**E-Shopping Zone**

Low-Level-Design (LLD) for Order-Management Microservice

**Date: 09/07/2022**

**Current Document Version: 1.0**

DOCUMENT APPROVAL

**Approvers of this document**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Department** | **Role** | **Signature** | **Date** |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Document Change History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Version #** | **Author** | **Date** | **Description** |
| 1.0 | Naga Manindra | 09/07/2022 | LLD for Order-Management Microservice |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Table of Contents**

1.0 Document Purpose 4

2.0 Intended Audience 4

3.0 Project Background 4

4.0 Design Pattern 5

5.0 Solution Steps 5

6.0 Classes 6

7.0 Data Model 6

8.0 API canvas 7

9.0 ENV Variable 7

10.0 HTTP Status Code 8

11.0 Response 8

# Document Purpose

This document describes the solution architecture for E-Shopping Zone Order-Management Microservice.

# Intended Audience

This document is intended as a reference for the following roles and stakeholders who are interested in the E-Shopping Zone Microservice technical architecture.

|  |  |
| --- | --- |
| Role | Nature of Engagement in WB Classics Portal Technical Architecture |
| Product Owners/SME | Key stakeholder to ensure that the architecture is aligned with business goals. |
| Business Analysts | Business analysts are one of the stakeholders who are informed with the key architectural decisions. |
| Enterprise Architects | To enforce E-Shopping Zone Platform Architecture is aligned to business goals and architecture, architectural guidelines. |
| Solution Architects | To ensure solution design and architecture is aligned to business requirements, architectural guidelines. |
| Developers | Use Technical Architecture Document as the guiding document for detail design and implantation approach to align with E-Shopping Zone Microservice |

# Project Background, Objective(s)

## Project Background

E-Shopping Zone is an ecommerce website, E-commerce is the activity of buying or selling of products on online services or over the internet. There are two roles one is merchant where he sells the products and other was the customers. Here customer can create account and login to their active account. Users can browse various types of products. Products on this site will be displayed category vies.

## Objective

Order-Management Microservice ​ will perform the following

* User should be able perform CRUD operations in this microservice.
* User should be able to place order by providing cartId.
* User should be able to get orders by orderId or username.
* User should be able to get orders by current date.
* It should be able to delete order.
* It should be able to update order Status.

# Design Pattern

|  |  |  |
| --- | --- | --- |
| # | Name | Description |
| 1 | API | Using HTTP requests, we will use the respective action to trigger various operations |

# 5.0 Solution Steps

**Order**

Adding Order:

When user clicks on place order, call reaches api gateway, Api gateway forwards call to orderController.createOder() which calls orderService.createOder () , which calls orderRepo.save() which saves the items in the cart as order into database.

Get current day Orders:

When user clicks on orders, call reaches api gateway, Api gateway forwards call to orderController.getTodaysOrders() in which order are filtered by date and calls orderService.getByCustomerId(), which calls orderRepo.findByCustomerId() which remove the order in database.

Get all Orders:

When user clicks on All orders, call reaches api gateway, Api gateway forwards call to orderController.getAllOrders() which calls orderService.getByCustomerId(), which calls orderRepo.findByCustomerId() which remove the order in database.

Update Orders:

After Successful payment, the call reaches api gateway. Api gateway forwards call to orderController.updateOrder() which calls orderService.getByOrderId() and orderService.updateOrder (), which calls orderRepo.save () which update the order status in database.

# 6.0 Classes

|  |  |  |
| --- | --- | --- |
| **#** | **Class** | **Description** |
| 1 | order | Model holds the order schema details |
| 2 | orderRepo | This class deals with the data accessibility for order. |
| 3 | orderService | It contains the core logic for the adding, removing order. Which calls the orderRepo class to connect to order database |
| 4 | orderController | This class deals with the data accessibility for cart creation, editing, removing and gets data from cart. |

# 7.0 Data Model

|  |  |
| --- | --- |
| Cart |  |
| id | int |
| totalPrice | double |
| items | List<Items> |

|  |  |
| --- | --- |
| Items |  |
| product | Product |
| subTotal | double |
| quantity | int |

|  |  |
| --- | --- |
| Product |  |
| productName | string |
| productId | int |
| price | double |

|  |  |
| --- | --- |
| Order |  |
| id | int |
| date | localDate |
| customerId | int |
| totalPrice | double |
| status | string |
| address | Address |
| number | long |
| items | List<Items> |

# 8.0 API canvas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Micro Service | Path | Verb | API Description | Role |
| Order-Management | /user/addOrder/{custId} | POST | Order creation | User |
| Order-Management | /user/getOrder/{custId} | GET | To get order list | User |
| Order-Management | /user/getTodayOrder/{custId} | GET | To get details of a current day order | User |
| Order-Management | /user/updateOrder/{orderId} | PUT | To update status of an order | NA |
| Order-Management | /user/deleteOrder/{orderId} | DELETE | To get details of a current day order | User |

# 9.0 ENV variables

server:

port: 9004

spring:

data:

mongodb:

uri: mongodb+srv://manindra08:Naga7550@cluster0.uwtgi.mongodb.net/order

cloud:

config:

uri: http://localhost:8888

# 10.0 HTTP status code

200 - Request succeeded

201 – Created

204 – No Content

208 - Already reported

400 – Inputs are invalid

404 –Page Not found

502 – Bad gateway

**11.0 Response**

**If order placed**

{

"message": "successfully"

}

status code: 201

**If server encounters unexpected error**

{

“message”: “Internal server error”

}

status code: 500

**If orderId not in database**

{

“message”: “order not found”

}

status code: 500

**If order deleted successfully**

{

“message”: “user deleted”

}

status code: 204