Software Requirement Specification (SRS)

Online Shopping System

The Perfect Pairs

Team Members:
Heer Dhandhukia - E015
Aryan Dalvi - E012
Kashmaya Khandelwal - E034

Mukesh Patel School of Technology, Management and Engineering

1. INTRODUCTION

1.1 Purpose

A Software Requirements Specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. It fully describes what the software will do and how it will be expected to perform. It is usually signed off at the end of requirements engineering phase. The development teams build the correct product with precise specifications and a Software Requirements Specifications enables you to lay the foundation for product development. A good SRS defines how an application will interact with system hardware, other programs and human users in a wide variety of real-world situations. In general, SRS documents contain three kinds of program requirements:

- ✓ Functional specifications that include measures to be performed by the system.
- ✓ Non-functional requirements determining the software system's performance attributes.
- ✓ Domain requirements that are device limits on the service domain.

If the SRS is written well, it will solve the following purposes.

- ✓ **Feedback to the Customer**

  The software requirement specification assures the project management stakeholders and client that the development team has really understood the business requirements documentation properly. This also provides confidence that the team will develop the functionality which has been detailed.

- ✓ **Breaking of the Requirements**

  The Software Requirement Specification is documented in such a way that it breaks the deliverables into smaller components. The information is organized in such a way that the developers will not only understand the boundaries within which they need to work, but also what functionality needs to be developed and in what order.

- ✓ **Facilitation of other Documents**

  The SRS forms the basis for a load of other important documents such as the Software Design Specification.

- ✓ **Product Validation**

  It basically helps in validating with the client that the product which is being delivered, meets the requirements what they have asked for.

1.2 Literature Survey

E-commerce is the most trending and the fastest growing platform. It is the process whereby consumers directly buy goods or services from a seller in real-time, over the internet. “The Perfect Pairs” is an online footwear website which offers a user-friendly interface where-in the customers will be able to shop in ease, fast and in an efficient manner. “The Perfect Pairs” makes sure that you get your favourite pair of shoes at your doorstep and that you get the feeling of offline shopping while you sit on your couch and shop your perfect pairs.

1.3 Characteristics of Software Requirements Specification

1.3.1 Accuracy

This is the first and foremost requirement. The development team will get nowhere if the SRS which will be the basis of the process of software development, is not accurate.

1.3.2 Clarity

SRS should be clearly stating what the user wants in the software.

1.3.3 Completeness

The software requirement specification should not be missing any of the requirements stated in the business requirements documentation that the user specified.

1.3.4 Consistency

The document should be consistent from beginning till the end. It helps the readers understand the requirements well.

1.3.5 Unambiguousness

The document should be free from any sort of ambiguities.

1.3.6 Verifiability

At the end of the project, the user should be able to verify that all that all the agreed deliverables have in fact been produced and meet the project management requirements specified.

1.3.7 Modifiability

The SRS should be written in such a way that it can be modified when the development team and user feel the need.

1.3.8 Traceability

Each requirement stated in the SRS should be uniquely associated to a source such as a use case or interaction document etc.

1.4 Product Scope

With advancing technologies and increasing competition, the e-commerce industry is constantly evolving. No matter what these changes are, the future of this robust and flourishing sector is bright and promising.

The e-commerce sector has undergone unprecedented growth in the recent few years. The worldwide epidemic fuelled this and made e-commerce an indispensable part of the global retail framework. Consumers from all countries of the world now benefit from the advantages of online shopping.

The number of online shoppers became 900 million more in 2021 than in 2020. Currently, there are 2.14 billion global online buyers. That is, 27.6 percent of the world’s population is shopping online. This number is expected to only rise.

By 2040, in about some 18 years from now, almost 95% of all purchases will be through e-commerce. The foremost reason why people purchase from e-commerce outlets is free delivery. Discounts and offers, customer

reviews, easy returns policy, and easy checkout process are other vital reasons.

The problems faced by the customers while they shop products online are for example, people often tend to check for details and the actual pictures of the product before buying it and also check if the product is verified and secured.

Considering the comfort of our customers our brand will provide an Omnichannel presence and support. This will include Video chat, Screen sharing, Co-browsing and Document Interaction with the customer.

We assure that the customers have a user-friendly platform. The users will get variety of shoes according to their favourite brand and style. By analysing the feedback that we received from people about through surveys, “The Perfect Pairs” launches its new variate where the customers will get the opportunity to customize their own pair of shoes according to their favourite colour, style, designs etc.

Customer experience, both in-store and online, matters a lot for the success of the e-commerce business. Hence, we ensure that our site loads fast and that there are no unnecessary interrupts. We would take care that the user’s site navigation is simple and easy. As people seek high resolution images whether or not to make a purchase, we will post only high-quality product images.

We will provide apt and honest product descriptions. This will help build our customers’ trust. According to the recent trends, new products will be updated on a regular basis.

The biggest trend in e-commerce now is personalization. People expect a shopping experience that is highly relevant to them based on their personal preferences. Studies say that over 78% of customers ignore impersonalized offers. Keeping this in mind, we would personalize our **customer’s interactions** based on their browsing behaviour, past purchases, and preferences specified by highly qualified AI and ML employees.

1.5 Final Problem Statement

Our goal is to match with the expectations of our customers by providing them our best products. We are looking forward to make this brand, “The Perfect Pairs” known to most people so that we get the opportunity to deliver to more and more happy faces every day.

1.6 Document Conventions

The SRS document is prepared using Microsoft Word 2021 and has used the font type 'Times New Roman'. The fixed font size that has been used to type this document is 14pt with 1.15 line spacing. It has used font size of 18pt and 20pt to set the headings of the document. Bold property is also used in some headings of the document. All pages are numbered chronologically, the numbers appear on the lower right- hand corner of the page. Every image and data table are numbered and referred to the data in the main text. The Standard IEEE template is the template used to organize the appearance of the document and its flow.

1.7 Intended Audience and Reading Suggestions

The intended audience of the document would be the client, specific workers, developers, employees etc to refer all the functional and non-functional requirements for developing a system that meets the standards and guidelines, the project managers and marketing staff who can review project’s capabilities and easily understand where their efforts should be targeted to improve or add more features to it.

The project testers, using this document as a base for their testing strategy can identify some bugs easily, thus making the testing more organised methodically.

The documentation writers can also view this document with the objective to refer and analyse the information. The SRS document can be used in any case regarding the requirements of the project and the solutions that have been taken.

The document would finally provide a clear idea about the system that is building.

1.8 Development Plan

Communication - The Website Order report system shall send an e-mail confirmation to the customer that the items they ordered will be delivered to the shipping address along with user identification and tracking details. The customers will also be able to contact us on our email and call us on the number mentioned on our website.

Planning - Calculation and regular updates of products. Estimation of the costs. Proper management of data of the customers.

Modelling - Design and analysis of software according to the requirements and later updating according to the customer feedback.

Construction - Each part of the user interface intends to be as user friendly as possible. The fonts and buttons used will be intended to be very fast and easy to load on web pages. The pages and the images of the products will be kept light in space so that it won't take a long time for the page to load. All the products will be bifurcated in different categories so that our customers can search their choice of product easily without any trouble.

Deployment - Our aim is to grow our business every day by introducing new updates and new products for our customers. We will provide good discounts and deals to our customers and also give complementary gifts on a specific amount of purchase.

1.9 Feasibility Study

- ✓ **Technical feasibility** - Nominal investments of hardware and software as per requirements.
- ✓ **Operational feasibility** - Professional and highly qualified artists to customize the footwears. Experienced software engineers and designers. Friendly and trained staff.
- ✓ **Economic feasibility** - Able to achieve all the economic funds for the project.

2. PRODUCT DESCRIPTION

2.1 Product Perspective

This website works online. It has a friendly user interface. The database is simple to understand so that customers can shop with ease and an efficient manner. A product database is maintained which handles all the interactions happening through the cart. A payment database is maintained to handle the online payment transactions.

We have collaborated with all high good quality brands so that our customers can shop variety of shoes of every brand and also get the opportunity to customize them as per their choices. An online seller database is maintained for all the products.

Once we have the products, as per the orders received, they are proceeded further for the customisation process which is done by the team of artists and designers.

After the product is ready, it then proceeds for the shipping/delivery process. A team of delivery staff is assigned to fulfil this process. A separate database is maintained for the same. We assure that the product is delivered in a perfect condition within a specific period of time according to the locations respectively.

2.2 Functional Requirements

- ✓ Web developer
- ✓ Artist, Designer – For customisation of the product.
- ✓ Registration and login - Data of the customer as well as Admin.
- ✓ Product master – Provides details of a product such as product number, name, category, brand, photos, size chart, requirements of items to be featured on the website etc. He also connects with the suppliers of various brands.

- ✓ Price master – He works with the quality of the products and the discounts and coupons which are applicable for a single product of the vendor/seller to offer the customer.
- ✓ Transactions – All the transactions of products carried on the website will be through the shopping cart.
- ✓ Reporting – This module deals with the management of the order report and the delivery report. Order report will include the list of the items that are ordered and also the details of the customers who requested the tracking of their products which are not shipped. Delivery report produces a list of items that are successfully shipped to the respective customers.
- ✓ Payment – Payment of the products will be accepted by online transactions through bank accounts. An option of cash on delivery is also available.
- ✓ Tracing Transactions – Product tracking details provided as it gets shipped.
- ✓ Customer care and feedback – An option of video calling, emails, contact number provided to the customers.

2.3 Non-Functional Requirements

- ✓ **Availability** – The website should be able to run every day for 24 hours.
- ✓ **Scalability** – Highly scalable.
- ✓ **Maintainability** – Able to maintain the data of the customers and to update new stocks on a regular basis according to the recent trends.
- ✓ **Portability** – Can be used on any Android or IOS mobile phones, laptops, computers etc.
- ✓ **Reliability** – Highly reliable.
- ✓ **Privacy** – Secured and verified products. The data of the customers is secured.
- ✓ **Environment friendly** – No plastic usage for packaging of products.

2.4 Ambiguities, Inconsistencies, Incompleteness from Requirements.

- ✓ **Ambiguity -** There are times when the customer enters a fake address. This leads to loss in the transportation and delivery costs. It also wastes a lot of time and eventually slows down the pace at which we are working. Hence address verification of the customers will be a tedious process.
- ✓ **Inconsistencies** – Customers providing incorrect details on their profile.

- ✓ **Incompleteness** - There is a policy of "Return and refund of the product within 15 days" if the customer is not satisfied with the product received. (Exclusive of customized products). Delay in delivery of products.

2.5 End User of the Solution

The end user is basically the customer (Open to all age groups) who is willing to shop from this website.

2.6 Operating Environment

- ✓ **User requirements**

  An electronic device (Android or IOS).
  A verified bank account (for online payment/transactions)

- ✓ **System Requirements**

  System with i3 processor or higher.
  Windows 7 or higher.
  Mac OS 10.12 Sierra or higher.
  4GB RAM or higher.

2.7 Design and Implementation Constraints

- ✓ System is wirelessly networked with an encryption.
- ✓ System is only accessible within globally.
- ✓ Database is password protected.
- ✓ Should use less RAM and processing power.
- ✓ Each user should have individual ID and password.
- ✓ Only administrator can access the whole system.
- ✓ Admin data is also password protected.

2.8 Assumptions and Dependencies

- ✓ Each user must have a valid user id and password.
- ✓ Server must be running for the system to run and function.
- ✓ Users must register to proceed for buying the product.
- ✓ Users must log in to the system to access any record.
- ✓ Only the Administrator can delete and update records.

- ✓ The database capacity is good enough for people to access the website worldwide.

2.9 Other Non - Functional Requirements

2.9.1 Performance Requirements

- ✓ Response time - The system will give responses within 0.5 second when selecting or opening any icons/categories/products on the website.
- ✓ Capacity - The system must support more than1000 people at a time.
- ✓ User interface - User interface screen will response within 3 seconds.
- ✓ Conformity - The system must conform to the Microsoft accessibility.
- ✓ The system is interactive and the delays involved are less or equal to zero.

2.9.2 Safety and Security Requirements

- ✓ If there is extensive damage to a wide portion of the database due to catastrophic failure, such as a disk crash, the recovery method restores a past copy of the database that was backed up to archival storage and reconstructs a more current state by reapplying or redoing the operations of committed transactions from the backed-up log, up to the time of failure.

- ✓ All the administrative and data entry operators have unique logins so system can understand who has logged in to the system right now and no intruders allowed. Except system administrative nobody can change records and valuable data of the system.

- ✓ Thinking about all aspects, there is a possibility of the database getting crashed at any time due to a virus or maybe an operating system failure. Therefore, a database backup is a must.
- ✓ Hacking can also be a threat to the system, thus the user id and password and must be hidden by the user.

- ✓ The system uses SSL (secured socket layer) in all transactions that include any confidential customer information.

- ✓ The system must automatically log out all customers after a period of inactivity.

- ✓ Since only authenticated users can have access to the server, it is highly secure. The 3 aspects covered are-

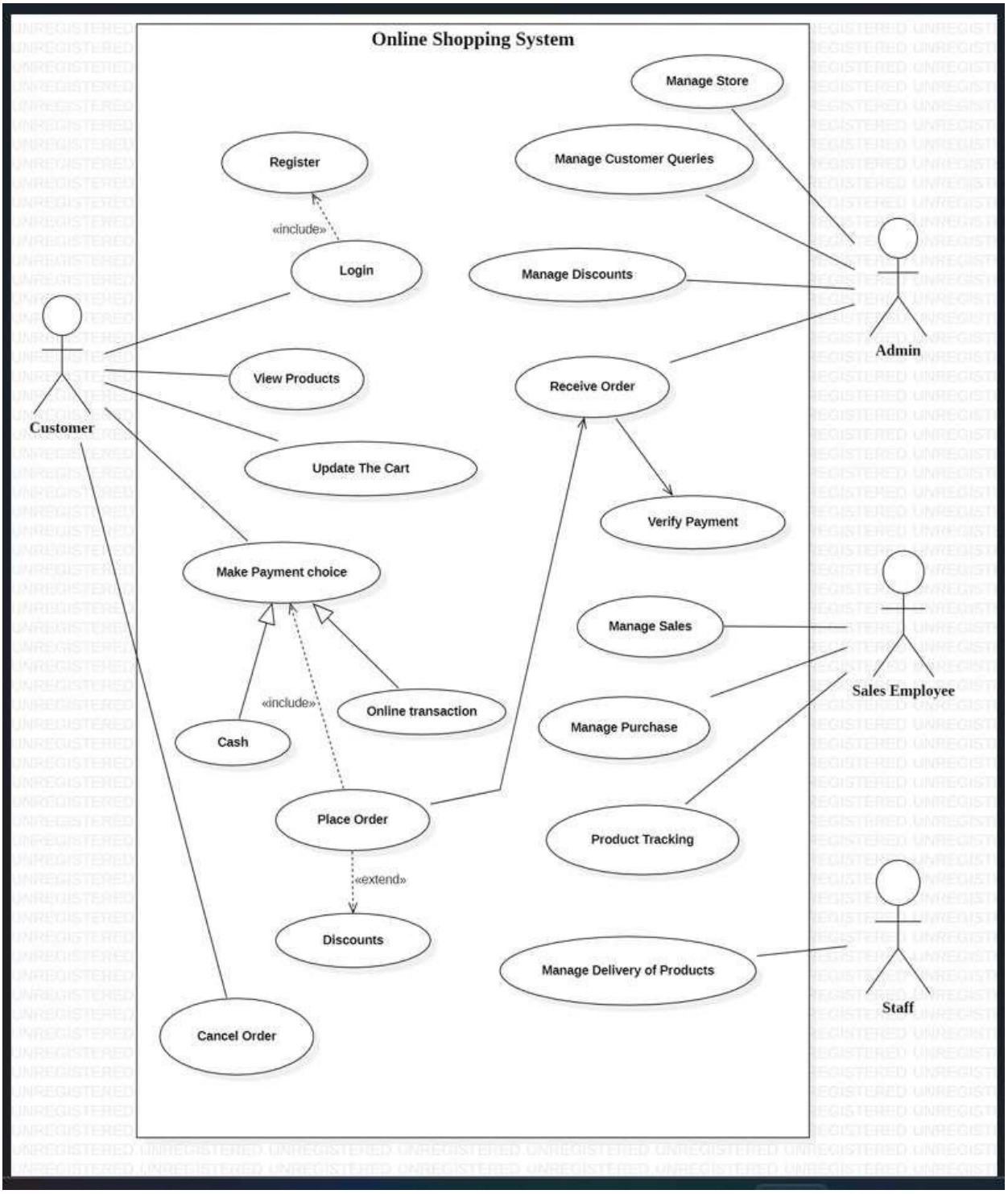
a. Confidentiality: The user shall ensure who gets to see their information
b. Integrity: Editing data shall be authorized only to owners of the accounts.
c. Availability: The application shall be always available at any hour of the day.

2.9.3 Software Quality Attributes

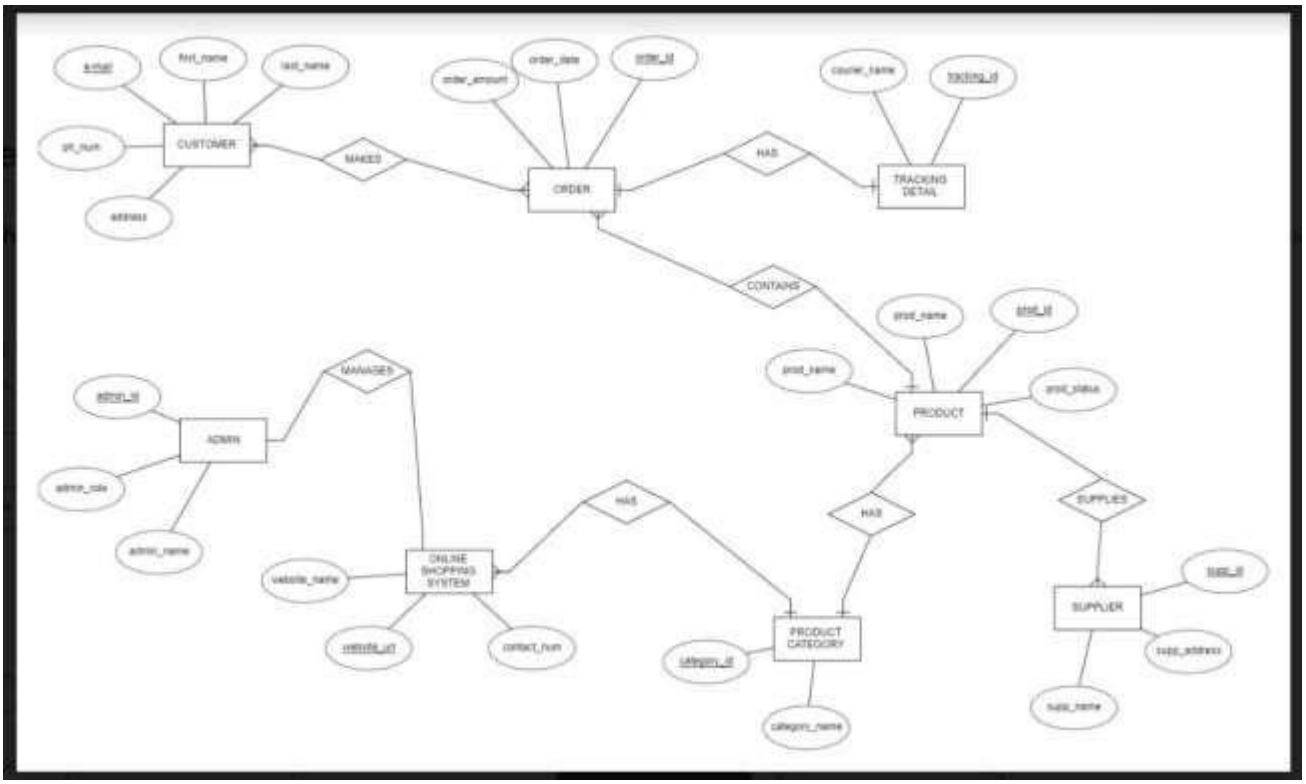
The 10 Software Quality Attributes are as follows –

1. **Availability:** The system shall be available all the time.
2. **Correctness:** A bug free software which fulfils the correct needs/requirements of the client.
3. **Maintainability:** The ability to maintain, modify information and update and repair fix problems of the system.
4. **Usability:** Software can be used again and again without distortion.
5. **Accessibility:** Administrator and many other users can access the system but the access level is controlled for each user according to their work scope.
6. **Reliability:** Systems are often liable to failure when the demand is high especially during crisis. Thus, our website should function smoothly and should be reliable. It is crucial that the system should be built taking into account the overload on the communication channels.
7. **Scalability:** Since our website is meant to be used by people all across the globe, it should be highly scalable.
8. **Extensibility:** Our website will have the ability to extend its requirements and to allow adding other services in the future. It should be able to adapt to new changes in specification.
9. **Reusability:** The website should be developed using a modular approach.
10. **Flexibility:** Should be flexible enough to modify. Adaptable to other products with which it needs interaction. Should be easy to interface with other standard 3rd party components.

3. USE CASE DIAGRAM

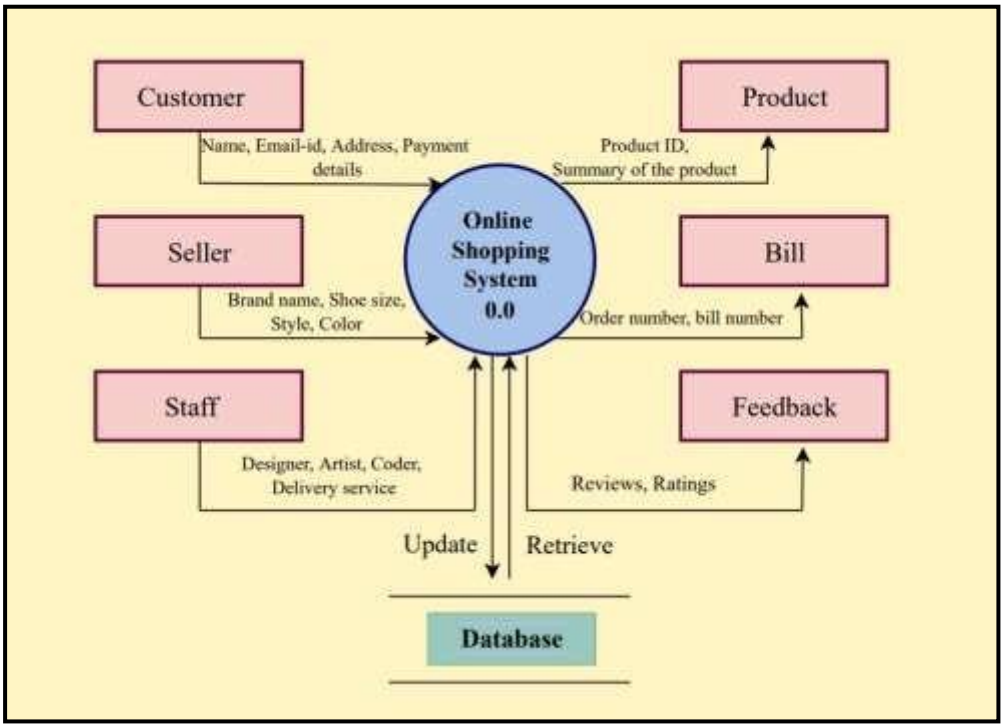


4. ERD

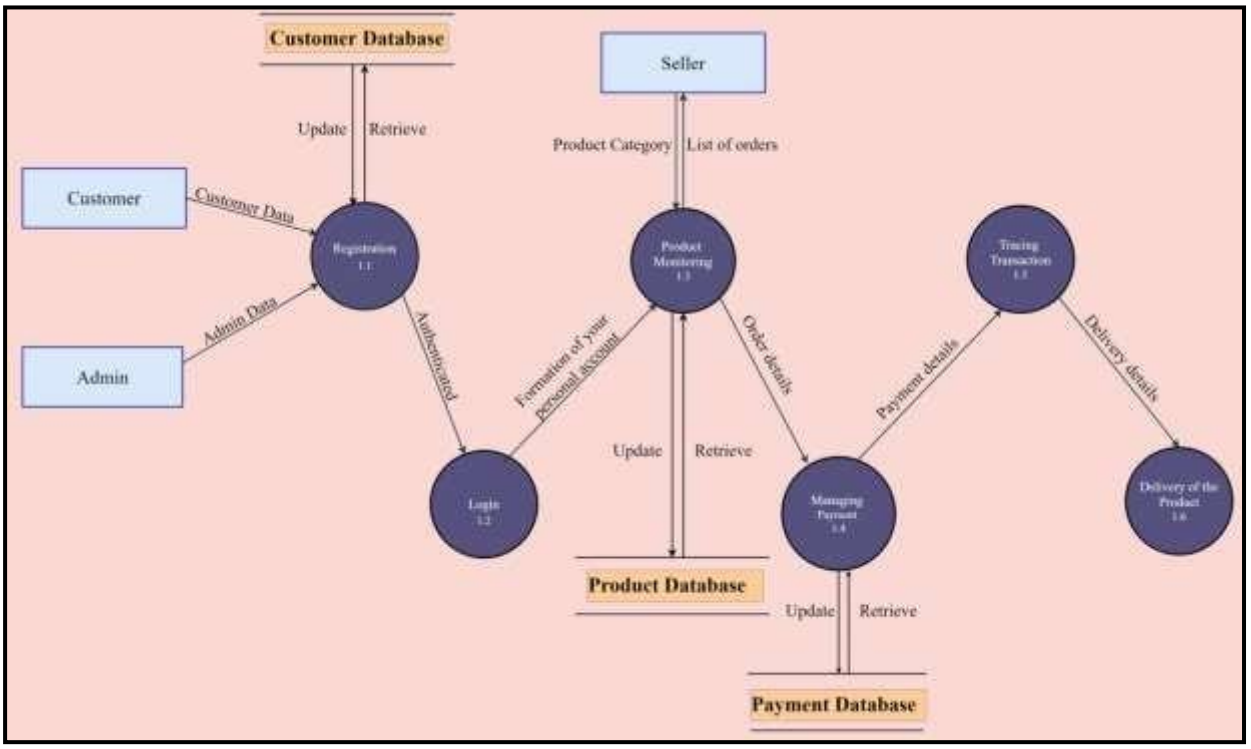


5. DATA FLOW DIAGRAMS

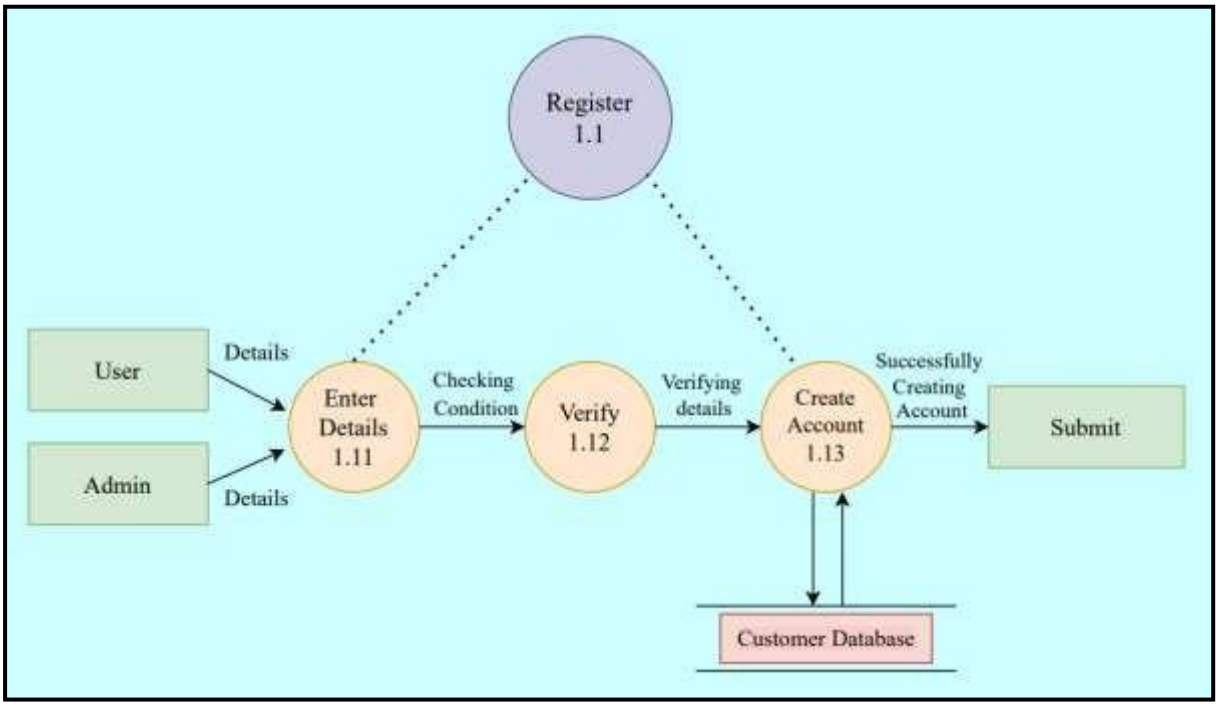
1. Level 0

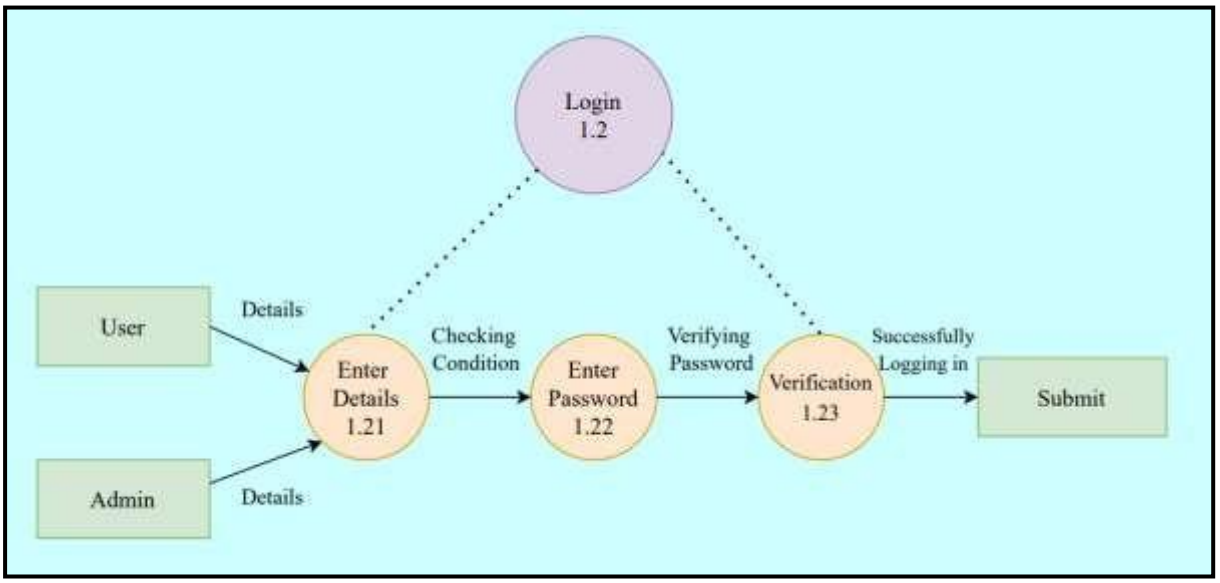


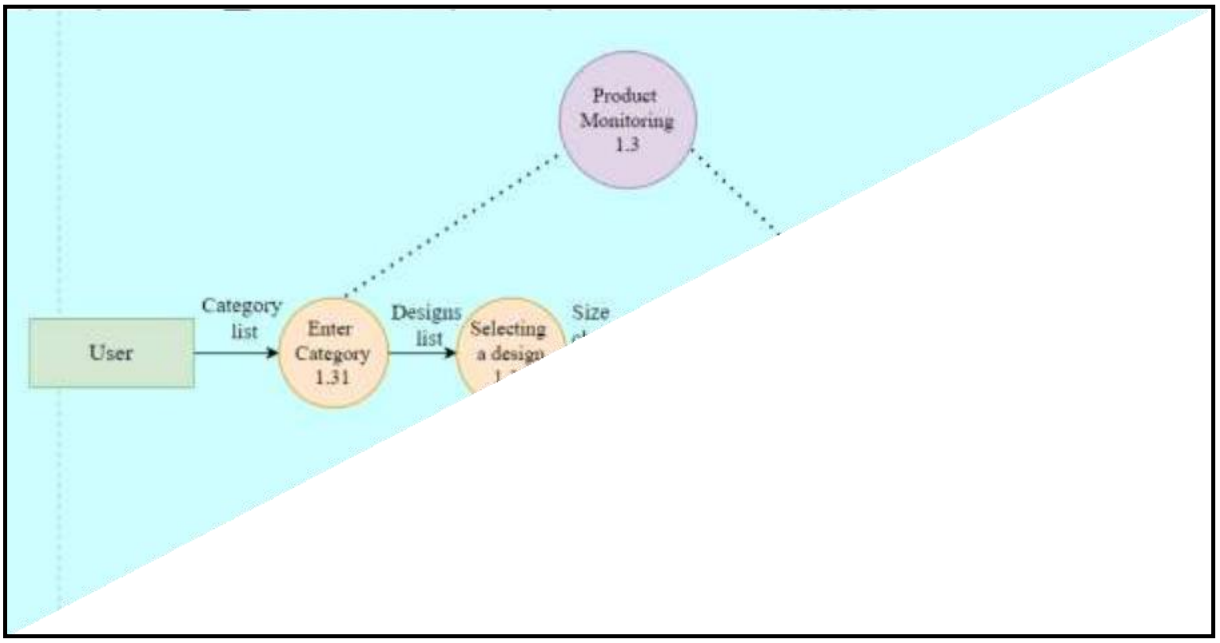
2. Level 1

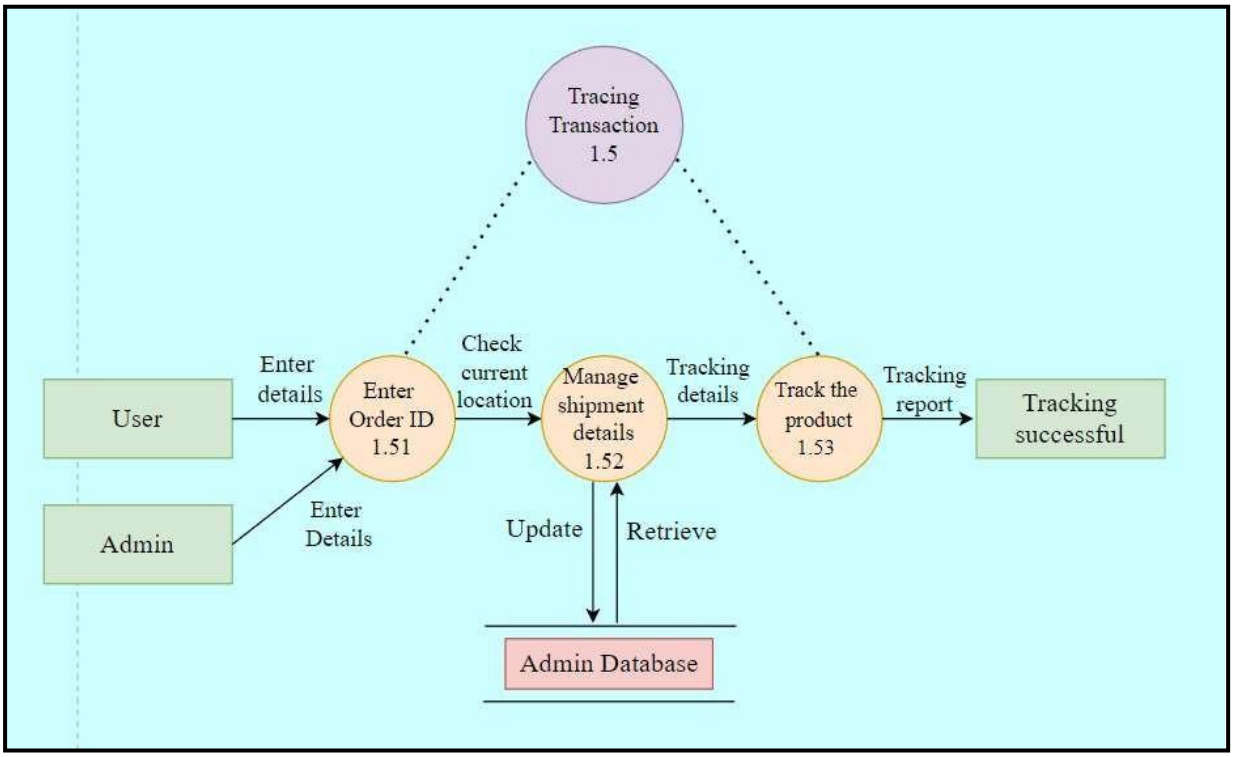


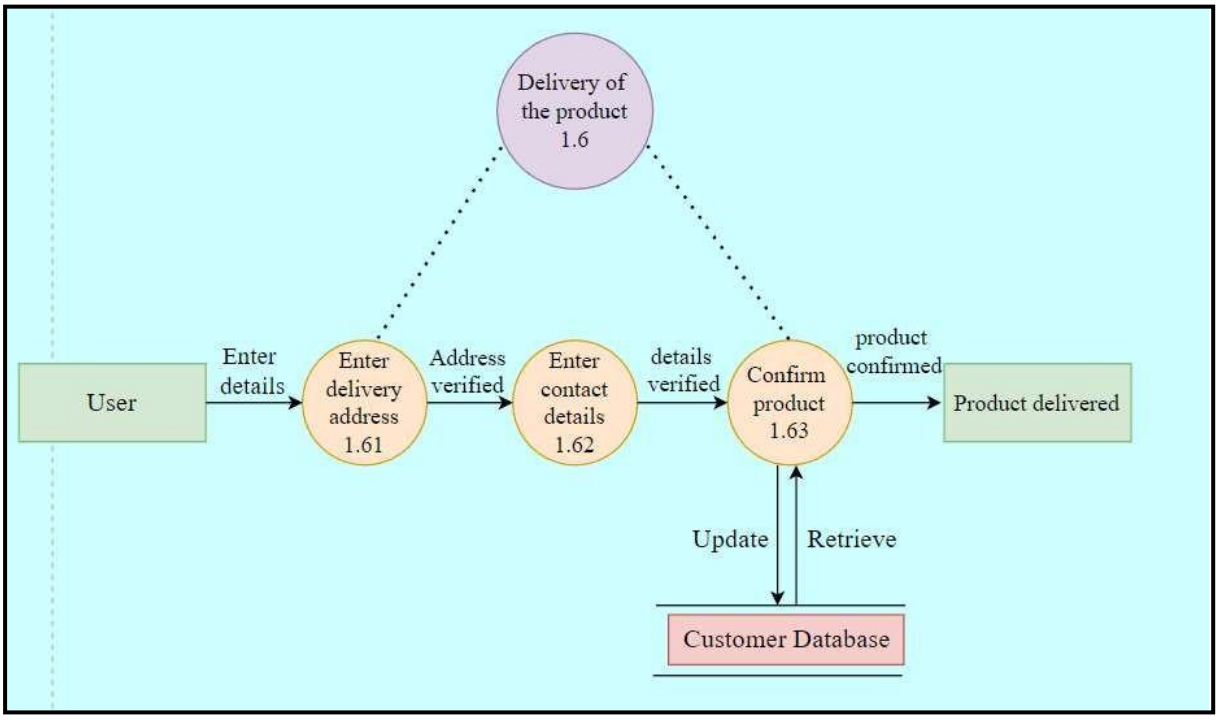
3. Level 2


Register
1.1
User
Details
Admin
Details
Enter
Details
1.11
Checking
Condition
Verify
1.12
Verifying
details
Create
Account
1.13
Successfully
Creating
Account
Submit
Customer Database


Login
1.2
User
Details
Admin
Details
Enter
Details
1.21
Checking
Condition
Enter
Password
1.22
Verifying
Password
Verification
1.23
Successfully
Logging in
Submit


Product
Monitoring
1.3
User
Category
list
Enter
Category
1.31
Designs
list
Selecting
a design
Size


Tracing Transaction 1.5
User
Enter details
Enter Order ID 1.51
Check current location
Manage shipment details 1.52
Tracking details
Track the product 1.53
Tracking report
Tracking successful
Admin
Enter Details
Update
Retrieve
Admin Database


Delivery of the product 1.6
User
Enter details
Enter delivery address 1.61
Address verified
Enter contact details 1.62
details verified
Confirm product 1.63
product confirmed
Product delivered
Update
Retrieve
Customer Database

Inputs:

Product Microservice:

Dependencies:

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>3.1.5</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>com.microservice</groupId>
   <artifactId>productService</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>productService</name>
   <description>Product service for microservice</description>
   <properties>
      <java.version>17</java.version>
   </properties>
   <repositories>
      <repository>
         <id>central</id>
         <url>https://repo.maven.apache.org/maven2</url>
      </repository>
   </repositories>
   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>

      <dependency>
         <groupId>com.mysql</groupId>
         <artifactId>mysql-connector-j</artifactId>
         <scope>runtime</scope>
      </dependency>
      <dependency>
         <groupId>org.projectlombok</groupId>
         <artifactId>lombok</artifactId>
         <optional>true</optional>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-test</artifactId>
         <scope>test</scope>
      </dependency>
      <dependency>
         <groupId>org.springframework.restdocs</groupId>
         <artifactId>spring-restdocs-mockmvc</artifactId>
         <scope>test</scope>
      </dependency>
   </dependencies>
```

```
   <build>
      <plugins>
         <plugin>
            <groupId>org.asciidoctor</groupId>
            <artifactId>asciidoctor-maven-plugin</artifactId>
            <version>2.2.1</version>
            <executions>
               <execution>
                  <id>generate-docs</id>
                  <phase>prepare-package</phase>
                  <goals>
                     <goal>process-asciidoc</goal>
                  </goals>
                  <configuration>
                     <backend>html</backend>
                     <doctype>book</doctype>
                  </configuration>
               </execution>
            </executions>
            <dependencies>
               <dependency>
                  <groupId>org.springframework.restdocs</groupId>
                  <artifactId>spring-restdocs-asciidoctor</artifactId>
                  <version>${spring-restdocs.version}</version>
               </dependency>
            </dependencies>
         </plugin>
         <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
               <excludes>
                  <exclude>
                     <groupId>org.projectlombok</groupId>
                     <artifactId>lombok</artifactId>
                  </exclude>
               </excludes>
            </configuration>
         </plugin>
      </plugins>
   </build>

</project>
```

Controller:

```
package com.microservice.productService.controller;

import com.microservice.productService.payload.request.ProductRequest;
import com.microservice.productService.payload.response.ProductResponse;
import com.microservice.productService.service.impl.ProductService;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/product")
@RequiredArgsConstructor
@Log4j2
public class ProductController {
    private final ProductService productService;
```

```
    @PostMapping
    public ResponseEntity<Long> addProduct(@RequestBody ProductRequest
productRequest) {

        log.info("ProductController | addProduct is called");

        log.info("ProductController | addProduct | productRequest : " +
productRequest.toString());

        long productId = productService.addProduct(productRequest);
        return new ResponseEntity<>(productId, HttpStatus.CREATED);
    }

    @GetMapping("/{id}")
    public ResponseEntity<ProductResponse> getProductById(@PathVariable("id") long
productId) {

        log.info("ProductController | getProductById is called");

        log.info("ProductController | getProductById | productId : " + productId);

        ProductResponse productResponse
                = productService.getProductById(productId);
        return new ResponseEntity<>(productResponse, HttpStatus.OK);
    }

    @PutMapping("/reduceQuantity/{id}")
    public ResponseEntity<Void> reduceQuantity(
            @PathVariable("id") long productId,
            @RequestParam long quantity
    ) {

        log.info("ProductController | reduceQuantity is called");

        log.info("ProductController | reduceQuantity | productId : " + productId);
        log.info("ProductController | reduceQuantity | quantity : " + quantity);

        productService.reduceQuantity(productId,quantity);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @DeleteMapping("/{id}")
    public void deleteProductById(@PathVariable("id") long productId) {
        productService.deleteProductById(productId);
    }
}
```

Entity:

```
package com.microservice.productService.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.springframework.boot.autoconfigure.web.WebProperties;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long productId;
```

```
    @Column(name = "PRODUCT_NAME")
    private String productName;

    @Column(name = "PRICE")
    private long price;

    @Column(name = "QUANTITY")
    private long quantity;

}
```

Implementation:

```
package com.microservice.productService.service.impl;

import com.microservice.productService.entity.Product;
import com.microservice.productService.exception.ProductServiceCustomException;
import com.microservice.productService.payload.request.ProductRequest;
import com.microservice.productService.payload.response.ProductResponse;
import com.microservice.productService.repository.ProductRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Service;

import static org.springframework.beans.BeanUtils.copyProperties;

@Service
@RequiredArgsConstructor
@Log4j2

public  class ProductService{
    private final ProductRepository productRepository;

    public long addProduct(ProductRequest productRequest) {
        log.info("ProductServiceImpl | addProduct is called");

        Product product
                = Product.builder()
                .productName(productRequest.getName())
                .quantity(productRequest.getQuantity())
                .price(productRequest.getPrice())
                .build();

        product = productRepository.save(product);

        log.info("ProductServiceImpl | addProduct | Product Created");
        log.info("ProductServiceImpl | addProduct | Product Id : " +
product.getProductId());
        return product.getProductId();
    }
    public ProductResponse getProductById(long productId) {

        log.info("ProductServiceImpl | getProductById is called");
        log.info("ProductServiceImpl | getProductById | Get the product for
productId: {}", productId);

        Product product
                = productRepository.findById(productId)
                .orElseThrow(
                        () -> new ProductServiceCustomException("Product with
given Id not found","PRODUCT_NOT_FOUND"));

        ProductResponse productResponse
                = new ProductResponse();
```

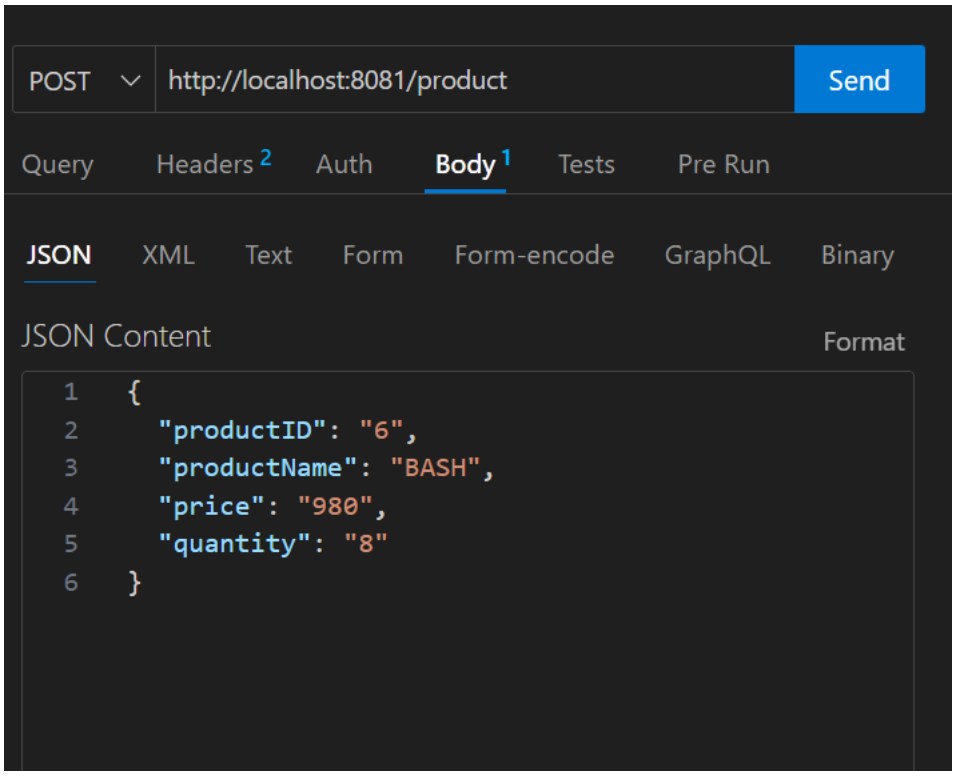
```
        copyProperties(product, productResponse);

        log.info("ProductServiceImpl | getProductById | productResponse :" +
productResponse.toString());

        return productResponse;
    }


    public void reduceQuantity(long productId, long quantity) {

        log.info("Reduce Quantity {} for Id: {}", quantity,productId);

        Product product
                = productRepository.findById(productId)
                .orElseThrow(() -> new ProductServiceCustomException(
                        "Product with given Id not found",
                        "PRODUCT_NOT_FOUND"
                ));

        if(product.getQuantity() < quantity) {
            throw new ProductServiceCustomException(
                    "Product does not have sufficient Quantity",
                    "INSUFFICIENT_QUANTITY"
            );
        }

        product.setQuantity(product.getQuantity() - quantity);
        productRepository.save(product);
        log.info("Product Quantity updated Successfully");
    }


    public void deleteProductById(long productId) {
        log.info("Product id: {}", productId);

        if (!productRepository.existsById(productId)) {
            log.info("Im in this loop {}",
!productRepository.existsById(productId));
            throw new ProductServiceCustomException(
                    "Product with given with Id: " + productId + " not found:",
                    "PRODUCT_NOT_FOUND");
        }
        log.info("Deleting Product with id: {}", productId);
        productRepository.deleteById(productId);

    }
}
```

Main Function:

```
package com.microservice.productService;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;


@SpringBootApplication
@EnableJpaRepositories("com.microservice.productService.repository")
public class ProductServiceApplication {

    public static void main(String[] args) {
```

```
        SpringApplication.run(ProductServiceApplication.class, args);
    }

}
```

Output:



product_id	price	product_name	quantity
1	0	NULL	0
2	200	NULL	12
52	980	NULL	8
NULL	NULL	NULL	NULL

Payment Microservice:

Inputs:

Entity:

```
package com.microservice.paymentservice.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.time.Instant;

@Data
@Entity
@Table(name = "TRANSACTION_DETAILS")
@AllArgsConstructor
@NoArgsConstructor
@Builder
public class TransactionDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "ORDER_ID")
    private long orderId;

    @Column(name = "MODE")
    private String paymentMode;

    @Column(name = "REFERENCE_NUMBER")
    private String referenceNumber;

    @Column(name = "PAYMENT_DATE")
    private Instant paymentDate;

    @Column(name = "STATUS")
    private String paymentStatus;

    @Column(name = "AMOUNT")
    private long amount;
}
```

Controller:

```
package com.microservice.paymentservice.controller;

import com.microservice.paymentservice.payload.PaymentRequest;
import com.microservice.paymentservice.payload.PaymentResponse;
import com.microservice.paymentservice.service.paymentservice1;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;


@RestController
@RequestMapping("/payment")
@RequiredArgsConstructor
```

```
@Log4j2
public class PaymentController {

    private final paymentservice1 paymentService;

    @PostMapping
    public ResponseEntity<Long> doPayment(@RequestBody PaymentRequest
paymentRequest) {

        log.info("PaymentController | doPayment is called");

        log.info("PaymentController | doPayment | paymentRequest : " +
paymentRequest.toString());

        return new ResponseEntity<>(
                paymentService.doPayment(paymentRequest),
                HttpStatus.OK
        );
    }
    @GetMapping("/order/{orderId}")
    public ResponseEntity<PaymentResponse>
getPaymentDetailsByOrderId(@PathVariable long orderId) {

        log.info("PaymentController | doPayment is called");

        log.info("PaymentController | doPayment | orderId : " + orderId);

        return new ResponseEntity<>(
                paymentService.getPaymentDetailsByOrderId(orderId),
                HttpStatus.OK
        );
    }
}
```

Implementation Class:

```
package com.microservice.paymentservice.service;

import com.microservice.paymentservice.exception.PaymentServiceCustomException;
import com.microservice.paymentservice.model.TransactionDetails;
import com.microservice.paymentservice.payload.PaymentRequest;
import com.microservice.paymentservice.payload.PaymentResponse;
import com.microservice.paymentservice.repository.TransactionDetailsRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.springframework.stereotype.Service;

import java.time.Instant;

@Service
@Log4j2
@RequiredArgsConstructor
public class paymentservice1 {
    private final TransactionDetailsRepository transactionDetailsRepository;

    public long doPayment(PaymentRequest paymentRequest) {

        log.info("PaymentServiceImpl | doPayment is called");

        log.info("PaymentServiceImpl | doPayment | Recording Payment Details: {}",
paymentRequest);

        TransactionDetails transactionDetails
                = TransactionDetails.builder()
```

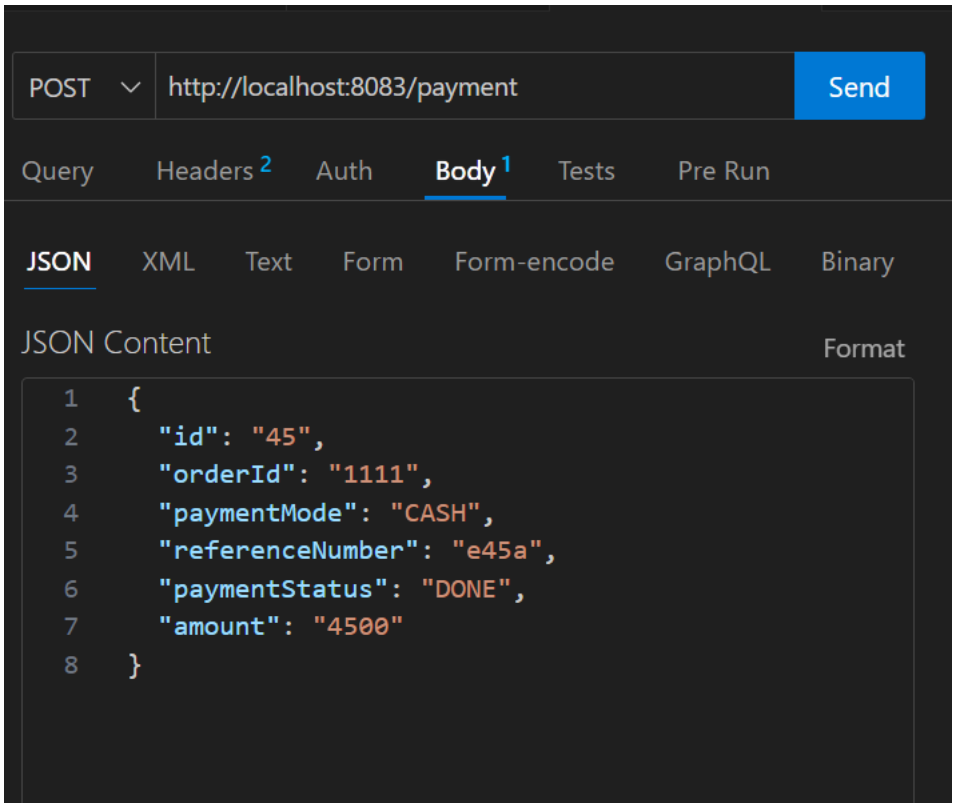
```
                .paymentDate(Instant.now())
                .paymentMode(paymentRequest.getPaymentMode())
                .paymentStatus("SUCCESS")
                .orderId(paymentRequest.getOrderId())
                .referenceNumber(paymentRequest.getReferenceNumber())
                .amount(paymentRequest.getAmount())
                .build();

        transactionDetails =
transactionDetailsRepository.save(transactionDetails);

        log.info("Transaction Completed with Id: {}", transactionDetails.getId());

        return transactionDetails.getId();
    }
    public PaymentResponse getPaymentDetailsByOrderId(long orderId) {

        log.info("PaymentServiceImpl | getPaymentDetailsByOrderId is called");

        log.info("PaymentServiceImpl | getPaymentDetailsByOrderId | Getting
payment details for the Order Id: {}", orderId);

        TransactionDetails transactionDetails
                = transactionDetailsRepository.findByOrderId(orderId)
                .orElseThrow(() -> new PaymentServiceCustomException(
                        "TransactionDetails with given id not found",
                        "TRANSACTION_NOT_FOUND"));

        PaymentResponse paymentResponse
                = PaymentResponse.builder()
                .paymentId(transactionDetails.getId())
                .paymentMode(String.valueOf(transactionDetails.getPaymentMode()))
                .paymentDate(transactionDetails.getPaymentDate())
                .orderId(transactionDetails.getOrderId())
                .status(transactionDetails.getPaymentStatus())
                .amount(transactionDetails.getAmount())
                .build();

        log.info("PaymentServiceImpl | getPaymentDetailsByOrderId |
paymentResponse: {}", paymentResponse.toString());

        return paymentResponse;
    }
}
```

Application:

```
package com.microservice.paymentservice;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@SpringBootApplication
@EnableJpaRepositories("com.microservice.paymentservice.repository")

public class PaymentserviceApplication {

   public static void main(String[] args) {
      SpringApplication.run(PaymentserviceApplication.class, args);
   }
```

```
}
```

Outputs:



7	231	0	2023-11-03 08:42:05.797950	NULL	SUCCESS
52	4500	1111	2023-11-05 13:37:25.578014	CASH	SUCCESS
NULL	NULL	NULL	NULL	NULL	NULL