Technical Design for DEMO Microservices

**Overview:**

Build an application based on microservices architecture using Spring cloud. Spring cloud provide libraries to apply common patterns needed in the distributed applications.Spring cloud with help of Netflix OSS addresses security,maintainablity,availability,fallback ,monitoring, Load Balancing issues in the microservices application. The configurations and the libraries which are used to address the above concerns are detailed in this document.

**Problem Statement:**

* Create a login microservice (Spring boot) and authenticate user using basic Spring security features.
* After authentication, redirect user to Product microservice (Spring boot) and show available products from the catalog (Have some dummy catalog created using in-memory/Cassandra/MongoDB)
* Create a microservice to fetch price and inventory information (Spring boot) from the Mongo DB database
* Create a microservice to add item to cart and display the item to cart page with price information, create operation like add/update/delete cart items.
* Create a microservice gateway (Zuul) and Eureka to access all your microservices.
* At least write for one microservice, Unit testing, integration testing, contract testing from the above using Junit, mockito, RestAssured.

An in Memory database (H2) is used to store all the repository data. A component which implements CommandLineRunner interface is used to populate the initial repository data (Product data, Price data, User data etc.) during application startup.

**Approach:**

Based on the above problem statement, following microservices will be developed based on Spring cloud architecture.

login microservice:

This service is used to authenticate the user using Spring security features. We will be using the starter dependency ‘<spring-boot-starter-security>’ to enable spring security and additional Configuration class which extends WebSecurityConfigurerAdapter to create an InMemory authentication and roles for the users. User will be redirected to Product Page once successfully logged in.

Product info service:

Product info service is used to populate Product detail page. Eureka service Discovery is used for Service Registration and auto discovery.

Price Service:

Price Service is used to fetch the Price information for the Product. The DAO methods of the Price Entity classes are annotated with @RepositoryRestResource to expose as RESTful services using Spring Data REST.Spring cloud Netflix Zuul and Eureka provides client side load balancing ( Netflix ribbon) and fallback capabilities (Netflix Hystrix).

Inventory Service:

Inventory Service is used to fetch the Stock information for the Product. Spring cloud Netflix Zuul and Eureka provides client side load balancing ( Netflix ribbon) and fallback capabilities (Netflix Hystrix).

Cart Service:

Cart service is used to support operations like add/update/delete cart items. A new Entity ‘Cart’ will be created to support the data structure needed for cart operations. All the DAO methods in the Cart Entity class will be exposed as RESTful services using Spring DATA REST. This service also in turn call Price service and Inventory service to get Price and inventory information. To improve the performance the individual calls to Price service and Inventory service are made asynchronous.



**Addressing cross-cutting concerns:**

External and Centralized Configuration management:

Configure Spring cloud config Server for achieving Centralized configuration management capabilities. A ConfigData folder in a GIT repository which supports versioning is used as source folder for Configuration Data.

Automatic Service Discovery:

Client services will be automatically registered and accessible to other client services using @EnableDiscoveryClient annotation (Passive Service Discovery). Eureka Servers are configured to run in multiple instances and can communicate with each other to ensure the availability.

Security:

We will be using the starter dependency ‘<spring-boot-starter-security>’ to enable spring security and additional Configuration class which extends WebSecurityConfigurerAdapter to create an InMemory authentication and roles for the users.

Cascading Failures and Fallback:

Spring cloud providers wrapper libraries to avoid cascading failures when some of the services are not available using Netflix OSS Hystrix. Hystrix also provides fallback methods when services are not available.

Monitoring:

Spring Boot actuator dependency brings in several Production grade services and endpoints for monitoring the Spring application.

Error Handling:

Spring Boot automatically map errors and exceptions to a default White label error page.

**Build Tool:**

Maven is used as build and dependency management tool.