**E-Shopping Zone**

Low-Level-Design (LLD)

**Date: 04/05/2022**

**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 | 04/05/2022 | E-Shopping Zone LLD |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**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 Diagram 5

6.0 Solution Steps 6

7.0 Classes 9

8.0 Data Model 10

9.0 API canvas 12

10.0 ENV Variable 13

11.0 HTTP Status Code 13

12.0 Response 13

# Document Purpose

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

## Project Objective

E-Shopping Zone Microservice​ will perform the following

* User can view the product without login and access the website.
* User can login with the account with which he registered in the website in order to be able to use the website.
* User can add or remove products from the cart.
* For placing the order user has to login or create the account if he doesn’t have active account.
* User can update the profile whenever he needs to update.
* User can view their previous orders.

# Design Pattern

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

# Solution Diagram

Diagram, schematic

Description automatically generated

# 6.0 Solution Steps

**User**

User Sign-up:

1. If user is not registered before, User will enter the required details such as username, fullName, email, password.
2. Call reaches api gateway.
3. API gateway does the routing and forward call to userController.createUser() to perform validation.
4. If validation fails, then it will return the error code and error description with status code
5. If validation is successful, then the function createUser() will call the userService.createUser() which will call the userRepo.save() to store the data in database
6. After successful registration and login user can view the cart and place orders.

User Sign-in:

1. If user had registered already, user can login by entering email and password.
2. Call reaches api gateway.
3. API gateway forwards call to userController.getUserByUserName () which calls userService.getByUserName() function which calls userRepo.findByUserName().
   1. If userName and password match as in database, then the user is logged in successfully.
   2. If userName and password does not match, then it throws an error code and error description with status code.
4. After successful login user can view cart and place orders.

User Update:

1. If user want to update his/her profile, then he/she can update his profile by providing updated details like fullName, email, gender, dob, mobile number, address.
2. Call reaches api gateway.
3. API gateway forwards call to userController.updateUser () which calls userService. updateUser () function which calls userRepo.save().
4. If validation fails, then it will return the error code and error description with status code
5. If validation is successful, then the function updateUser () will calls userService.updateUser () function which calls userRepo.save(). to update the data in database
6. After successful update user can view the changes.

User Delete:

1. If user want to delete his/her profile, user can delete by his/her profile.
2. Call reaches api gateway.
3. API gateway forwards call to userController.deleteUser () which calls userService.deleteByUserName () function which calls userRepo.findByUserName().
   1. If userName is present in database, then the profile will be deleted.
   2. If userName is no found in database, then it throws an error code and error description with status code.

**Product**

All products:

When user want to view all the products, call reaches api gateway, Api gateway forwards call to UserProductController.getAllProducts() which calls productService.getAll() function which calls productRepo.findAll() which returns all the products stored in database.

Search by Category:

If user want search product by category, call reaches api gateway, Api gateway forwards call to UserProductController. getProductByCategory () which calls productService.getProductByCategory() function which calls productRepo.findProductByCategory () which returns all the products by category stored in database.

Search by Id:

If user want search product by category, call reaches api gateway, Api gateway forwards call to UserProductController. getProductById() which calls productService.getProductById() function which calls productRepo.findProductById() which returns all the products by id stored in database.

Search by Type:

If user want search product by category, call reaches api gateway, Api gateway forwards call to UserProductController.getProductByType() which calls productService.getProductByType() function which calls productRepo.findProductByType() which returns all the products by type stored in database.

Search by Name:

If user want search product by category, call reaches api gateway, Api gateway forwards call to UserProductController. getProductByName() which calls productService.getProductByName() function which calls productRepo.findProductByName() which returns all the products by Name stored in database.

Add Product:

If admin want to add Product, call reaches api gateway, Api gateway forwards call to AdminProductController.addProduct() which consumes Product details contain productId, productName, Type, Category, price, image, rating, description and calls productService.createProduct() function which calls productRepo.save() which creates new document in database.

Delete Product:

If admin want to delete product, call reaches api gateway, Api gateway forwards call to AdminProductController.deleteProduct() which calls productService.getProductById() and calls productService.delete() function which calls productRepo.remove() and which deletes the product stored in database.

Update Product:

If admin want to update product, call reaches api gateway, Api gateway forwards call to AdminProductController.updateProduct() which calls productService.getProductById() and calls productService.update() function which calls productRepo.save() and which update the product stored in database.

**Cart**

Adding to Cart:

When user clicks on add to cart, call reaches api gateway, Api gateway forwards call to cartController.addCart() which gets userName, productId and calls cartService.createCart() function which calls cartRepo.save() to save to the cart database.

Show Cart:

When user clicks on cart, call reaches api gateway, Api gateway forwards call to cartController.getCartByUserId() which calls cartService.getCart() function which calls cartRepo.findByCartId() to show all the added products in cart database.

Edit Cart:

When user make changes in cart, call reaches api gateway, Api gateway forwards call to cartController.updateItemsInCart() which gets userName, productId and calls cartService.updateCart() in which sub function getById() gets the cart details to update and it calls cartRepo.save() to save changes to cart database.

Remove Cart:

When user click on delete in cart, call reaches api gateway, Api gateway forwards call to cartController.deleteItemFromCart() which gets userId, productId and calls cartService.deleteCart() in which sub function getById() gets the cart details to delete and it calls cartRepo.save() to remove item from cart database.

**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.

# 7.0 Classes

|  |  |  |
| --- | --- | --- |
| **#** | **Class** | **Description** |
| 1 | user | Model holds the user schema details |
| 2 | userRepo | This class deals with the data accessibility for user |
| 3 | userService | It contains the core logic for the adding, removing, updating, getting users. Which calls the userRepo class to connect to user database |
| 4 | userController | This class deals with the data accessibility for user registration. |
| 5 | product | Model holds the product schema details |
| 6 | productRepo | This class deals with the data accessibility for product |
| 7 | productService | It contains the core logic for the adding, removing, updating, getting products. Which calls the productRepo class to connect to product database |
| 8 | productController | This class deals with the data accessibility for getting products. |
| 9 | cart | Model holds the cart schema details |
| 10 | cartRepo | This class deals with the data accessibility for cart. |
| 11 | cartService | It contains the core logic for the adding, removing, updating, getting cart. Which calls the cartRepo class to connect to cart database |
| 12 | cartController | This class deals with the data accessibility for cart creation, editing, removing, displaying and getting data from product and user. |
| 13 | order | Model holds the order schema details |
| 14 | orderRepo | This class deals with the data accessibility for order. |
| 15 | orderService | It contains the core logic for the adding, removing order. Which calls the orderRepo class to connect to order database |
| 16 | orderController | This class deals with the data accessibility for cart creation, editing, removing and gets data from cart. |

# 8.0 Data Model

|  |  |
| --- | --- |
| User |  |
| userName | int |
| name | string |
| email | string |
| number | long |
| dob | LocalDate |
| gender | string |
| role | string |
| password | string |
| address | address |

|  |  |
| --- | --- |
| Address |  |
| houseNo | int |
| streetName | string |
| colonyName | string |
| city | string |
| state | string |
| pincode | int |
| Product |  |
| id | int |
| name | string |
| image | string |
| type | string |
| price | double |
| category | string |
| description | string |

|  |  |
| --- | --- |
| 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> |

# 9.0 API canvas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Micro Service | Path | Verb | API Description | Role |
| User-Management | /user/new/register | POST | User registration | No |
| User-Management | /admin/getAll | GET | To get user list | Admin |
| User-Management | /user/update/{userName} | PUT | To update user details | No |
| User-Management | /user/{userName} | GET | To get details of a particular user | No |
| User-Management | /user/delete/{userName} | DELETE | To remove the user | No |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Micro Service | Path | Verb | API Description | Role |
| Product-Management | /admin/addProduct | POST | Product creation | Admin |
| Product-Management | /admin/getAll | GET | To get product list | Admin |
| Product-Management | /admin/update/{id} | PUT | To update product details | Admin |
| Product-Management | /admin/delete/{id} | DELETE | To remove the product | Admin |
| Product-Management | /admin/getById/id | GET | To get details of a particular product | Admin |
| Product-Management | /admin/getByName/name | GET | To get details of a particular product | Admin |
| Product-Management | /admin/getByType/type | GET | To get details of a particular product | Admin |
| Product-Management | /admin/getByCateogory/category | GET | To get details of a particular product | Admin |
| Product-Management | /user/getById/id | GET | To get details of a particular product | No |
| Product-Management | /user/getByName/name | GET | To get details of a particular product | No |
| Product-Management | /user/getByType/type | GET | To get details of a particular product | No |
| Product-Management | /user/getByCateogory/category | GET | To get details of a particular product | No |
| Product-Management | /user/getAll | GET | To get product list | No |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Micro Service | Path | | | Verb | | API Description | | Role | |
| Cart-Management | /user/additem/{cartId}/{productId} | | | POST | | Cart creation | | User | |
| Cart-Management | /user/getcart/{cartId} | | | GET | | To get cart list | | user | |
| Cart-Management | /user/updateitem/{id}/{productId}/  {quantity} | | | PUT | | To update cart details | | User | |
| Cart-Management | /user/deleteitem/{cartId}/{productId} | | | DELETE | | To remove the cart | | User | |
| 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 | |

# 10.0 ENV variables

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

# 11.0 HTTP status code

200 - Request succeeded

400 – Inputs are invalid

404 –Page Not found

502 – Bad gateway

**12.0 Response**

**If valid details**

{

"message": "successfully"

}

status code: 200

**If invalid details**

{

“message”: “Inputs are not valid”

}

status code: 400

**If server encounters unexpected error**

{

“message”: “Internal server error”

}

status code: 500