Senior Java Developer Case Study.

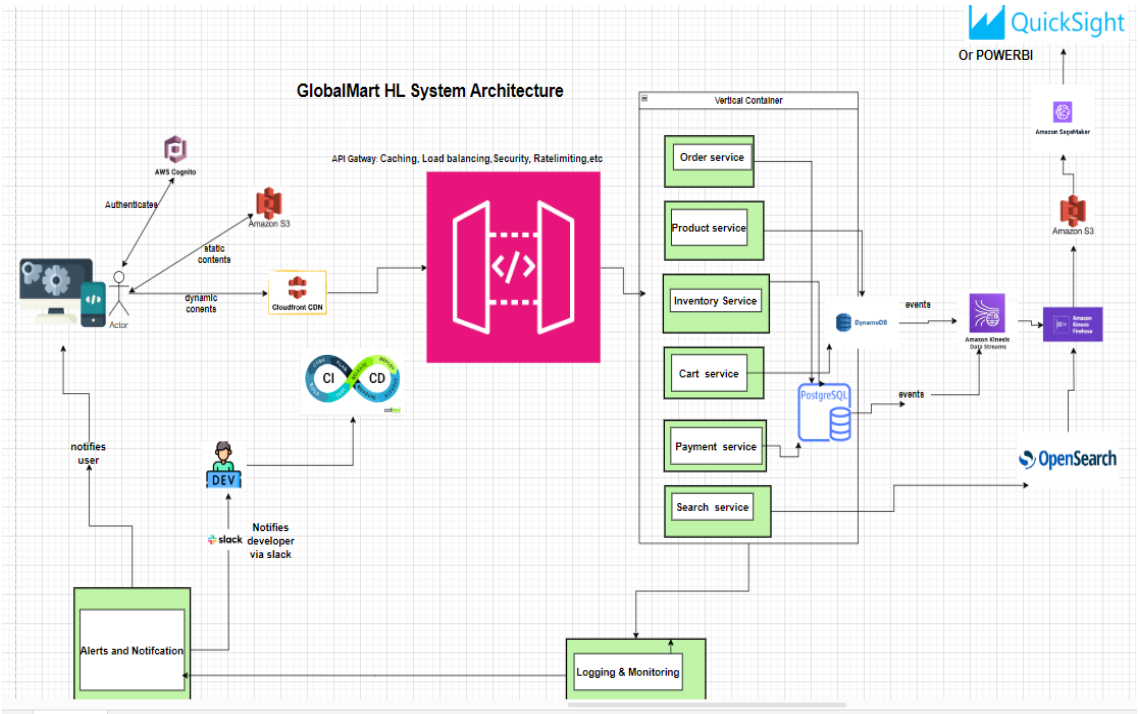
Duration: 3 Days

Your company is launching a global multi-vendor e-commerce marketplace called "GlobalMart." This platform allows multiple sellers to list their products and enables buyers from around the world to purchase them. The system needs to handle millions of users and products, support multiple languages and currencies, and offer real-time inventory and order tracking.

Tasks:

1. **System Design:**
    - Design the high-level architecture of the GlobalMart system.

        Solution:



    - Describe the main components such as user management, product catalog, order processing, payment integration, and inventory management.

Answer:

As shown in the above diagram: It is made up Aws Cognito (for effective and federated authentication and user management. Then others (product, inventory, payment, search and cart services) are shown and they handle the respective functions of the e-commerce according to their names.

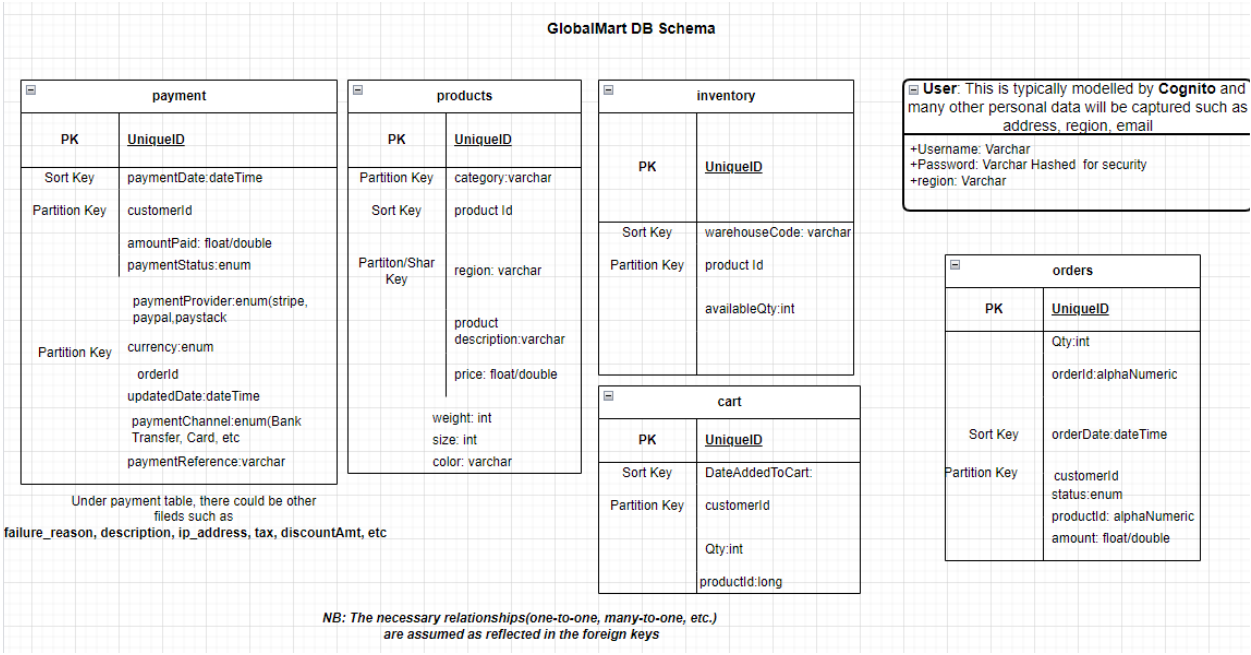
- Explain how you would handle scalability, availability, and fault tolerance, considering the global nature of the platform.

Answer:

Cloud Front is a robust and scalable regional-conscious CDN. Cognito is also designed to scale. Finally, the API Gateway shown above handles other scalable and fault tolerant requirements such as load-balancing, availability, throttling/Rate Limiting, Caching and Security

2. **Data Modeling:**
   - Design the database schema for GlobalMart.
   - Include tables for users, sellers, products, orders, payments, and inventory.

Solution:



- Justify your choice of database (SQL/NoSQL) and specific technologies, considering aspects like data consistency, partitioning, and regional data storage.

Answer:

Because this is an e-commerce application where scalability, availability and partition-tolerant are key, according to CAP theorem. Hence, I choose Hybrid: Amazon Dynamo NoSQL (for products reviews, histories, cart) for the above state reasons of scalability and availability. However, due to strong need for consistency in transactions (also according to CAP theorem), I choose PostgreSQL for payment, inventory and order tables. However, in accordance with GDPR and NDPR requirement, I am open to making adjustments for regional legal compliance depending on the customer region.

3. **Security Considerations:**

- Discuss how you would handle security concerns such as data protection, secure communication, and authentication, especially in a multi-vendor environment.

Answer:

Amazon Cognito is already robust for authentication. It also has federated-sign-in capabilities (OpenID, SAML2, LDAP). However, I can also

use Keycloak or Okta. These two also implement latest OAuh2 specifications. Finally, Multi-Factor Authentication will be implemented.

Notice that the adopted API gateway is already implementing some security features at the application-layer.

- Describe measures to prevent common vulnerabilities (e.g., SQL injection, cross-site scripting), and how to secure sensitive information like payment details.

Answer:

To a large extent, Spring Security enables one to tackle some of these out of the-box (Cross-site scripting, Clickjacking, Session Fixation, sniffing, CSRF, etc.). However, SQL Injection could be further handled with sanitization and proper validation of inputs both at the front-end and back-end. The need to use TLS/SSL cannot be over-emphasized. Finally, the adopted load-balancer in the API gateway and at CloudFront CDN ensure that DDos attacks don't happen especially via RateLimiting.

Part 2: Critical Thinking and Problem-Solving

GlobalMart's order processing system experiences delays, particularly during global sales events. The system must process, validate, and store each order while ensuring that inventory is updated in real-time across different regions.

Tasks:

1. **Performance Analysis:**
    - Identify potential bottlenecks in the current system, particularly in order processing and inventory management.
    - Suggest improvements or optimizations to enhance performance, considering the global scale and peak load times.

Solution:

After careful analysis, I will do the following:

1) *Check the caching of orders at the DB and Server level.*
2) *Check if necessary, indexes and even global secondary indexes are implemented at the DB.*
3) *Find out if proper replications and sharding are done based on the region using edge servers.*
4) *Check the implementations of Loading balancing and if possible, the type of algorithms (round-robin, hashing, etc) used*
5) *Finally check CDN edge servers if they are pointing to the closet destination of the customers to reduce latency.*

2. **Algorithm Design:**

- Propose an algorithm to prioritize order processing during peak times, ensuring that high-priority orders (e.g., expedited shipping, high-value orders) are handled first.

  Solution:

  After careful analysis, similar to OS level technique of prioritizing processes and threads, the algorithm will be like this:

  1) *Clearly and unambiguously define a priority criterion such as expedited-shipping, high-value orders, loyalty, age and region.*
  2) *Define and assign weights **wi** to each defined priority criteria in step 1. The weight should be in descending order of priority. For instance, expedited-shipping may weight 10, while region criterion may weigh 2.*
  3) *Assign priority score to all the defined criteria of step 1. For instance, high-value orders over $20,000 may have a score of 10 while the one of $100 may have a score of 1.*
  4) *Calculate priority score of each order multiplying the values assigned in step 2 and step 3.*
  5) *Use Java PriorityQueue data structure specifically PriorityBlockingQueue class. Remember to pass an Order Comaparator(that is used to compare the priority score of two orders) to the Priority Queue constructor.*
  6) *Finally, add orders to the PriortiyQueue and, process and poll when necessary.*
  7) *It is also important to optimize, do batch processing, when necessary, based on similar orders/items and regions. Real-Time monitoring using ML or MLOPs is key for instance, if a specific carrier is experiencing delay or logistic issue, the carrier orders can receive lower priority score/weight.*

- Describe how the algorithm manages concurrency, maintains data consistency, and ensures fairness among different vendors.

  Solution:

Concurrency is managed using PriorityBlockingQueue. Fairness is ensured by not defining priority criteria based on vendor.

3. **Scalability Planning:**
    - How would you ensure the system scales to handle a significant increase in users and orders during global promotions?

        Solutions:

        As mentioned during performance suggestions above, I will implement caching at the server, CDN and database levels/layers. For API requests, Load Balancing is key. Finally, at the DB level, Data partition, Replication/Sharding, Query Optimizations and proper Indexing is very important.

    - Discuss the use of caching, load balancing, and database sharding, and how to handle cross-region data synchronization.

        Solution:

        1) *Caching ensures low latency by minimizing unnecessary server and database requests instead of using saved and last fetched contents (unless they have expired). It can be done at the browser, server, CDN and DB level. Redis is a popular key-value database used for server and DB level caching. Load balancing ensures that the applications are scaled horizontally (scaling out) and no-one server is inundated with requests simultaneously ensuring availability and preventing distributed denial of service (DDos) attacks. Popular Load balancers are AWS ELB, Nginx, HAProxy, etc.*
        2) *As regards cross-region data sync, I will implement multi-master replication strategy for each region and more importantly adopt highly distributed Amazon Aurora Global DB or Cassandra which are both very good at this.*

Part 3: Algorithm Development and Coding Challenge

You need to develop a feature for the inventory management system that tracks the stock levels of products and automatically reorders items when they fall below a certain threshold. The system must support multiple warehouses across different regions.

Tasks:

1. **Algorithm Design:**

- Design an algorithm that monitors stock levels across different warehouses and triggers reordering when necessary.
- Consider factors like lead time, reorder quantities, supplier availability, and regional variations in demand.

2. **Implementation:**
    - Implement the algorithm in Java.
    - Include error handling, logging, and optimizations for efficiency and scalability.
    - Ensure you implement required security and performance considerations appropriate for this question.
3. **Unit Testing:**
    - Write unit tests for your algorithm to ensure its correctness, robustness, and ability to handle edge cases.