Design Doc

Toy Store Application: Detailed Design Document

System Architecture Overview

The toy store application consists of three primary microservices:

1. **Front-end Service**
2. **Catalog Service**
3. **Order Service**

Each service is designed to handle specific functions within the application. Below is a detailed breakdown of each service, including their respective APIs and functionalities.

1. Front-end Service

Description

The Front-end Service acts as the gateway for user interactions, managing API requests related to products and orders, and directing them to the appropriate backend services.

The FrontendService is listening at /api API.

APIs

GET /api/products/{product_name}

- **Purpose**: Retrieve product details from the cache or catalog service.
- **Process**:
    - Check the cache for product details.
    - If the product is not in the cache, request details from the Catalog Service, store in the cache, and return the response.
    - If cached, return the cached response directly.

POST /api/orders

- **Purpose**: Place a new order.
- **Process**:
  - Send the order request to the Order Service (leader node).
  - Return the order number and details provided by the Order Service.

GET /api/orders/{order_number}

- **Purpose**: Query existing orders.
- **Process**:
  - Request order details from the Order Service (leader node).
  - Return order details including `number`, `name`, and `quantity`.

POST /api/invalidate_cache/{productName}

- **Purpose**: Invalidate cache entries upon product update.
- **Process**:
  - Invalidate the cache entry for the specified product upon receiving an invalidation request from the Catalog Service.

2. Catalog Service

Description

The Catalog Service manages product information and inventory, including restocking and updating product availability.

APIs

GET /products/{productName}

- **Purpose**: Provide product details and availability.
- **Process**:
  - Return details of the requested product if available.

- If the product does not exist, return a not found response.

POST /products/{productName}/update

- **Purpose**: Update the stock quantity for a specific product.
- **Process**:
  - Update the product stock based on the input quantity.
  - Send cache invalidation request to the Front-end Service after the update.

3. Order Service

Description

The Order Service is responsible for processing and storing order transactions. It is implemented with replication for high availability and fault tolerance.

APIs

POST /orders

- **Purpose**: Record a new order.
- **Process**:
  - Store the order details in the database.
  - Propagate the order details to follower nodes for synchronization.

GET /orders/{order_number}

- **Purpose**: Retrieve specific order details.
- **Process**:
  - Return details of the requested order if it exists.
  - If the order does not exist, return a not found response.

POST /orders/replicate

- **Purpose**: Synchronize order data among replicas.
- **Process**:

- Accept data from the leader and replicate it in the follower's database.

GET /orders/health

- **Purpose**: Check the health of the Order Service node.
- **Process**:
  - Return a success response to indicate that the node is functional.

GET /orders/synchronize/{lastOrderNumber}

- **Purpose**: Sync missing orders for a recovering replica.
- **Process**:
  - Retrieve orders that have been missed since the last known order number and synchronize them.

POST /orders/leader/{node_id}

- **Purpose**: Notify all replicas of the leader node selection.
- **Process**:
  - Receive the leader node identifier.
  - Set internal flags to recognize the leader.
  - All nodes adjust their roles based on the leader ID (leader vs follower).