

**Applied Software Project Report**

By

<Nourth Arvinder Pal Singh >

**A Master’s Project Report submitted to Scaler Neovarsity - Woolf in partial fulfillment of the requirements for the degree of Master of Science in Computer Science**

<Month of Submission March, 2025>



**Scaler Mentee Email ID :** <arvinderpalsingh2321@gmail.com>

**Thesis Supervisor :** Anurag Khanna

**Date of Submission :** 25/03/2025

**Certification**

I confirm that I have overseen / reviewed this applied project and, in my judgment, it adheres to the appropriate standards of academic presentation. I believe it satisfactorily meets the criteria, in terms of both quality and breadth, to serve as an applied project report for the attainment of Master of Science in Computer Science degree. This applied project report has been submitted to Woolf and is deemed sufficient to fulfill the prerequisites for the Master of Science in Computer Science degree.

Anurag Khanna

…………………

Project Guide / Supervisor

**DECLARATION**

I confirm that this project report, submitted to fulfill the requirements for Master of Science in Computer Science degree, completed by me from 30-05-2024 to 20-11-2024 is the result of my own individual endeavor. The Project has been made on my own under guidance of my supervisor with proper acknowledgment and without plagiarism. Any contributions from external sources or individuals, including the use of AI tools, are appropriately through citation. By making this declaration. I acknowledge that any violation if this statement constitutes academic misconduct. I understand that such misconduct may lead to expulsion from the program and/or disqualification from receiving degree.

**<Nourth Arvinder Pal Singh>**

**<Signature of the Candidate>                                   Date: 25 March 2025**

**ACKNOWLEDGMENT**

I would like to express my gratitude to my family (Tunzala, and Sahib Singh) for their support, encouragement and belief in me throughout this journey. Their patience and motivation have been my greatest strength. I am thankful to the dedicated instructors and mentors of Scaler community, whose guidance, knowledge, and insightful teachings have played a crucial role in shaping my understanding of computer science and programming. I am thankful to everyone who has inspired and motivated me to push my limits and successfully complete this program. This journey would not have been possible without your support,

**Table of Contents**

Contents

[List of Figures 8](#_Toc202859682)

[**List of Tables** 9](#_Toc202859683)

[1. Applied Software Project 10](#_Toc202859684)

[Abstract 10](#_Toc202859685)

[1.1 Project Description 11](#_Toc202859686)

[1.2 Key Components: 12](#_Toc202859687)

[Technologies and tools used: 12](#_Toc202859688)

[Java mail configuration 13](#_Toc202859689)

[Kafka Topic 13](#_Toc202859690)

[Kafka configuration Yml file 13](#_Toc202859691)

[Inside Services Configurations 14](#_Toc202859692)

[Eureka Server Yml File 14](#_Toc202859693)

[Redis Configuration 15](#_Toc202859694)

[Api-Gateway server 15](#_Toc202859695)

[Stripe Configuration 16](#_Toc202859696)

[1.3 Project work flow 17](#_Toc202859697)

[2. Project Development Process 31](#_Toc202859698)

[1. Requirements Gathering 32](#_Toc202859699)

[1.1 Functional Requirements 32](#_Toc202859700)

[1.2 Non Functional Requirements 33](#_Toc202859701)

[2. Users and Use Cases 33](#_Toc202859702)

[2.1 User Types 33](#_Toc202859703)

[2.2 Use Case Diagram 34](#_Toc202859704)

[Use Case List of features: 35](#_Toc202859705)

[3. Class Diagrams 36](#_Toc202859706)

[3.1 User Service 36](#_Toc202859707)

[3.5 Role Authority Service 39](#_Toc202859708)

[3.6 Authrisation server Registered clinet or OICD client 40](#_Toc202859709)

[3.7 Product-service (Category-service) 41](#_Toc202859710)

[3.8 Product service 42](#_Toc202859711)

[3.9 Cart service 43](#_Toc202859712)

[3.10 Order service 44](#_Toc202859713)

[3.11 Payment service 45](#_Toc202859714)

[3.12 Delivery service 46](#_Toc202859715)

[3.13 Delivery service (user implementation) 47](#_Toc202859716)

[4. Database Schema Design 48](#_Toc202859717)

[4.1 User service schema diagram 48](#_Toc202859718)

[4.2 Product service 49](#_Toc202859719)

[4.3 Cart service 49](#_Toc202859720)

[4.4 Order service 50](#_Toc202859721)

[4.5 Delivery service 50](#_Toc202859722)

[5. Textual schema design 51](#_Toc202859723)

[5.1 User Service textual schema 51](#_Toc202859724)

[5.2 Product Service textual schema 51](#_Toc202859725)

[Cardinalities 52](#_Toc202859726)

[5.3 Cart Service textual schema 52](#_Toc202859727)

[Cartitems relationship 53](#_Toc202859728)

[5.4 Order Service textual schema 53](#_Toc202859729)

[5.5 Delivery Service textual schema 54](#_Toc202859730)

[6. Feature Development Process 56](#_Toc202859731)

[6.1 Topic : Track delivery from warehouse to user delivery address 56](#_Toc202859732)

[6.3 Service handling request 58](#_Toc202859733)

[6.4 MVC Architecture 58](#_Toc202859734)

[7. Deployment Flow 59](#_Toc202859735)

[7.1 GIT commit and version control 59](#_Toc202859736)

[7.2 CI (Continues integration) 59](#_Toc202859737)

[7.3 CD (Continues Deployment) 59](#_Toc202859738)

[7.4 AWS -Production deployment architect 60](#_Toc202859739)

[7.5 Post-Deployment 61](#_Toc202859740)

[7.6 Rollback strategy 61](#_Toc202859741)

[7.7. Deployment Diagram 62](#_Toc202859742)

[8.Technologies Used 63](#_Toc202859743)

[8.1 Kafka 63](#_Toc202859744)

[8.2 Mysql (relational database management system(RDBMS) 64](#_Toc202859745)

[8.3 Spring boot 64](#_Toc202859746)

[8.4 Spring security with Oauth2 65](#_Toc202859747)

[8.5 Redis (in memory data storage) 65](#_Toc202859748)

[8.6 (AWS) Amazon Web Services 66](#_Toc202859749)

[Conclusion 67](#_Toc202859750)

[9. Key takeaways 68](#_Toc202859751)

[9.1 Spring boot and java 17 68](#_Toc202859752)

[9.2 Microservices architectures 68](#_Toc202859753)

[9.3 Authentication And Security With Oauth2 and JWT 68](#_Toc202859754)

[9.4 Mysql database 68](#_Toc202859755)

[9.5 AWS cloud 68](#_Toc202859756)

[Practical applications 68](#_Toc202859757)

[Web applications 68](#_Toc202859758)

[Data caching 69](#_Toc202859759)

[e-commerce flows 69](#_Toc202859760)

[Eureka server (service discovery) 69](#_Toc202859761)

[Limitations 69](#_Toc202859762)

[Manual deployment overhead 69](#_Toc202859763)

[Complexity of service coordination 69](#_Toc202859764)

[Caching consistency 69](#_Toc202859765)

[Final thoughts 70](#_Toc202859766)

[References 71](#_Toc202859767)

[For websites and articles 71](#_Toc202859768)

List of Figures

[Figure : 1:1 Microservice Layer Architecture 11](#_Toc202859769)

[Figure : 1:2 Project Flow 30](#_Toc202859770)

[Figure : 2:1 1Project Development Process 31](#_Toc202859771)

[Figure : 2:2 Use Case Diagram 34](#_Toc202859772)

[Figure : 3:1 User Service Implementation 36](#_Toc202859773)

[Figure : 3:2 User service Authorization server consent service 37](#_Toc202859774)

[Figure : 3:3 JpaAuthorisation service 38](#_Toc202859775)

[Figure : 3:4 Role Authority Service 39](#_Toc202859776)

[Figure : 3:5 Authrisation server Registered clinet or OICD client 40](#_Toc202859777)

[Figure : 3:6 Product-service (Category-service) 41](#_Toc202859778)

[Figure : 3:7 Product service 42](#_Toc202859779)

[Figure : 3:8 Cart Service 43](#_Toc202859780)

[Figure : 3:9 Order service 44](#_Toc202859781)

[Figure : 3:10 Payment service 45](#_Toc202859782)

[Figure : 3:11 Delivery Service 46](#_Toc202859783)

[Figure : 3:12 Delivery Service(user implementation) 47](#_Toc202859784)

[Figure : 4:1 User service schema diagram 48](#_Toc202859785)

[Figure : 4:2 Product service 49](#_Toc202859786)

[Figure : 4:3 Cart service 49](#_Toc202859787)

[Figure : 4:4 Order service 50](#_Toc202859788)

[Figure : 4:5 Delivery service 50](#_Toc202859789)

[Figure : 7:1 Deployment flow Diagram 62](#_Toc202859790)

**List of Tables**

# 

[Table : 2‑1 Feature List 35](#_Toc202858744)

# **Applied Software Project**

## Abstract

Today E-commerce is one of the widely used application across the world. Its impact become more on the customers especially after covid-19 breakout. As most of people now prefer to buy the products rather visit market, as example of from children’s toys, medicine, grocery, furniture, food almost everything is possible to buy online. it saved valuable time and due to fast delivery services, it becomes more convenient and gain the trustable market.

In this e-commerce Project, I aimed to develop a secure, efficient and scalable platform using backend technologies like the system uses token based security system Oauth2 implemented by spring boot to secure user data and all passwords are encrypted by BcryptPassword Encoder.

To increase the performance Redis is integration so that customer will get fastest browsing experience, the platform also implements role based authentication which separates general users and admin and provide them access to appropriately.

For secure and reliable payments, the system integration stripe which I widely used in most of e-commerce platforms because of it stability and flexible. to provide customer real time update Kafka and java mail notification system provided to customer, the delivery service accurate calculate and displays estimated delivery time after success purchase of item.

To persist data MySQL is best of my knowledge which offers reliable and structured storage for transactional systems.

By completing this project the aim is to show how backend skills and real time integration with API’s can provide better results for both users and businesses in the online shopping industry.

## 1.1 Project Description

this E-commerce backend platform project is based on spring boot microservice layered architect, which includes UserService, ProductService, CartService, OrderService, PaymentService, DeliveryServcie, Kafka and JavaMail service, and API-Gateway for API request routing, eureka server for service discovery, for data storage MySQL, this project will provide seamless communication between services by RestTemplates.

The best of this project is user can freely browse products can category but when user want to purchase then signup will be required.

**Microservice Layer Architecture Diagram**

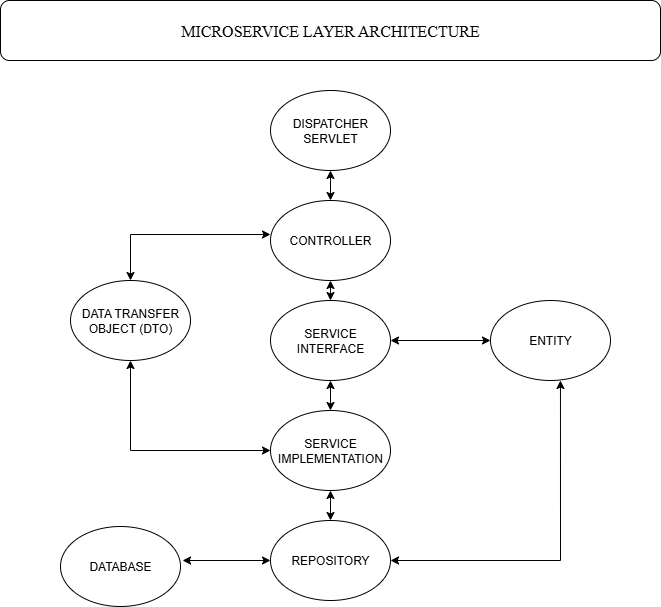


Figure : 1:1 Microservice Layer Architecture

### 1.2 Key Components:

UserService: Role based access, managed by authentication and authorization registration for protection of profile and BcryptPasswordEncorder to encode passwords.

ProductService: handling products and category crud operations, for better performance used Redis integration.

CartService: add or remove product from cart according to customer choice.

OrderService: if cart is confirmed then this service processing orders and making product ready to sell.

PaymentService: most advance payment service as Stripe by providing safe and secure transactions.

DeliveryService: delivery to European countries only and track order status.

KakfaService and JavaMail: used for sending email notification.

Api-Gateway: used as api gateway for product unified entry point for all client requests and routing them to appropriate services.

EurekaServer: it make sure each microservice communicate properly and can be tracked by this server.

## ****Technologies and tools used:****

* **Programming Language:** Java 17
* **Framework:** Spring Boot
* **Database:** MySQL, database for storing data, user, product and category,cart, orders, and delivery destinations.
* **Notifications and JavaMail :** Kafka, configured via Docker, is used and JavaMail for sending email notifications
* **Redis:** used for cache via docker
* Spring Boot Starter OAuth2 Resource Server (for authentication
* **JWT(JSON WebTokens for security)**
* **API-Gateway : routing requests to services.**
* **Eureka server:** microservice communicate
* RestTemplate : to provide smooth communication between microservices

**Server ports**

* Cart-service : 8085
* Order service : 8086
* Payment service : 8087
* Api-Gateway : 8088
* Product-service : 8089
* User-service : 8090
* Delivery service :8091
* Eureka server : 8761
* Kafka-server : 9000

### Java mail configuration

(recommended to add Javamail dependency)

spring.mail.host=smtp.gmail.com  
spring.mail.username=Pattorney0@gmail.com  
spring.mail.password=${PASS}  
spring.mail.properties.mail.smtp.auth=true  
spring.mail.port=587  
spring.mail.protocol=smtp  
spring.mail.properties.mail.smtp.starttls.enable=true  
spring.mail.properties.mail.smtp.ssl.trust=smtp.gmail.com

Kafka Topic : run via docker topic name = “sendemail” groupId =emailServcie

**Run Docker :** Install Kafka ,Zookeeper, Redis in docker

### Kafka configuration Yml file

spring:  
 kafka:  
 bootstrap-servers: localhost:9092  
 consumer:  
 group-id: emailService  
 auto-offset-reset: earliest  
 key-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
 value-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
 listener:  
 missing-topics-fatal: false

### Inside Services Configurations

spring:  
 kafka:  
 bootstrap-servers: localhost:9092  
 consumer:  
 group-id: email-service  
 auto-offset-reset: earliest  
 key-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
 value-deserializer: org.apache.kafka.common.serialization.StringDeserializer  
 producer:  
 key-serializer: org.apache.kafka.common.serialization.StringSerializer  
 value-serializer: org.apache.kafka.common.serialization.StringSerializer

### Eureka Server Yml File

spring:  
 application:  
 name: eurekaserver  
server:  
 port: 8761  
eureka:  
 client:  
 register-with-eureka: false  
 fetch-registry: false  
logging:  
 level:  
 com.netflix.eureka: OFF  
 com.netflix.discovery: OFF

### Redis Configuration

spring.data.redis.host=localhost  
spring.data.redis.port=6379

### Api-Gateway server

spring.application.name=apigateway  
server.port=8088  
eureka.client.register-with-eureka=true  
eureka.client.fetch-registry=true  
eureka.client.service-url.defaultZone=http://localhost:8761/eureka  
spring.cloud.compatibility-verifier.enabled=true  
instance.prefer-ip-address=true  
logging.level.com.netflix.discovery=DEBUG  
logging.level.com.netflix.eureka=DEBUG  
spring.cloud.gateway.routes[0].id=productservice  
spring.cloud.gateway.routes[0].predicates[0]=Path=/product/\*  
spring.cloud.gateway.routes[0].uri=lb://productservice  
spring.cloud.gateway.routes[1].id=cartservice  
spring.cloud.gateway.routes[1].predicates[0]=Path=/cart/\*  
spring.cloud.gateway.routes[1].uri=lb://cartservice  
spring.cloud.gateway.routes[2].id=orderservice  
spring.cloud.gateway.routes[2].predicates[0]=Path=/order/\*  
spring.cloud.gateway.routes[2].uri=lb://orderservice  
spring.cloud.gateway.routes[3].id=userservice  
spring.cloud.gateway.routes[3].predicates[0]=Path=/user/\*  
spring.cloud.gateway.routes[3].uri=lb://userservice  
  
spring.cloud.gateway.routes[4].id=paymentservice  
spring.cloud.gateway.routes[4].predicates[0]=Path=/pay/\*  
spring.cloud.gateway.routes[4].uri=lb://paymentservicespring.cloud.gateway.routes[5].id=deliveryservice  
spring.cloud.gateway.routes[5].predicates[0]=Path=/delivery/\*  
spring.cloud.gateway.routes[5].uri=lb://deliveryservice

### Stripe Configuration

stripe.secret.key=${SECRET\_KEY}  
stripe.public.key=${PUBLIC\_KEY}

## 

## ****1.3 Project work flow****

**PRODUCT-SERVICE**

**GET ALL CATEGORY**

Endpoint : GET - <http://localhost:8089/category/>

Retrieve all available product categories. Choose your preferred category from the displayed list.

Reponse

[

    {

        "id": 1,

        "categoryName": "ELECTRONICS",

        "categoryDescription": "computers and laptops",

        "productResponseDtos": **null**

    },

    {

        "id": 2,

        "categoryName": "SHOES",

        "categoryDescription": "leather",

        "productResponseDtos": **null**

    }]

**GET PRODUCT BY CATEGORY NAME**

Example: Search for products in the "PHONES" category.

Endpoint : GET - [http://localhost:8089/category/searchByCategoryName/{categoryName}](http://localhost:8089/category/searchByCategoryName/%7bcategoryName%7d)

Response

{

    "id": 5,

    "categoryName": "FURNITURE",

    "categoryDescription": "wooden and steal and iron",

    "productResponseDtos": [

        {

            "id": 21,

            "name": "Wooden Dining Table",

            "description": "6-seater dining table",

            "brand": "Ikea",

            "price": 400,

            "stock": 0,

            "image": "https://example.com/dining-table",

            "categoryes": **null**

        },

        {

            "id": 22,

            "name": "Comfortable Sofa Set",

            "description": "3-piece upholstered sofa set",

            "brand": "Ashley Furniture",

            "price": 1500,

            "stock": 0,

            "image": "https://example.com/sofa-set",

            "categoryes": **null**

        },

        {

            "id": 23,

            "name": "King Size Bed",

            "description": "Solid wood king size bed",

            "brand": "Wayfair",

            "price": 800,

            "stock": 0,

            "image": "https://example.com/king-size-bed",

            "categoryes": **null**

        },

**GET ALL PRODUCTS**

Alternatively, you can fetch all products using the endpoint:

Endpoint : GET - <http://localhost:8089/product/>

{

"id": 1,

"name": "Dell XPS 13 Laptop",

"description": "Compact and high-performance laptop",

"brand": "DELL",

"price": 1800,

"stock": 20,

"image": "//GOOGLE.com/dell-xps",

"categoryes": null

},

{

"id": 2,

"name": "HP Pavilion Desktop",

"description": "Powerful desktop PC for home and office",

"brand": "HP",

"price": 800,

"stock": 20,

"image": "https://GOOGLE.com/hp-pavilion",

"categoryes": null

}}

**GET PRODUCT BY PRODUCT NAME**

Endpoint :GET - <http://localhost:8089/product/getProductByName/hp>

Reponse

[

    {

        "id": 2,

        "name": "HP Pavilion Desktop",

        "description": "Powerful desktop PC for home and office",

        "brand": "HP",

        "price": 800,

        "stock": 20,

        "image": "https://GOOGLE.com/hp-pavilion",

        "categoryes": **null**

    }

]

**CART-SERVICE**

**POST PUBLIC ADD TO CART**

Add products to your cart by specifying the product ID and quantity in the request payload.

Endpoint : POST - [**http://localhost:8085/cart/add**](http://localhost:8085/cart/add)

REQUEST :

{

"item": [

{

"productId": 1,

"quantity": 5

},

{

"productId": 2,

"quantity": 3

}

]

}

RESPONSE :

{

    "cartStatus": "ACCEPTED",

    "cartId": 1,

    "items": [

        {

            "cartId": 1,

            "productId": 1,

            "productName": "Dell XPS 13 Laptop",

            "quantity": 5,

            "price": 1800.0

        },

        {

            "cartId": 2,

            "productId": 2,

            "productName": "HP Pavilion Desktop",

            "quantity": 3,

            "price": 800.0

        }

    ],

    "total": 11400,

    "cartCreatedTime": "2025-07-07T09:06:04.5721319",

    "balanceStock": 17

}

**GET SAVED CARD BY ID**

Confirm the cart details and proceed to the order service.

Endpoint : GET - [**http://localhost:8085/cart/getCartById/1**](http://localhost:8085/cart/getCartById/1)

**RESPONSE :**

{

    "cartStatus": "IN\_PROGRESS",

    "cartId": 1,

    "items": [

        {

            "cartId": 1,

            "productId": 1,

            "productName": "Dell XPS 13 Laptop",

            "quantity": 5,

            "price": 1800.0

        },

        {

            "cartId": 2,

            "productId": 2,

            "productName": "HP Pavilion Desktop",

            "quantity": 3,

            "price": 800.0

        }

    ],

    "total": 11400,

    "cartCreatedTime": "2025-07-07T09:06:04.572132",

    "balanceStock": 17

}

**ORDER-SERVICE**

**1ST STEP ADD CART TO ORDER**

Place an order

Endpoint : POST - <http://localhost:8086/order/getCartById/1>

REPONSE :

{

    "orderid": 1,

    "cartId": 1,

    "orderStatus": "SUCESSFULL",

    "price": 11400

}

**FINAL ORDER**

If user already exists then you can directly fetch and add address of user in final order request if not exists then move to SIGN UP

Endpoint : GET - [http://localhost:8086/order/getUserLogin/1/{EMAIL}](http://localhost:8086/order/getUserLogin/1/%7bEMAIL%7d)

REPONSE :{

{

    "cartId": 1,

    "orderStatus": "SUCESSFULL",

    "price": 11400,

    "userDto": {

        "userId": 352,

        "userName": "abhishek",

        "userPhone": "935170075",

        "userEmail": "av0304773@gmail.com",

        "userCity": "BRATISLAVA",

        "userState": "ZAK",

        "userCountry": "SLOVAKIA",

        "userPostelCode": 1442690,

        "userHouseNumber": "1111",

        "userStreet": "paris street",

        "userLandMark": "HOSPITAL"

    },

}

IF USER NOT EXISTS THEN MOVE TO USER-SERVICE AND SIGNUP

**USER-SERVICE**

If user not exists then first need to signup as registered client to get app access

**POST - ADMIN REGISTRATION FORM**

Endpoint : POST - <http://localhost:8090/client/register>

REQUEST :

{

    "clientId":"john",

    "clientSecret":"johni123",

    "clientName":"john",

    "redirectUris":"https://oauth.pstmn.io/v1/callback",

    "postLogoutRedirectUris":"http://127.0.0.1:8090/"

}

**POST CREATE ROLE**

Create role according to role

Endpoint : POST - <http://localhost:8090/role/create>

REQUEST

{

"role":”USER"

}

**POST -SIGN UP USER**

**SIGNUP as a new user**

Endpoint : POST - <http://localhost:8090/user/signup>

REQUEST:

{

    "userName":"\*\*\*\*\*\*\*\*”,

    "userPhone":"\*\*\*\*\*\*\*\*”,

    "userPassword":"\*\*\*\*\*\*\*\*”,

    "userEmail":"\*\*\*\*\*\*\*\*”,

    "roles": ["USER","ADMIN"],

    "userHouseNumber":"\*\*\*\*\*\*\*\*”,

    "userStreet":"\*\*\*\*\*\*\*\*”,

    "userLandMark":"\*\*\*\*\*\*\*\*”,

    "city":"\*\*\*\*\*\*\*\*”,

    "state":"\*\*\*\*\*\*\*\*”,

    "country":"\*\*\*\*\*\*\*\*”,

    "postelCode":"\*\*\*\*\*\*\*\*”,

}

**FINAL ORDER**

**Now you can check your order**

Endpoint : GET - [http://localhost:8086/order/getUserLogin/1/{EMAIL}](http://localhost:8086/order/getUserLogin/1/%7bEMAIL%7d)

REPONSE :{

{

    "cartId": 1,

    "orderStatus": "SUCESSFULL",

    "price": 11400,

    "userDto": {

        "userId": 352,

        "userName": "abhishek",

        "userPhone": "935170075",

        "userEmail": "av0304773@gmail.com",

        "userCity": "BRATISLAVA",

        "userState": "ZAK",

        "userCountry": "SLOVAKIA",

        "userPostelCode": 1442690,

        "userHouseNumber": "1111",

        "userStreet": "paris street",

        "userLandMark": "HOSPITAL"

    },

}

**PAYMENT-SERVICE**

You are ready to pay please check your email for payment link or you can copy payment link from response and open in browser

GET PAY

Endpoint : GET - [http://localhost:9095/pay/1/{EMAIL}](http://localhost:9095/pay/1/%7bEMAIL%7d)

REPONSE

{

    "status": "SUCCESSFUL",

    "message": "HERE IS YOUR LINK TO PAY",

    "lineItems": [

        {

            "adjustableQuantity": {

                "enabled": **true**,

                "extraParams": **null**,

                "maximum": 10,

                "minimum": 1

            },

            "extraParams": **null**,

            "price": "price\_1Ri99IFKjxizuB9szlvE6OyO",

            "quantity": 1

        }

    ],

    "url": "https://buy.stripe.com/test\_00wfZh0wofHAgEHa0Jawo1Z"

}

**DELIVERY-SERVICE**

If payment successful then your browser will show the status of payment and estimate time of order delivery OR user can check order status by using below endpoint

EndPoint : GET - [http://localhost:8091/deliveryUser/getOrder/{email}](http://localhost:8091/deliveryUser/getOrder/%7bemail%7d)

Reponse

{

"id": 10,

"createdAt": "2025-07-08T05:27:40",

"cartId": 1,

"paymentStatus": "PAID",

"amount": 11400,

"deliveryStatus": "READY\_TO\_DELIVER",

"message": "PARCEL WILL DELIVER IN MAXIMUM 2 days "

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**For Returning Customers**

• Delivery service uses a hardcoded database to calculate the distance between the warehouse and the user’s location based on the country, state, city, and postal code. The estimated delivery time is then provided.

**Project flow diagram**

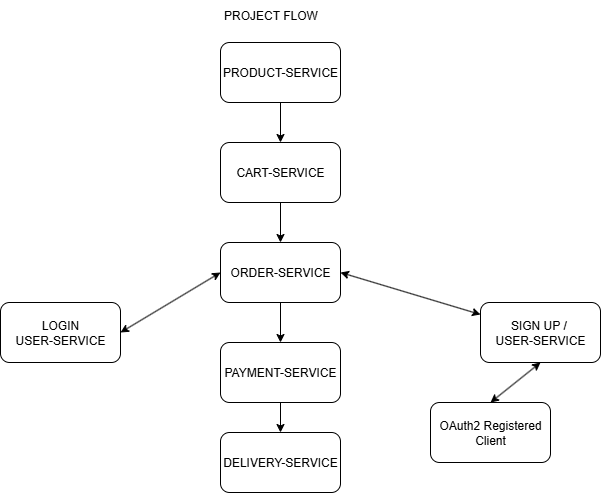


Figure : 1:2 Project Flow

# **Project Development Process**

**E-commerce Project Development Process**

Figure : 2:1 1Project Development Process

**Figure 2.1**: Project Development Process

* **Team formation** (since I am working solo, I took responsibility for all roles)
* **Topic Selection -**I selected theme of e-comProject Platform because of its real word relevance and potential to showcase multiple backend technologies.and most important popularity of e-commerce websites these day which will give me good learning experience with hands on this project.
* **Creating Project** -I created a detailed plan outlining project goals where I choose microservice architect rather then monolith architect because of it independency of functionality and scalability as (UserService, ProductService, CartService, OrderService, PaymentService, DeliveryService, and separate server for Kafka, Api-Gateway, and Eureka Server)

**Requirement Gathering**

* User-friendly
* Fast
* Loosely coupled
* Scalable
* secure

**User Requirements**:

1. Users should be able to browse products and categories without authentication or authorization.
2. Users must sign up to add products to the cart.
3. After successful signup or login, users should receive a confirmation email.
4. Users can change the quantity of selected products before placing an order.
5. Once the order is confirmed, users should receive a payment link.
6. After payment confirmation, users should receive a delivery notification.
7. Users should have the option to add or update the delivery address before the order is dispatched.
8. Users should see the estimated delivery time or date.

**Admin Requirements** :

1. Admin can add, delete, and manage all services.
2. Admin has access to API Gateway, Eureka Server, and Kafka topics.
3. Admin should have the ability to control and manage APIs.

## 1. Requirements Gathering

### 1.1 Functional Requirements

* User Registration and Role based Authentication (Signup, Login, get all users, delete user, check roles)
* Product Management (Add, Update, Delete, View Products,Delete Products,)
* Shopping Cart Management (Add, Remove, Update Items)
* Order Processing (Place Order, Delete Order, Cancel Order)
* Payment Integration (Secure Online Stripe Payments)
* Delivery Tracking (hardcoded calculations)
* Notification System (Order Updates via Kafka,user login and signup, delivery status)
* Service Discovery and API Gateway for Communication

### 

### 1.2 Non Functional Requirements

* Scalability: The system should support concurrent usage by thousands of customers, especially peak sale periods.
* Security: user data or personal data should be encrypted and handled securely using authentication mechanisms.
* Performance: api reponse should be within 300ms under average load
* Availability: application should have uptime with redundancy across critical services not less then 96%
* Maintainability: each funcanality and service should be independent to each other.

## 2. Users and Use Cases

### 2.1 User Types

* **Customers:** Browse products, select and add to cart,confirm order, make payments, and track orders.
* **Admin:**delete and add products , update products, delete users, check list of users, check list of users, check role based users, edit stock of products after purchase.
* **Delivery Personnel:** Update delivery status and track orders.

### 2.2 Use Case Diagram

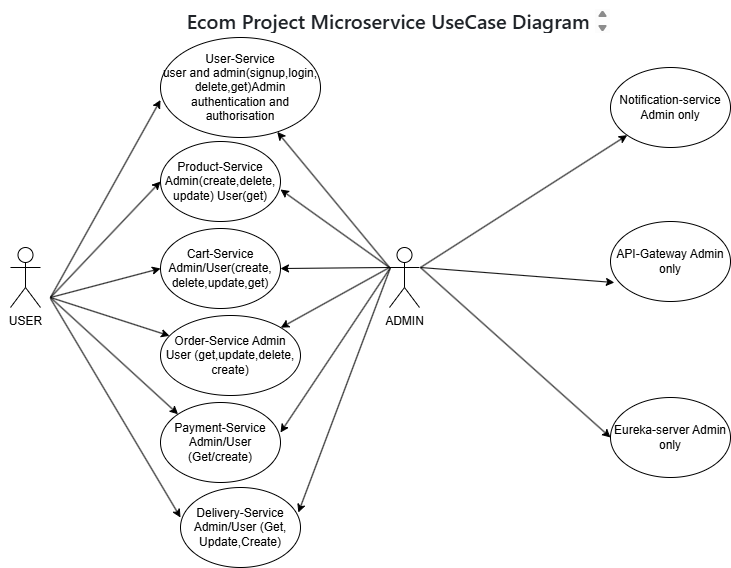


Figure : 2:2 Use Case Diagram

### Use Case List of features:

|  |  |  |
| --- | --- | --- |
| **Feature ID** | **Feature Name** | **Description** |
| 1 | User Authentication | Users can register, log in, and manage profiles securely. |
| 2 | Product Management | Admins can add, update, and delete products. Users can view products. |
| 3 | Shopping Cart | Users can add, remove, and update items in the cart. |
| 4 | Order Processing | Users can place and track orders. Admins can manage orders. |
| 5 | Payment Integration | Secure payment processing for orders. |
| 6 | Delivery Tracking | Users and delivery personnel can track order status. |
| 7 | Notification System | Kafka-based notifications for order updates. |
| 8 | Service Discovery | Eureka-based service registry for microservices communication. |

Table : 2‑1 Feature List

# **Class Diagrams**

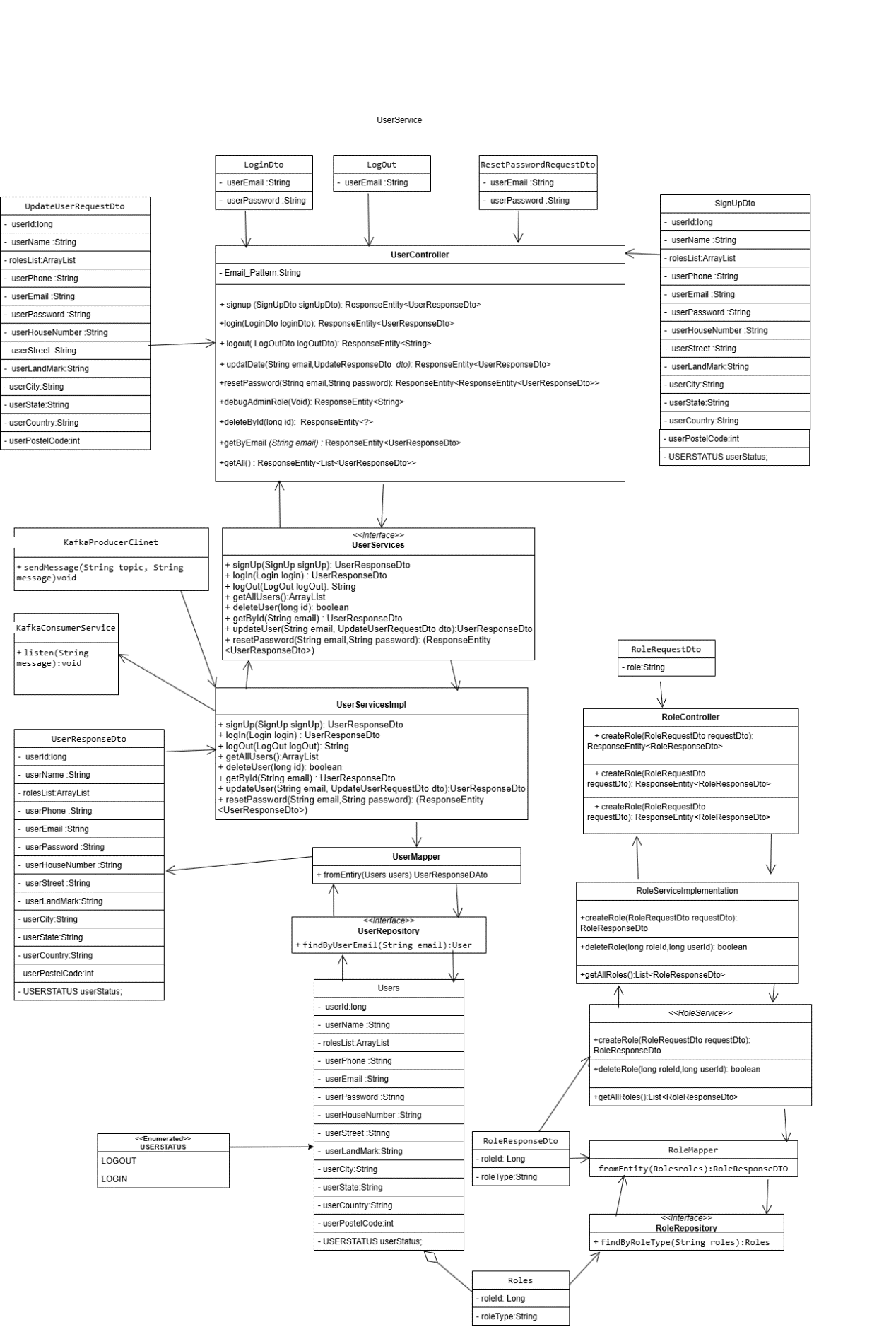
3.1 User Service

Figure : 3:1 User Service Implementation

3.2 User service Authorisation server concent service

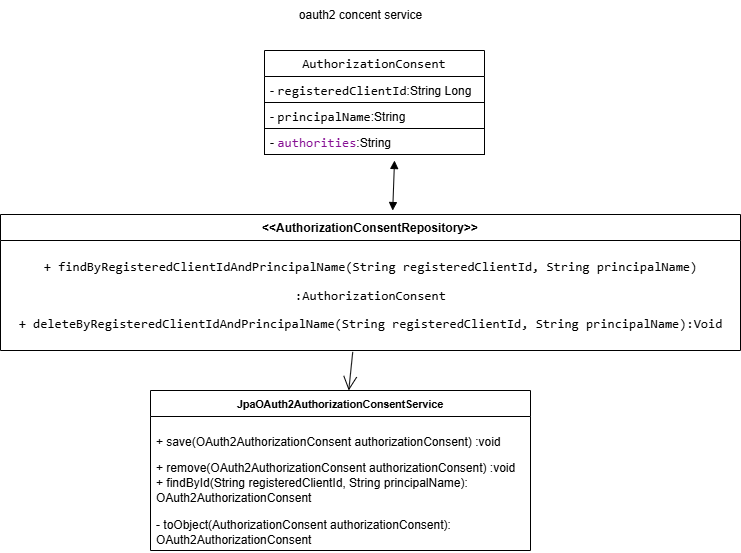


Figure : 3:2 User service Authorization server consent service

3.3 **JpaAuthorisation service**

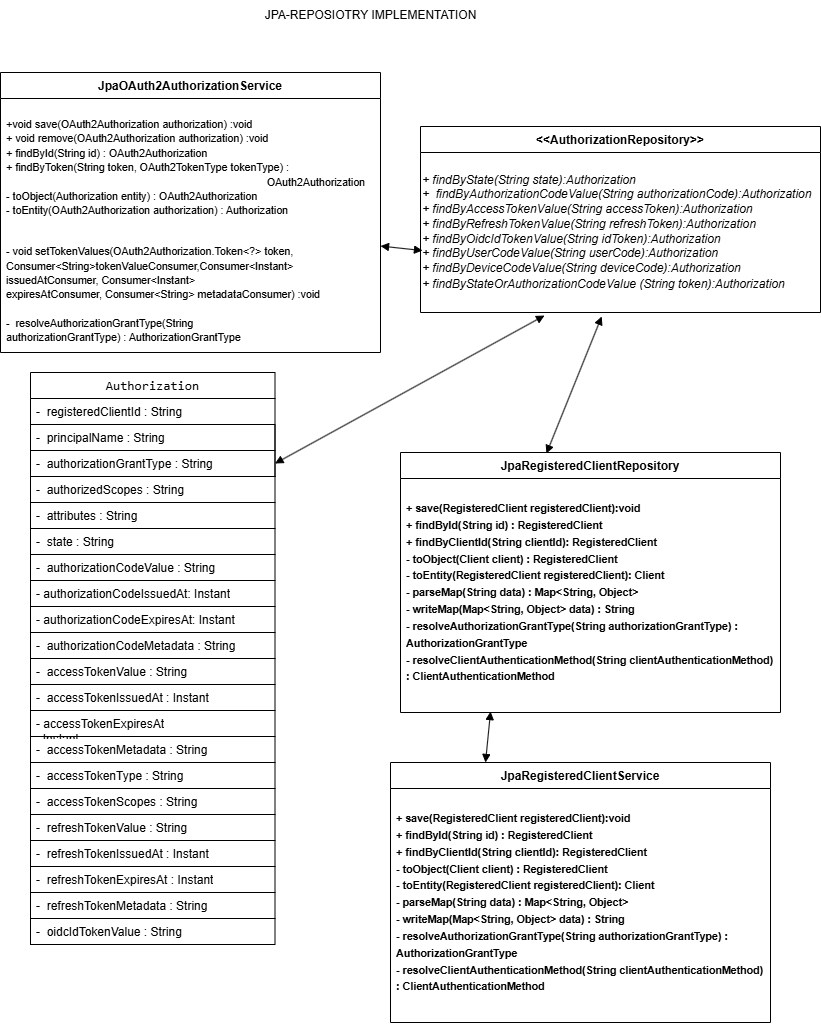


Figure : 3:3 JpaAuthorisation service

## 3.5 Role Authority Service

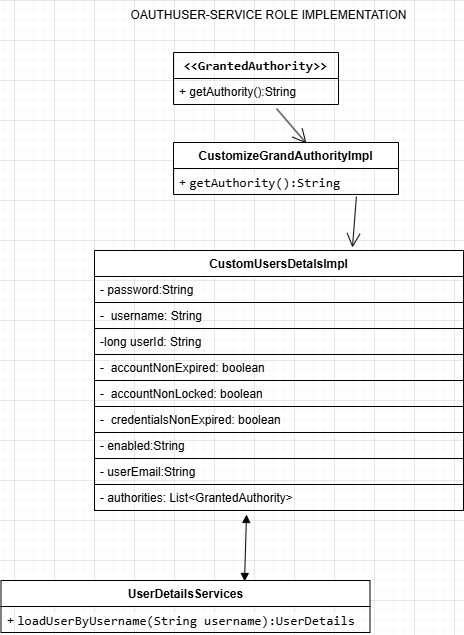


Figure : 3:4 Role Authority Service

## 

## 3.6 Authrisation server Registered clinet or OICD client

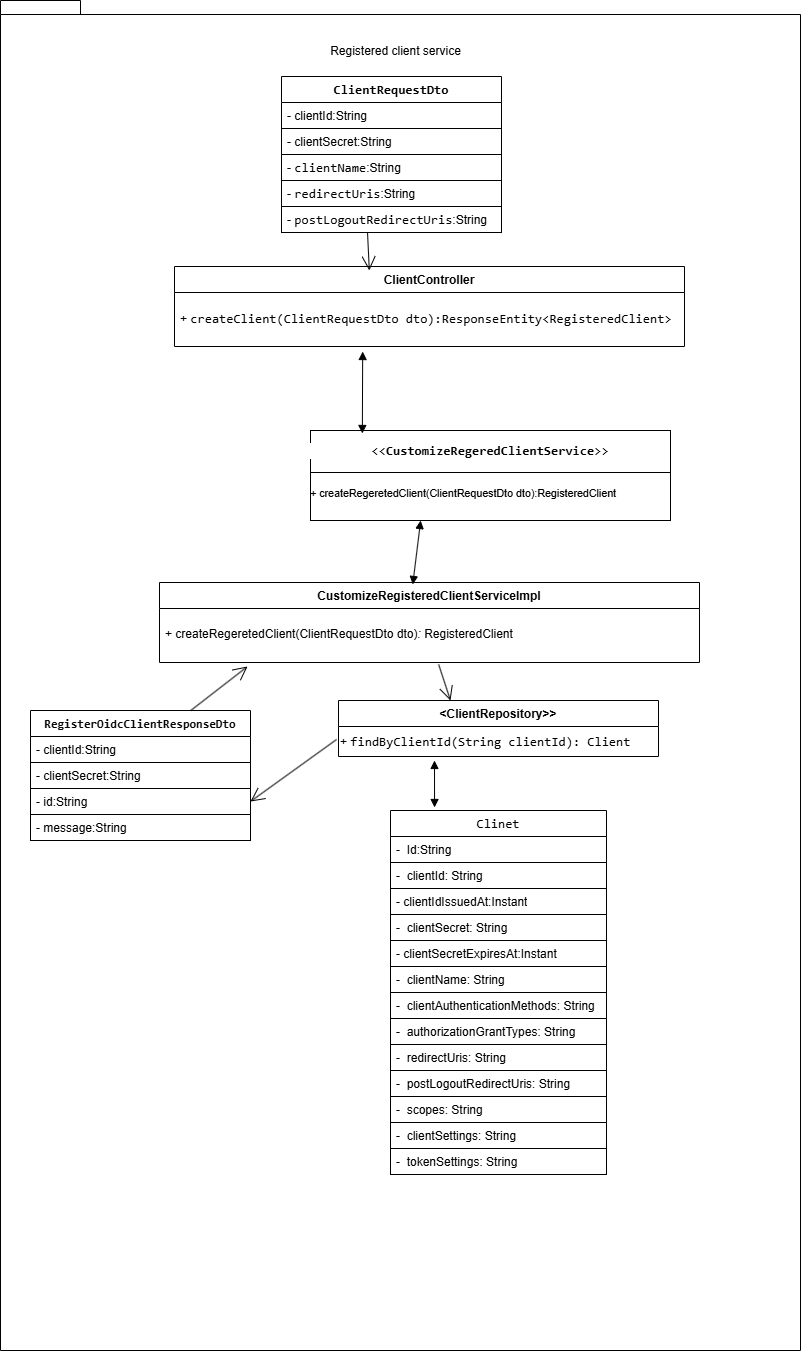


Figure : 3:5 Authrisation server Registered clinet or OICD client

## 3.7 Product-service (Category-service)

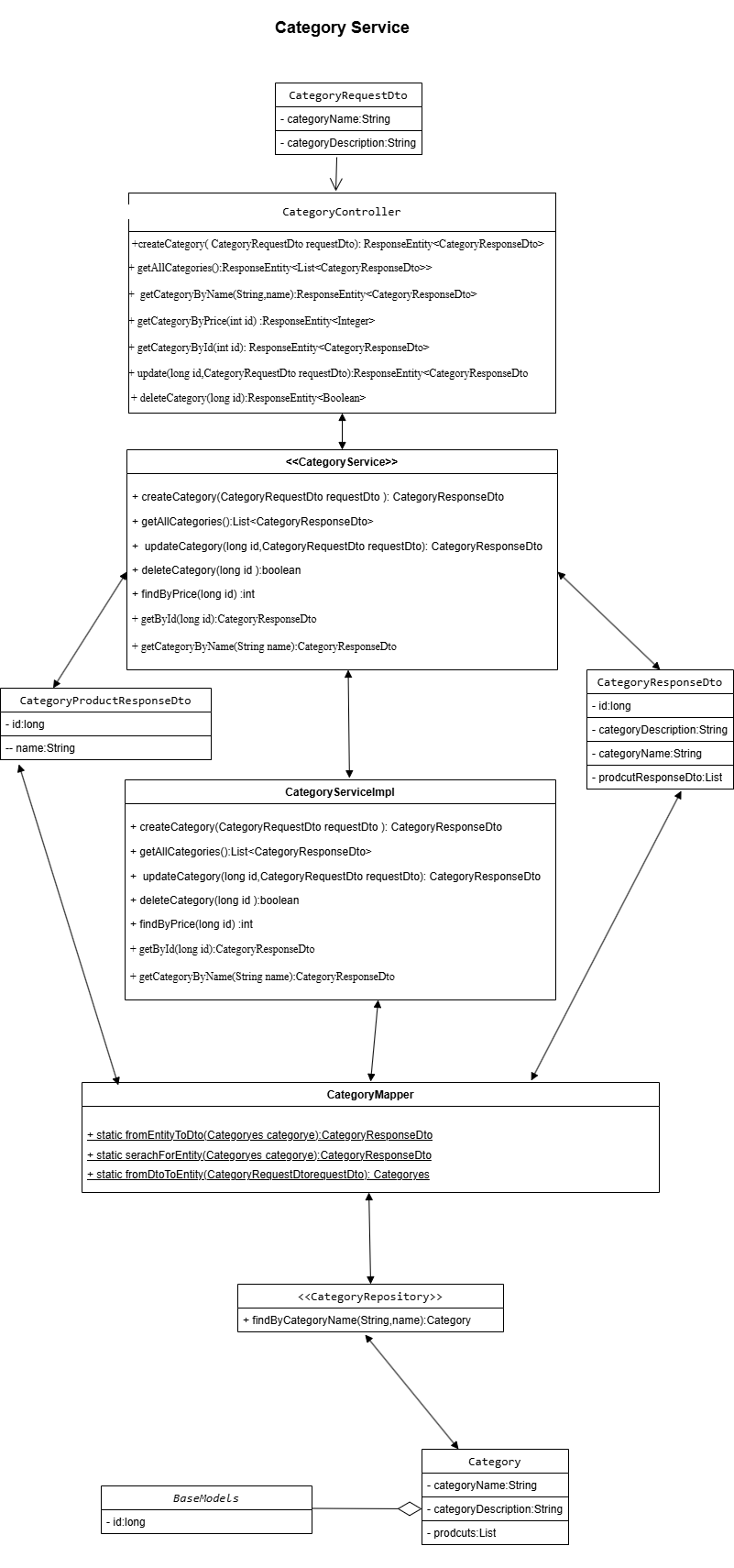


Figure : 3:6 Product-service (Category-service)

## 3.8 Product service

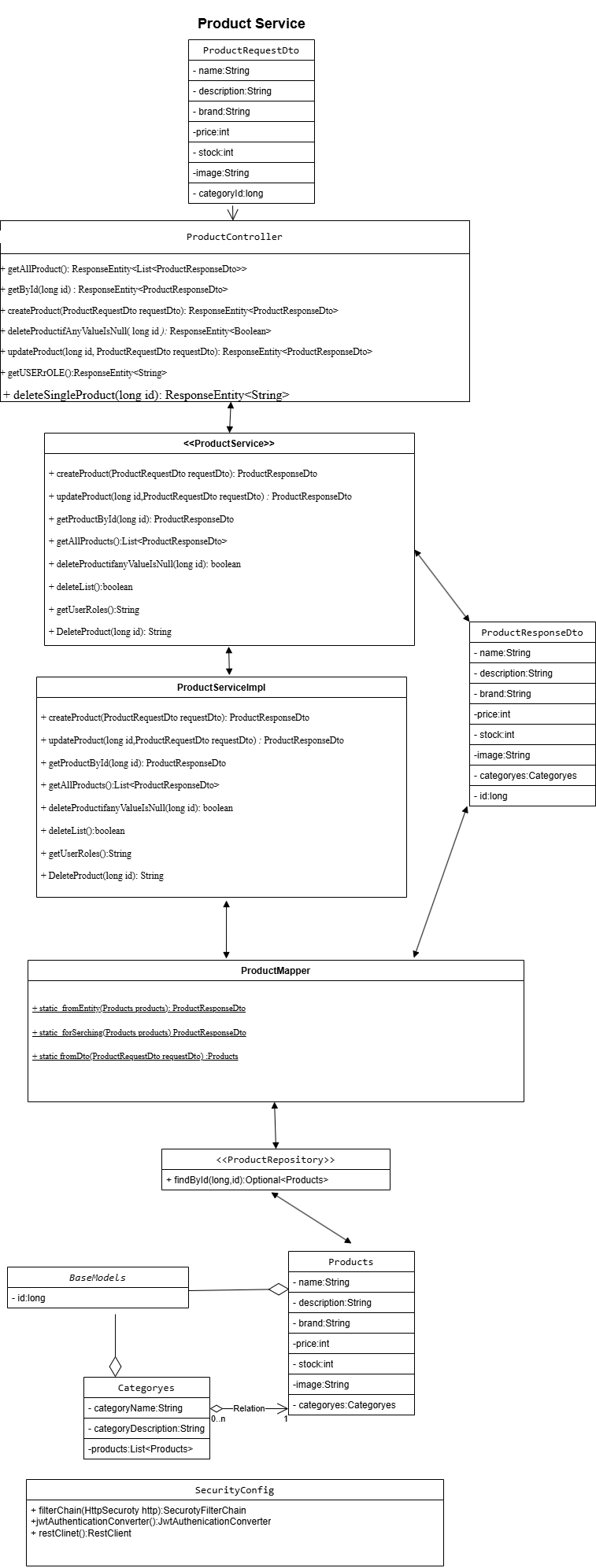


Figure : 3:7 Product service

## 3.9 Cart service

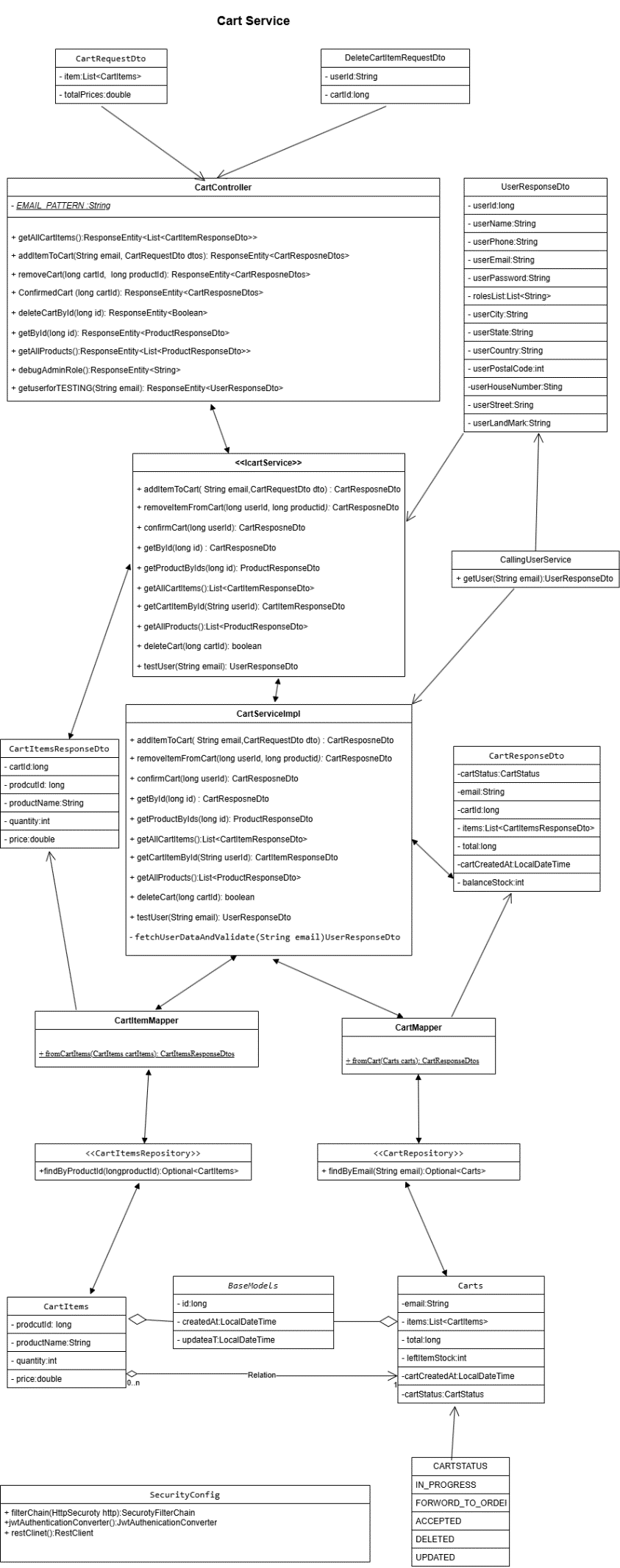


Figure : 3:8 Cart Service

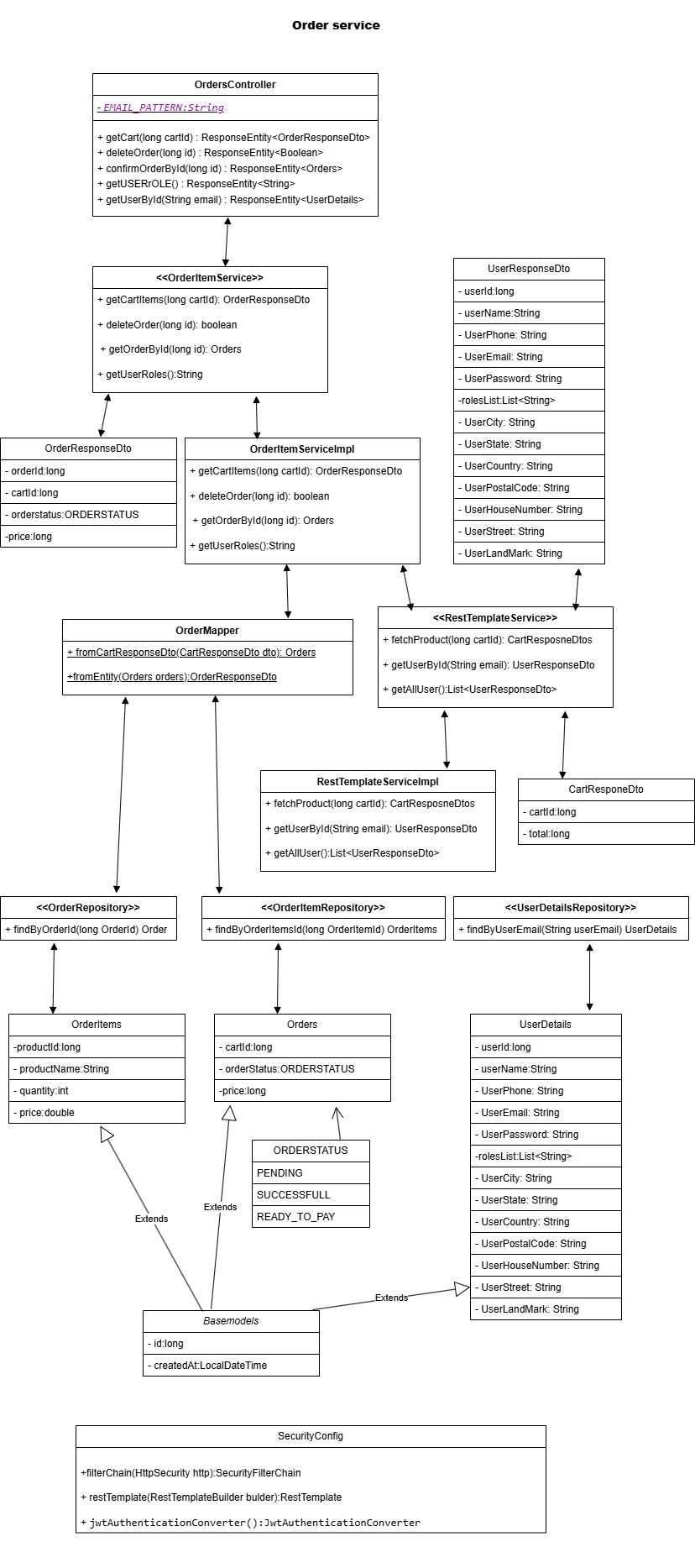
3.10 Order service

Figure : 3:9 Order service

## 3.11 Payment service

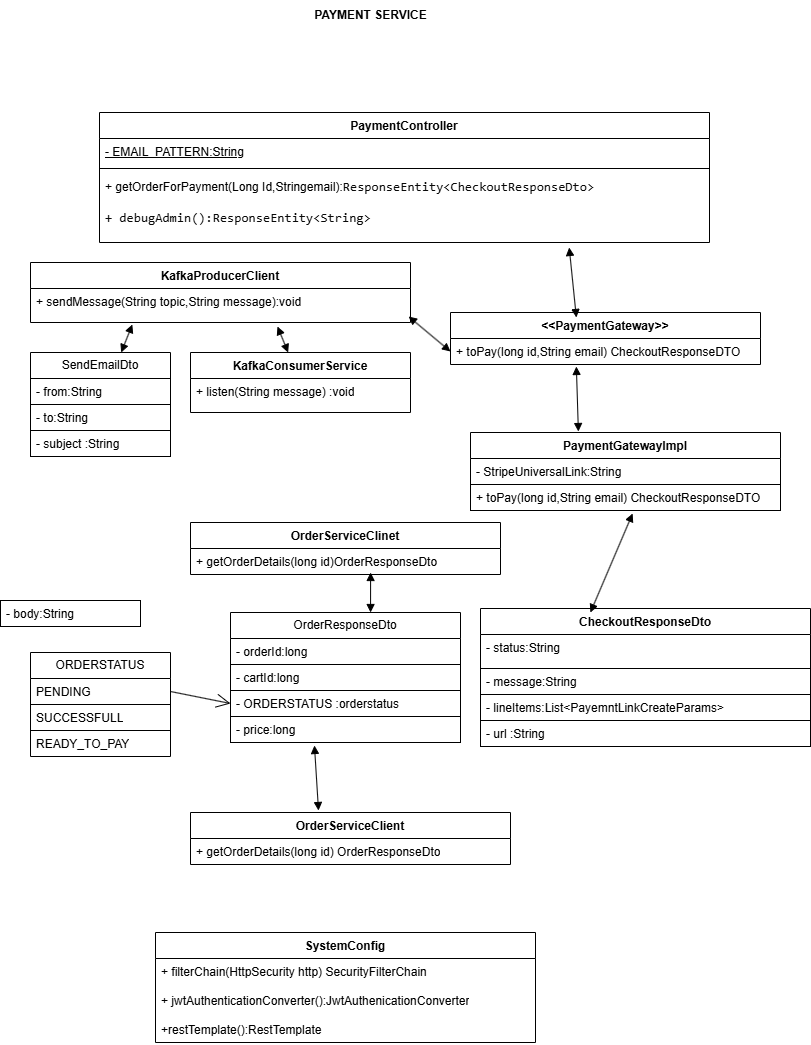


Figure : 3:10 Payment service

## 3.12 Delivery service

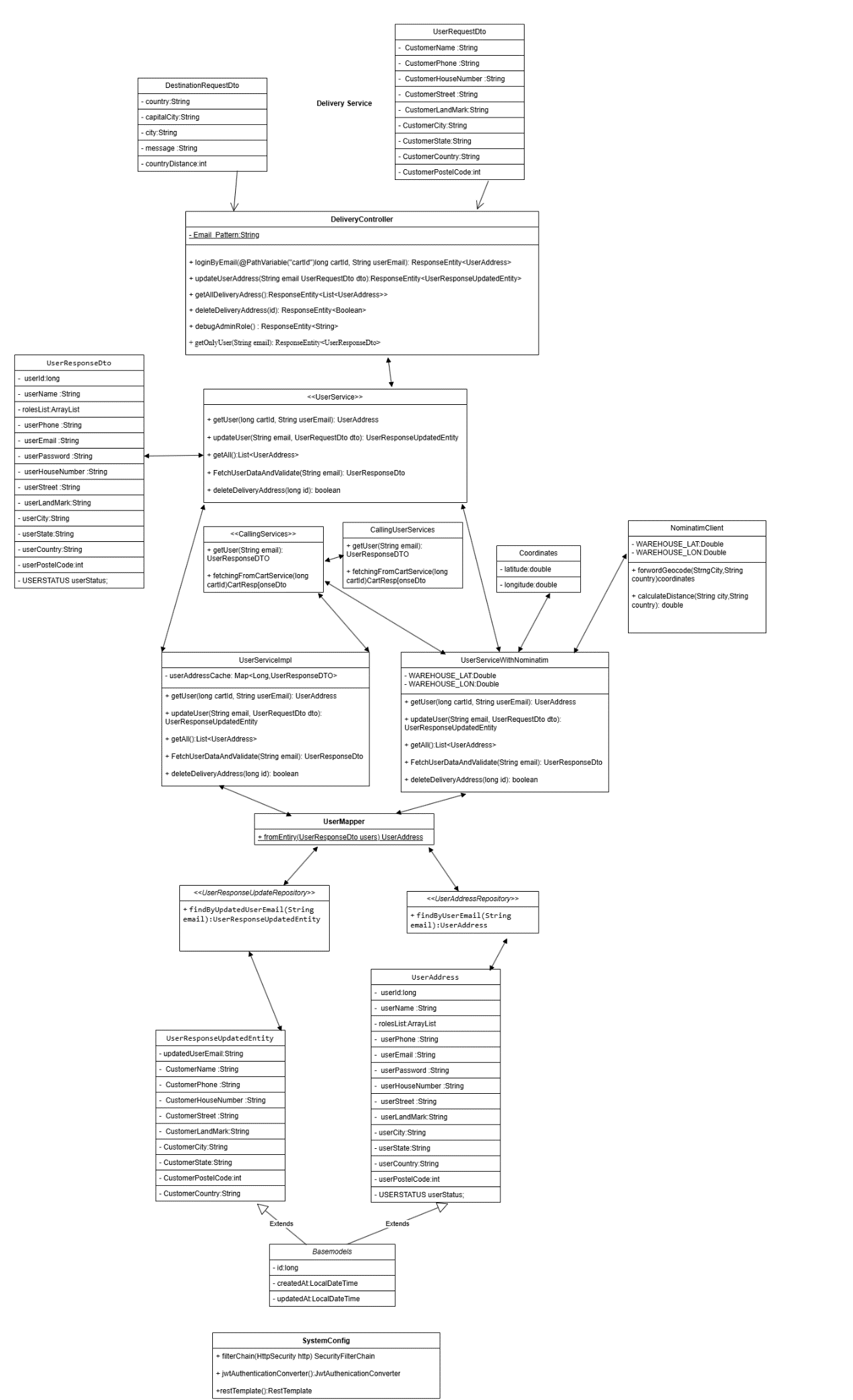


Figure : 3:11 Delivery Service

## 3.13 Delivery service (user implementation)

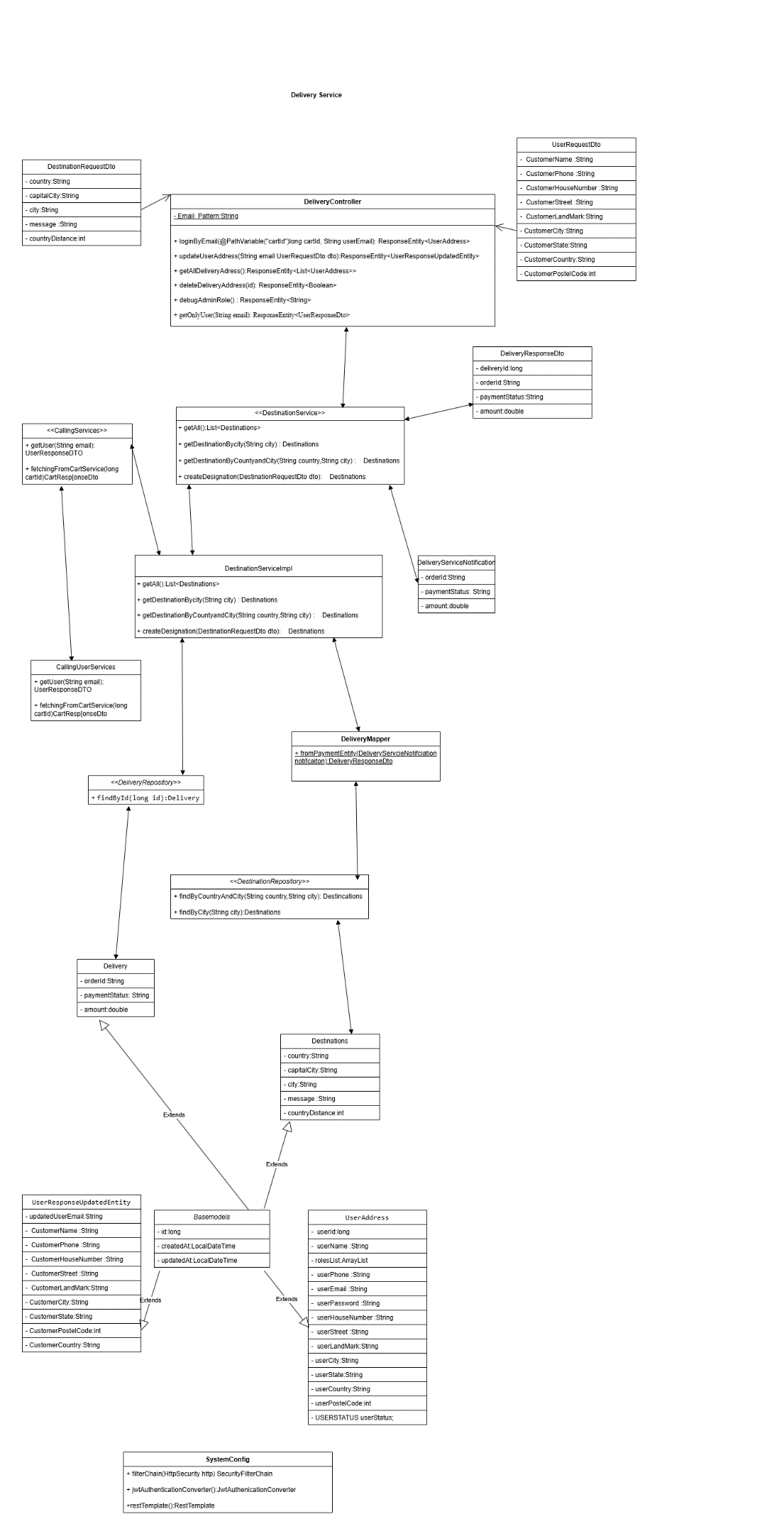


Figure : 3:12 Delivery Service(user implementation)

# **Database Schema Design**

## 4.1 User service schema diagram

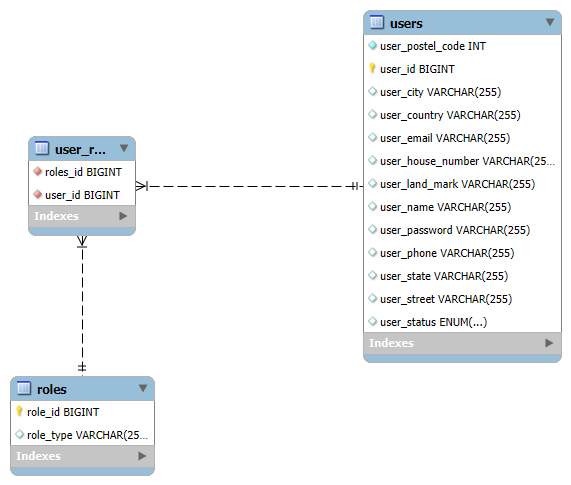


Figure : 4:1 User service schema diagram

## 4.2 Product service

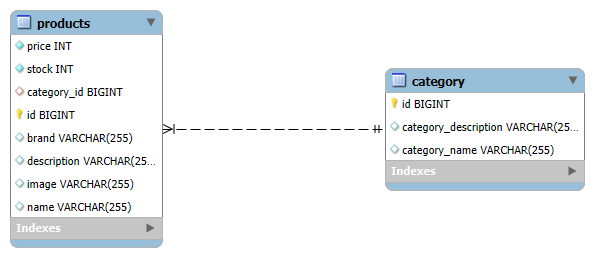


Figure : 4:2 Product service

## 4.3 Cart service

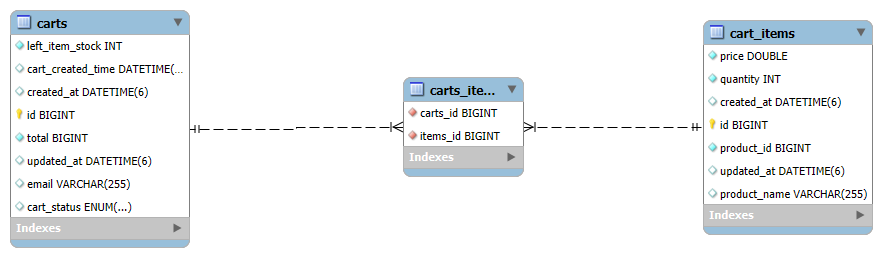


Figure : 4:3 Cart service

## 4.4 Order service

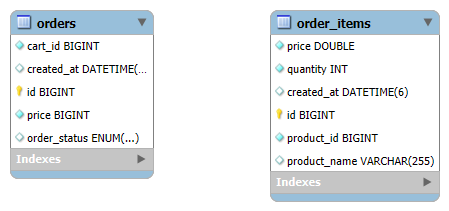


Figure : 4:4 Order service

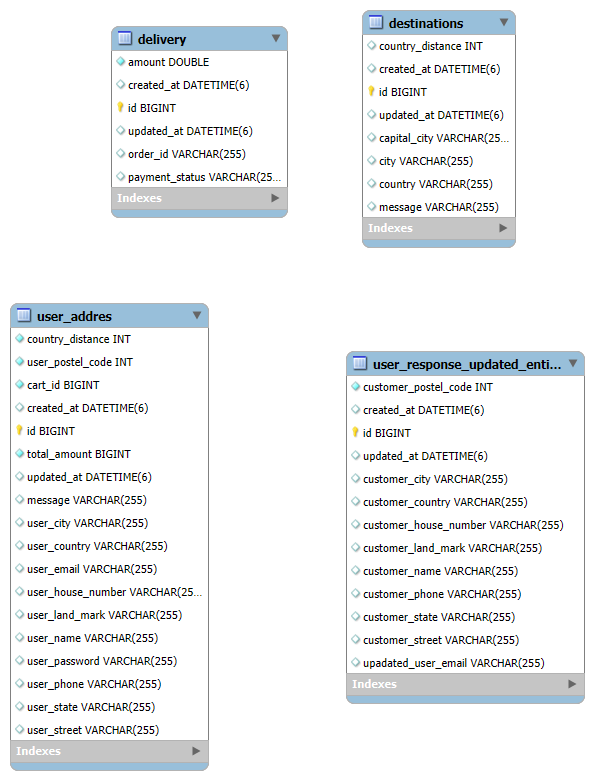
4.5 Delivery service 

Figure : 4:5 Delivery service

# **Textual schema design**

## 5.1 User Service textual schema

**Roles**

* role\_Id (PK, auto\_increment)
* roleType

**Users**

* user\_Id(PK, auto\_increment)
* user\_Name
* user\_Phone
* user\_Email
* user\_Password
* user\_HouseNumber
* user\_Street
* user\_LandMark
* user\_City
* user\_State
* user\_Country
* user\_PostelCode
* user\_Status (Enum:LOGIN,LOGOUT)

**user\_role**

* user\_id(FK -> Users.user\_id)
* roles\_id(FK->Roles.role\_id)
* Composite Primary Key (user\_id,roles\_id)

Between users and role : m:n

## 5.2 Product Service textual schema

**Categoryes**

* Id (PK, inherited from Basemodels)
* Category\_name
* Category\_description

**Products**

* Id (PK,inherited from BaseModels)
* Name
* Description
* Brand
* Price
* Stock
* Image
* Category\_id(FK -> Categoryes.id)

### Cardinalities

* Cardinality to Products (1:m)
* Products to categoryes (m:1)

## 5.3 Cart Service textual schema

**CartItems**

* Id (pk,Inherited from BaseModels)
* Product\_id
* Product\_name
* Quantity
* Price
* Created\_at (Inherited from BaseModels)
* Updated\_at(Inherited from BaseModels)

**Cart**

* Id (PK, Inherited from BaseModels)
* Email
* Total
* Cart\_created\_at
* Left\_item\_stock
* Cart\_status(Enum: IN\_PROGRESS, FORWORD\_TO\_ORDER\_SERVICE, DELETED, ACCEPTED, UPDATED)
* created\_at(Inherited from BaseModels)
* updated\_at(Inherited from BaseModels)

### Cartitems relationship

Cart to cartItem (1:m)

## 5.4 Order Service textual schema

**OrderItems**

* Id (PK, inherited from Basemodel)
* Product\_id
* Product\_name
* Quantity
* Price
* Created\_at

**Orders**

* Id (PK, inherited from Basemodel)
* cart\_id
* order\_status(Enum: PENDING,SUCCESSFUL, READY\_TO\_PAY )
* Price
* Created\_at

**UserDetails**

* id (PK, inherited from BaseModels)
* user\_id
* user\_name
* user\_phone
* user\_email
* user\_password
* roles\_list
* user\_city
* user\_state
* user\_country
* user\_postal\_code
* user\_house\_number
* user\_street
* user\_landmark
* created\_at

## 5.5 Delivery Service textual schema

**Delivery**

* Id(PK, inherited from Basemodel)
* Order\_id
* Payment\_status
* Amount
* Created\_at
* Updated\_at

**Destinations**

* id (PK, inherited from BaseModels)
* country
* capital\_city
* city
* message
* country\_distance
* created\_at
* updated\_at

**UserAddress**

* id (PK, inherited from BaseModels)
* user\_name
* user\_phone
* user\_email
* user\_password
* user\_house\_number
* user\_street
* user\_land\_mark
* user\_city
* user\_state
* user\_country
* user\_postal\_code
* message
* country\_distance
* cart\_id (Reference to the cart – optional FK)

• total\_amount (Amount to be paid)

• created\_at

• updated\_at

**UserReponseUpdatedEntity**

* id (PK, inherited from BaseModels)
* updated\_user\_email
* customer\_name
* customer\_phone
* customer\_house\_number
* customer\_street
* customer\_land\_mark
* customer\_city
* customer\_state
* customer\_country
* customer\_postal\_code
* created\_at
* updated\_at

# **Feature Development Process**

## 6.1 Topic : Track delivery from warehouse to user delivery address

This feature involes real time delivery of product for customer after succesfull purchase, it communicate with other microservices to fetch user and cart data , and calculate distance from warehouse location and estimate time using hardcoded logic

Microservice communication

User service –confim and get customer address details.

Cart service – fetch cart details such as cartid and total amount.

Delivery service – this is combination of both user and cart service and return delivery status.

**6.2 Features**

Delivery service

Main feature in delivery service is once product is purchased custumer will get choice to use delivery address, and after selecter selection of delivery address , our application will calculate total distance between warehouse to destination point , and it will response according to distance in kilometers and convey message in how many days product will reach destination point.

Api:- GET http://localhost:8091/deliveryUser/\*\*userEmail\*\*

Repones:

{

    "id": 9,

    "createdAt": "2025-07-02T14:46:00",

    "updatedAt": "2025-07-02T14:46:00",

    "userName": "\*\*\*\*\*\*",

    "userPhone": "\*\*\*\*\*\*\*\*",

    "userEmail": "\*\*\*\*\*\*\*\*\*",

    "userPassword": "NOT VISIBLE BECAUSE OF PRIVCY RASONS",

    "userHouseNumber": "1421",

    "userSteet": "\*\*\*\*\*\*\*\*",

    "userLandMark": "CMC HOSPITAL",

    "userCity": "UZHHOROD",

    "userState": "PUNJAB",

    "userCountry": "UKRAINE",

    "userPostelCode": 1442690,

    "message": "PARCEL WILL DELIVER IN MAXIMUM 2 days ",

    "countryDistance": 260,

    "cartId": 1,

    "totalAmount": 2600

}

**Payload explanation**

* Id: unique id for delivery
* Created\_At used for timestamp when it was generated
* username: name of customer
* userPhone: phone of customer
* userEmail: email address of customer
* userPassword: "NOT VISIBLE BECAUSE OF PRIVCY RASONS",
* userHouseNumber": house address
* userSteet: street
* userLandMark: any landmark
* userCity: city of customer
* userState: state of customer
* userCountry: country of customer
* userPostelCode: postal code of customer
* message: estimate delivery time if customer distance from warehouse is less then 60 kilometres then parcel will reach in 24 hours
* If distance is more then 60 kilometres then parcel will take 2 days or more
* countryDistance: distance of destination from warehouse
* cartId: cart id
* totalAmount: total amount paid by customer

## 6.3 Service handling request

Controller - DeliveryController

Service – UserServices(interface) to UserServicesImpl

Calling other services – CallingServices(interface) to CallingServicesImpl

CallingServicesImpl calls -: (user service by UserResponseDto getUser(String emailId)

And

Calling Cart service by CartResposneDtos fetchingFromCartServcie(long cartId)

Repository layer - Useraddress repository

## 6.4 MVC Architecture

Customer ->GET/deliveryUser/ {email)-> controller->deliveryController-> service->

UserServiceImpl ->calling layer -> userservice(get user by email)

UserServiceImpl ->calling layer -> cartservice(get cart by by)

business Logic-> Calculate distance using hardcoded mappings

business Logic-> Estimate delivery days based on distance (<60km → 1 day, >60km → 2+ days)

repository -> save response to db

return userAddress to client

**6.5 Performance Improvement**

**before optimization**

Api query performance time 3.2 s

after optimization

Api query performance time 48 ms by using Redis template

performance percentage = ((3200-48)/3200) \*100= 98.5%

# **Deployment Flow**

Deployment flow is journey of e-commerce project which involves end to end processing which start from development environment to production environment. This includes building and testing and deploying and monitoring,

## 7.1 GIT commit and version control

* Source control – after finishing and testing we commit code to version control

## 7.2 CI (Continues integration)

the Continues integration pipeline is only triggered when code is pushed to version control system. This ensures that the code is compiled, tested and packaged.

**Build :**

in this code is checked and build by maven.

After successfully compile the code, resolves dependencies and create package

As example this is product service jar file for app

{ productservice-0.0.1-SNAPSHOT.jar}

**Unit test and integration test**

Unit and integration tests are done to ensure that individual components and their interactions are working as expected. And Junit tool is used to run tests.

## 7.3 CD (Continues Deployment)

Continues deployment automate the deployment process to different environments from staging to production.

**Continues deployment steps –**

* staging deployment -: application deployed to stating enjoinment that mirrors production
* deployment automation -: tools like Kubernetes automate the deployment process
* smoke testing-: basic checks are to be done to check if app works properly
* performance testing -: took like JMeter simulate traffic to its performance under load
* approval gate-: a manual review is done by developer
* production deployment-: approved changes are deployed to production

## ****7.4 AWS -Production deployment architect****

**Amazon web services is the best in market most of big tech companies are using AWS for deployment, to insure scalability ,high availability, and ease of maintenance the is deployed on AWS Cloud using below services**

* **EC2(elastic compute) instance**

**EC2 instances host the applications JAR files.**

* **VPC (Virtual private network)**

**This is secure and isolated network containing subsets for public facing services and private services.**

* **Security groups**

**Its virtual securities that control traffic like EC2 and RDS allowing ports like 8080 for application and 3306 from MYSQL and 6379 for Redis**

* **RDS(Relational database system)**

**Manage database instances where application data stored .**

* **ElasticCache(Redis)**

**Used for caching for frequent access of data, and improving performance and reduce database loads.**

* **Elastic beanstalk**

**It provides managed infrastructure for deploying applications without handling EC2 directly , you can upload JAR and EBS manage deployment , scaling up and down, and monitoring and health checks.**

## ****7.5 Post-Deployment****

**Once app is deployed to production the system needs to be checked and maintained.**

**This contains -**

* **Monitoring and logging**
* **Error tracking**
* **Auto-scaling**
* **Continues monitoring**

## ****7.6 Rollback strategy****

**If application having any issues then rollback strategy is must have choice**

## ****7.7. Deployment Diagram****

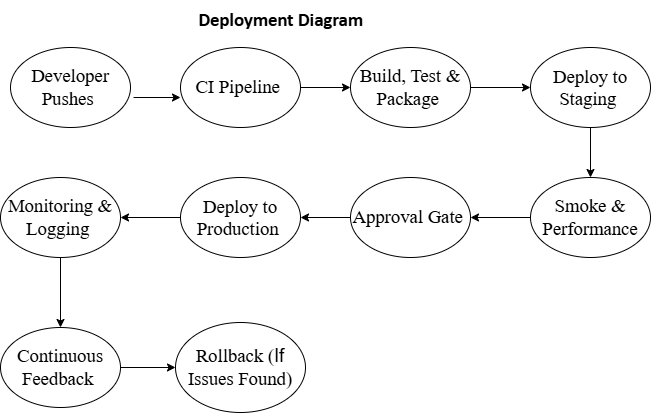


Figure : 7:1 Deployment flow Diagram

# **8.Technologies Used**

In this e-commerce project I used several technologies which used to enhance performance, scalability and smooth functionality across microservices, databases and other services, below I provide details descriptions of each technology used and explain their real life use cases with examples of real world application.

### ****8.1 Kafka****

**Overview**

**Apache Kafka is an open source distributed even streaming platform used to build real time data pipelines and streaming applications.**

**User case in project**

**In my e-commerce project kafka is used for notification to send email which used in user service order service and delivery service, so user will be informed and to confirm email address.**

**Real life use case**

**Kafka is very famous and widely used real time data processing open source application below are some real life applications which use kafka.**

**Real time analytics : companies like Netflix, uber and LinkedIn user kafak for real time user analytics and recommendation engines;**

**Log aggregation : Kafka is used for collection logs from various services and making it real time processing and monitoring.**

**Examples :**

**Netflix – Netflix uses kafka to stream million of events per second such as user action actionsl like play the video,pause skip and service metrics ,to monitor service health and personalize recommendations in real time.**

## ****8.2 Mysql (relational database management system(RDBMS)****

**Overview**

MySQL is one of the most popular open source database management system, it stores data in tables and support SQL queries for data retravel, CRUD (Create, Read, Update and Delete) operations.

**Use case in project**

In my project MySQL community version is used which free to store structured data such and user profile, order, product, categories and delivery. MySQL ensures consistency and enables complex queries that aggregate data from different tables.

**Real life use case**

MySQL is mostly used where structured relational data storage required.

* **e-commerce application :** Amazon , eBay are biggest examples which are using MySQL to store data and user profiles.
* **Content management systems (CMS):** Companies like WordPress and Joomla use MySQL to store blog posts, user information and metadata.

**Examples**

Airbnb uses MySQL as primary database for storing listings, bookings, user data and payments,

## 8.3 Spring boot

**Overview**

Spring boot is popular java framework used to create stand alone production grade spring based applications, it simplifies the development of microservices by providing build in configurations and other utilizes

**Use case in project**

My whole project fully based on spring boot to create microservice architect like user service, product service, cart service, order service, payment service and delivery service because of it auto configurations and it provides rest Apis manage scanning dependencies and making it easier to deploy and scale each service independently.

**Real life use case**

After spring applications spring boot top one of the real word applications used to quickly build scalable and efficient web application and microservices. Below are real life use cases

* **E-commerce application:** Alibaba user spring boot to develop and scale their microservices.
* **Banking applications:** JPMorgan Chase bank uses spring boot for developing banking applications with highly available Apis.

**Examples**

Spotify – Spotify uses spring boot to build its backend microservices that handle music streaming, user data, playlists etc.

## 8.4 Spring security with Oauth2

**Overview**

Spring security with oauth2 is used to secure apis using industry standerd Oauth2 authentication and authorization mechanisms. It enables single sign on , token based authentication and access control in spring boot application.

**Use case in project**

In my project spring Oauth2 authentication and authorization used to secure all microservices.

**Real life use case**

Spring security used in applications that requires user identity verification as examples

* Google, LinkedIn, Facebook use it for secure logins.
* Enterprise application – spring security requiring role based access.

**Examples**

Google it best example of spring oauth2 used for authentication and authorization of user.

## 8.5 Redis (in memory data storage)

**overview**

Redis is an open source in memory data store often used for caching and real time data storage, it supports various data structures like strings, lists, sets, hashes and providing high speed read and write operations.

**Uses case in project**

Because Redis is used for caching frequently accessed data such and product details and user sessions I used Redis to cache that and store In memory which helps improving Api performance, and reducing load on database.

**Real life use case**

Redis is used where high speed outs required for data retrieval

* Social media – twitter and Instagram use Redis to cache feeds and user data for faster load times.
* e-commerce- eBay and amazon user Redis for caching product information

**Examples**

* GitHub – GitHub use Redis to cache data.

## 8.6 (AWS) Amazon Web Services

**Overview**

AWS (amazon web services) is on top in market because of it cloud computing platforms in real world. It provides services like EC2, RDS, S3(cloud storage) and many more, AWS helps developers and companies to build scalable , secure, and cost effective applications without managing hardware of machine. Its famous because of its supports from startups to enterprise level applications .

**User case in project**

In this project I used AWS to deploy and manage all my microservices, each service like user service, product service, cart service etc. was deployed using EC2 instances. I used RDS (relation database system) for managing MySQL database, which took care of backups and maintenance. For quick and easy deployment I used ELASTIC BEANSTACK (EBS) which automatically handled load balancing , scaling and monitoring and security groups are used so that only authorize users and services can access each component.

**Real life use case**

AWS is used by top most companies for hosting and managing their applications.

* Netflix -Netflix use AWS to stream content to millions of users globally by dynamically scaling their infrastructure.
* Airbnb – Airbnb runs its bookings and search platform entirely of AWS.
* NASA- NASA user AWS to store and analyze massive amounts of space mission data.

**Examples**

* Netflix – it stores AWS EC2 and S3 to service and store video files, scaling during peak demand.
* Airbnb – this application relies of AWS service like RDS and EC2 to host millions of listing and handle user traffic.

# **Conclusion**

**Building this project offered a deep dive into backend development using microservice architect, communication between services , secure Apis , real time notifications and cloud technology, it wasn’t just an application it was for me understanding how in real world spring boot applications works and systems are structured. Here are some important takeaways , and few challenges I faced along the way**

# **Key takeaways**

## ****9.1 Spring boot and java 17****

**Spring Boot with java 17 provided a clear and efficient way to build restful Apis, its simplified configuration, development especially features like auto configuration, embedded servers, starter dependencies, which made the code base more readable and efficient.**

## ****9.2 Microservices architectures****

**Working on microservices helped me understand how large application can be broken down into smaller, independent services, and how its beneficial in production as each service had clear responsibility, which make development and debugging more manageable.**

## ****9.3 Authentication And Security With Oauth2 and JWT****

**Implementing Oauth2 resouse server with JWT tokens gave the application reobust security. Each request to protect endpoints was verified using signed token, ensuring that onlu authenticated users could access senstice operations.**

## ****9.4 Mysql database****

**Mysql was as good choice for managing and storing structured data such as user data, product and category and other services data.**

## ****9.5 AWS cloud****

**Deploying services on AWS introduced me to cloud development . services like EC2,RDS, EBS helped me more beyond local deployment and understand how real production systems are hosted and maintained at scale.**

### ****Practical applications****

### ****Web applications****

**Using Spring boot in microservices architectures showed how real world platforms like amazon or flipchart can scale individual services independently.**

## ****Data caching****

**Redis proved to very useful In speeding up data retrieval , I learned how companies like Instagram , twitter (X) improve their performance for millions of users.**

## ****e-commerce flows****

**with implementing Oauth2 and JWT reflects real practices used by banking , e-commerce, where secure, token based access is critical.**

## ****Eureka server (service discovery)****

**Eureka servers abilities to dynamically locate services without manual configuration is crucial in real time systems where services might scale up or down frequently.**

## ****Limitations****

## ****Manual deployment overhead****

**Deploying services manually took more time and was more complex for me. This could become a challenged in large teams or environments .**

## ****Complexity of service coordination****

**Managing multiple services even with eureka, added a layer of complexity . keeping track of which service handles what and ensuring communication happens correctly required careful planning and clear Api contracts.**

## ****Caching consistency****

**While Redis boosted performance, managing cache consistency introduced a few challenges that needed attention to avoid serving outdated data.**

# **Final thoughts**

**Overall this project gave me solid experience in building a real world backend system using technologies that are widely used in production environments. The combination of spring boot, Oauth2 Redis, eureka, Api gateway, rest template , MySQL, RestTemplate allowed me to design a secure efficient backend structure that can scaled or extended easily in the future**

**By focusing on secure communication, performance optimization, and modular design , I learned how enterprise level systems manage complexity while ensuring reliability. At the same time , I also saw first hand how architecture choices like using service discovery and caching can significantly impact the responsiveness and maintainability of the project.**

**This project not only enhanced my technical knowledge but also deepened my understanding of how to approach backend syst design from both a developer’s and users perspective. I believe now I can build more application and will deploy on cloud and enhance my experience as a java backend developer.**

# 

# **References**

## For websites and articles

1. Spring.io, Referred on 10 January 2025, Spring Team, *"Spring Initializr – Start Your Spring Boot Project"*, [<https://start.spring.io/>].  
   2. Spring.io, Referred on 12 March 2025, Spring Team, *"Spring Authorization Server – Secure OAuth2 Implementation"*, [<https://spring.io/projects/spring-authorization-server>].  
   3. Apache Kafka, Referred on 12 March 2025, Apache Software Foundation, *"Apache Kafka Documentation and Downloads"*, [<https://kafka.apache.org/>].  
   4. Baeldung.com, Referred on 10 April 2025, Eugen Paraschiv, *"Guide to RestTemplate in Spring"*, [<https://www.baeldung.com/rest-template>].  
   5. Redis.io, Referred on 4 April 2025, Redis Labs, *"What is Redis?"*, [<https://redis.io/>].

**6.** AWS Documentation, Referred on 3 July 2025, Amazon Web Services, *"Amazon EC2 Overview"*, [<https://docs.aws.amazon.com/ec2/>].  
**7.** JWT.io, Referred on 15 March 2025, Auth0, *"Introduction to JSON Web Tokens"*, [https://jwt.io/introduction].