**E-Shopping Zone**

Low-Level-Design (LLD) for User-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 User-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 7

8.0 API canvas 7

9.0 ENV Variable 8

10.0 HTTP Status Code 8

11.0 Response 8

# Document Purpose

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

User-Management Microservice ​ will perform the following

* User should be able perform CRUD operations in this microservice.
* User should be able to create his account, where username should be unique.
* User should be able to get his account details.
* User should be able to update his profile by provide necessary details, which have validations.
* User should be able to delete his profile.
* Only Admin should be able to get all user details.

# Design Pattern

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

# 5.0 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.

Admin Get:

1. If admin want to get all user’s profile.
2. Call reaches API gateway.
3. API gateway forwards call to userController.getAllUsers() which calls userService.getAll() function which calls userRepo.findAll().
4. Then all user in database will be displayed.

# 6.0 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, update, delete, get by userName |
| 5 | adminController | This class deals with admin to get all users |
| 6 | userNotFound | This class deals with exception handling, whenever wrong username us called this class is accessed. |

# 7.0 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 |

# 8.0 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 |

# 9.0 ENV variables

server:

port: 9007

spring:

data:

mongodb:

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

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 valid details**

{

"message": "successfully"

}

status code: 201

**If invalid details**

{

“message”: “Inputs are not valid”

}

status code: 400

**If server encounters unexpected error**

{

“message”: “Internal server error”

}

status code: 500

**If user not in database**

{

“message”: “user not found”

}

status code: 500

**If user already exist in database**

{

“message”: “user already exist”

}

status code: 208

**If user deleted successfully**

{

“message”: “user deleted”

}

status code: 204