Architectural and Quality Requirements for AI Trip Creator

The Rolling Capstones

27 September 2024

Architectural Requirements

Quality Requirements

Performance

The system should perform well to maintain user engagement and interaction. The maximum response time for any user request should be as low as possible, ensuring a smooth and satisfactory user experience. Flight and accommodation data aggregation and scraping processes should be completed within a reasonably low time frame. The system must be capable of handling many concurrent users without performance degradation.

Scalability

The system should scale according to demand, ensuring it can accommodate a growing user base. This includes handling a significant increase in user base per year and an increase in data volume per year without performance issues.

Security

User data must be protected using industry-standard encryption algorithms. All data must be encrypted in transit using HTTPS and at rest. The system should implement OAuth 2.0 for secure authentication and role-based access control for authorization. Vulnerability assessments should be conducted bi-annually, with critical vulnerabilities resolved within 30 days.

Usability

The system should be easy to navigate with an intuitive user interface. The application should comply with WCAG 2.1 AA standards to ensure accessibility. Additionally, user support should be available through chatbots and email, with a response time of 24 hours for user inquiries.

Reliability

The system should have an uptime of 99.9% to ensure high availability. The error rate for user transactions and requests should be less than 0.1%. Data should be backed up

every 24 hours, with a recovery time objective (RTO) of 2 hours and a recovery point objective (RPO) of 1 hour.

Maintainability

The codebase must adhere to clean coding principles and be well-documented to facilitate future maintenance and updates by development teams. The code should maintain a maximum cyclomatic complexity of 10 for any function/method. Comprehensive documentation, including API documentation, user manuals, and developer guides, must be provided. The system should achieve at least 80% code coverage with automated unit and integration tests.

Compatibility

The system should work across a variety of current browsers and devices to provide a consistent user experience. This ensures users can access the system regardless of the browser or device being used.

Architectural Patterns

1. Microservices Architecture

1.1 Description

Decompose the application into loosely coupled, independently deployable services. Each service corresponds to a specific business functionality.

1.2 Benefits

- **Scalability**: Individual services can be scaled independently.
- **Flexibility**: Easier to update and deploy individual components without affecting the entire system.
- **Resilience**: Failure in one service does not bring down the entire system.

Components:

- **User Interface Service**: Handles user inputs and displays data.
- **Flight Data Service**: Integrates with flight APIs to fetch data.
- **Accommodation Service**: Scrapes and aggregates accommodation data.
- **Itinerary Service**: Generates and manages travel itineraries.
- **User Management Service**: Manages user authentication and profiles.
- **Social Sharing Service**: Handles sharing functionalities.

2. Serverless Architecture

2.1 Description

A serverless architecture allows building and running applications and services without the need to manage servers. The cloud provider handles infrastructure management tasks like scaling, patching, and maintaining servers, allowing focus solely on code.

2.2 Benefits

- **Cost-Efficiency**: Pay only for the computation time consumed.
- **Automatic Scaling**: Automatically scales to handle varying loads.
- **Simplified Operations**: Reduced need for server management and maintenance.

2.3 Components

- **Frontend (SPA/PWA)**: A single-page application (SPA) or progressive web app (PWA) to provide a responsive user interface.
- **API Gateway**: Routes user requests to appropriate serverless functions.
- **Lambda Functions**: Small, single-purpose functions that execute backend logic in response to events like HTTP requests from the API Gateway. They are stateless and designed to run for a short duration (e.g., fetching flight data, generating itineraries).
- **Database (DynamoDB, Firestore)**: Stores user data, preferences, and itineraries.

3. Model-View-Controller (MVC) Pattern

3.1 Description

Separate the application into three interconnected components: Model, View, and Controller.

3.2 Benefits

- **Clear separation of concerns**: Each component handles specific aspects of the application, making it easier to manage and develop.
- **Reusability**: Components can be reused across different parts of the application.
- **Maintainability**: Easier to update and maintain due to the separation.

3.3 Components

- **Model**: Manages the data and business logic (e.g., user preferences, itinerary data).
- **View**: Displays data to the user and handles user interactions.
- **Controller**: Processes user input, interacts with the model, and updates the view.

Architectural Diagram