Microservices Architecture Design Document

1.Introduction

This document outlines the design and development plan for implementing a microservices architecture for an e-commerce application. The architecture aims to provide scalability, flexibility, and maintainability by decomposing the application into smaller, independent services.

2. Microservices Overview

The application will be decomposed into the following microservices:

1. **Frontend Service**: Handles client-side interactions, user interface rendering, and presentation logic.
2. **Cart Service**: Manages shopping cart functionality, including adding/removing items, updating quantities, and calculating totals.
3. **Items Service**: Manages product catalog, including CRUD operations for products and retrieval of product information.
4. **Order Service**: Handles order management, including placing orders, processing payments, and managing order history.
5. **Authentication Service**: Responsible for user authentication, authorization, and management of user profiles.

3. Communication Protocols

Microservices will communicate with each other via RESTful APIs over HTTP/HTTPS. Each microservice will expose a set of well-defined endpoints for inter-service communication.

4. Data Models and Schemas

Each microservice will have its own database to store relevant data. The following outlines the data models and schemas for each microservice:

1. **Authentication Service**:
    - User: { id, username, email, password_hash, role, created_at, updated_at }

2. **Items Service**:
    - Product: { id, name, description, price, category, image_url, created_at, updated_at }
3. **Order Service**:
    - Order: { id, user_id, status, total_price, created_at, updated_at }
    - Order Item: { id, order_id, product_id, quantity, price }
4. **Cart Service**:
    - Cart: { id, user_id, items }

5. Programming Languages and Frameworks

- **Frontend Service**: React.js for the frontend framework.
- **Cart Service**: Node.js with Express.js framework.
- **Items Service**: Node.js with Express.js framework.
- **Order Service**: Node.js with Express.js framework.
- **Authentication Service**: Node.js with Express.js framework.

6. Modules

- **Authentication Service**:
    - Endpoints for user registration, login, profile management.
    - Authentication middleware to verify JWT tokens.
- **Items Service**:
    - Endpoints for CRUD operations on products.
    - Search functionality to query products by name or category.
- **Order Service**:
    - Endpoints for placing orders, retrieving order history.
    - Integration with payment gateway for processing payments.
- **Cart Service**:
    - Endpoints for managing shopping cart functionality.

7. Database Connectivity

Each microservice will utilize a MongoDB cluster as the underlying database solution. The MongoDB cluster will be a public database accessible to all microservices. Database connections will be managed using appropriate drivers and libraries for interacting with MongoDB databases in Node.js environments.

8.Conclusion

This microservices architecture design provides a scalable and modular approach for developing the e-commerce application. By decomposing the application into smaller services, it enables easier maintenance, independent deployment, and scalability of individual components.