THE ROLLING CAPSTONES

AI TRIP CREATOR

ARCHITECTURAL REQUIREMENTS DOCUMENT

TABLE OF CONTENTS

1. Architectural strategy

Design Based on Quality Requirements

Description: This strategy involves designing the system architecture to meet specific quality requirements such as performance, security, scalability, reliability, etc.

Justification:

- **Quality Focus:** Ensures that the system meets critical quality attributes essential for its success.
- **Optimization:** The architecture can be optimized for specific quality attributes, leading to better performance, security, and reliability.
- **Alignment with Goals:** Helps align the system design with business goals and user expectations, ensuring that the most critical aspects of the system are prioritized.

When to Use:

- When the system has stringent quality requirements that must be met.
- When performance, security, and other quality attributes are more important than functional decomposition.
- When specific non-functional requirements are crucial for the system's success.

2. Architectural design strategies

1. Microservices Architecture

Description: Break down the application into smaller, independent services. Each service handles a specific part of the application such as user preferences, itinerary generation, web scraping, flights information, and accommodation details.

Benefits:

- Scalability: Easily scale each service independently based on demand.
- Maintainability: Simplifies debugging and updating individual services without affecting the entire system.
- Flexibility: Allows integration with various third-party services and APIs.

Implementation:

- Use containers (e.g., Docker) and orchestration tools (e.g., Kubernetes) to deploy and manage microservices.
- Implement RESTful APIs or gRPC for inter-service communication.

2. Event-Driven Architecture

Description: Leverage an event-driven approach where different parts of the system react to events (e.g., user submits preferences) rather than continuously polling for data.

Benefits:

- Efficiency: Reduces resource consumption by only acting on specific events.

- Responsiveness: Improves the app's responsiveness to user actions.

Implementation:

- Use message brokers like Apache Kafka or RabbitMQ to manage event streams and communication between services.
- Implement event handlers to process events asynchronously and update the system state or notify users.

3. Data Aggregation Layer

Description: Introduce a dedicated layer for aggregating and processing data from various sources such as web scrapers, flight APIs, and accommodation databases.

Benefits:

- Centralized Data Handling: Simplifies data retrieval and processing logic.
- Improved Performance: Enhances performance by consolidating data operations.

Implementation:

- Use data processing frameworks like Apache Spark for batch processing or Apache Flink for stream processing.
- Implement caching mechanisms to store frequently accessed data for faster retrieval.

4. User Personalization Engine

Description: Develop a robust personalization engine that customizes itineraries based on user preferences, past behavior, and other relevant data.

Benefits:

- Enhanced User Experience: Provides users with highly relevant and personalized recommendations.
- Increased Engagement: Keeps users engaged by offering content tailored to their interests.

Implementation:

- Use machine learning models to analyze user data and predict preferences.
- Implement a recommendation system (e.g., collaborative filtering, content-based filtering) to suggest activities, flights, and accommodations.

5. Web Scraping and Data Integration

Description: Design a reliable web scraping mechanism to gather information from various travel and tourism websites. Ensure compliance with the websites' terms of service and legal regulations.

Benefits:

- Comprehensive Data Collection: Collects a wide range of up-to-date information.
- Flexibility: Allows integration with multiple data sources.

Implementation:

- Use frameworks like Scrapy or Beautiful Soup for web scraping.
- Implement rate limiting and request throttling to avoid IP blocking and comply with website policies.
- Use APIs from travel aggregators (e.g., Skyscanner, Expedia) for reliable data.

6. API Gateway and Load Balancer

Description: Implement an API Gateway to manage client requests and route them to the appropriate microservices. Use load balancers to distribute traffic evenly across services.

Benefits:

- Security: Provides a single entry point for security measures like authentication and rate limiting.
- Traffic Management: Ensures the application remains responsive under heavy load.

Implementation:

- Use API gateway solutions like AWS API Gateway or Kong.
- Implement load balancers (e.g., Nginx, HAProxy) to manage traffic and ensure high availability.

7. Cloud Services and Scalability

Description: Deploy the application on a cloud platform to take advantage of on-demand resources and scalability.

Benefits:

- Scalability: Easily scale up or down based on user demand.
- Cost Efficiency: Pay for only the resources you use.

Implementation:

- Use cloud services like AWS, Google Cloud, or Azure for infrastructure needs.
- Implement auto-scaling groups to handle varying loads.

8. Security and Data Privacy

Description: Incorporate robust security measures to protect user data and ensure compliance with data privacy regulations.

Benefits:

- User Trust: Builds trust by safeguarding user data.
- Compliance: Ensures adherence to legal requirements such as GDPR.

Implementation:

- Implement HTTPS for secure data transmission.
- Use encryption for sensitive data storage.
- Perform regular security audits and vulnerability assessments.

9. Continuous Integration and Continuous Deployment (CI/CD)

Description: Adopt CI/CD practices to automate the process of code integration and deployment, ensuring rapid and reliable updates.

Benefits:

- Faster Development: Accelerates the delivery of new features and bug fixes.
- Reduced Risk: Minimizes the risk of deployment issues.

Implementation:

- Use CI/CD tools like Jenkins, GitLab CI, or CircleCI.
- Implement automated testing and code quality checks in the CI/CD pipeline.

10. Monitoring and Analytics

Description: Set up comprehensive monitoring and analytics to track application performance and user behavior.

Benefits:

- Proactive Issue Detection: Identifies issues before they impact users.
- Data-Driven Decisions: Provides insights for improving the application.

Implementation:

- Use monitoring tools like Prometheus and Grafana for system metrics.
- Implement user analytics using tools like Google Analytics or Mixpanel.

3. Architectural quality requirements

1. Scalability

As the application gains popularity, it needs to handle an increasing number of users and requests without performance degradation.The AI trip creator should be able to process large volumes of data efficiently to generate recommendations in real-time or near real-time, regardless of the number of concurrent users.

2. Usability

A user-friendly interface is critical for ensuring that users can easily input their preferences, understand

Recommendations, and make modifications to their plans.

3. Reliability

The application must provide accurate and reliable recommendations for destinations, itineraries, accommodations, and activities. Inaccurate information can lead to user dissatisfaction and loss of trust in the application. The Applicarion should accurately interpret and adapt to user preferences and requirements to provide personalized travel plans that meet their needs.

4. Performance

The application should provide quick responses to user queries and recommendations. Delays can frustrate users and lead to abandonment of the service.

Resource Management: Efficient handling of data and resources ensures smooth operation even during peak usage times, providing a consistent user experience.

5. Security

The application will handle sensitive user data, including personal information and travel preferences. Ensuring data privacy and protection is crucial to prevent data breaches and maintain user trust.

4. Architectural design & pattern

1. Microservices Architecture

Description: Divide the application into independent microservices, each responsible for a specific domain or function (e.g., user management, itinerary generation, flight and accommodation search).

Components:

- **User Service**: Manages user data, authentication, and preferences.
- **Itinerary Service**: Generates personalized itineraries based on user preferences and available data.
- **Flight Service**: Integrates with flight data APIs to fetch and manage flight options.
- **Accommodation Service**: Handles accommodation data and integrates with hotel APIs.
- **Web Scraper Service**: Collects additional data from travel websites.
- **Notification Service**: Manages notifications and communications with users.
- **Recommendation Engine**: Uses machine learning to personalize suggestions.

Benefits:

- **Scalability**: Services can be scaled independently based on load.
- **Fault Isolation**: Failures in one service do not affect others.
- **Flexibility**: Facilitates the integration of new services or updates.

2. Model-View-Controller (MVC) Pattern

Description: Use the MVC pattern for structuring the front-end and possibly the API layer. MVC separates the application into three interconnected components: Model, View, and Controller.

Components:

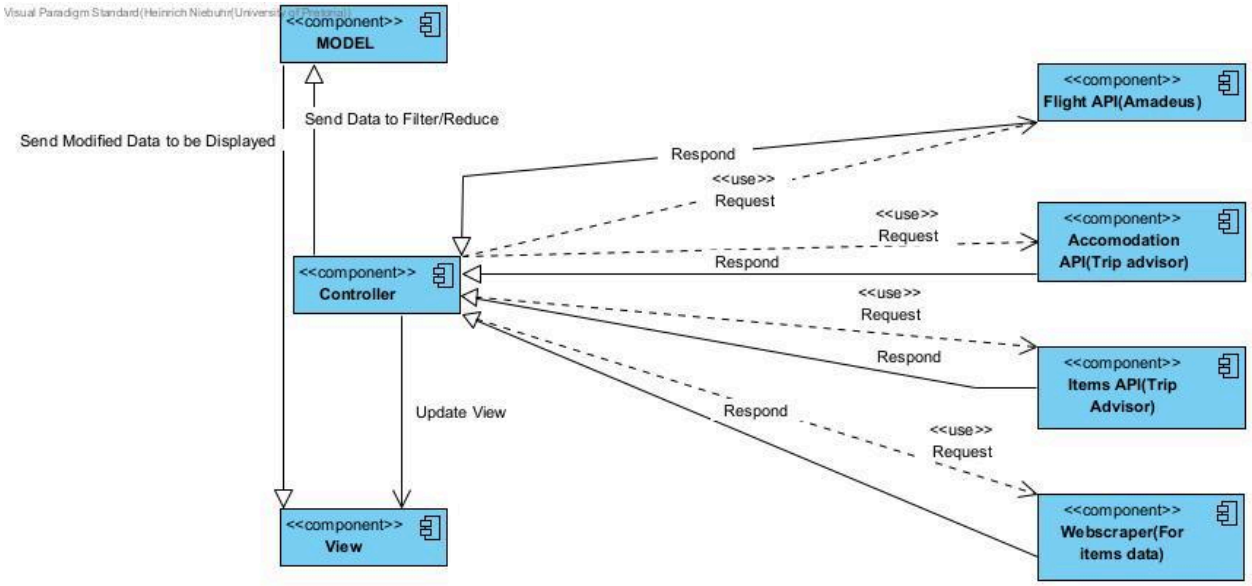
- **Model**: Represents the application's data and business logic (e.g., user preferences, itinerary data).
- **View**: Handles the presentation layer, displaying data to users and collecting user inputs.
- **Controller**: Manages user inputs, interacts with the Model, and updates the View.

Benefits:

- **Separation of Concerns**: Enhances maintainability by separating business logic, user interface, and input control.
- **Reusability**: Promotes the reuse of code across different parts of the application.

Implementation Example:

- **Frontend**: Use frameworks like React, Angular, or Vue.js to implement the View and Controllers.
- **Backend**: Use web frameworks like Spring (Java), Django (Python), or Express.js (Node.js) to handle API requests and data processing.



5. Architectural constraints

1. Firebase - must be used (specified by client).
2. Real-time responsiveness - The application functionality must be in real-time and interact with all parts of the stack as it is used.
3. Design pattern usage - A minimum of 1 design pattern must be used when creating the system.
4. Microservices has to be used in order to gather data.

6. Technology Choices

1. React for Front End

React's component-based architecture allows for modular and reusable UI components, making it easier to maintain and scale the application.

2. Express for Backend

Express is a minimalistic framework that provides a robust set of features for web and mobile applications without adding unnecessary complexity. It is a rich middleware ecosystem that allows easy integration of functionalities such as authentication, logging, and error handling. It is highly scalable and can handle a large number of simultaneous connections, making it ideal for applications expecting high traffic.

3. Python for Data Scraping

Reasons:

Python offers powerful libraries such as BeautifulSoup, Scrapy, and Selenium for efficient web scraping and data extraction.

It also has simple and readable Syntax which makes it easy to write and maintain web scraping scripts.

4. Material UI for Front End

Reasons:

Pre-Built Components: Material UI provides a rich set of pre-built, customizable components that follow Google's Material Design guidelines, ensuring a modern and consistent look and feel.

Responsive Design: Built-in support for responsive design ensures that your application looks great on all devices, from desktops to mobile phones.

Theming: Easy theming and customization options allow you to tailor the UI to match your brand's style and preferences.

5. Firebase for Hosting and Database

Reasons:

Firebase's real-time database allows for instant data synchronization across all clients, which is crucial for providing up-to-date travel information.

Firebase can scale effortlessly to handle growing amounts of data and user traffic, ensuring that the application remains performant.

Hosting and Deployment: Firebase provides easy-to-use hosting and deployment services, enabling quick deployments and application management .

Authentication: Built-in support for authentication simplifies the process of user sign-up, login, and security.

Integration: Firebase offers seamless integration with other Google services and third-party tools, enhancing the overall capabilities of your application.