Assignment Title: Designing a Scalable Architecture for a Smart Ticketing System Online Website

Overview The microservices architecture provides a robust, scalable framework for the Smart Ticketing System Online. This design allows each service to be independently scaled and managed, offering high availability, fault tolerance, and ease of updates. By leveraging modern cloud services and efficient data management practices, the system is well-equipped to handle current demands and future growth, ensuring a seamless user experience even during peak traffic periods. **Create website Html, Css,Tailwind,Daisy Ui farmwork ,java-script, Dom , jQuery.** This architecture not only meets current needs but also provides a foundation for future enhancements and scalability.

Diu Paribahan is apparently a family-owned transport company specializing in transportation of passenger bus services since 1990. From a humble beginning of local services, our transport system encompasses all reachable areas of Bangladesh and also beyond the border, extending our reach to Kolkata in India.

We worked hard and honest, we put our vision forward and explored ways and means to continuously improve passenger comfort, and as a result, we were able to introduce the 1st ever Air- Conditioned bus services in Bangladesh. We take pride in mentioning that our fleet of buses includes the most luxurious models of VOLVO and SCANIA Imported from Europe, which provide the ultimate in passenger comfort and safety.

The company at present operates more then 60 (Sixty) buses on schedule routes employing over 200 trained staff and safely transporting over a million passengers a year.

Task :1

Research on Scalable Web Application Architectures

Monolithic Architecture:

Description: A single, unified codebase where all components are interlinked.

Pros: Simple development and deployment, easier testing.

Cons: Hard to scale, difficult to maintain with growing complexity, single point of failure.

Microservices Architecture:

Description: Breaks down the application into smaller, independent services.

Pros: Independent scaling, fault isolation, easier to manage and update individual components.

Cons: Complex development and deployment, requires robust inter-service communication mechanisms.

Serverless Architecture:

Description: Runs code in response to events and automatically manages server resources.

Pros: No server management, automatic scaling, cost-effective for variable workloads.

Cons: Cold start latency, limited execution duration, dependency on third-party services.

Other Architectures:

Container-based Architecture: Uses containers for consistent environments across development, testing, and production.

Event-Driven Architecture: Focuses on producing and consuming events for decoupled components.

Scalability Requirements Analysis for Online Ticketing System

Expected User Base: Large and growing user base due to the popularity of events.

Traffic Patterns: High traffic during event announcements and booking periods, with peaks and troughs.

Data Volume: Significant volume of transactional data, including user details, bookings, and payment information.

Future Growth Projections: Anticipated increase in users and events, requiring the system to handle larger loads without performance degradation.

Task 2: Architecture Design

Choosing the Architectural Pattern

Microservices Architecture is chosen for the following reasons:

Scalability: Individual services can be scaled independently based on load.

Fault Isolation: Issues in one service do not affect the entire system.

Agility: Easier to develop, test, and deploy smaller, independent services.

Defining Components/Modules and Their Responsibilities

User Interface (UI)

Responsibility: Provides a responsive and user-friendly interface for users.

Interaction: Communicates with backend services through RESTful APIs.

Authentication Service

Responsibility: Manages user authentication and authorization.

Interaction: Provides JWT tokens to the UI for secure communication with other services.

Ticket Management Service

Responsibility: Handles ticket creation, updating, and deletion.

Interaction: Interacts with the Database Service for storing ticket information and the Notification Service for booking confirmations.

Payment Gateway

Responsibility: Processes user payments securely.

Interaction: Integrates with third-party payment processors and updates the Ticket Management Service upon successful transactions.

Notification Service

Responsibility: Sends notifications to users, such as booking confirmations and reminders.

Interaction: Triggered by the Ticket Management Service and Payment Gateway.

Analytics Service

Responsibility: Analyzes usage patterns and generates reports.

Interaction: Collects data from all services for comprehensive analytics.

Database Service

Responsibility: Stores user profiles, ticket details, and transaction records.

Interaction: Central repository accessed by all services needing persistent data storage.

Load Balancing, Caching, and Data Storage

Load Balancing:

Strategy: Use AWS Elastic Load Balancer to distribute incoming traffic across multiple instances of services, ensuring even load distribution and high availability.

Caching Mechanisms:

Strategy: Implement Redis for caching frequently accessed data, such as event listings and seat availability, to reduce database load and improve response times.

Data Storage:

Relational Database: Use PostgreSQL for structured data, such as user profiles and ticket transactions.

NoSQL Database: Use MongoDB for flexible and scalable storage of event-related information and logs.

Communication Between Components

Internal Communication: Use RESTful APIs over HTTP for simplicity and compatibility. Services communicate using lightweight JSON messages.

External Communication: Secure communication using HTTPS for all interactions between the UI and backend services to protect user data.