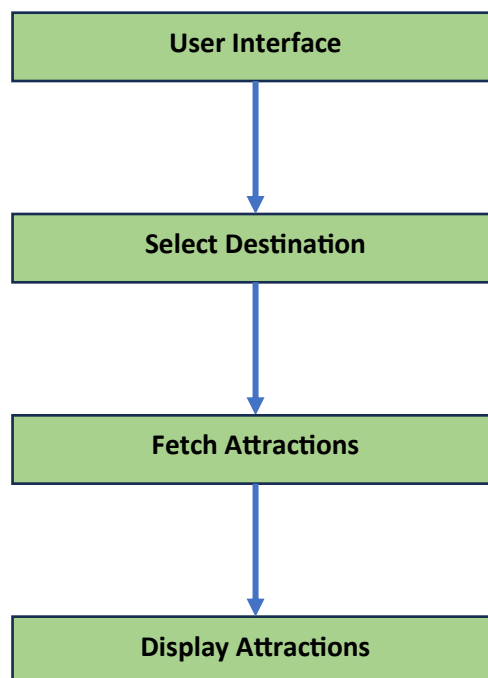**Fetch Attractions**

↓

Display Attractions

Figure 9: Control Flow Diagram for Viewing Attractions

Control Flow Diagram for Generating Itinerary
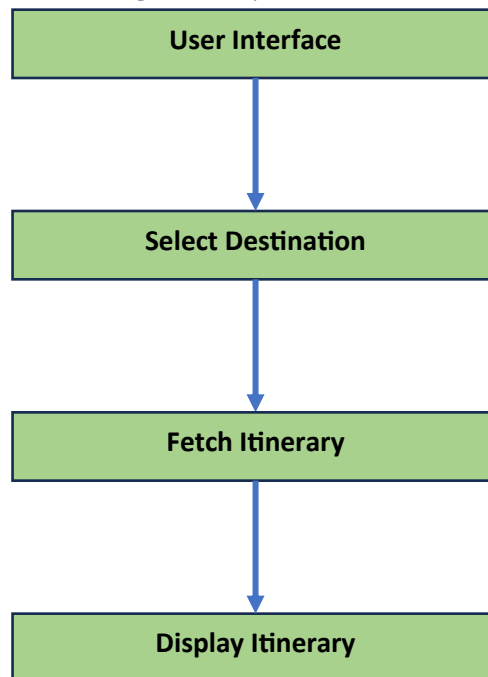
Figure 10: Control Flow Diagram for Generating Itinerary

Explanation of Control Flow Diagrams

1. **Viewing Destinations**

   o The user interacts with the user interface to select a continent and then a country.

   o The system fetches the list of destinations for the selected country using the backend API.

   o The system displays the list of destinations to the user.

2. **Viewing Attractions**

   o The user selects a destination from the user interface.

   o The system fetches the list of attractions for the selected destination using the backend API.

   o The system displays the list of attractions to the user.

3. **Generating Itinerary**

   o The user selects a destination from the user interface.

   o The system fetches the itinerary for the selected destination using the backend API.

   o The system displays the generated itinerary to the user.

State Diagram

State diagrams, also known as state machine diagrams, illustrate the states an object can be in and the transitions between those states. Below are the state diagrams for key components of the Trip Planner Application, such as viewing destinations, attractions, and generating itineraries.

State Diagram for Viewing Destinations

Initial State

Select Continent

Select Country

Fetch Destinations

Display Destinations

Final State
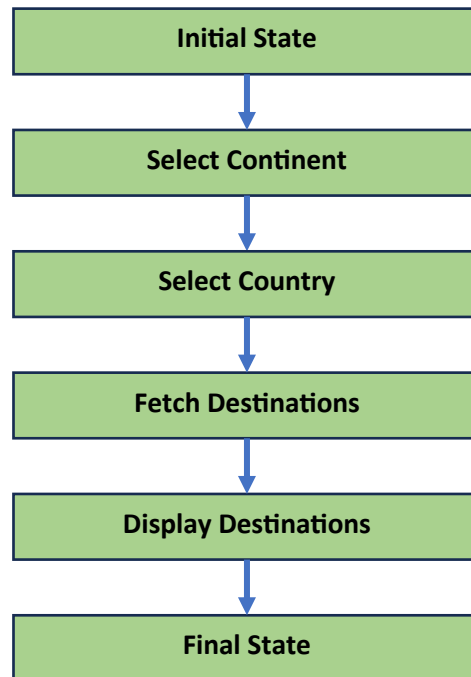
Figure 11: State Diagram for Viewing Destinations

State Diagram for Viewing Attractions

Initial State

Select Destination

Fetch Attractions
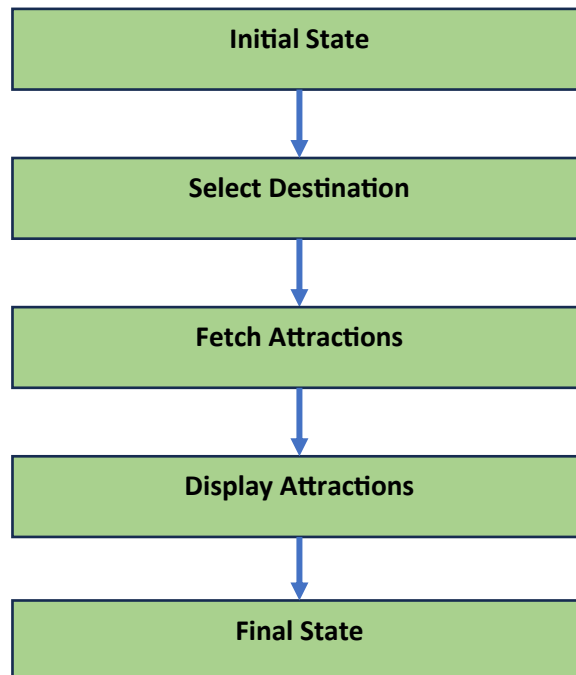
Display Attractions

Final State

Figure 12: State Diagram for Viewing Attractions

State Diagram for Generating Itinerary
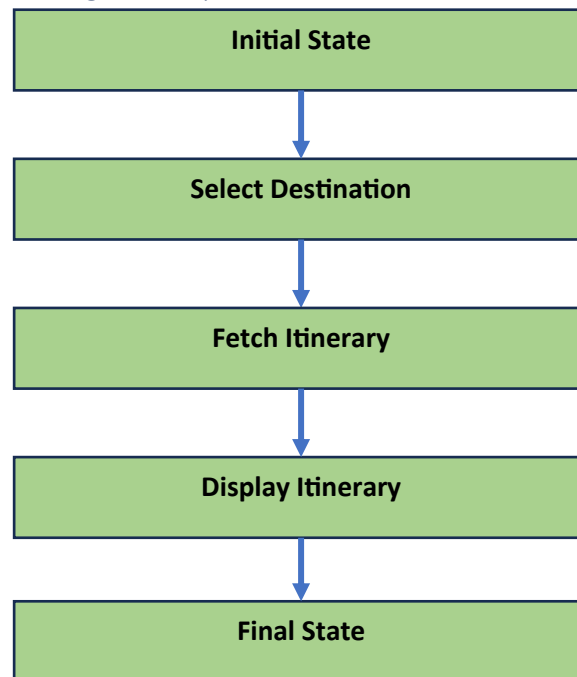
Figure 13: State Diagram for Generating Itinerary

1. **Viewing Destinations**

   o **Initial State**: The system is in the initial state.

   o **Select Continent**: The user selects a continent.

   o **Select Country**: The user selects a country within the selected continent.

   o **Fetch Destinations**: The system fetches the list of destinations for the selected country.

   o **Display Destinations**: The system displays the list of destinations to the user.

   o **Final State**: The system reaches the final state after displaying the destinations.

2. **Viewing Attractions**

   o **Initial State**: The system is in the initial state.

   o **Select Destination**: The user selects a destination.

   o **Fetch Attractions**: The system fetches the list of attractions for the selected destination.

   o **Display Attractions**: The system displays the list of attractions to the user.

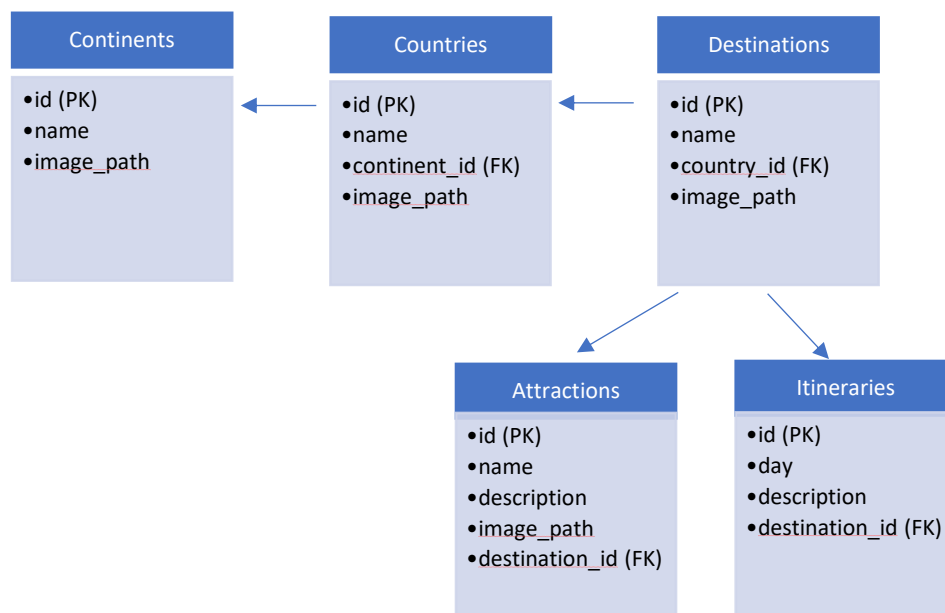   o **Final State**: The system reaches the final state after displaying the attractions.

3. **Generating Itinerary**

- o **Initial State**: The system is in the initial state.

- o **Select Destination**: The user selects a destination.

- o **Fetch Itinerary**: The system fetches the itinerary for the selected destination.

- o **Display Itinerary**: The system displays the generated itinerary to the user.

- o **Final State**: The system reaches the final state after displaying the itinerary.

Entity Relationship Diagram (ERD)

Continents
- •id (PK)
- •name
- •image_path

Countries
- •id (PK)
- •name
- •continent_id (FK)
- •image_path

Destinations
- •id (PK)
- •name
- •country_id (FK)
- •image_path

Attractions
- •id (PK)
- •name
- •description
- •image_path
- •destination_id (FK)

Itineraries
- •id (PK)
- •day
- •description
- •destination_id (FK)

1. **Continents**

   - o id: Primary key, unique identifier for each continent.

   - o name: Name of the continent.

   - o image_path: Path to the image representing the continent.

2. **Countries**

   - o id: Primary key, unique identifier for each country.

   - o name: Name of the country.

   - o continent_id: Foreign key, references the id in the Continents table.

   - o image_path: Path to the image representing the country.

3. **Destinations**

   o   id: Primary key, unique identifier for each destination.

   o   name: Name of the destination.

   o   country_id: Foreign key, references the id in the Countries table.

   o   image_path: Path to the image representing the destination.

4. **Attractions**

   o   id: Primary key, unique identifier for each attraction.

   o   name: Name of the attraction.

   o   description: Description of the attraction.

   o   image_path: Path to the image representing the attraction.

   o   destination_id: Foreign key, references the id in the Destinations table.

5. **Itineraries**

   o   id: Primary key, unique identifier for each itinerary item.

   o   day: Day number in the itinerary.

   o   description: Description of the activities for the day.

   o   destination_id: Foreign key, references the id in the Destinations table.

Use Case Diagram:

Use-case diagrams illustrate the interactions between users (actors) and the system to achieve specific goals. Below are the use-case diagrams for the Trip Planner Application, covering key functionalities such as viewing destinations, attractions and generating itineraries.

Use-case Diagram for Trip Planner Application

Actors:

User: A person using the application to plan trips.

Admin: A person managing the application data.

Use-cases:

1. **View Continents**

2. **View Countries**

3. **View Destinations**

4. **View Attractions**

5. **View Hotels**

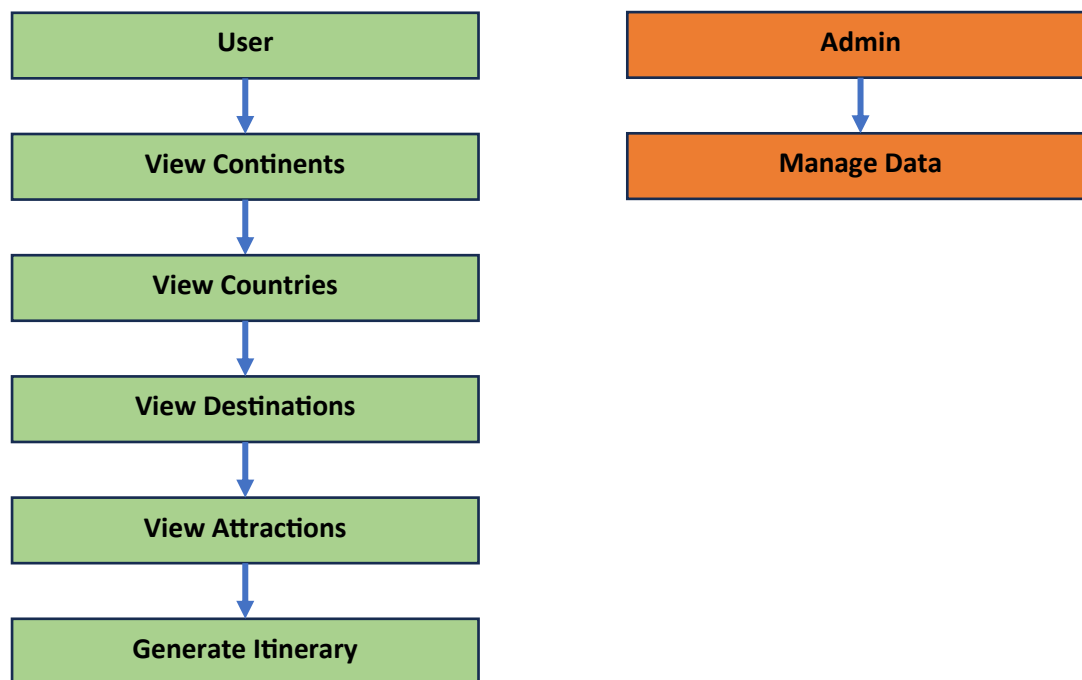6. **Generate Itinerary**

7. **Manage Data** (Admin)

Figure 14: Use-case Diagram

Explanation of Use-case Diagram

1. View Continents
   - o Actor: User
   - o Description: The user can view a list of continents available in the application.
2. View Countries
   - o Actor: User
   - o Description: The user can view a list of countries within a selected continent.
3. View Destinations
   - o Actor: User
   - o Description: The user can view a list of destinations within a selected country.
4. View Attractions
   - o Actor: User
   - o Description: The user can view a list of attractions for a selected destination.
5. Generate Itinerary
   - o Actor: User
   - o Description: The user can generate a personalized itinerary for a selected destination.
6. Manage Data

- o Actor: Admin
- o Description: The admin can manage the application data, including adding, updating, and deleting continents, countries, destinations, attractions.

Detailed Use-case Descriptions

Use-case: View Continents

- **Actor**: User

- **Precondition**: The user is on the home page.

- **Postcondition**: The user sees a list of continents.

- **Main Flow**:

  1. The user navigates to the home page.

  2. The system displays a list of continents.

Use-case: View Countries

- **Actor**: User

- **Precondition**: The user has selected a continent.

- **Postcondition**: The user sees a list of countries within the selected continent.

- **Main Flow**:

  1. The user selects a continent.

  2. The system displays a list of countries within the selected continent.

Use-case: View Destinations

- **Actor**: User

- **Precondition**: The user has selected a country.

- **Postcondition**: The user sees a list of destinations within the selected country.

- **Main Flow**:

  1. The user selects a country.

  2. The system displays a list of destinations within the selected country.

Use-case: View Attractions

- **Actor**: User

- **Precondition**: The user has selected a destination.

- **Postcondition**: The user sees a list of attractions for the selected destination.

- **Main Flow**:

1. The user selects a destination.

2. The system fetches and displays a list of attractions for the selected destination.

Use-case: Generate Itinerary

- **Actor**: User

- **Precondition**: The user has selected a destination.

- **Postcondition**: The user sees a generated itinerary for the selected destination.

- **Main Flow**:

    1. The user selects a destination.

    2. The system fetches and displays a generated itinerary for the selected destination.

Use-case: Manage Data

- **Actor**: Admin

- **Precondition**: The admin is logged in.

- **Postcondition**: The admin can manage the application data.

- **Main Flow**:

    1. The admin logs in.

    2. The admin can add, update, or delete continents, countries, destinations, attractions.

System Design

Configuration Management

Repository Structure

The repository is organized into directories and files to maintain a clean and manageable structure.
Below is an example of the repository structure:

```
smart_guide/
├── doc/
├── src/
│   ├── app.py
│   ├── db.py
│   ├── templates/
│   │   ├── index.html
│   │   ├── countries.html
│   │   ├── destinations.html
│   │   ├── attractions.html
│   │   ├── itinerary.html
│   │   ├── admin.html
│   │   ├── admin_login.html
│   │   ├── about.html
│   │   ├── contact.html
│   ├── static/
│   │   ├── css/
│   │   │   ├── styles.css
│   │   ├── js/
│   │   │   ├── scripts.js
│   │   ├── images/
│   │   │   ├── asia.jpg
│   │   │   ├── europe.jpg
│   │   │   ├── america.jpg
│   │   │   ├── india.jpg
├── tests/
│   ├── test_app.py
│   ├── test_integration.py
│   ├── test_system.py
├── README.md
├── requirements.txt
├── .gitignore
├── LICENSE
```

Repository Files and Directories

- **src/**: Contains the source code of the application.

    o **app.py**: Main Flask application file containing routes and controllers.

    o **db.py**: Script for creating and initializing the database schema.

    o **templates/**: Directory containing HTML templates for the application.

    o **static/**: Directory containing static files such as CSS, JavaScript, and images.

- **tests/**: Contains test scripts for unit, integration, and system testing.

  o **test_app.py**: Unit test cases for the application.

  o **test_integration.py**: Integration test cases for the application.

  o **test_system.py**: System test cases for the application.

- **README.md**: Provides an overview of the project, installation instructions, and usage guidelines.

- **requirements.txt**: Lists the dependencies required for the project.

- **.gitignore**: Specifies files and directories to be ignored by Git.

- **LICENSE**: Contains the license information for the project.

Version Control Practices

- **Branching Strategy**: Use a branching strategy such as GitFlow to manage feature development, bug fixes, and releases.

  o **main**: The main branch containing the stable version of the application.

  o **develop**: The development branch where new features and bug fixes are integrated.

  o **feature/**: Branches for developing new features.

  o **bugfix/**: Branches for fixing bugs.

  o **release/**: Branches for preparing releases.

- **Commit Messages**: Use clear and descriptive commit messages to document changes.

  o Example: Add user authentication feature, Fix bug in itinerary generation, Update README with installation instructions.

- **Pull Requests**: Use pull requests to review and merge changes into the main or develop branches.

  o Ensure that all tests pass before merging a pull request.

  o Conduct code reviews to maintain code quality and consistency.

Modularisation Details of the Trip Planner Application

Modularisation involves dividing the application into distinct, manageable modules, each responsible for a specific aspect of the system. This approach enhances maintainability, scalability, and collaboration among team members. Below are the modularisation details for the Trip Planner Application.

Modules

1. **User Interface (UI) Module**

2. **Backend Module**

3. **Database Module**

4. **Admin Module**

5. **API Integration Module**

User Interface (UI) Module

Responsibilities:

- Manage the presentation layer of the application.

- Provide an intuitive and user-friendly interface for users to interact with the system.

- Handle user inputs and display data fetched from the backend.

Components:

- **HTML Templates**: Define the structure of web pages.

- **CSS Stylesheets**: Define the styling and layout of web pages.

- **JavaScript**: Handle client-side interactions and AJAX requests.

Key Files:

- templates/index.html: Home page displaying continents.

- templates/countries.html: Page displaying countries within a selected continent.

- templates/destinations.html: Page displaying destinations within a selected country.

- templates/attractions.html: Page displaying attractions for a selected destination.

- templates/hotels.html: Page displaying hotels for a selected destination.

- static/css/styles.css: Stylesheet for the application.

- static/js/scripts.js: JavaScript file for handling client-side interactions.

Backend Module

Responsibilities:

- Handle server-side logic and processing.

- Manage API endpoints for fetching and updating data.

- Implement business logic and data validation.

Components:

- **Flask Application**: Main application framework.

- **Routes**: Define API endpoints and handle HTTP requests.

- **Controllers**: Implement business logic and data processing.

Key Files:

- app.py: Main Flask application file containing routes and controllers.

Database Module

Responsibilities:

- Manage data storage and retrieval.

- Ensure data integrity and consistency.

- Handle database migrations and schema updates.

Components:

- **SQLite Database**: Database management system.

- **Database Schema**: Define tables, relationships, and constraints.

- **Data Access Layer**: Handle database queries and transactions.

Key Files:

- db.py: Script for creating and initializing the database schema.

- trip_planner.db: SQLite database file.

Admin Module

Responsibilities:

- Provide administrative functionalities for managing application data.

- Allow admins to add, update, and delete records in the database.

- Implement authentication and authorization for admin access.

Components:

- **Admin Interface**: Web pages for admin functionalities.

- **Admin Controllers**: Handle admin-specific logic and data processing.

- **Authentication**: Implement login and access control for admins.

Key Files:

- templates/admin.html: Admin dashboard page.

- templates/admin_login.html: Admin login page.

- app.py: Contains routes and controllers for admin functionalities.

API Integration Module

Responsibilities:

- Integrate external APIs for additional functionalities.

- Handle API requests and responses.

- Process and format data received from external APIs.

Components:

- **API Clients**: Implement API requests and handle responses.

- **Data Processing**: Process and format data received from APIs.

Key Files:

- app.py: Contains routes and controllers for API integration.

Module Interactions

1. **User Interface (UI) Module** interacts with the **Backend Module** to fetch and display data.

2. **Backend Module** interacts with the **Database Module** to retrieve and update data.

3. **Admin Module** interacts with the **Backend Module** to manage application data.

4. **API Integration Module** interacts with external APIs and processes data for the **Backend Module**.

Example Interaction Flow

1. **User views destinations**:

   o User selects a continent and country in the UI.

   o UI sends a request to the Backend Module.

   o Backend Module fetches data from the Database Module.

   o Backend Module sends the data back to the UI.

   o UI displays the list of destinations.

2. **Admin adds a new destination**:

   o Admin logs in and accesses the admin interface.

   o Admin fills out a form to add a new destination.

   o Admin Module sends the data to the Backend Module.

   o Backend Module validates the data and updates the Database Module.

o   Database Module stores the new destination.

Data Integrity and Constraints of the Trip Planner Application

Ensuring data integrity and applying constraints are crucial aspects of database design. They help maintain the accuracy, consistency, and reliability of the data stored in the database. Below are the details of data integrity and constraints applied in the Trip Planner Application.

Types of Data Integrity

1.   Entity Integrity

2.   Referential Integrity

3.   Domain Integrity

Entity Integrity

Entity integrity ensures that each table has a primary key and that the primary key is unique and not null. This guarantees that each record in the table can be uniquely identified.

Referential Integrity

Referential integrity ensures that relationships between tables remain consistent. This is achieved by using foreign keys to link tables and enforcing rules to maintain these relationships.

Example:

- The continent_id in the countries table is a foreign key that references the id in the continents table.

- The country_id in the destinations table is a foreign key that references the id in the countries table.

- The destination_id in the attractions and itineraries tables are foreign keys that reference the id in the destinations table.

Domain Integrity

Domain integrity ensures that the data entered into a column falls within a specific range or domain. This is achieved by defining data types, constraints, and rules for each column.

Example:

- The name column in the continents, countries, destinations, attractions tables is defined as TEXT NOT NULL, ensuring that a name must be provided and it must be a text value.

- The day column in the itineraries table is defined as INTEGER NOT NULL, ensuring that a day value must be provided and it must be an integer.

Database Design of the Trip Planner Application

The database design for the Trip Planner Application involves defining the schema, tables, relationships, and constraints to ensure data integrity and efficient data retrieval. Below is a detailed description of the database design, including the schema and SQL commands for creating the tables.

Database Schema

The database schema consists of the following tables:

1. **Continents**

2. **Countries**

3. **Destinations**

4. **Attractions**

5. **Hotels**

6. **Itineraries**

Table Definitions

1. Continents Table

- **Purpose:** Stores information about continents.
- **Columns:**
  - id: Primary key, unique identifier for each continent.
  - name: Name of the continent.
  - image_path: Path to the image representing the continent.

```
CREATE TABLE IF NOT EXISTS continents (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT NOT NULL UNIQUE,

    image_path TEXT

);
```

2. Countries Table

- **Purpose:** Stores information about countries.

- **Columns:**

  - id: Primary key, unique identifier for each country.

  - name: Name of the country.

  - continent_id: Foreign key, references the id in the Continents table.

  - image_path: Path to the image representing the country.

```
CREATE TABLE IF NOT EXISTS countries (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT NOT NULL,

    continent_id INTEGER,

    image_path TEXT,

    FOREIGN KEY (continent_id) REFERENCES continents (id)

),
```

3. Destinations Table

- **Purpose:** Stores information about destinations.

- **Columns:**

  - id: Primary key, unique identifier for each destination.

  - name: Name of the destination.

  - country_id: Foreign key, references the id in the Countries table.

  - image_path: Path to the image representing the destination.

```
CREATE TABLE IF NOT EXISTS destinations (

    id INTEGER PRIMARY KEY AUTOINCREMENT,

    name TEXT NOT NULL,

    country_id INTEGER,

    image_path TEXT,

    FOREIGN KEY (country_id) REFERENCES countries (id)

);
```

4. Attractions Table

- **Purpose:** Stores information about attractions.

- **Columns:**

- o id: Primary key, unique identifier for each attraction.

- o name: Name of the attraction.

- o description: Description of the attraction.

- o image_path: Path to the image representing the attraction.

- o destination_id: Foreign key, references the id in the Destinations table.

```
CREATE TABLE IF NOT EXISTS attractions (

   id INTEGER PRIMARY KEY AUTOINCREMENT,

   name TEXT NOT NULL,

   description TEXT,

   image_path TEXT,

   destination_id INTEGER,

   FOREIGN KEY (destination_id) REFERENCES destinations (id)
```

5. Itineraries Table

- **Purpose:** Stores information about itineraries.

- **Columns:**

- o id: Primary key, unique identifier for each itinerary item.

- o day: Day number in the itinerary.

- o description: Description of the activities for the day.

- o destination_id: Foreign key, references the id in the Destinations table.

```
CREATE TABLE IF NOT EXISTS itineraries (

   id INTEGER PRIMARY KEY AUTOINCREMENT,

   day INTEGER NOT NULL,

   description TEXT NOT NULL,

   destination_id INTEGER,

   FOREIGN KEY (destination_id) REFERENCES destinations (id)
```

Relationships

Continents to Countries: *One-to-Many relationship*
- A continent can have multiple countries.
- The continent_id in the countries table references the id in the continents table.

Countries to Destinations: *One-to-Many relationship*
- A country can have multiple destinations.
- The country_id in the destinations table references the id in the countries table.

Destinations to Attractions: *One-to-Many relationship*
- A destination can have multiple attractions.
- The destination_id in the attractions table references the id in the destinations table.

Destinations to Itineraries: *One-to-Many relationship*
- A destination can have multiple itinerary items.
- The destination_id in the itineraries table references the id in the destinations table.

Data Integrity and Constraints
- **Primary Key Constraints:** Ensure that each table has a unique identifier.
  Example: id in all tables.
- **Foreign Key Constraints:** Ensure referential integrity between related tables.
  Example: continent_id in the countries table references id in the continents table.
- **Not Null Constraints:** Ensure that certain columns cannot have null values.
  Example: name in the continents, countries, destinations, attractions, and hotels tables.
- **Unique Constraints:** Ensure that certain columns have unique values.
  Example: name in the continents table.

Procedural Design for the Trip Planner Application

Procedural design involves defining the procedures or functions that will be used to implement the functionality of the application. This includes specifying the inputs, outputs, and processing steps for each function. Below is the procedural design for the Trip Planner Application, covering key functionalities such as viewing continents, countries, destinations, attractions and generating itineraries.

Key Procedures
1. View Continents
2. View Countries
3. View Destinations
4. View Attractions
5. View Hotels
6. Generate Itinerary
7. Manage Data (Admin)

1. View Continents
Procedure: view_continents
Description: Fetches and displays the list of continents.

Inputs: None
Outputs: List of continents
Steps:

1. Connect to the database.
2. Execute a query to fetch all continents.
3. Return the list of continents.

Pseudo-code:

```
def view_continents():

    # Connect to the database

    conn = sqlite3.connect('trip_planner.db')

    cursor = conn.cursor()

    # Execute query to fetch all continents

    cursor.execute('SELECT id, name, image_path FROM continents')

    continents = cursor.fetchall()

    # Close the database connection

    conn.close()

    # Return the list of continents
```

2. *View Countries*

Procedure: view_countries
Description: Fetches and displays the list of countries for a selected continent.
Inputs: continent_id (integer)
Outputs: List of countries
Steps:
1. Connect to the database.
2. Execute a query to fetch countries for the given continent ID.
3. Return the list of countries.
Pseudo-code:

```
def view_countries(continent_id):

    # Connect to the database

    conn = sqlite3.connect('trip_planner.db')

    cursor = conn.cursor()

        # Execute query to fetch countries for the given continent ID

    cursor.execute('SELECT id, name, image_path FROM countries WHERE continent_id = ?',
(continent_id,))

    countries = cursor.fetchall()

        # Close the database connection

    conn.close()

        # Return the list of countries
```

3. View Destinations

Procedure: view_destinations

Description: Fetches and displays the list of destinations for a selected country.

Inputs: country_id (integer)

Outputs: List of destinations

Steps:

1. Connect to the database.
2. Execute a query to fetch destinations for the given country ID.
3. Return the list of destinations.

Pseudo-code:

```
def view_destinations(country_id):

    # Connect to the database

    conn = sqlite3.connect('trip_planner.db')

    cursor = conn.cursor()

    # Execute query to fetch destinations for the given country ID

    cursor.execute('SELECT id, name, image_path FROM destinations WHERE country_id = ?',
(country_id,))

    destinations = cursor.fetchall()

    # Close the database connection

    conn.close()

    # Return the list of destinations
```

4. View Attractions

Procedure: view_attractions

Description: Fetches and displays the list of attractions for a selected destination.

Inputs: destination_id (integer)

Outputs: List of attractions

Steps:

1. Connect to the database.
2. Execute a query to fetch attractions for the given destination ID.
3. Return the list of attractions.

Pseudo-code:

```python
def view_attractions(destination_id):

    # Connect to the database

    conn = sqlite3.connect('trip_planner.db')

    cursor = conn.cursor()

    # Execute query to fetch attractions for the given destination ID

    cursor.execute('SELECT id, name, description, image_path FROM attractions WHERE
destination_id = ?', (destination_id,))

    attractions = cursor.fetchall()

  # Close the database connection

    conn.close()

    # Return the list of attractions
```

5. Generate Itinerary

Procedure: generate_itinerary

Description: Fetches and displays the itinerary for a selected destination.

Inputs: destination_id (integer)

Outputs: List of itinerary items

Steps:

1. Connect to the database.
2. Execute a query to fetch itinerary items for the given destination ID.
3. Return the list of itinerary items.

Pseudo-code:

```
def generate_itinerary(destination_id):

    # Connect to the database

    conn = sqlite3.connect('trip_planner.db')

    cursor = conn.cursor()

    # Execute query to fetch itinerary items for the given destination ID

    cursor.execute('SELECT day, description FROM itineraries WHERE destination_id = ?',
(destination_id,))

    itinerary = cursor.fetchall()

    # Close the database connection

    conn.close()

    # Return the list of itinerary items
```

6. *Manage Data (Admin)*

Procedure: add_continent

Description: Adds a new continent to the database.

Inputs: name (string), image_path (string)

Outputs: Success or failure message

Steps:

1. Connect to the database.
2. Execute an insert query to add the new continent.
3. Return a success or failure message.

Pseudo-code:

```
def add_continent(name, image_path):

    # Connect to the database

    conn = sqlite3.connect('trip_planner.db')

    cursor = conn.cursor()


    # Execute insert query to add the new continent

    try:

        cursor.execute('INSERT INTO continents (name, image_path) VALUES (?, ?)', (name,
image_path))

        conn.commit()

        result = "Continent added successfully."

    except sqlite3.IntegrityError:

        result = "Error: Continent already exists."


    # Close the database connection

    conn.close()
```

Procedure: add_country
Description: Adds a new country to the database.
Inputs: name (string), continent_id (integer), image_path (string)
Outputs: Success or failure message
Steps:
1. Connect to the database.
2. Execute an insert query to add the new country.
3. Return a success or failure message.

Pseudo-code:

```
def add_country(name, continent_id, image_path):

    # Connect to the database

    conn = sqlite3.connect('trip_planner.db')

    cursor = conn.cursor()


    # Execute insert query to add the new country

    try:

        cursor.execute('INSERT INTO countries (name, continent_id, image_path) VALUES (?, ?, ?)',
(name, continent_id, image_path))

        conn.commit()

        result = "Country added successfully."

    except sqlite3.IntegrityError:

        result = "Error: Country already exists."


    # Close the database connection

    conn.close()
```

Procedure: add_destination
Description: Adds a new destination to the database.
Inputs: name (string), country_id (integer), image_path (string)
Outputs: Success or failure message
Steps:
1. Connect to the database.
2. Execute an insert query to add the new destination.
3. Return a success or failure message.

Pseudo-code:

```
def add_destination(name, country_id, image_path):

    # Connect to the database

    conn = sqlite3.connect('trip_planner.db')

    cursor = conn.cursor()


    # Execute insert query to add the new destination

    try:

        cursor.execute('INSERT INTO destinations (name, country_id, image_path) VALUES (?, ?, ?)',
(name, country_id, image_path))

        conn.commit()

        result = "Destination added successfully."

    except sqlite3.IntegrityError:

        result = "Error: Destination already exists."


    # Close the database connection

    conn.close()
```

User Interface Design for the Trip Planner Application

The User Interface (UI) design for the Trip Planner Application focuses on creating an intuitive and user-friendly experience for users. The design includes various pages and components that allow users to navigate through continents, countries, destinations, attractions, hotels, and itineraries. Below is a detailed description of the UI design, including wireframes and key elements for each page.
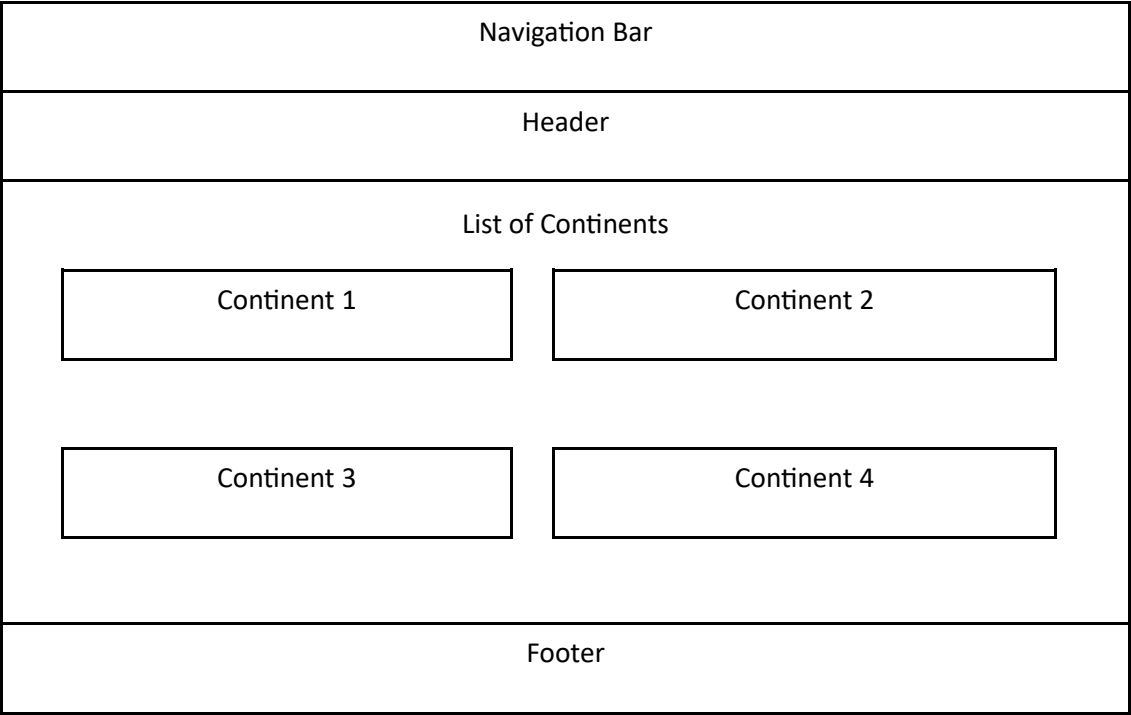
Key Pages

1. Home Page
2. Countries Page
3. Destinations Page
4. Attractions Page
5. Hotels Page
6. Itinerary Page

7. Admin Dashboard
8. Admin Login

1. Home Page

Description: The home page displays a list of continents that users can select to start their trip planning.

Wireframe:

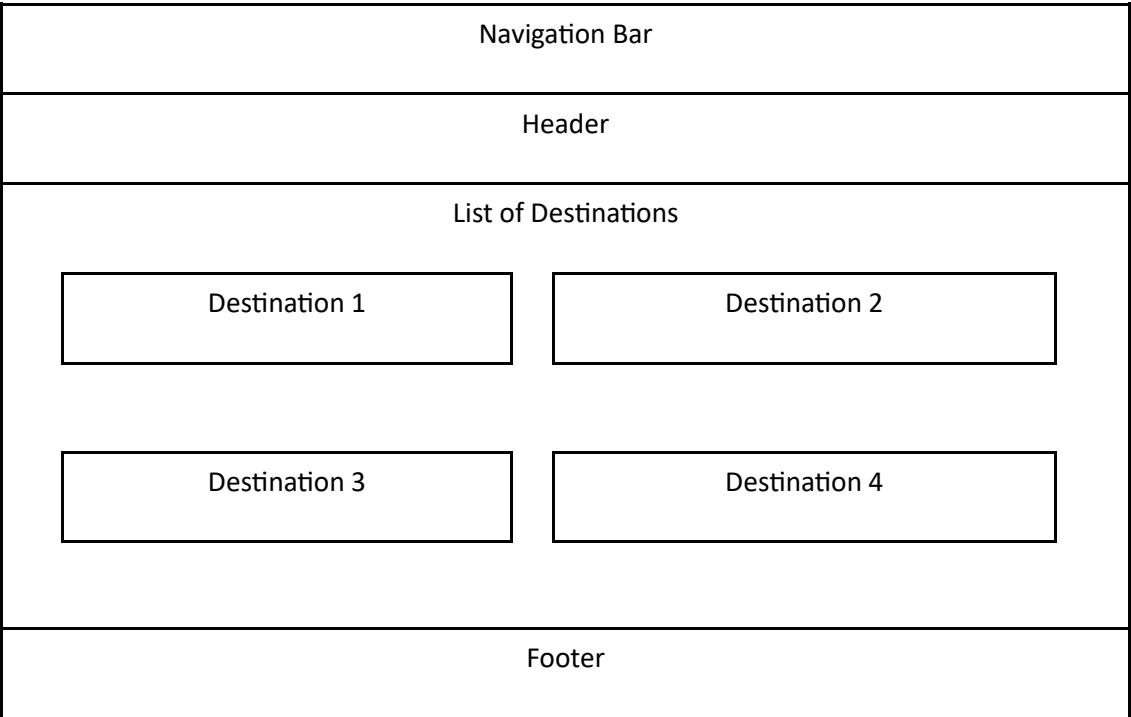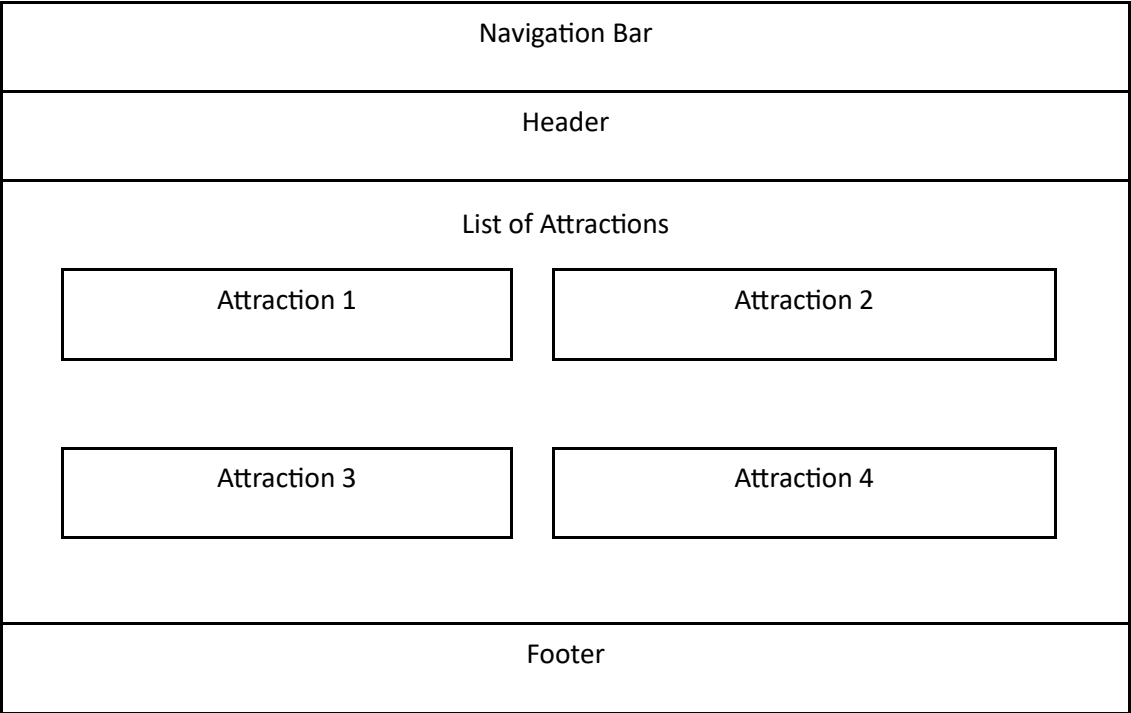| Navigation Bar |
|---|
| Header |
| List of Continents

Continent 1   Continent 2

Continent 3   Continent 4 |
| Footer |

Key Elements:

- **Navigation Bar:** Links to Home, Admin, About, and Contact pages.
- **Header**: Title of the page (e.g., "Trip Planner").
- **List of Continents**: Grid of continent cards with images and names.
- **Footer**: Additional links or information.

2. Countries Page

Description: The countries page displays a list of countries within the selected continent.

Wireframe:

Navigation Bar

Header

List of Countries

Country 1	Country 2
Country 3	Country 4

Footer

Key Elements:

- **Navigation Bar:** Links to Home, Admin, About, and Contact pages.

- **Header**: Title of the page (e.g., "Countries in [Continent Name]").

- **List of Countries**: Grid of country cards with images and names.

- **Footer**: Additional links or information.

3. Destinations Page

Description: The destinations page displays a list of destinations within the selected country.

Wireframe:

Navigation Bar

Header

List of Destinations

Destination 1	Destination 2

Destination 3	Destination 4

Footer

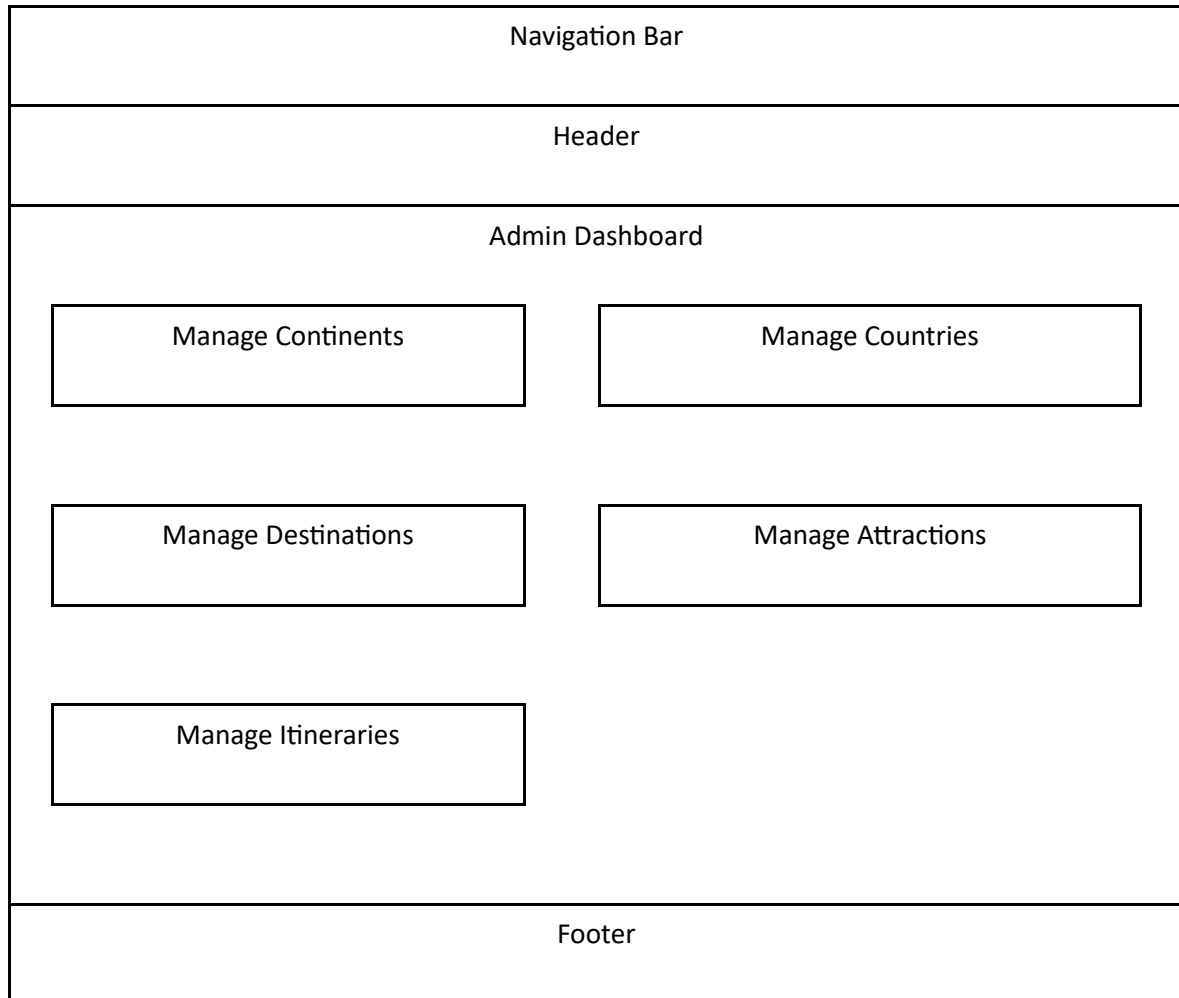**Key Elements:**
- **Navigation Bar:** Links to Home, Admin, About, and Contact pages.
- **Header**: Title of the page (e.g., "Destinations in [Country Name]").
- **List of Destinations:** Grid of destination cards with images and names.
- **Footer**: Additional links or information.

4. Attractions Page

Description: The attractions page displays a list of attractions for the selected destination.

Wireframe:

Navigation Bar

Header

List of Attractions

Attraction 1	Attraction 2
Attraction 3	Attraction 4

Footer

Key Elements:

- **Navigation Bar:** Links to Home, Admin, About, and Contact pages.
- **Header:** Title of the page (e.g., "Attractions in [Destination Name]").
- **List of Attractions:** Grid of attraction cards with images, names, and descriptions.
- **Footer:** Additional links or information.

5.   Itinerary Page

Description: The itinerary page displays the generated itinerary for the selected destination.

Wireframe:

Navigation Bar
Header
Itinerary Details

Day 1	Day 2
Day 3	Day 4

Footer

Key Elements:

- **Navigation Bar:** Links to Home, Admin, About, and Contact pages.

- **Header:** Title of the page (e.g., "Itinerary for [Destination Name]").

- **Itinerary Details:** Grid of itinerary items with day numbers and descriptions.

- **Footer:** Additional links or information.

6. Admin Dashboard

Description: The admin dashboard allows administrators to manage application data, including adding, updating, and deleting records.

Wireframe:

Navigation Bar
Header

Admin Dashboard

Manage Continents	Manage Countries
Manage Destinations	Manage Attractions
Manage Itineraries	

Footer

Key Elements:

- **Navigation Bar:** Links to Home, Admin, About, and Contact pages.

- **Header:** Title of the page (e.g., "Admin Dashboard").

- **Admin Dashboard:** Grid of management options for continents, countries, destinations, attractions, hotels, and itineraries.

- **Footer:** Additional links or information.

7. Admin Login

Description: The admin login page allows administrators to log in to access the admin dashboard.

Wireframe:

Navigation Bar
Header
Admin Login   **UserName**   **Password**   **Login Button**
Footer

Key Elements:

- **Navigation Bar:** Links to Home, Admin, About, and Contact pages.

- **Header:** Title of the page (e.g., "Admin Login").

- **Admin Login:** Form with fields for username and password, and a login button.

- **Footer:** Additional links or information.

Unit Testing

Creating unit test cases involves writing tests for individual functions or methods to ensure they work as expected. Below are some unit test cases for the Trip Planner Application, focusing on key functionalities such as fetching continents, countries, destinations, attractions and itineraries.

Test Case 1: Fetch Continents

- **Test Case ID**: TC-001

- **Description**: Verify that the system fetches the list of continents correctly.

- **Precondition**: The database contains continent data.

- **Test Steps**:

    1. Call the get_continents method.

- **Expected Result**: The method returns a list of Continent objects.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

Test Case 2: Fetch Countries by Continent

- **Test Case ID**: TC-002

- **Description**: Verify that the system fetches the list of countries for a given continent correctly.

- **Precondition**: The database contains country data.

- **Test Steps**:

    1. Call the get_countries method with a valid continent ID.

- **Expected Result**: The method returns a list of Country objects.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

Test Case 3: Fetch Destinations by Country

- **Test Case ID**: TC-003

- **Description**: Verify that the system fetches the list of destinations for a given country correctly.

- **Precondition**: The database contains destination data.

- **Test Steps**:

    1. Call the get_destinations method with a valid country ID.

- **Expected Result**: The method returns a list of Destination objects.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

Test Case 4: Fetch Attractions by Destination

- **Test Case ID**: TC-004

- **Description**: Verify that the system fetches the list of attractions for a given destination correctly.

- **Precondition**: The database contains attraction data.

- **Test Steps**:

    1. Call the get_attractions method with a valid destination ID.

- **Expected Result**: The method returns a list of Attraction objects.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

Test Case 5: Fetch Itinerary by Destination

- **Test Case ID**: TC-005

- **Description**: Verify that the system fetches the itinerary for a given destination correctly.

- **Precondition**: The database contains itinerary data.

- **Test Steps**:

    1. Call the get_itinerary method with a valid destination ID.

- **Expected Result**: The method returns a list of Itinerary objects.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

System Testing

Creating system test cases involves testing the entire system to ensure that all components work together as expected. Below are some system test cases for the Trip Planner Application, focusing on key functionalities such as viewing continents, countries, destinations, attractions and generating itineraries.

Test Case 1: View Continents

- **Test Case ID**: STC-001

- **Description**: Verify that the user can view the list of continents.

- **Precondition**: The database contains continent data.

- **Test Steps**:

    1. Navigate to the home page.

    2. Verify that the list of continents is displayed.

- **Expected Result**: The system displays a list of continents with images and names.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

Test Case 2: View Countries
- **Test Case ID**: STC-002

- **Description**: Verify that the user can view the list of countries for a selected continent.

- **Precondition**: The database contains country data.

- **Test Steps**:

    1. Navigate to the home page.

    2. Select a continent.

    3. Verify that the list of countries is displayed.

- **Expected Result**: The system displays a list of countries with images and names.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

Test Case 3: View Destinations
- **Test Case ID**: STC-003

- **Description**: Verify that the user can view the list of destinations for a selected country.

- **Precondition**: The database contains destination data.

- **Test Steps**:

    1. Navigate to the home page.

    2. Select a continent.

    3. Select a country.

    4. Verify that the list of destinations is displayed.

- **Expected Result**: The system displays a list of destinations with images and names.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

Test Case 4: View Attractions

- **Test Case ID**: STC-004

- **Description**: Verify that the user can view the list of attractions for a selected destination.

- **Precondition**: The database contains attraction data.

- **Test Steps**:

    1. Navigate to the home page.

    2. Select a continent.

    3. Select a country.

    4. Select a destination.

    5. Verify that the list of attractions is displayed.

- **Expected Result**: The system displays a list of attractions with images, names, and descriptions.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

Test Case 5: Generate Itinerary

- **Test Case ID**: STC-005

- **Description**: Verify that the user can generate an itinerary for a selected destination.

- **Precondition**: The database contains itinerary data.

- **Test Steps**:

    1. Navigate to the home page.

    2. Select a continent.

    3. Select a country.

    4. Select a destination.

    5. Click the "Generate Itinerary" button.

    6. Verify that the itinerary is displayed.

- **Expected Result**: The system displays the generated itinerary with day numbers and descriptions.

- **Actual Result**: (To be filled after testing)

- **Status**: (Pass/Fail)

Coding

SQL Commands for the Trip Planner Application

Database Creation (along with constraints)

The following SQL commands create the necessary tables for the Trip Planner Application, including primary keys, foreign keys, and other constraints to ensure data integrity.

```sql
-- Create Continents Table
CREATE TABLE IF NOT EXISTS continents (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    image_path TEXT
);

-- Create Countries Table
CREATE TABLE IF NOT EXISTS countries (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    continent_id INTEGER,
    image_path TEXT,
    FOREIGN KEY (continent_id) REFERENCES continents (id)
);

-- Create Destinations Table
CREATE TABLE IF NOT EXISTS destinations (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    country_id INTEGER,
    image_path TEXT,
    FOREIGN KEY (country_id) REFERENCES countries (id)
);

-- Create Attractions Table
CREATE TABLE IF NOT EXISTS attractions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    description TEXT,
    image_path TEXT,
    destination_id INTEGER,
    FOREIGN KEY (destination_id) REFERENCES destinations (id)
);

-- Create Hotels Table
CREATE TABLE IF NOT EXISTS hotels (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    description TEXT,
    image_path TEXT,
    destination_id INTEGER,
    FOREIGN KEY (destination_id) REFERENCES destinations (id)
```

```
);

-- Create Itineraries Table
CREATE TABLE IF NOT EXISTS itineraries (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    day INTEGER NOT NULL,
    description TEXT NOT NULL,
    destination_id INTEGER,
    FOREIGN KEY (destination_id) REFERENCES destinations (id)
);
```

Data Insertion in Tables

The following SQL commands insert initial data into the tables.

```
-- Insert initial data into continents table
INSERT INTO continents (name, image_path) VALUES ('Asia', 'images/asia.jpg');
INSERT INTO continents (name, image_path) VALUES ('Europe', 'images/europe.jpg');
INSERT INTO continents (name, image_path) VALUES ('America', 'images/america.jpg');

-- Insert initial data into countries table
INSERT INTO countries (name, continent_id, image_path) VALUES ('India', 1, 'images/india.jpg');
INSERT INTO countries (name, continent_id, image_path) VALUES ('France', 2, 'images/france.jpg');
INSERT INTO countries (name, continent_id, image_path) VALUES ('USA', 3, 'images/usa.jpg');

-- Insert initial data into destinations table
INSERT INTO destinations (name, country_id, image_path) VALUES ('Goa', 1, 'images/goa.jpg');
INSERT INTO destinations (name, country_id, image_path) VALUES ('Paris', 2, 'images/paris.jpg');
INSERT INTO destinations (name, country_id, image_path) VALUES ('New York', 3,
'images/newyork.jpg');

-- Insert initial data into attractions table
INSERT INTO attractions (name, description, image_path, destination_id) VALUES ('Beach',
'Beautiful beach', 'images/beach.jpg', 1);
INSERT INTO attractions (name, description, image_path, destination_id) VALUES ('Eiffel Tower',
'Iconic tower', 'images/eiffel.jpg', 2);
INSERT INTO attractions (name, description, image_path, destination_id) VALUES ('Statue of
Liberty', 'Famous statue', 'images/statue.jpg', 3);

-- Insert initial data into itineraries table
INSERT INTO itineraries (day, description, destination_id) VALUES (1, 'Arrival and check-in', 1);
INSERT INTO itineraries (day, description, destination_id) VALUES (2, 'City tour and sightseeing', 1);
INSERT INTO itineraries (day, description, destination_id) VALUES (3, 'Free day for personal
activities', 1);
```

Standardization of the Coding for the Trip Planner Application

Standardizing the coding practices ensures consistency, readability, maintainability, and collaboration among team members. Below are the coding standards and best practices applied to the Trip Planner Application.

Naming Conventions

Variables and Attributes

- Use descriptive names that clearly indicate the purpose of the variable or attribute.

- Use snake_case for variable and attribute names.

- Example: continent_name, country_id, image_path.

Functions and Methods

- Use descriptive names that clearly indicate the purpose of the function or method.

- Use snake_case for function and method names.

- Example: get_continents, add_country, fetch_attractions.

Classes

- Use CamelCase for class names.

- Example: Continent, Country, Destination.

Code Structure and Organization

Modules and Packages

- Organize code into modules and packages based on functionality.

- Example: Separate modules for database operations, API endpoints, and utility functions.

Indentation and Spacing

- Use 4 spaces for indentation.

- Separate functions and classes with two blank lines.

- Separate logical sections within functions with a single blank line.

Documentation and Comments

Docstrings

- Use docstrings to document classes, methods, and functions.

- Include a brief description, parameters, and return values.

- Example:

```
class Continent:
    """

    Represents a continent.

    Attributes:
        id (int): The unique identifier for the continent.
        name (str): The name of the continent.
        image_path (str): The path to the image representing the continent.
    """

    def __init__(self, id, name, image_path):
        """

        Initializes a Continent instance.

        Args:
            id (int): The unique identifier for the continent.
            name (str): The name of the continent.
            image_path (str): The path to the image representing the continent.
        """
        self.id = id
        self.name = name
        self.image_path = image_path
```

Inline Comments

- Use inline comments to explain complex or non-obvious code.

- Keep comments concise and relevant.

- Example:

```
def get_continents(self):
    """

    Fetches and returns a list of continents.

    Returns:
        list: A list of Continent objects.
    """
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute('SELECT id, name, image_path FROM continents')
    continents = [Continent(*row) for row in cursor.fetchall()]
    conn.close()
    return continents
```

Error Handling

- Use try-except blocks to handle exceptions and provide meaningful error messages.

- Log errors for debugging and troubleshooting.

- Example:

```
def add_continent(self, name, image_path):
    """
    Adds a new continent to the database.

    Args:
        name (str): The name of the continent.
        image_path (str): The path to the image representing the continent.

    Returns:
        str: A success or failure message.
    """
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    try:
        cursor.execute('INSERT INTO continents (name, image_path) VALUES (?, ?)', (name,
image_path))
        conn.commit()
        result = "Continent added successfully."
    except sqlite3.IntegrityError:
        result = "Error: Continent already exists."
    finally:
        conn.close()
    return result
```

Code Efficiency

- Use efficient algorithms and data structures to optimize performance.

- Avoid redundant code and unnecessary computations.

- Example:

```
def get_countries(self, continent_id):
    """
    Fetches and returns a list of countries for a given continent.

    Args:
        continent_id (int): The unique identifier for the continent.

    Returns:
        list: A list of Country objects.
    """
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    cursor.execute('SELECT id, name, continent_id, image_path FROM countries WHERE
continent_id = ?', (continent_id,))
    countries = [Country(*row) for row in cursor.fetchall()]
    conn.close()
    return countries
```

Parameters Calling/Passing
- Use clear and consistent parameter names.

- Pass parameters in a logical order.

- Example:

```python
def add_country(self, name, continent_id, image_path):
    """
    Adds a new country to the database.

    Args:
        name (str): The name of the country.
        continent_id (int): The unique identifier for the continent.
        image_path (str): The path to the image representing the country.

    Returns:
        str: A success or failure message.
    """
    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    try:
        cursor.execute('INSERT INTO countries (name, continent_id, image_path) VALUES (?, ?, ?)',
(name, continent_id, image_path))
        conn.commit()
        result = "Country added successfully."
    except sqlite3.IntegrityError:
        result = "Error: Country already exists."
    finally:
        conn.close()
    return result
```

Validation Checks
- Validate input data to ensure it meets the required criteria.

- Provide meaningful error messages for invalid input.

- Example:

```python
def add_destination(self, name, country_id, image_path):
    """
    Adds a new destination to the database.

    Args:
        name (str): The name of the destination.
        country_id (int): The unique identifier for the country.
        image_path (str): The path to the image representing the destination.

    Returns:
        str: A success or failure message.
    """
    if not name or not country_id:
```

```
        return "Error: Name and country ID are required."

    conn = sqlite3.connect(self.db_path)
    cursor = conn.cursor()
    try:
        cursor.execute('INSERT INTO destinations (name, country_id, image_path) VALUES (?, ?, ?)',
(name, country_id, image_path))
        conn.commit()
        result = "Destination added successfully."
    except sqlite3.IntegrityError:
        result = "Error: Destination already exists."
    finally:
        conn.close()
    return result
```

Testing

Testing Techniques and Testing Strategies Used

Testing is a crucial part of the software development process to ensure the quality, reliability, and performance of the application. Below are the testing techniques and strategies used in the Trip Planner Application.

Testing Techniques

1. Unit Testing

2. Integration Testing

3. System Testing

4. Acceptance Testing

5. Regression Testing

6. Performance Testing

Unit Testing

Unit testing involves testing individual components or functions of the application in isolation to ensure they work as expected.

Tools Used: unittest framework in Python

Example:

```python
import unittest
from trip_planner_app import TripPlannerApp, Continent

class TestTripPlannerApp(unittest.TestCase):
    def setUp(self):
        self.app = TripPlannerApp('trip_planner.db')

    def test_get_continents(self):
        continents = self.app.get_continents()
        self.assertIsInstance(continents, list)
        self.assertGreater(len(continents), 0)
        self.assertIsInstance(continents[0], Continent)

if __name__ == '__main__':
    unittest.main()
```

Integration Testing

Integration testing involves testing the interaction between different components or modules of the application to ensure they work together as expected.

Tools Used: unittest framework in Python

Example:

```
import unittest
from trip_planner_app import TripPlannerApp, Country

class TestTripPlannerIntegration(unittest.TestCase):
    def setUp(self):
        self.app = TripPlannerApp('trip_planner.db')

    def test_get_countries_and_destinations(self):
        countries = self.app.get_countries(1)  # Assuming 1 is a valid continent ID
        self.assertGreater(len(countries), 0)
        for country in countries:
            destinations = self.app.get_destinations(country.id)
            self.assertIsInstance(destinations, list)

if __name__ == '__main__':
    unittest.main()
```

System Testing

System testing involves testing the entire application as a whole to ensure it meets the specified requirements and works as expected in the target environment.

Tools Used: Selenium WebDriver, unittest framework in Python

Example:

```
import unittest
from selenium import webdriver

class TestTripPlannerSystem(unittest.TestCase):
    def setUp(self):
        self.driver = webdriver.Chrome()

    def test_view_continents(self):
        driver = self.driver
        driver.get("http://localhost:5000/")
        continents = driver.find_elements_by_class_name("continent-item")
        self.assertGreater(len(continents), 0)
        for continent in continents:
            self.assertTrue(continent.find_element_by_tag_name("img"))
            self.assertTrue(continent.find_element_by_tag_name("p"))

    def tearDown(self):
        self.driver.quit()

if __name__ == "__main__":
    unittest.main()
```

Acceptance Testing

Acceptance testing involves verifying that the application meets the business requirements and is ready for deployment. It is typically performed by the end-users or stakeholders.

Tools Used: Manual testing

Example:

- Verify that the user can view the list of continents, countries, destinations, attractions, and hotels.

- Verify that the user can generate an itinerary for a selected destination.

Regression Testing

Description: Regression testing involves re-running previously conducted tests to ensure that new changes or enhancements have not introduced any new defects.

Tools Used: unittest framework in Python, Selenium WebDriver

Example:

- Re-run unit tests and system tests after adding a new feature or fixing a bug to ensure existing functionality is not affected.

Performance Testing

Performance testing involves testing the application's performance under various conditions, such as load, stress, and scalability.

Tools Used: Apache JMeter

Example:

- Simulate multiple users accessing the application simultaneously to test its performance and identify any bottlenecks.

Testing Strategies

Test Planning

Test planning involves defining the scope, objectives, resources, schedule, and approach for testing activities.

Activities:

- Define the scope and objectives of testing.

- Identify the resources required for testing.

- Create a test schedule and timeline.

- Define the testing approach and techniques to be used.

Test Design

Description: Test design involves creating detailed test cases and test scripts based on the requirements and design specifications.

Activities:

- Identify test scenarios and create test cases.

- Define the expected results for each test case.

- Create test scripts for automated testing.

Test Execution

Description: Test execution involves running the test cases and test scripts to verify the functionality of the application.

Activities:

- Execute test cases and test scripts.

- Record the actual results and compare them with the expected results.

- Identify and report any defects or issues.

Defect Reporting and Tracking

Description: Defect reporting and tracking involve documenting and managing defects identified during testing.

Activities:

- Log defects in a defect tracking system.

- Assign defects to the appropriate team members for resolution.

- Track the status of defects until they are resolved.

Test Closure

Description: Test closure involves finalizing the testing activities and preparing test closure reports.

Activities:

- Review and analyze the test results.

- Prepare test closure reports and documentation.

- Conduct a test closure meeting to discuss the findings and lessons learned.

Test Plan

1. Introduction

1.1 Purpose

The purpose of this test plan is to outline the testing strategy, objectives, scope, resources, schedule, and deliverables for the Trip Planner Application. This document serves as a guide for the testing team to ensure that the application meets the specified requirements and functions as expected.

1.2 Objectives

- Verify that the application meets the functional and non-functional requirements.

- Ensure that the application is free of defects and performs reliably.

- Validate that the application provides a user-friendly experience.

- Confirm that the application integrates seamlessly with external APIs and databases.

2. Scope

2.1 In-Scope

- Functional testing of key features such as viewing continents, countries, destinations, attractions and generating itineraries.

- Integration testing to ensure seamless interaction between different modules.

- System testing to validate the overall functionality of the application.

- Acceptance testing to verify that the application meets business requirements.

- Regression testing to ensure that new changes do not introduce defects.

- Performance testing to assess the application's performance under various conditions.

2.2 Out-of-Scope

- Testing of third-party APIs beyond their integration with the application.

- Testing of non-functional requirements such as security and compliance.

3. Testing Strategy

3.1 Testing Techniques

- Unit Testing

- Integration Testing

- System Testing

- Acceptance Testing

- Regression Testing

- Performance Testing

3.2 Testing Tools

- unittest framework in Python for unit and integration testing.

- Selenium WebDriver for system and acceptance testing.

- Apache JMeter and Locust for performance testing.

4. Test Environment

4.1 Hardware

- Development and testing machines with sufficient processing power and memory.

- Web server to host the application.

4.2 Software

- Operating System: Windows/Linux/MacOS

- Web Browser: Chrome, Firefox, Safari

- Database: SQLite

- Development Environment: Python, Flask

- Testing Tools: unittest, Selenium WebDriver, Apache JMeter, Locust

5. Test Schedule

Activity	Start Date	End Date	Responsible
Test Planning	15/10/2024	16/10/2024	Test Lead
Test Design	17/10/2024	25/10/2024	Test Team
Test Environment Setup	11/11/2024	15/11/2024	DevOps Team
Unit Testing	18/11/2024	19/11/2024	Dev Team
Integration Testing	19/11/2024	20/11/2024	Test Team
System Testing	20/11/2024	21/11/2023	Test Team
Acceptance Testing	21/11/2024	22/11/2024	Stakeholders
Regression Testing	25/11/2024	25/11/2024	Test Team
Performance Testing	25/11/2024	25/11/2024	Test Team
Test Closure	26/11/2024	26/11/2024	Test Lead

6. Test Deliverables

- Test Plan Document

- Test Cases and Test Scripts

- Test Data

- Test Execution Reports

- Defect Reports

- Test Summary Report

- Test Closure Report

7. Test Cases

7.1 Unit Test Cases

Test Case ID	Description	Precondition	Test Steps	Expected Result
TC-001	Fetch Continents	Database contains continent data	Call the get_continents method.	Returns a list of Continent objects.
TC-002	Fetch Countries by Continent	Database contains country data	Call the get_countries method with a valid continent ID.	Returns a list of Country objects.
TC-003	Fetch Destinations by Country	Database contains destination data	Call the get_destinations method with a valid country ID.	Returns a list of Destination objects.
TC-004	Fetch Attractions by Destination	Database contains attraction data	Call the get_attractions method with a valid destination ID.	Returns a list of Attraction objects.
TC-005	Fetch Itinerary by Destination	Database contains itinerary data	Call the get_itinerary method with a valid destination ID.	Returns a list of Itinerary objects.

7.2 System Test Cases

Test Case ID	Description	Precondition	Test Steps	Expected Result
STC-001	View Continents	Database contains continent data	Navigate to the home page. Verify that the list of continents is displayed.	Displays a list of continents with images and names.
STC-002	View Countries	Database contains country data	Navigate to the home page. Select a continent. Verify that the list of countries is displayed.	Displays a list of countries with images and names.
STC-003	View Destinations	Database contains destination data	Navigate to the home page. Select a continent. Select a country. Verify that the list of destinations is displayed.	Displays a list of destinations with images and names.
STC-004	View Attractions	Database contains attraction data	Navigate to the home page. Select a continent. Select a country. Select a destination. Verify that the list of attractions is displayed.	Displays a list of attractions with images, names, and descriptions.
STC-005	Generate Itinerary	Database contains itinerary data	Navigate to the home page. Select a continent. Select a country. Select a destination. Click the "Generate Itinerary" button. Verify that the itinerary is displayed.	Displays the generated itinerary with day numbers and descriptions.

8. Defect Management

8.1 Defect Reporting

- Use a defect tracking system (e.g., JIRA, Bugzilla) to log and manage defects.

- Include detailed information such as defect ID, description, steps to reproduce, severity, priority, and status.

8.2 Defect Tracking

- Assign defects to the appropriate team members for resolution.

- Track the status of defects from identification to resolution.

- Conduct regular defect review meetings to discuss and prioritize defects.

9. Test Closure

9.1 Test Closure Activities

- Review and analyze the test results.

- Prepare test closure reports and documentation.

- Conduct a test closure meeting to discuss the findings and lessons learned.

9.2 Test Closure Report

- Summary of testing activities and results.

- List of defects identified and their status.

- Recommendations for future testing and improvements.

Test reports for Unit Test Cases

TC ID	Description	Precondition	Test Steps	Expected Result	Actual result	Status
TC-001	Fetch Continents	Database contains continent data	Call the get_continents method.	Returns a list of Continent objects.	A list of Continent objects is returned	Pass
TC-002	Fetch Countries by Continent	Database contains country data	Call the get_countries method with a valid continent ID.	Returns a list of Country objects.	A list of Country objects is returned	Pass
TC-003	Fetch Destinations by Country	Database contains destination data	Call the get_destinations method with a valid country ID.	Returns a list of Destination objects.	A list of Destination objects is returned	Pass
TC-004	Fetch Attractions by Destination	Database contains attraction data	Call the get_attractions method with a valid destination ID.	Returns a list of Attraction objects.	A list of Attraction objects is returned	Pass
TC-	Fetch Itinerary by	Database contains	Call the get_itinerary method	Returns a list of Itinerary objects.	A list of Itinerary objects is returned	Pass

005	Destination	itinerary data	with a valid destination ID.			

Test reports for System Test Cases

Test Case ID	Description	Precondition	Test Steps	Expected Result	Actual result	Status
STC-001	View Continents	Database contains continent data	Navigate to the home page. Verify that the list of continents is displayed.	Displays a list of continents with images and names.	A list of continents with images and names is displayed.	Pass
STC-002	View Countries	Database contains country data	Navigate to the home page. Select a continent. Verify that the list of countries is displayed.	Displays a list of countries with images and names.	A list of countries with images and names is displayed.	Pass
STC-003	View Destinations	Database contains destination data	Navigate to the home page. Select a continent. Select a country. Verify that the list of destinations is displayed.	Displays a list of destinations with images and names.	A list of destinations with images and names is displayed.	Pass
STC-004	View Attractions	Database contains attraction data	Navigate to the home page. Select a continent. Select a country. Select a destination.	Displays a list of attractions with images, names, and descriptions.	A list of attractions with images, names, and descriptions is displayed.	Pass

			Verify that the list of attractions is displayed.			
STC-005	Generate Itinerary	Database contains itinerary data	Navigate to the home page. Select a continent. Select a country. Select a destination. Click the "Generate Itinerary" button. Verify that the itinerary is displayed.	Displays the generated itinerary with day numbers and descriptions.	The generated itinerary with day numbers and descriptions is displayed.	Pass

Cost Estimation and its Model

The cost model provides a straightforward approach to estimating the cost of a project based on key factors such as the number of team members, their hourly rates, and the estimated duration of the project. Below is a cost model for the Trip Planner Application.

Key Factors

1. Number of Team Members

2. Hourly Rate

3. Estimated Duration

Estimation for the Trip Planner Application

Step 1: Determine the Number of Team Members

Assume the project team consists of the following members:

- 1 Project Manager

- 1 Developers

- 1 Designer

- 1 QA Engineer

Total number of team members: 4

Step 2: Determine the Hourly Rate

Assume the average hourly rate for each team member is as follows:

- Project Manager: $80/hour

- Developer: $60/hour

- Designer: $50/hour

- QA Engineer: $40/hour

Step 3: Estimate the Duration

Assume the estimated duration of the project is 2 months (approximately 8 weeks).

Step 4: Calculate the Total Hours

Assume each team member works 40 hours per week.

Total hours per team member: [8 weeks x 40 hours = 1040 hours]

Step 5: Calculate the Cost for Each Role

- **Project Manager**: [1040 hours x $80 = $83,200]

- **Developers**: [1040 hours x $60 = $62,400]

- **Designer**: [1040 hours x $50 = $52,000]

- **QA Engineer**: [1040 hours x $40 = $41,600]

Step 6: Calculate the Total Cost

Total cost: [$83,200 (Project Manager) + $62,400 (Developer) + $52,000 (Designer) + $41,600 (QA Engineer)]

[Total Cost = $239,200]

Summary of Cost Estimation

- **Number of Team Members**: 5

- **Hourly Rate**:

  - Project Manager: $80/hour

  - Developer: $60/hour

  - Designer: $50/hour

  - QA Engineer: $40/hour

- **Estimated Duration**: 2 months (8 weeks)

- **Total Cost**: $239,200

Future Scope

The Trip Planner Application has the potential for significant enhancements and expansions to improve user experience, increase functionality, and cater to a broader audience. Below are some potential future scope areas for the application:

User Authentication and Personalization

User Authentication

- **Description**: Implement user authentication to allow users to create accounts, log in, and access personalized features.

- **Benefits**: Enhances security, enables personalized experiences, and allows users to save their preferences and itineraries.

User Profiles

- **Description**: Allow users to create and manage their profiles, including personal information, travel preferences, and saved itineraries.

- **Benefits**: Provides a personalized experience and allows users to easily access their saved data.

Advanced Search and Filtering

Advanced Search

- **Description**: Implement advanced search functionality to allow users to search for destinations, attractions, and hotels based on various criteria such as location, price, ratings, and amenities.

- **Benefits**: Improves user experience by making it easier to find relevant information.

Filtering Options

- **Description**: Provide filtering options to allow users to narrow down search results based on specific criteria.

- **Benefits**: Enhances user experience by allowing users to quickly find what they are looking for.

Reviews and Ratings

User Reviews

- **Description**: Allow users to leave reviews and ratings for destinations, attractions, and hotels.

- **Benefits**: Provides valuable feedback for other users and helps improve the quality of the content.

Rating System

- **Description**: Implement a rating system to allow users to rate destinations, attractions, and hotels based on their experiences.

- **Benefits**: Helps users make informed decisions based on the experiences of others.

Social Sharing and Collaboration

Social Media Integration

- **Description**: Integrate social media sharing options to allow users to share their itineraries and travel experiences on platforms like Facebook, Twitter, and Instagram.

- **Benefits**: Increases user engagement and promotes the application through social media.

Collaborative Planning

- **Description**: Allow users to collaborate with friends and family to plan trips together.

- **Benefits**: Enhances user experience by enabling group planning and decision-making.

Mobile Application

Mobile App Development

- **Description**: Develop a mobile application for iOS and Android to provide users with a seamless experience on their smartphones and tablets.

- **Benefits**: Increases accessibility and convenience for users who prefer using mobile devices.

Offline Access

- **Description**: Implement offline access to allow users to view their itineraries and travel information without an internet connection.

- **Benefits**: Enhances user experience by providing access to important information even in areas with limited connectivity.

Integration with Travel Services

Flight and Hotel Booking

- **Description**: Integrate with flight and hotel booking services to allow users to book their travel arrangements directly through the application.

- **Benefits**: Provides a one-stop solution for trip planning and booking, increasing convenience for users.

Local Transportation and Activities

- **Description**: Integrate with local transportation and activity booking services to allow users to book transportation and activities at their destination.

- **Benefits**: Enhances user experience by providing comprehensive travel planning options.

Enhanced Itinerary Features

Customizable Itineraries

- **Description**: Allow users to customize their itineraries by adding, removing, or rearranging activities and attractions.

- **Benefits**: Provides flexibility and personalization, allowing users to create itineraries that suit their preferences.

Real-Time Updates

- **Description**: Provide real-time updates to itineraries based on factors such as weather, traffic, and availability.

- **Benefits**: Enhances user experience by providing up-to-date information and helping users adjust their plans accordingly.

Multilingual Support

Language Options

- **Description**: Implement multilingual support to provide the application in multiple languages.

- **Benefits**: Increases accessibility and usability for users from different linguistic backgrounds.

Bibliography

The following sources were referenced and utilized in the development of the Trip Planner Application and the associated documentation:

Books and Articles

1. **"Software Engineering: A Practitioner's Approach" by Roger S. Pressman**

   o This book provided foundational knowledge on software engineering principles, methodologies, and best practices.

2. **"The Pragmatic Programmer: Your Journey to Mastery" by Andrew Hunt and David Thomas**

   o This book offered practical advice on coding standards, software design, and development practices.

3. **"Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides**

   o This book was referenced for understanding and implementing design patterns in the application.

Online Resources

1. **Flask Documentation**

   o URL: https://flask.palletsprojects.com/

   o The official Flask documentation was used extensively for understanding the framework and implementing the backend of the application.

2. **SQLite Documentation**

   o URL: https://www.sqlite.org/docs.html

   o The SQLite documentation provided guidance on database design, SQL commands, and database management.

3. **OpenAI API Documentation**

   o URL: https://beta.openai.com/docs/

   o The OpenAI API documentation was referenced for integrating AI-based itinerary generation (if applicable).

4. **Selenium WebDriver Documentation**

   o URL: https://www.selenium.dev/documentation/en/

   o The Selenium WebDriver documentation was used for implementing automated system and acceptance testing.

5. **Apache JMeter Documentation**

    o   URL: https://jmeter.apache.org/usermanual/index.html

    o   The Apache JMeter documentation provided guidance on performance testing and load testing.

Appendices

Source Code

app.py

```python
from flask import Flask, render_template, request, redirect, url_for, flash, session, jsonify
import sqlite3
import os
import openai

app = Flask(__name__)
app.secret_key = 'your_secret_key'  # Replace with your actual secret key

ADMIN_PASSCODE = '1234'  # Replace with your actual passcode

def get_continents():
    db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('SELECT id, name, image_path FROM continents')
    continents = cursor.fetchall()
    conn.close()
    return continents

def get_countries():
    db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('SELECT id, name FROM countries')
    countries = cursor.fetchall()
    conn.close()
    return countries

def get_destinations():
    db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('SELECT id, name FROM destinations')
    destinations = cursor.fetchall()
    conn.close()
    return destinations

def get_countries_by_continent(continent_id):
    db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('''
        SELECT countries.id, countries.name, countries.image_path
        FROM countries
        WHERE countries.continent_id = ?
    ''', (continent_id,))
    countries = cursor.fetchall()
```

```
    conn.close()
    return countries

def get_destinations_by_country(country_id):
    db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('''
        SELECT destinations.id, destinations.name, destinations.image_path
        FROM destinations
        WHERE destinations.country_id = ?
    ''', (country_id,))
    destinations = cursor.fetchall()
    conn.close()
    return destinations

def get_all_countries_and_destinations():
    db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('''
        SELECT countries.id, countries.name, destinations.id, destinations.name
        FROM countries
        JOIN destinations ON countries.id = destinations.country_id
    ''')
    data = cursor.fetchall()
    countries = {}
    for country_id, country_name, destination_id, destination_name in data:
        if country_id not in countries:
            countries[country_id] = [country_name, []]
        countries[country_id][1].append((destination_id, destination_name))
    conn.close()
    return countries

def get_attractions_by_destination(destination_id):
    db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
    cursor.execute('''
        SELECT name, description, image_path
        FROM attractions
        WHERE destination_id = ?
    ''', (destination_id,))
    attractions = cursor.fetchall()
    conn.close()
    return attractions

def get_itineraries_by_destination(destination_id):
    db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
    conn = sqlite3.connect(db_path)
    cursor = conn.cursor()
```

```
    cursor.execute('''
        SELECT day, description
        FROM itineraries
        WHERE destination_id = ?
    ''', (destination_id,))
    itineraries = cursor.fetchall()
    conn.close()
    return itineraries


@app.route('/')
def index():
    continents = get_continents()
    return render_template('index.html', continents=continents)


@app.route('/countries/<int:continent_id>')
def countries(continent_id):
    countries = get_countries_by_continent(continent_id)
    return render_template('countries.html', countries=countries)


@app.route('/destinations/<int:country_id>')
def destinations(country_id):
    destinations = get_destinations_by_country(country_id)
    return render_template('destinations.html', destinations=destinations)


@app.route('/attractions/<int:destination_id>')
def attractions(destination_id):
    attractions = get_attractions_by_destination(destination_id)
    return jsonify({'attractions': [{'name': attraction[0], 'description': attraction[1], 'image_path':
attraction[2]} for attraction in attractions]})


@app.route('/itineraries/<int:destination_id>')
def itineraries(destination_id):
    itineraries = get_itineraries_by_destination(destination_id)
    return jsonify({'itineraries': [{'day': itinerary[0], 'description': itinerary[1]} for itinerary in
itineraries]})


@app.route('/admin', methods=['GET', 'POST'])
def admin():
    if request.method == 'POST':
        if 'passcode' in request.form:
            passcode = request.form['passcode']
            if passcode == ADMIN_PASSCODE:
                session['admin_authenticated'] = True
                return redirect(url_for('admin'))
            else:
                flash('Incorrect passcode. Please try again.')
                return redirect(url_for('admin'))
        if not session.get('admin_authenticated'):
            flash('You must enter the passcode to access the admin page.')
            return redirect(url_for('admin'))
        if 'continent_name' in request.form and 'continent_image_path' in request.form:
```

```python
        continent_name = request.form['continent_name']
        continent_image_path = request.form['continent_image_path']
        db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()
        cursor.execute('''
            INSERT INTO continents (name, image_path)
            VALUES (?, ?)
        ''', (continent_name, continent_image_path))
        conn.commit()
        conn.close()
        flash('Continent added successfully!')
    elif 'country_name' in request.form and 'continent_id' in request.form and
'country_image_path' in request.form:
        country_name = request.form['country_name']
        continent_id = request.form['continent_id']
        country_image_path = request.form['country_image_path']
        db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()
        cursor.execute('''
            INSERT INTO countries (name, continent_id, image_path)
            VALUES (?, ?, ?)
        ''', (country_name, continent_id, country_image_path))
        conn.commit()
        conn.close()
        flash('Country added successfully!')
    elif 'destination_name' in request.form and 'country_id' in request.form and
'destination_image_path' in request.form:
        destination_name = request.form['destination_name']
        country_id = request.form['country_id']
        destination_image_path = request.form['destination_image_path']
        db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()
        cursor.execute('''
            INSERT INTO destinations (name, country_id, image_path)
            VALUES (?, ?, ?)
        ''', (destination_name, country_id, destination_image_path))
        conn.commit()
        conn.close()
        flash('Destination added successfully!')
    elif 'attraction_name' in request.form and 'attraction_description' in request.form and
'destination_id' in request.form and 'attraction_image_path' in request.form:
        attraction_name = request.form['attraction_name']
        attraction_description = request.form['attraction_description']
        destination_id = request.form['destination_id']
        attraction_image_path = request.form['attraction_image_path']
        db_path = os.path.join(os.path.dirname(__file__), 'db', 'trip_planner.db')
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()
```

Project Final Report – AI-Based-Virtual-Travel-Guide

```python
        cursor.execute('''
            INSERT INTO attractions (name, description, image_path, destination_id)
            VALUES (?, ?, ?, ?)
        ''', (attraction_name, attraction_description, attraction_image_path, destination_id))
        conn.commit()
        conn.close()
        flash('Attraction added successfully!')
    elif 'itinerary_day' in request.form and 'itinerary_description' in request.form and
'destination_id' in request.form:
        itinerary_day = request.form['itinerary_day']
        itinerary_description = request.form['itinerary_description']
        destination_id = request.form['destination_id']
        db_path = os.path.join(os.path.dirname(__file__), 'src/db', 'trip_planner.db')
        conn = sqlite3.connect(db_path)
        cursor = conn.cursor()
        cursor.execute('''
            INSERT INTO itineraries (day, description, destination_id)
            VALUES (?, ?, ?)
        ''', (itinerary_day, itinerary_description, destination_id))
        conn.commit()
        conn.close()
        flash('Itinerary added successfully!')
    return redirect(url_for('admin'))
    if not session.get('admin_authenticated'):
        return render_template('admin_login.html')
    continents = get_continents()
    countries = get_countries()
    destinations = get_destinations()
    all_countries_and_destinations = get_all_countries_and_destinations()
    #session.pop('admin_authenticated', None)  # Clear the session variable after each request
    return render_template('admin.html', continents=continents, countries=countries,
destinations=destinations, all_countries_and_destinations=all_countries_and_destinations)


@app.route('/about')
def about():
    return render_template('about.html')


@app.route('/contact', methods=['GET', 'POST'])
def contact():
    if request.method == 'POST':
        # Handle form submission here (e.g., save to database, send email, etc.)
        name = request.form['name']
        email = request.form['email']
        message = request.form['message']
        # For now, just print the form data to the console
        print(f"Name: {name}, Email: {email}, Message: {message}")
        return redirect(url_for('contact'))
    return render_template('contact.html')


if __name__ == '__main__':
    app.run(debug=True)
```

P a g e 79 | 125

db.py

```
import sqlite3
import os

# Ensure the db directory exists
os.makedirs('./src/db', exist_ok=True)

# Connect to SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect('./src/db/trip_planner.db')

# Create a cursor object
cursor = conn.cursor()

# Create tables
cursor.execute('''
CREATE TABLE IF NOT EXISTS continents (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    image_path TEXT
)
''')

cursor.execute('''
CREATE TABLE IF NOT EXISTS countries (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    continent_id INTEGER,
    image_path TEXT,
    FOREIGN KEY (continent_id) REFERENCES continents (id)
)
''')

cursor.execute('''
CREATE TABLE IF NOT EXISTS destinations (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    country_id INTEGER,
    image_path TEXT,
    FOREIGN KEY (country_id) REFERENCES countries (id)
)
''')

cursor.execute('''
CREATE TABLE IF NOT EXISTS attractions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    description TEXT,
    image_path TEXT,
    destination_id INTEGER,
```

```
    FOREIGN KEY (destination_id) REFERENCES destinations (id)
)
''')

# Insert initial data into continents table
continents = [
    ('Asia', 'images/asia.jpg'),
    ('Europe', 'images/europe.jpg'),
    ('America', 'images/america.jpg')
]

cursor.executemany('''
INSERT INTO continents (name, image_path)
VALUES (?, ?)
''', continents)

# Insert initial data into countries table
countries = [
    ('Japan', 1, 'images/japan.jpg'),
    ('China', 1, 'images/china.jpg'),
    ('Thailand', 1, 'images/thailand.jpg'),
    ('France', 2, 'images/france.jpg'),
    ('Germany', 2, 'images/germany.jpg'),
    ('Italy', 2, 'images/italy.jpg'),
    ('USA', 3, 'images/usa.jpg'),
    ('Canada', 3, 'images/canada.jpg'),
    ('Brazil', 3, 'images/brazil.jpg')
]

cursor.executemany('''
INSERT INTO countries (name, continent_id, image_path)
VALUES (?, ?, ?)
''', countries)

# Insert initial data into destinations table
destinations = [
    ('Tokyo', 1, 'images/tokyo.jpg'),
    ('Beijing', 2, 'images/beijing.jpg'),
    ('Bangkok', 3, 'images/bangkok.jpg'),
    ('Paris', 4, 'images/paris.jpg'),
    ('Berlin', 5, 'images/berlin.jpg'),
    ('Rome', 6, 'images/rome.jpg'),
    ('New York', 7, 'images/new_york.jpg'),
    ('Toronto', 8, 'images/toronto.jpg'),
    ('Rio de Janeiro', 9, 'images/rio.jpg')
]

cursor.executemany('''
INSERT INTO destinations (name, country_id, image_path)
VALUES (?, ?, ?)
''', destinations)
```

```
# Insert initial data into attractions table
attractions = [
    ('Tokyo Tower', 'A communications and observation tower in the Shiba-koen district of Minato,
Tokyo, Japan.', 'images/tokyo_tower.jpg', 1),
    ('Great Wall of China', 'A series of fortifications made of stone, brick, tamped earth, wood, and
other materials.', 'images/great_wall_of_china.jpg', 2),
    ('Grand Palace', 'A complex of buildings at the heart of Bangkok, Thailand.',
'images/grand_palace.jpg', 3),
    ('Eiffel Tower', 'A wrought-iron lattice tower on the Champ de Mars in Paris, France.',
'images/eiffel_tower.jpg', 4),
    ('Brandenburg Gate', 'An 18th-century neoclassical monument in Berlin, Germany.',
'images/brandenburg_gate.jpg', 5),
    ('Colosseum', 'An ancient amphitheater in Rome, Italy.', 'images/colosseum.jpg', 6),
    ('Statue of Liberty', 'A colossal neoclassical sculpture on Liberty Island in New York Harbor.',
'images/statue_of_liberty.jpg', 7),
    ('CN Tower', 'A 553.3 m-high concrete communications and observation tower in downtown
Toronto, Ontario, Canada.', 'images/cn_tower.jpg', 8),
    ('Christ the Redeemer', 'An Art Deco statue of Jesus Christ in Rio de Janeiro, Brazil.',
'images/christ_the_redeemer.jpg', 9)
]

cursor.executemany('''
INSERT INTO attractions (name, description, image_path, destination_id)
VALUES (?, ?, ?, ?)
''', attractions)

# Commit the changes and close the connection
conn.commit()
conn.close()

print("Database and tables created successfully.")
```

index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smart Guide</title>
    <style>
        .tab {
            display: inline-block;
            margin-left: 20px;
        }
    </style>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
    <script>
        function handleClick(continentId) {
```

```
            fetch(`/countries/${continentId}`)
                .then(response => response.text())
                .then(html => {
                    document.getElementById('country-list').innerHTML = html;
                });
        }
    </script>
</head>
<body>
    <header>
        <nav class="navbar">
            <a href="/">Home</a>
            <a href="/admin">Admin</a>
            <a href="/about">About</a>
            <a href="/contact">Contact</a>
        </nav>
        <h1 style="color: Green"><span class="tab"></span>Smart Guide</h1>
        <h4 style="color: brown"><span class="tab"></span>The AI-based Virtual Travel Guide</h4>
    </header>
    <main>
        <section id="button-click">
            <h2 style="color: Orange">Select the continent to proceed:</h2>
            <div class="continent-list">
                {% for continent in continents %}
                    <div class="continent-item" onclick="handleClick({{ continent[0] }})" style="cursor:
pointer;">
                        <img src="{{ url_for('static', filename=continent[2]) }}" alt="{{ continent[1] }}"
class="continent-image">
                        <p class="continent-name">{{ continent[1] }}</p>
                    </div>
                {% endfor %}
            </div>
        </section>
        <section id="country-list">
            <h2></h2>
        </section>
    </main>
    {% include 'footer.html' %}
</body>
</html>
```

admin.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Page</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
```

```html
<!--<style>
  .submenu {
    display: none;
  }
  .submenu.active {
    display: block;
  }
  .submenu-nav a {
    margin-right: 10px;
    cursor: pointer;
    text-decoration: underline;
    color: blue;
  }
</style>-->
<script>
  function showSubMenu(subMenuId) {
    document.querySelectorAll('.submenu').forEach(subMenu => {
      subMenu.classList.remove('active');
    });
    document.getElementById(subMenuId).classList.add('active');
  }
</script>
</head>
<body>
  <header>
    <nav class="navbar">
      <a href="/">Home</a>
      <a href="/admin">Admin</a>
      <a href="/about">About</a>
      <a href="/contact">Contact</a>
    </nav>
    <!--<h1>Admin Page</h1>-->
  </header>
  <main>
    <div class="submenu-nav">
      <a onclick="showSubMenu('add-continent')">Add Continent</a>
      <a onclick="showSubMenu('add-country')">Add Country</a>
      <a onclick="showSubMenu('add-destination')">Add Destination</a>
      <a onclick="showSubMenu('add-attraction')">Add Attraction</a>
    </div>
    <div id="add-continent" class="submenu">
      <h2>Add Continent</h2>
      <form action="{{ url_for('admin') }}" method="post">
        <label for="continent_name">Continent Name:</label>
        <input type="text" id="continent_name" name="continent_name" required>
        <label for="continent_image_path">Image Path:</label>
        <input type="text" id="continent_image_path" name="continent_image_path" required>
        <button type="submit">Add Continent</button>
      </form>
    </div>
    <div id="add-country" class="submenu">
```

```html
<h2>Add Country</h2>
<form action="{{ url_for('admin') }}" method="post">
  <label for="country_name">Country Name:</label>
  <input type="text" id="country_name" name="country_name" required>
  <label for="continent_id">Continent:</label>
  <select id="continent_id" name="continent_id" required>
    {% for continent in continents %}
      <option value="{{ continent[0] }}">{{ continent[1] }}</option>
    {% endfor %}
  </select>
  <label for="country_image_path">Image Path:</label>
  <input type="text" id="country_image_path" name="country_image_path" required>
  <button type="submit">Add Country</button>
</form>
</div>
<div id="add-destination" class="submenu">
  <h2>Add Destination</h2>
  <form action="{{ url_for('admin') }}" method="post">
    <label for="destination_name">Destination Name:</label>
    <input type="text" id="destination_name" name="destination_name" required>
    <label for="country_id">Country:</label>
    <select id="country_id" name="country_id" required>
      {% for country in countries %}
        <option value="{{ country[0] }}">{{ country[1] }}</option>
      {% endfor %}
    </select>
    <label for="destination_image_path">Image Path:</label>
    <input type="text" id="destination_image_path" name="destination_image_path"
required>
    <button type="submit">Add Destination</button>
  </form>
</div>
<div id="add-attraction" class="submenu">
  <h2>Add Attraction</h2>
  <form action="{{ url_for('admin') }}" method="post">
    <label for="attraction_name">Attraction Name:</label>
    <input type="text" id="attraction_name" name="attraction_name" required>
    <label for="attraction_description">Description:</label>
    <textarea id="attraction_description" name="attraction_description"
required></textarea>
    <label for="destination_id">Destination:</label>
    <select id="destination_id" name="destination_id" required>
      {% for destination in destinations %}
        <option value="{{ destination[0] }}">{{ destination[1] }}</option>
      {% endfor %}
    </select>
    <label for="attraction_image_path">Image Path:</label>
    <input type="text" id="attraction_image_path" name="attraction_image_path" required>
    <button type="submit">Add Attraction</button>
  </form>
</div>
```

```
      {% with messages = get_flashed_messages() %}
      {% if messages %}
        <div class="flash-messages">
          {% for message in messages %}
            <p>{{ message }}</p>
          {% endfor %}
        </div>
      {% endif %}
    {% endwith %}
    </main>
    {% include 'footer.html' %}
</body>
</html>
```

destinations.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Travel Attractions</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <header>
      <nav class="navbar">
        <a href="/">Home</a>
        <a href="/admin">Admin</a>
        <a href="/about">About</a>
        <a href="/contact">Contact</a>
      </nav>
      <h1>Travel Attractions in {{ destination_name }}</h1>
    </header>
    <main>
      <section>
        <h2>Top Attractions</h2>
        <div class="attraction-list">
          {% for attraction in attractions %}
            <div class="attraction-item">
              <img src="{{ url_for('static', filename=attraction[2]) }}" alt="{{ attraction[0] }}"
class="attraction-image">
              <h3>{{ attraction[0] }}</h3>
              <p>{{ attraction[1] }}</p>
            </div>
          {% endfor %}
        </div>
      </section>
    </main>
    {% include 'footer.html' %}
```

```
</body>
</html>
```

Admin_login.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Admin Login</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <header>
    <nav class="navbar">
      <a href="/">Home</a>
      <a href="/admin">Admin</a>
      <a href="/about">About</a>
      <a href="/contact">Contact</a>
    </nav>
    <h1>Admin Login</h1>
  </header>
  <main>
    {% with messages = get_flashed_messages() %}
      {% if messages %}
        <div class="flash-messages">
          {% for message in messages %}
            <p>{{ message }}</p>
          {% endfor %}
        </div>
      {% endif %}
    {% endwith %}
    <form action="{{ url_for('admin') }}" method="post">
      <label for="passcode">Enter Passcode:</label>
      <input type="password" id="passcode" name="passcode" required>
      <button type="submit">Submit</button>
    </form>
  </main>
  {% include 'footer.html' %}
</body>
</html>
```

about.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>About Us</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <header>
        <nav class="navbar">
            <a href="/">Home</a>
            <a href="/admin">Admin</a>
            <a href="/about">About</a>
            <a href="/contact">Contact</a>
        </nav>
        <!--<h1>About Us</h1>-->
    </header>
    <main>
        <section>
            <h2>Welcome to the Smart Guide</h2>
            <p>Our mission is to help you plan your perfect trip. Whether you're looking for adventure,
relaxation, or cultural experiences, we've got you covered.</p>
            <p>Explore various destinations, manage your itinerary, and keep track of your budget all in
one place.</p>
        </section>
        <section>
            <h1></h1>
        </section>
    </main>
    {% include 'footer.html' %}
</body>
</html>
```

contact.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Contact Us</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <header>
        <nav class="navbar">
            <a href="/">Home</a>
            <a href="/admin">Admin</a>
            <a href="/about">About</a>
            <a href="/contact">Contact</a>
        </nav>
        <!--<h1>Contact Us</h1>-->
    </header>
    <main>
```
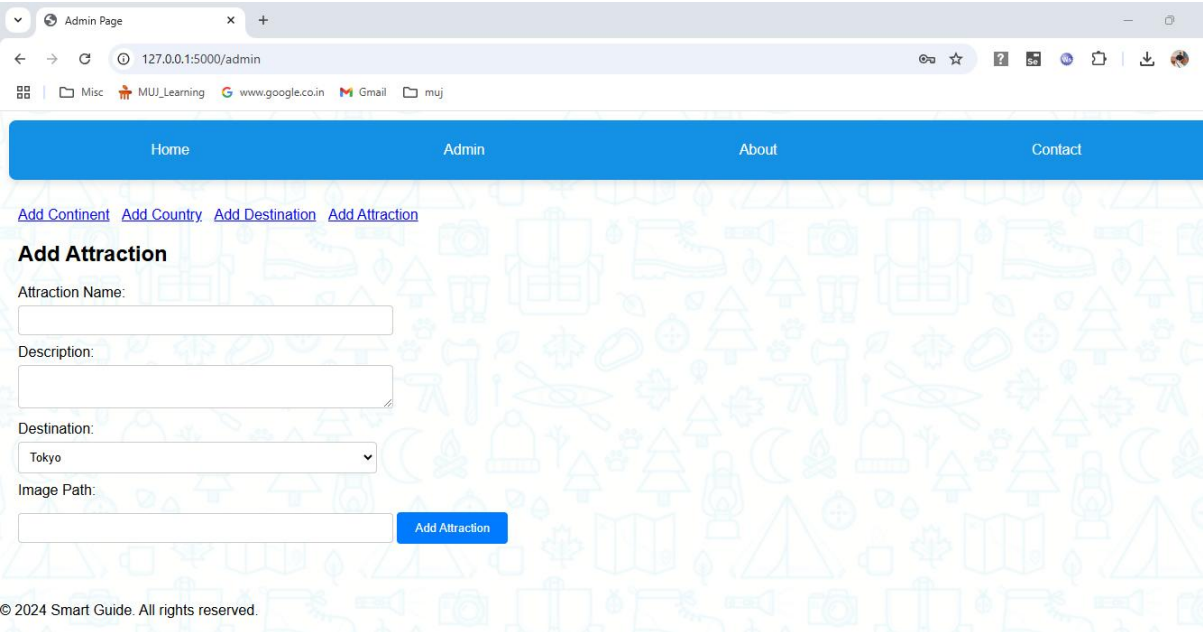
```html
    <section>
      <h2>Get in Touch</h2>
      <p>If you have any questions or need assistance, please feel free to contact us.</p>
      <form action="/contact" method="post">
        <label for="name">Name:</label>
        <input type="text" id="name" name="name" required>
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required>
        <label for="message">Message:</label>
        <textarea id="message" name="message" required></textarea>
        <p></p>
        <button type="submit">Send Message</button>
      </form>
    </section>
  </main>
  {% include 'footer.html' %}
</body>
</html>
```

footer.html

```html
<!-- footer.html -->
<footer>
  <p>&copy; 2024 Smart Guide. All rights reserved.</p>
</footer>
```

styles.css

```css
body {
    background-image: url('/static/images/background.png'); /* Path to your background image */
    background-size: auto; /* Ensure the image covers the entire background */
    background-position: center; /* Center the background image */
    background-repeat: repeat; /* Prevent the background image from repeating */
    font-family: Arial, sans-serif; /* Set a default font */
    margin: 0;
    padding: 0;
}

.navbar {
    display: flex;
    justify-content: space-around;
    background-color: #1491e4; /* Windows 11 blue color */
    padding: 1rem;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /* Shadow */
    border-radius: 8px; /* Rounded corners */
    margin: 10px; /* Margin around the navbar */
}

.navbar a {
    color: white;
```

```css
    text-decoration: none;
    padding: 0.5rem 1rem;
    transition: background-color 0.3s, color 0.3s, transform 0.3s;
    border-radius: 4px; /* Rounded corners for links */
}

.navbar a:hover {
    background-color: #005A9E; /* Darker blue on hover */
    color: white;
    transform: translateY(-2px); /* Lift effect */
    box-shadow: 0 6px 12px rgba(0, 0, 0, 0.2); /* Enhanced shadow */
}

.continent-list {
    display: flex;
    flex-direction: row;
    gap: 20px;
    align-items: center;
}

.continent-image {
    width: 150px; /* Set the desired width */
    height: 150px; /* Set the desired height */
    object-fit: cover; /* Ensure the image covers the area without distortion */
    transition: transform 0.3s ease;
}

.continent-image:hover {
    transform: scale(1.2);
}
.continent-name {
    text-align: center;
    margin-top: 10px;
}

.country-list {
    display: flex;
    flex-wrap: nowrap;
    overflow-x: auto;
    gap: 20px;
}

.country-item {
    text-align: center;
    flex: 0 0 auto;
}

.country-image {
    width: 150px; /* Set the desired width */
    height: 150px; /* Set the desired height */
    object-fit: cover; /* Ensure the image covers the area without distortion */
```

```css
    transition: transform 0.3s ease;
}

.country-image:hover {
    transform: scale(1.2);
}

.destination-list {
    display: flex;
    flex-wrap: wrap;
    gap: 20px;
}

.destination-item {
    text-align: center;
    border: 2px solid #ccc; /* Adjust the color and thickness of the border as needed */
    padding: 15px; /* Add some padding inside the border */
    margin: 10px 0; /* Add some space between items */
    border-radius: 8px; /* Optional: Add rounded corners */
    transition: all 0.3s ease; /* Optional: Add a transition effect for hover */
}
.destination-item:hover {
    border-color: #555; /* Change border color on hover */
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* Optional: Add a shadow on hover */
}

.destination-image {
    width: 150px; /* Set the desired width */
    height: 150px; /* Set the desired height */
    object-fit: cover; /* Ensure the image covers the area without distortion */
    transition: transform 0.3s ease;
    max-width: 100%; /* Ensure the image fits within the border */
    border-radius: 8px; /* Optional: Same radius as the container */

}

.destination-image:hover {
    transform: scale(1.2);

}

.attractions-overlay {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
    background-color: rgba(0, 0, 0, 0.7); /* Black background with transparency */
    color: white;
    display: flex;
    flex-direction: column;
```

```css
        justify-content: center;
        align-items: center;
        opacity: 0;
        transition: opacity 0.3s ease;
}

.destination-image-container:hover .attractions-overlay {
        opacity: 1;
}

main {
        padding: 20px; /* Add padding to the main content */
}

#itinerary-section {
        margin-top: 20px;
        background-color: rgba(255, 255, 255, 0.8); /* White background with transparency */
        padding: 20px;
        border-radius: 8px; /* Rounded corners */
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /* Shadow */
}

#itinerary-details {
        display: flex;
        flex-wrap: wrap;
        gap: 20px;
}

.day-item {
        flex: 1 1 calc(33.333% - 20px); /* Three columns with gap */
        background-color: rgba(255, 255, 255, 0.8); /* White background with transparency */
        padding: 10px;
        box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /* Shadow */
        text-align: center;
        border: 2px solid #007BFF;
        border-color: #007BFF;
        border-radius: 8px; /* Rounded corners */
}

#itinerary-details p {
        margin: 10px 0;
}

#attractions-section {
        margin-top: 20px;
}

.attraction-list {
        flex: 1 1 calc(33.333% - 20px); /* Three columns with gap */
        display: flex;
        flex-wrap: wrap;
```

```css
    gap: 20px;
}

.attraction-item {
    flex: 1 1 calc(33.333% - 20px); /* Three columns with gap */
    background-color: rgba(255, 255, 255, 0.8); /* White background with transparency */
    padding: 10px;
    border-radius: 8px; /* Rounded corners */
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1); /* Shadow */
    margin-bottom: 10px;
    text-align: left;
}

.attraction-image {
    margin: 0 auto; /* Center align*/
    width: 150px; /* Set the desired width */
    height: 150px; /* Set the desired height */
    object-fit: cover; /* Ensure the image covers the area without distortion */
    transition: transform 0.3s ease;
}

.icon-image {
    width: 30px; /* Set the desired width */
    height:30px; /* Set the desired height */
    object-fit: cover; /* Ensure the image covers the area without distortion */
    transition: transform 0.3s ease;
    margin-left: 5px; /* Space between the icon and the text */
    vertical-align: middle; /* Align icons with text */
}

.icon-image:hover {
    transform: scale(1.2);
}

.attraction-image:hover {
    transform: scale(1.2);
}

form {
    margin-bottom: 20px;
}

label {
    display: block;
    margin-bottom: 5px;
}

input[type="text"], textarea, select {
    width: 30%;
    padding: 8px;
    margin-bottom: 10px;
```

```css
    border: 1px solid #ccc;
    border-radius: 4px;
}

button {
    display: inline-block;
    margin-top: 10px;
    padding: 10px 20px;
    background-color: #007BFF;
    color: white;
    text-decoration: none;
    border: none;
    border-radius: 4px;
    cursor: pointer;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #0056b3;
}

.flash-messages {
    background-color: #d4edda;
    color: #155724;
    border: 1px solid #c3e6cb;
    padding: 10px;
    border-radius: 4px;
    margin-bottom: 20px;
}

.submenu {
    display: none;
}

.submenu.active {
    display: block;
}

.submenu-nav a {
    margin-right: 10px;
    cursor: pointer;
    text-decoration: underline;
    color: blue;
}

.hotel-search-button {
    display: inline-block;
    margin-top: 10px;
    padding: 10px 20px;
    background-color: #007BFF;
    color: white;
```

```css
    text-decoration: none;
    border-radius: 4px;
    transition: background-color 0.3s;
}

.send-message-button {
    display: inline-block;
    margin-top: 10px;
    padding: 10px 20px;
    background-color: #007BFF;
    color: white;
    text-decoration: none;
    border-radius: 4px;
    transition: background-color 0.3s;
}
.hotel-search-button:hover {
    background-color: #0056b3;
}

.icon-link {
    display: inline-block;
    margin-top: 10px;
    color: #007BFF;
    text-decoration: none;
    font-size: 1.2em;
    transition: color 0.3s;
}

.icon-link:hover {
    color: #0056b3;
}
```

Project Screenshots

Home Page

Admin Login Page
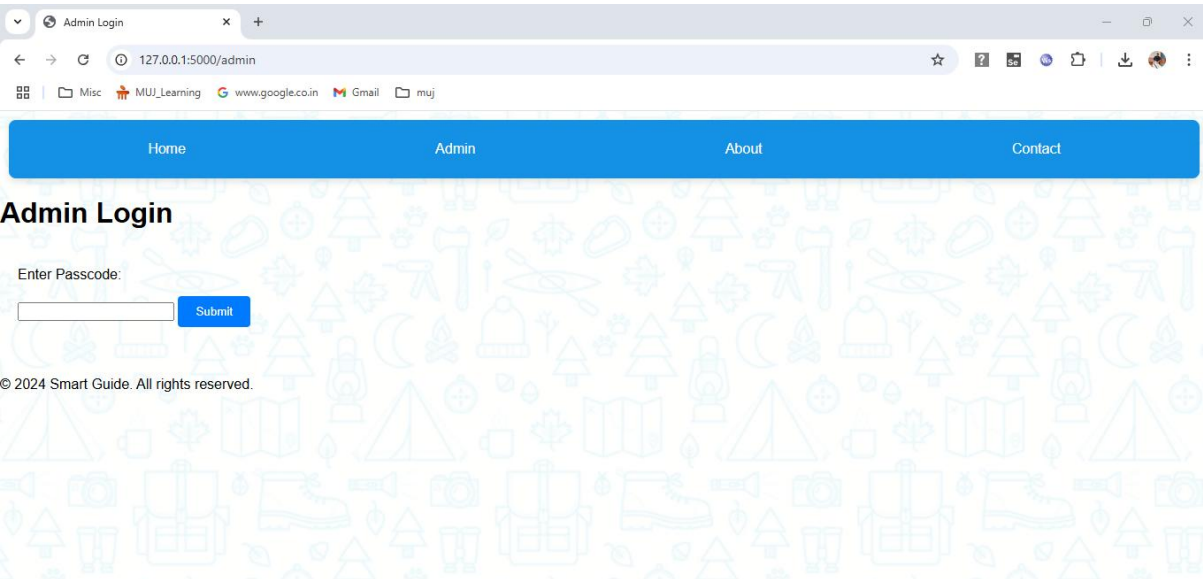
Admin page
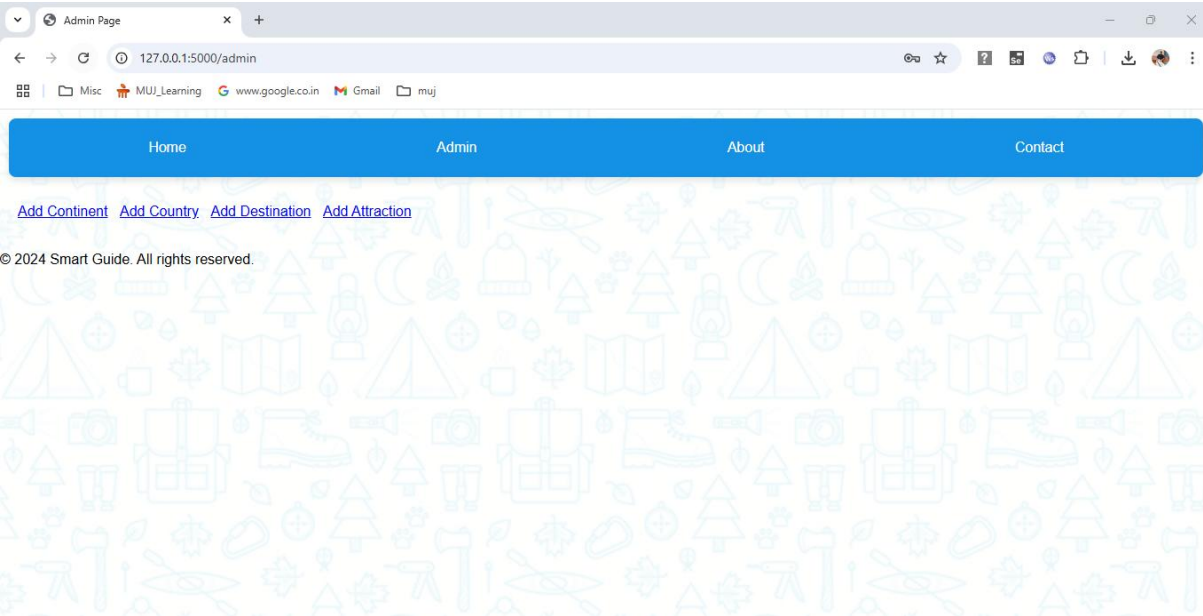
Add Continent
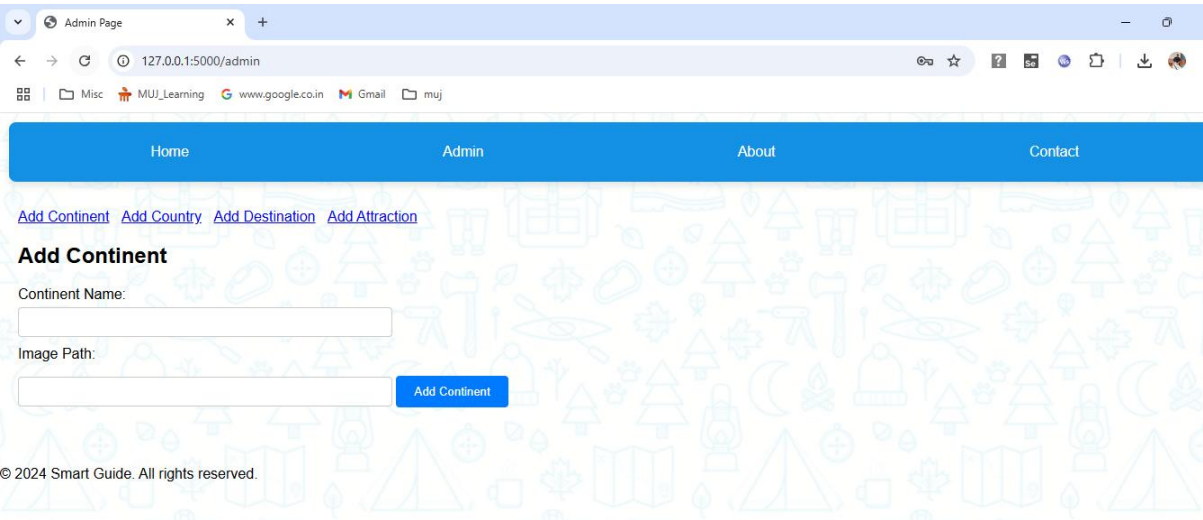
Add Country

Add Destination

Add Attraction

Countries Listing Page
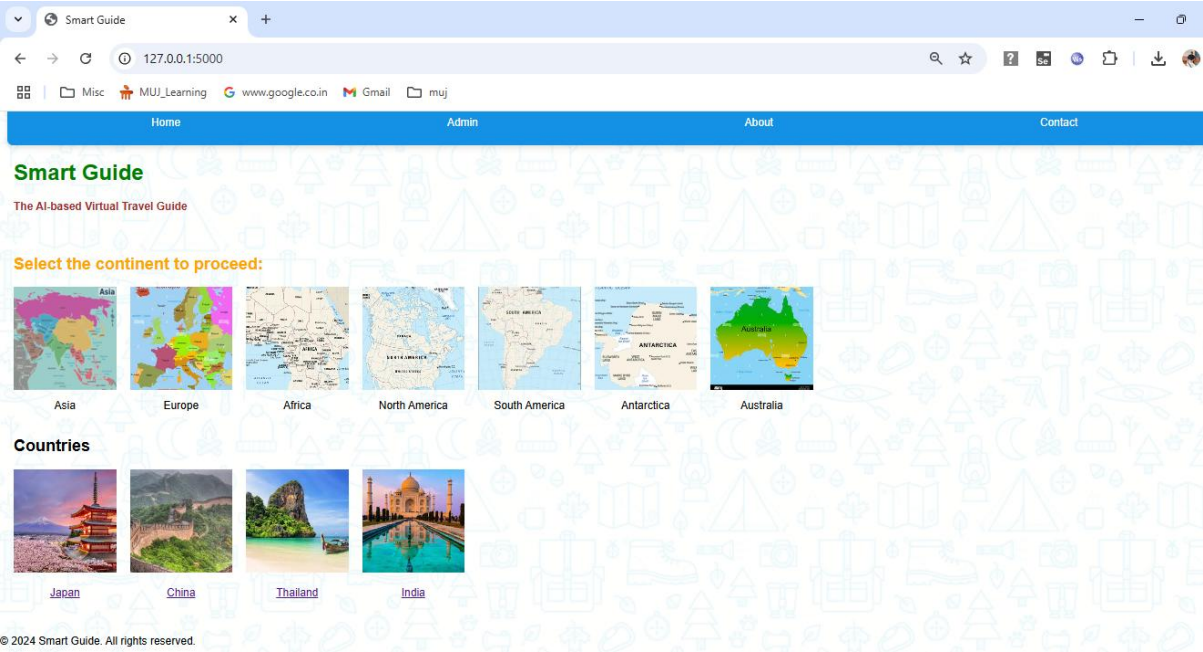
Destinations Page
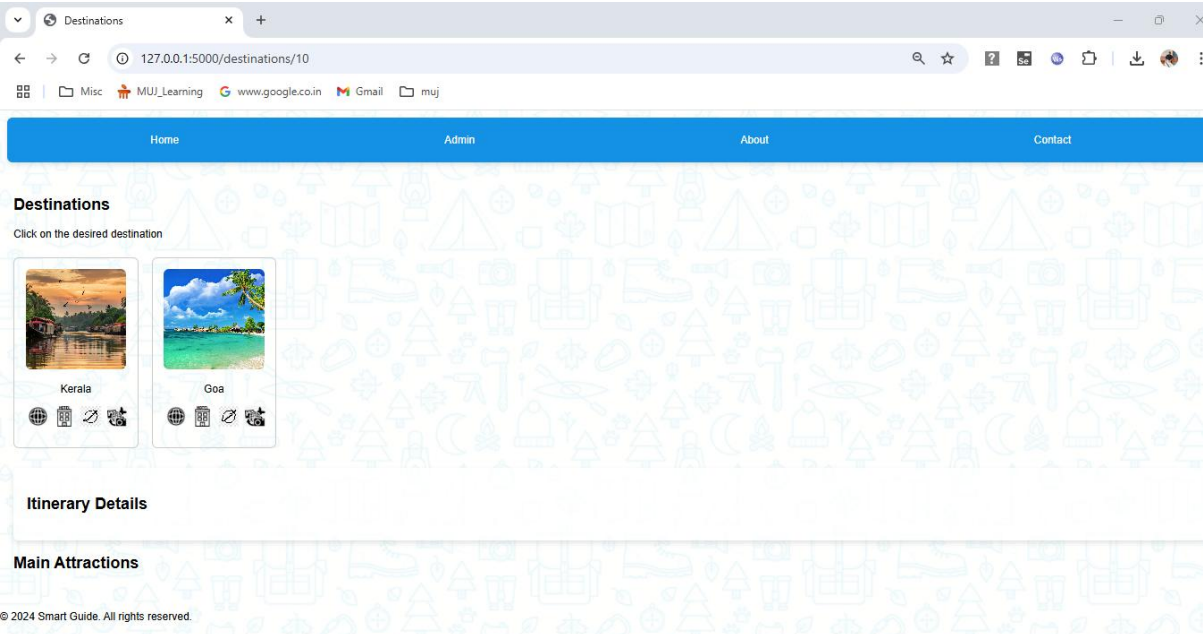
View Attractions and itinerary
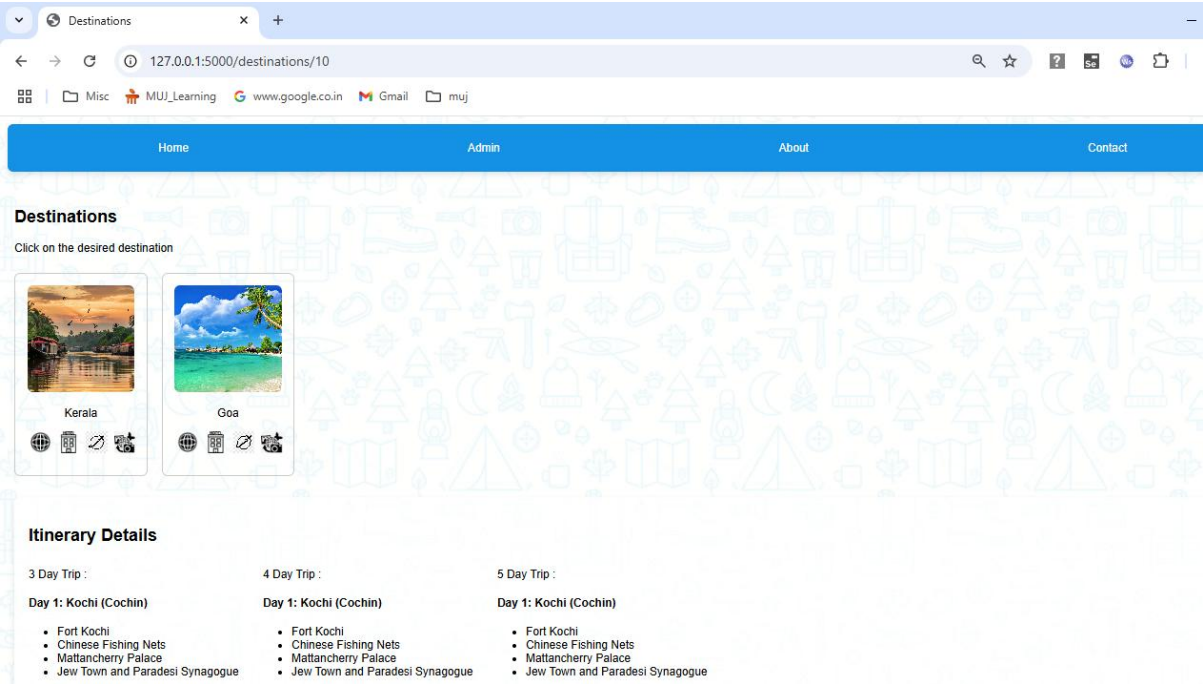
Itinerary Details

3 Day Trip :	4 Day Trip :	5 Day Trip :
Day 1: Kochi (Cochin)	**Day 1: Kochi (Cochin)**	**Day 1: Kochi (Cochin)**
• Fort Kochi	• Fort Kochi	• Fort Kochi
• Chinese Fishing Nets	• Chinese Fishing Nets	• Chinese Fishing Nets
• Mattancherry Palace	• Mattancherry Palace	• Mattancherry Palace
• Jew Town and Paradesi Synagogue	• Jew Town and Paradesi Synagogue	• Jew Town and Paradesi Synagogue
Day 2: Alleppey (Alappuzha) Backwaters	**Day 2: Alleppey (Alappuzha) Backwaters**	**Day 2: Alleppey (Alappuzha) Backwaters**
• Houseboat Cruise	• Houseboat Cruise	• Houseboat Cruise
• Marari Beach	• Marari Beach	• Marari Beach
• Alleppey Beach	• Alleppey Beach	• Alleppey Beach
Day 3: Munnar	**Day 3: Munnar**	**Day 3: Munnar**
• Tea Plantations	• Tea Plantations	• Tea Plantations
• Eravikulam National Park	• Eravikulam National Park	• Eravikulam National Park
• Top Station	• Top Station	• Top Station
	Day 4: Thekkady	**Day 4: Thekkady**
	• Periyar Wildlife Sanctuary	• Periyar Wildlife Sanctuary
	• Spice Plantations	• Spice Plantations
	• Periyar Lake	• Periyar Lake
		Day 5: Kovalam and Trivandrum
		• Kovalam Beach
		• Padmanabhaswamy Temple
		• Napier Museum
		• Shanghumukham Beach

Main Attractions

Fort Kochi

A charming seaside area, Fort Kochi is known for its Dutch, Portuguese, and British colonial architecture, and elaborate bamboo fishing nets at Fort Kochi Beach. St. Francis Church was the original burial site of explorer Vasco da Gama. Upscale eateries and chic cafes serve Keralan specialties, and quaint shops sell cotton clothing and handmade souvenirs. Heritage buildings house contemporary art galleries.

Mattancherry Palace

The Mattancherry Palace is a palace popularly known as the Dutch Palace, in Mattancherry, Kochi, in the Indian state of Kerala which features Kerala murals depicting portraits and exhibits of the Rajas of Kochi. The palace was included in the tentative list of UNESCO World Heritage Sites

Eravikulam National Park

Eravikulam National Park is a 97 km² national park located along the Western Ghats in the Idukki and Ernakulam districts of Kerala in India. The park is situated between 10°05'N and 10°20' north, and 77°0' and 77°10' east and is the first national park in Kerala. It was established in 1978

Periyar Wildlife Sanctuary

Periyar National Park, also known as Periyar Tiger Reserve, is in the mountainous Western Ghats of Kerala, southern India. This wildlife sanctuary is home to tigers and a significant elephant population, as well as rare lion-tailed macaques, sambar deer, leopards and Indian bison. In the park's north, Periyar Lake is popular for boat rides. Farther north, spice plantations surround the town of Kumily.

Kovalam Beach

Kovalam is an internationally renowned beach with three adjacent crescent beaches. It has been a favourite haunt of tourists since the 1930s. A massive rocky promontory on the beach has created a beautiful bay of calm waters ideal for sea bathing. The leisure options at this beach are plenty and

Padmanabhaswamy Temple

The Padmanabhaswamy Temple is a Hindu temple dedicated to Vishnu in Thiruvananthapuram, the capital of the state of Kerala, India. It is one of the 108 Divya Desams which are considered the sacred abodes of Vishnu in the Sri Vaishnava tradition.

Shanghumukham Beach

Shankumugham Beach is a beach in Thiruvananthapuram district of Kerala, south India. The beach is on the western side of Thiruvananthapuram and is very close to Thiruvananthapuram International Airport. Shanghumugham beach is considered as the 'Katturkadavu' of Sri Ananthapadmanabhan, the

About Page:
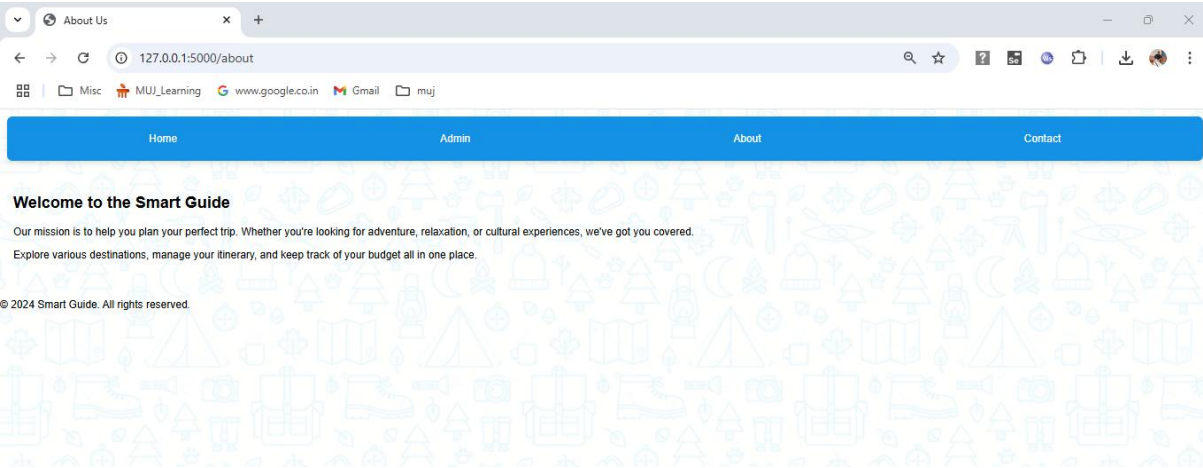
Contact Us Page
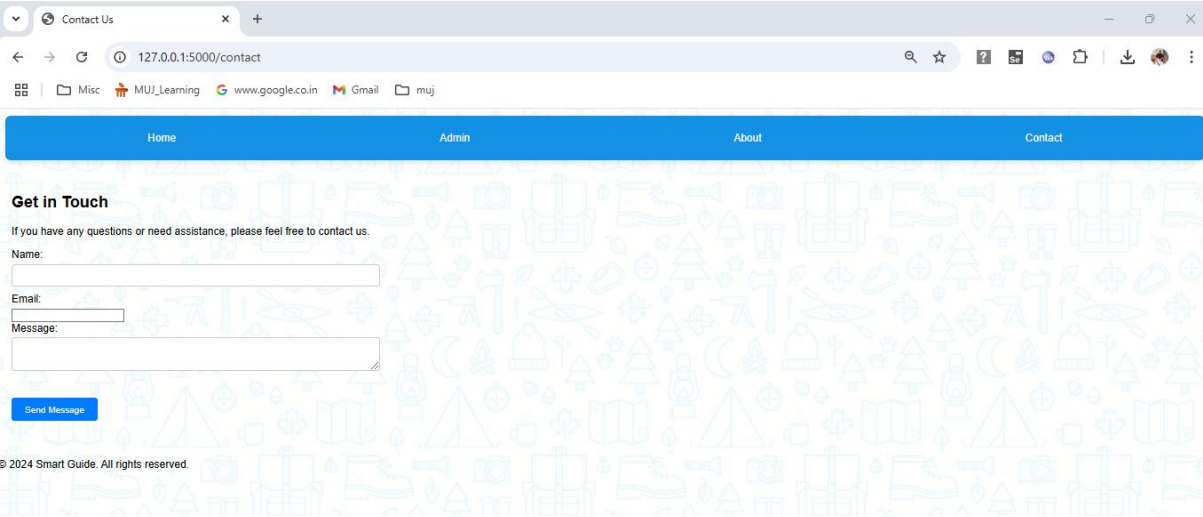
Intended Blank Page

Project Presentation

Project Presentation

On

AI Based Virtual Travel Guide

AI BASED VIRTUAL TRAVEL GUIDE

NAME: JACOB BABY

ROLL NO: 2214100795

PROGRAM: BACHELOR OF COMPUTER APPLICATIONS (BCA)

SEMESTER: VI

SESSION: MARCH 2024

DATE: 10 DECEMBER 2024

AGENDA

- Introduction
- Technologies used
- Data flow diagram(DFD)
- Control Flow Diagram (CFD)
- Entity Relationship Diagram (ERD)
- Project Pages
- Future Scope
- Conclusion
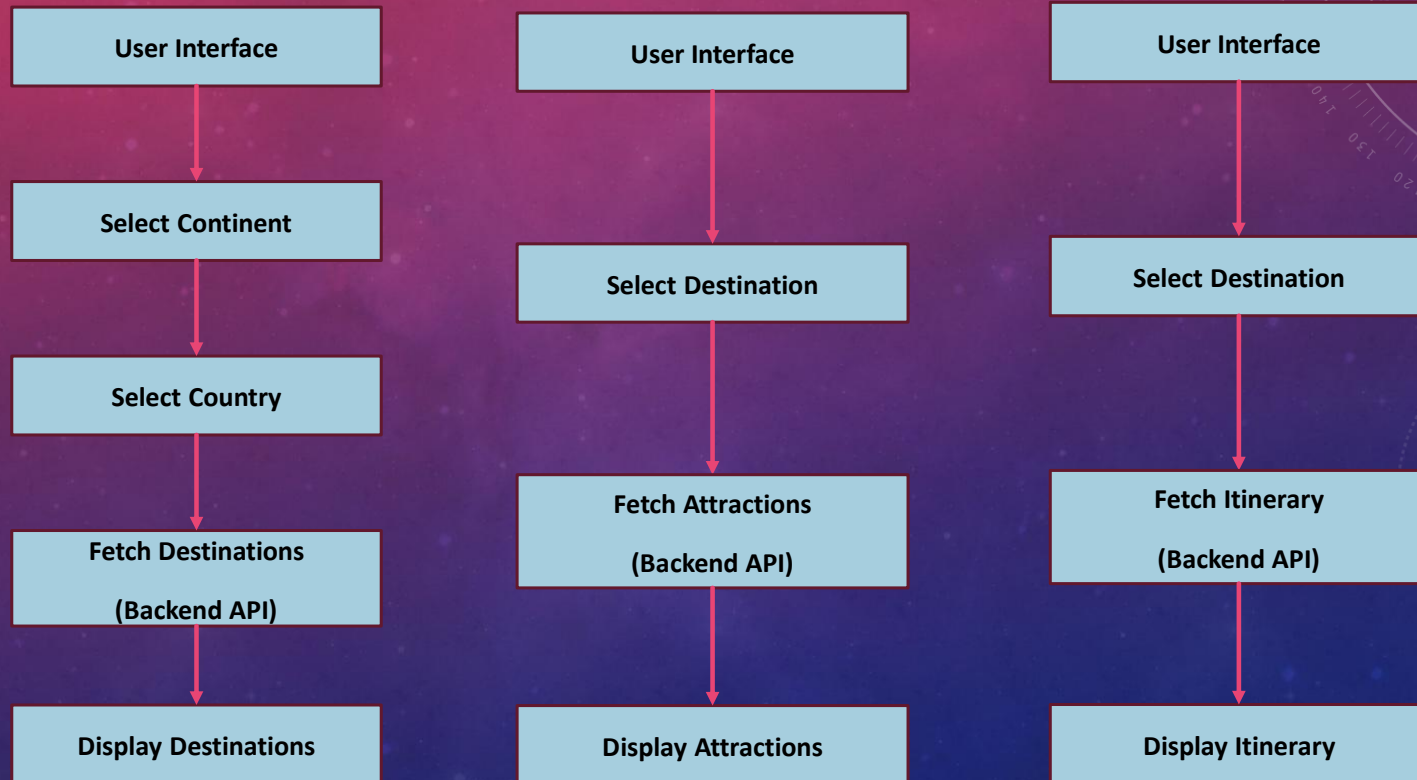
INTRODUCTION TO AI BASED VIRTUAL TRAVEL GUIDE

AI Based Virtual Travel Guide:

- Designed to help users plan their trips
    - Provides information about various destinations, attractions and itineraries.
- Aims to enhance the travel planning experience by offering detailed information and personalized itineraries.

TECHNOLOGIES USED:

- Python

- HTML

- SQLite

- Unit Test Framework

- Wikipedia API's

DATA FLOW DIAGRAM(DFD)

User

↓

Trip Planner App

↓

View Destinations

↓

View Attractions

↓

Generate Itinerary

Admin

↓

Trip Planner App

↓

Manage Data

↓

Update Database

CONTROL FLOW DIAGRAM (CFD)

User Interface	User Interface	User Interface
Select Continent	Select Destination	Select Destination
Select Country	Fetch Attractions (Backend API)	Fetch Itinerary (Backend API)
Fetch Destinations (Backend API)	Display Attractions	Display Itinerary
Display Destinations		

PROJECT PAGES: HOME PAGE

PROJECT PAGES: ADMIN LOGIN PAGE

PROJECT PAGES: ADMIN PAGE

PROJECT PAGES: ADMIN PAGE - CONTINUED

PROJECT PAGES: COUNTRY LISTING PAGE

PROJECT PAGES: DESTINATIONS PAGE
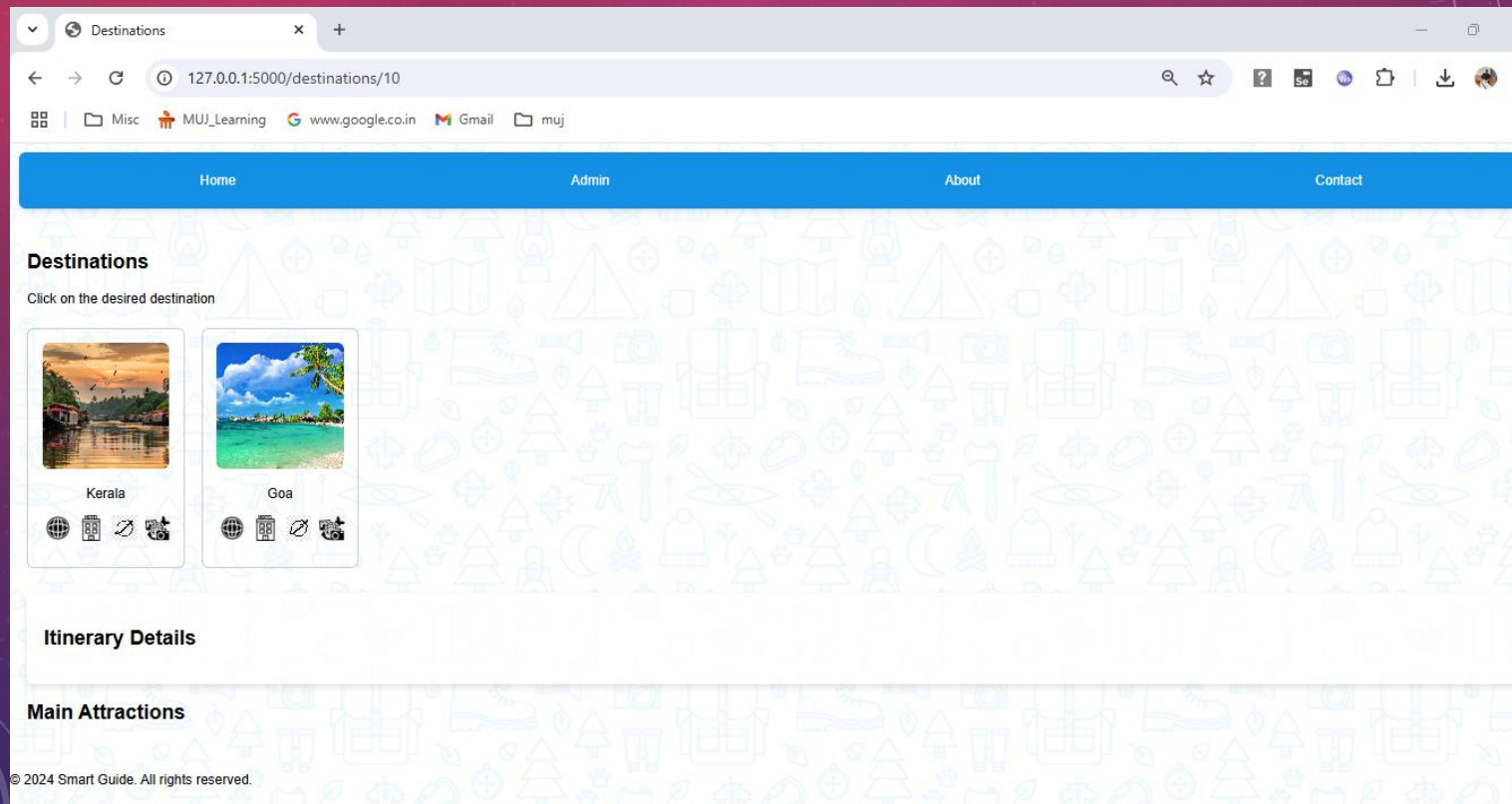
PROJECT PAGES: VIEW ATTRACTIONS AND ITINERARY
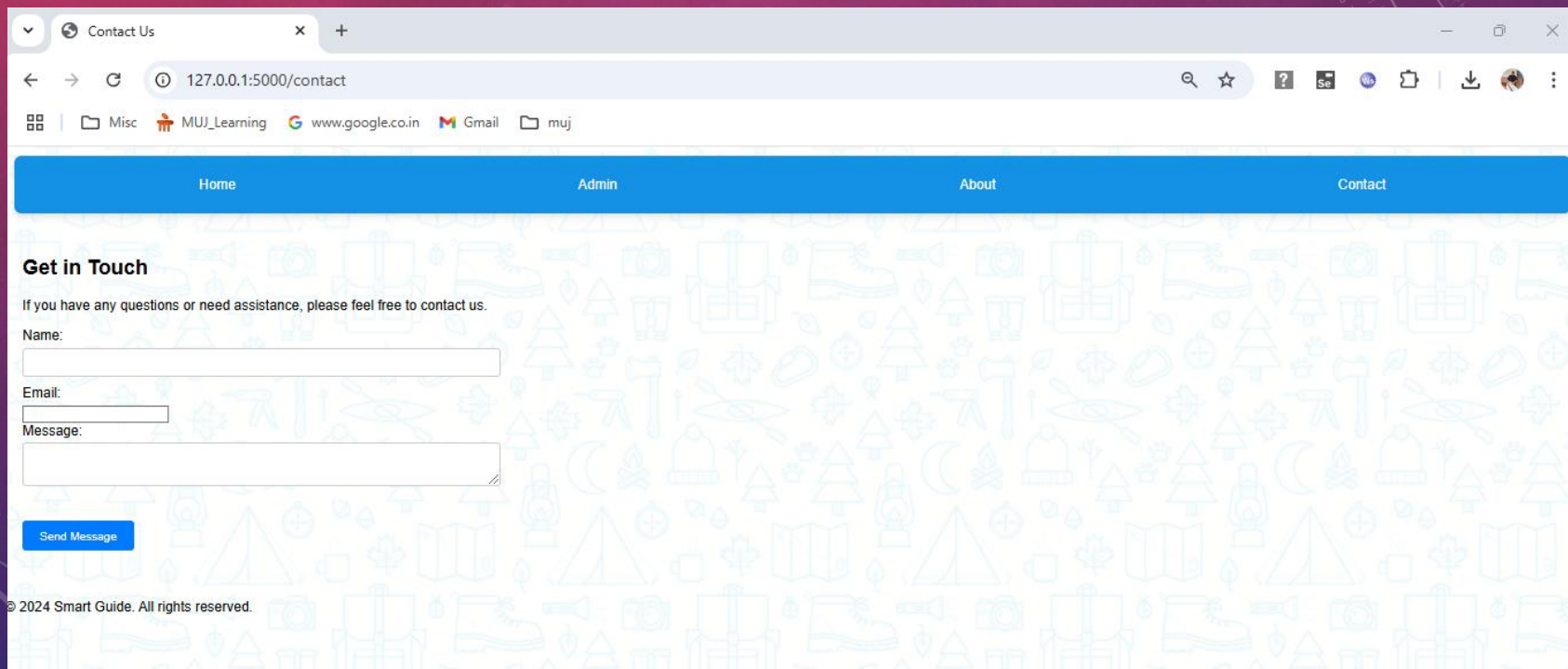
PROJECT PAGES: VIEW ATTRACTIONS AND ITINERARY - CONTINUED

PROJECT PAGES: VIEW ATTRACTIONS AND ITINERARY - CONTINUED

PROJECT PAGES: ABOUT PAGE

PROJECT PAGES: CONTACT US PAGE

FUTURE SCOPE

- User Authentication and Personalization
- Advanced Search and Filtering
- Reviews and Ratings
- Social Sharing and Collaboration
- Mobile Application
- Integration with Travel Services
- Enhanced Itinerary Features
- Multilingual Support