ShopSmart
Software Design Description

1. Revision History

Version	Date	Author	Change Description
1.0	12/05/2024	Mert Tazeoğlu	Creation of document

2. INTRODUCTION

2.1 Purpose and Scope

Purpose

The purpose of this Software Design Description (SDD) is to provide a comprehensive architectural overview of the ShopSmart e-commerce application. This document is intended to guide the development team through the design implementation of the application and serve as an informative resource for stakeholders to understand the technical framework and functionality of ShopSmart. Additionally, the SDD will detail the system architecture, components, and interfaces, ensuring alignment with the project requirements and objectives.

Scope

This e-commerce system is designed to enable users to browse, select, and purchase products online with ease. The scope of this document includes detailed descriptions of the system's architecture, components, interfaces, and data design. It covers the software setup, including the back-end services built with Java and Spring, the database integration with PostgreSQL, and the front-end client interfaces. The document also addresses the system's performance, scalability, security measures, and compliance with relevant standards and regulations.

2.2 Document Overview

System Overview: This section provides a high-level description of the complete system, including its architecture and the interactions between components, typically represented through a commented drawing.

Definitions, Acronyms, and Abbreviations: This section lists and defines all non-standard terms, acronyms, and abbreviations used throughout this document to ensure clarity and a common understanding among all stakeholders.

Design Constraints and Decisions: This section discusses the constraints and decisions shaping the system design, including general design constraints like project scope and resources, hardware constraints impacting performance, software environment constraints such as operating systems and programming languages, and key design decisions regarding component interfaces, data models, security, and privacy.

Design Details: This section details the components of the system design, including: the software components used within the system, their responsibilities, and interactions; the behavior of software components under various scenarios, the data model, presented through an entity-relationship diagram, and discussion of the database schema, the user interface design, focusing on user interaction flows and visual design.

Requirements Traceability: This section maps out how each software requirement is addressed in the design specifications, ensuring all requirements are met.

Annexes: This section contains supplementary materials that support the main text, such as additional diagrams, code snippets, and reference documents. Provides a detailed table mapping each team member's responsibilities to specific design elements and tasks to ensure accountability.

2.3 System Overview

The system designed is a comprehensive web-based e-commerce platform, enabling users to browse, select, and purchase products conveniently online. The architecture is modular, built on a robust framework of Java and Spring, supporting scalability and maintainability. Below is a general description of the system, complemented by a commented drawing that illustrates the major components and their interactions.

• **User Interface (UI):** A responsive web interface that adapts to various devices, providing a seamless shopping experience for users. It communicates with backend services via RESTful APIs.

• **Business Logic Layer:** Implements the core functionality of the e-commerce platform, including product management, user accounts, and order processing. This layer handles the business rules and ensures data integrity and transaction management.

• **Data Access Layer:** Manages communication between the business logic layer and the database. It uses Spring Data to abstract and encapsulate all access to the PostgreSQL database.

• **Database:** Utilizes PostgreSQL to store and manage all persistent data, including user profiles, product catalogues, and order history. The database is designed for high availability and consistency.

• **Security Module:** Ensures the confidentiality, integrity, and availability of user data. It handles authentication (using JWT for secure token generation) and authorization, protecting against common security threats.

• **Infrastructure:** In the future, we suggest to host project on a combination of on-premise servers and cloud-based services, providing necessary scalability and reliability. It includes a continuous integration/continuous deployment (CI/CD) pipeline for streamlined development and maintenance.

2.4 Definitions, Acronyms, and Abbreviations

Term/Acronym	***Definition***
IEEE	Institute of Electrical and Electronics Engineers: A professional association dedicated to advancing technological innovation and excellence for the benefit of humanity.
SRS	System Requirements Specification: A comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform.
API	Application Programming Interface: A set of rules that allows different software entities to communicate with each other.
GDPR	General Data Protection Regulation: A legal framework that sets guidelines for the collection and processing of personal information from individuals who live in the European Union.
JWT	JSON Web Token: A compact, URL-safe means of representing claims to be transferred between two parties.
UX	User Experience: The overall experience of a person using a product such as a website or computer application, especially in terms of how easy or pleasing it is to use.
REST	Representational State Transfer: An architectural style designed for distributed systems, particularly for use in web services.
HTTP	Hypertext Transfer Protocol: An application protocol used for transmitting hypermedia documents, such as HTML, over the internet.
SQL	Structured Query Language: A standardized programming language used for managing relational databases and performing various operations on the data in them.
CI/CD	Continuous Integration/Continuous Deployment: A method to frequently deliver apps to customers by introducing automation into the stages of app development.

3. Design Constraints and Decisions

3.1 General Design Constraints

The design process of our project is constrained by several factors including time, tools, and resources. The technologies employed and the size of our team, along with the set delivery deadlines, directly influence the scope and the detailed design of the modules. For instance, the requirement to complete the project by a certain date may necessitate the simplification of some advanced features or the postponement of certain module developments.

3.2 Hardware Constraints

Hardware limitations, particularly the limitations on storage and processing power, significantly impact the design of our system. In this project, the performance of the PostgreSQL database system must be optimized based on the capacity of the server hardware used. This may require the adoption of more efficient data storage techniques and the offloading of certain processes to the client side.

3.3 Software Environment Constraints

The software environment introduces various constraints, including the operating system, programming languages, and libraries used. As this project is built on Java and the Spring Framework, the tools and libraries offered by these technologies shape our design. It may also need to be optimized to operate on a specific operating system or platform. Under normal conditions, it should work smoothly in internet browsers on computers with Java installed.

3.4 Design Decisions

• Component Interfaces: The interfaces between components in our system are designed to be provided. This facilitates easy integration of various frontend and backend components.

• Data Model: Our data model is structured in a normalized form, including basic entities such as products, users, and orders. PostgreSQL is used to support the ACID properties adequately.

• Security and Privacy: Security and privacy are ensured using JWT-based authentication. Additionally, encrypted data communication over HTTPS and GDPR-compliant policies should be implemented to protect user data.

• Development Infrastructure: The development process is standardized using Docker containers. This ensures consistent integration and deployment of the code.

• Scalability and Performance: The system should be developed to handle high user and transaction volumes and is developed using a special architecture to allow horizontal scalability. This approach enables dynamic scaling based on traffic increases.

• User Experience (UX) Design: To optimize user experience, the interface design is crafted considering user feedback and best UX practices. Responsive design techniques are applied to ensure a consistent user experience across mobile and desktop platforms.

• Database Optimization: Techniques such as indexing, query optimization, and the use of appropriate isolation levels should be implemented to enhance database performance. These optimizations aim to minimize response times, especially when working with large data sets.

• Fault Tolerance and Reliability: The system should be equipped with load balancing and automatic failover mechanisms to provide fault tolerance. This ensures that the application operates continuously even in the event of service disruptions.

• Security Enhancements: Security measures such as protections against SQL injection and XSS attacks, encryption of user data, and regular security audits should be included. Advanced encryption techniques are also should be applied for processing and storing sensitive data. On the other hand, hash functions should be used when storing the passwords in order to prevent attacks to see them if they will be able to capture database.

Note: In this section, the features that should be present in a real large-scale project are listed, and a significant part of them is not present in our project (in other words exists in documents but not in the code).

4. Design Details

4.1 Software Components

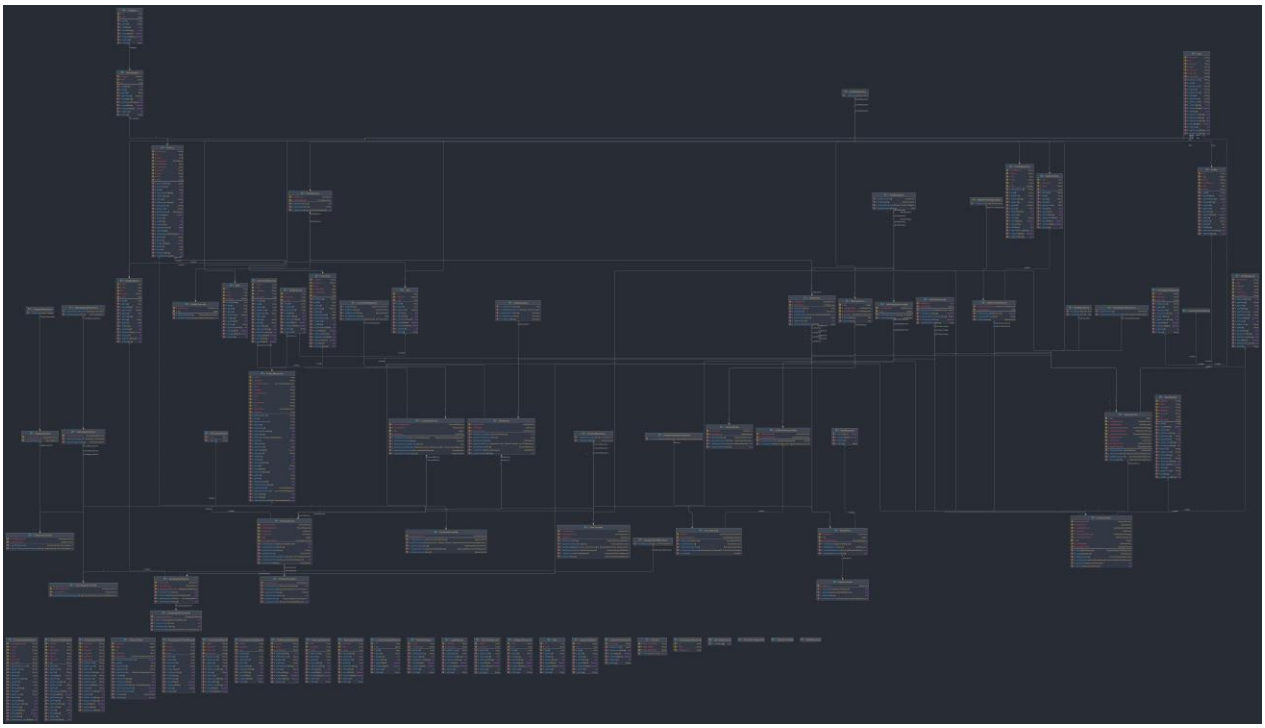


Figure-1:Updated Class Diagram for Show Product Use Case (@ Asım)
Note: Full sized version of class diagram is available on GitHub DEL-4 repository.

4.2 Software Behavior

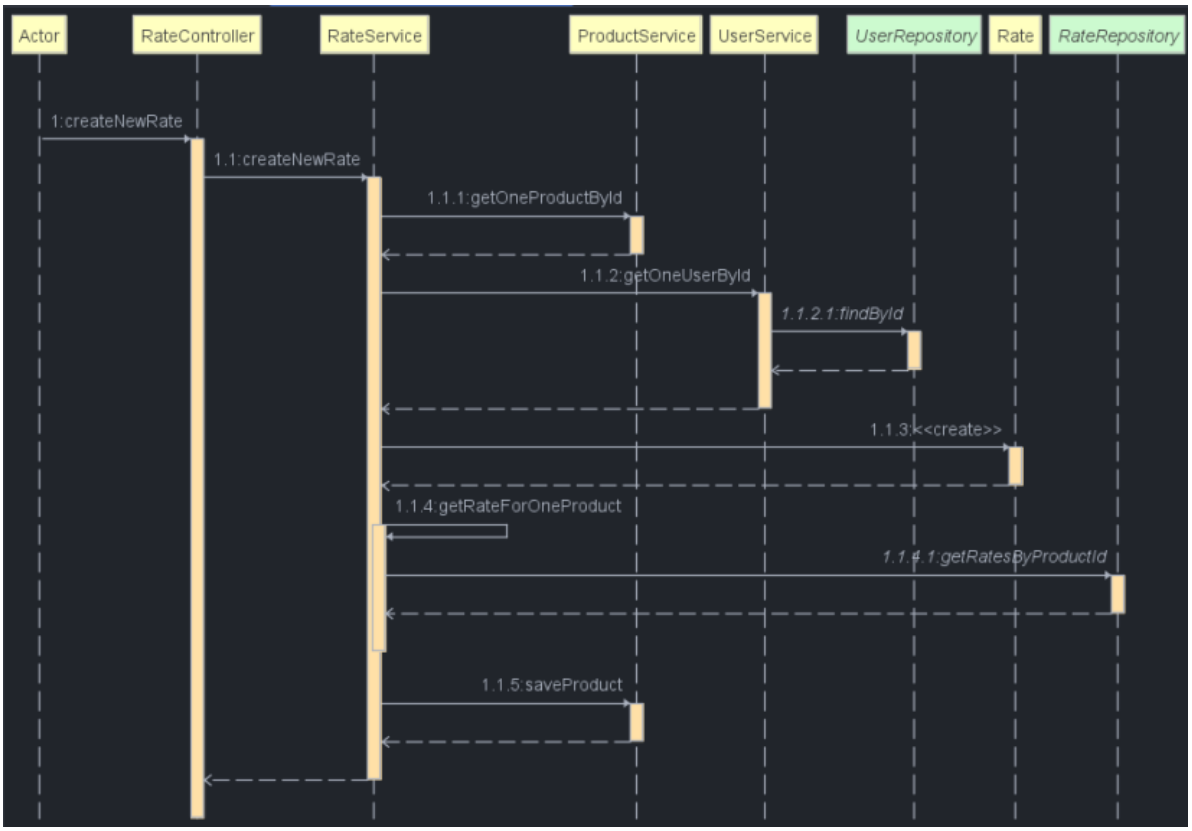
4.2.1 - Sequence Diagram for Show Product Use Case (@Kenan)

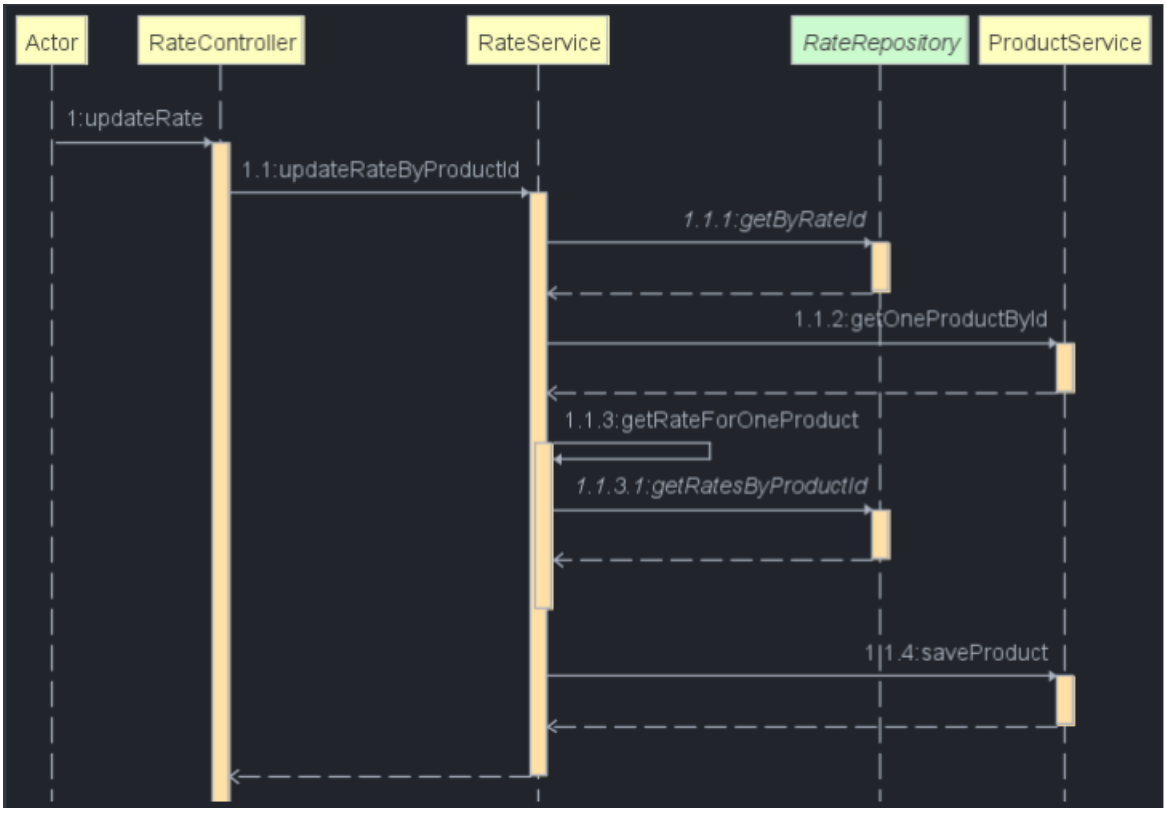
// This part was Kenan Gökdeniz Acet's responsibility, but he didn't do it

4.2.2 - Sequence Diagram for Manage Product Use Case (@Kenan)

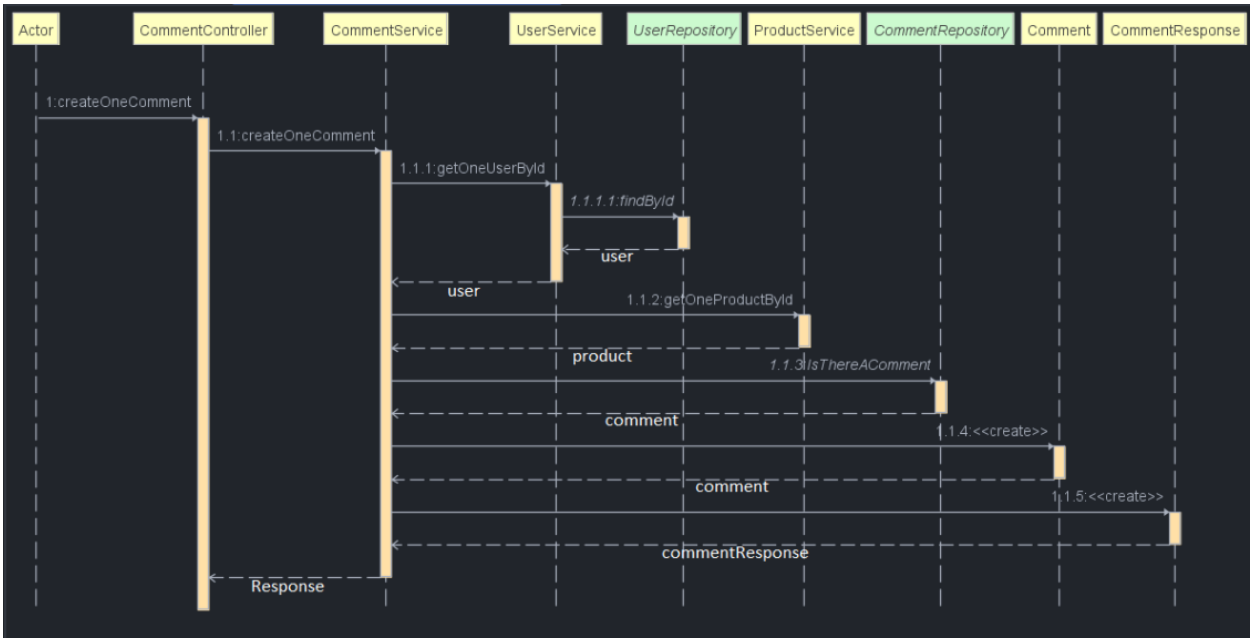
// This part was Kenan Gökdeniz Acet's responsibility, but he didn't do it

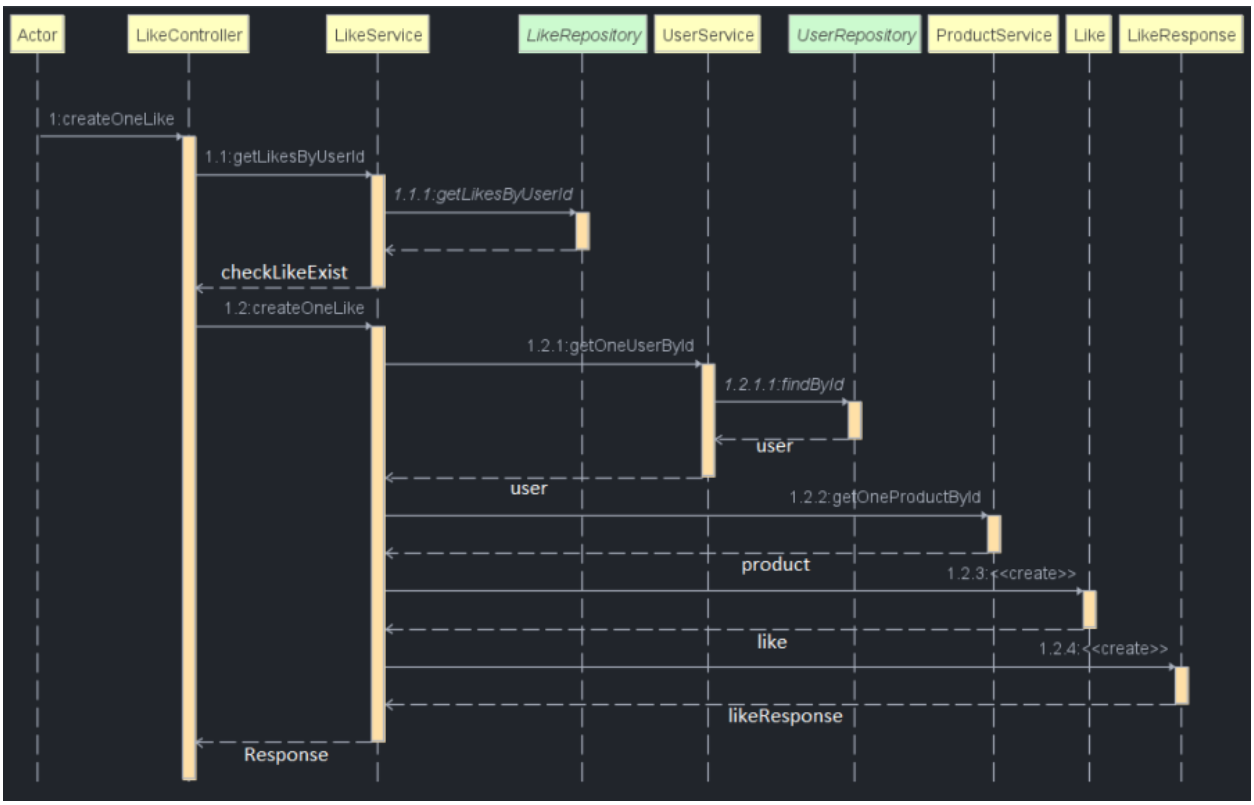
4.2.3 - 2 Sequence Diagrams for Rate Product Use Case (@Asım)





4.2.4 - 2 Sequence Diagrams for Like / Comment Product Use Case (@Asım)





4.2.5 - Sequence Diagram for Edit Own Profile Use Case (@Emre)

// This part was Emre Can Şahin's responsibility, but he didn't do it

4.2.6 - Sequence Diagram for Manage User Use Case (@Emre)

// This part was Emre Can Şahin's responsibility, but he didn't do it

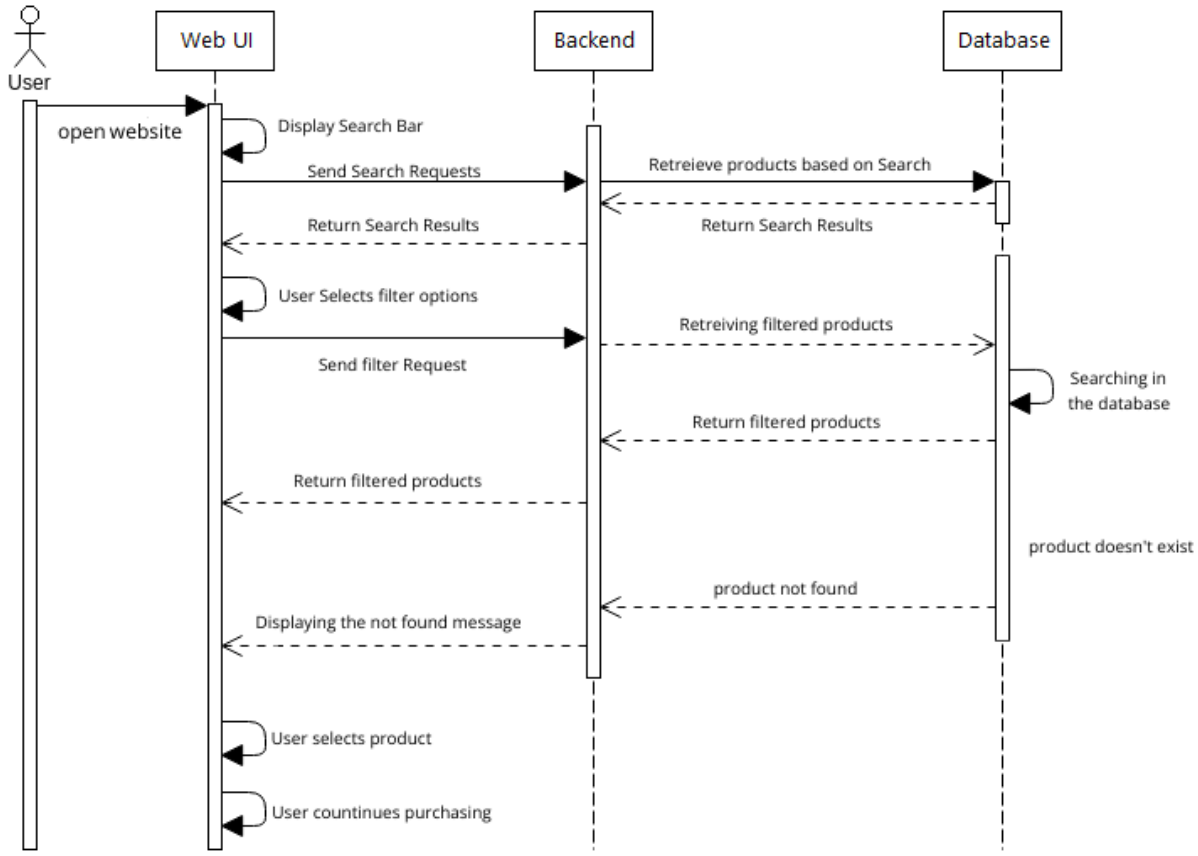
4.2.7 - Sequence Diagram for Add Product To Shopping Cart Use Case (@Tarık)

// This part was Tarık Sümer's responsibility, but he didn't do it

4.2.8 - Sequence Diagram for Buy Product Use Case (@Tarık)

// This part was Tarık Sümer's responsibility, but he didn't do it

4.2.9 - Sequence Diagram for Search Product Use Case (@Mert)



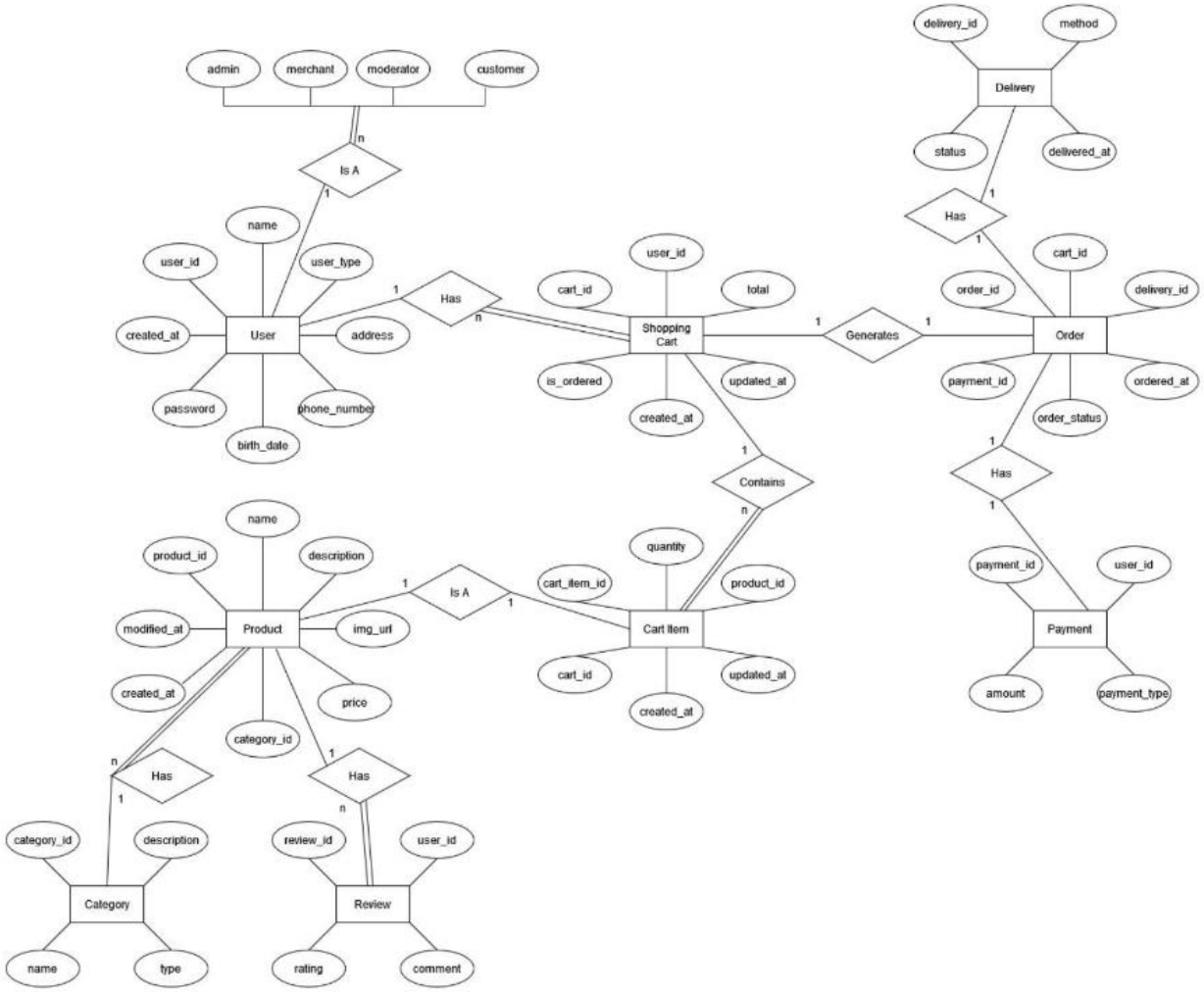
4.3 Data Model (E-R Diagram)

The data model of the Shopsmart e-commerce application consists of various tables that support the functionality of the application and optimize the user experience. This model covers essential e-commerce components such as users, products, orders, payments, deliveries, and product reviews. At the core of the data model are the "USER", "SHOPPING CART", "ORDER", and "REVIEW" tables, which manage user account information, shopping cart details, order history, and product reviews. These tables enable users to personalize their shopping experience and easily review past orders.

Product information is stored in the "PRODUCT" and "CATEGORY" tables, where each product can belong to multiple categories, supporting various classifications of a product within the database. Payment and delivery processes are managed through the "PAYMENT" and "DELIVERY" tables, allowing user preferences for payment and delivery to be flexibly handled. The "CART ITEM" table links the details of products in the shopping cart with the user's cart, including the quantity, price, and related product information of each order item.

Overall, Shopsmart's data model is designed to enable users to shop effectively, gather information about products, and experience a smooth payment and delivery process. This model helps the business manage its operations efficiently and make strategic decisions aimed at enhancing customer satisfaction.

We can summarize tables, their entities and relationships between tables like in the ER diagram below:



4.4 User Interface Design

For the User Interface Design section of the Shopsmart e-commerce application, we'll focus on designing interfaces that are both intuitive and visually appealing, while also addressing core principles of user interaction. By adhering to some basic principles, the user interface of Shopsmart will enhance the shopping experience, leading to higher user satisfaction and increased usability. This strategic focus on user-centered design will not only support efficient shopping but also foster positive user interactions with the Shopsmart platform.

• **Visibility of System Status:** The interface of Shopsmart will consistently provide feedback within a reasonable time after user actions. For example, when a user adds an item to their shopping cart, a visible confirmation message will appear, and the cart icon will update to reflect the new item count. This immediate feedback helps users understand that their actions have been successfully recognized and executed by the system.

• **User Control and Freedom:** Shopsmart's interface will support user autonomy by allowing easy navigation back and forth throughout their shopping experience. Features like a clearly labeled "back" button, breadcrumbs at the top of every page, and the ability to undo actions (such as removing an item from the cart) without having to navigate through multiple screens, will empower users to explore and make changes effortlessly.

• **Error Prevention:** The design will incorporate elements that help prevent errors before they occur. For instance, form entries for payment information will include real-time validation that checks for common input errors, like an incomplete credit card number or an expired date. By catching these errors early, the system minimizes the chances of failed transactions, enhancing user satisfaction and reducing frustration.

• **Aesthetic and Minimalist Design:** Shopsmart's interface will embrace a minimalist design philosophy, focusing on user needs without unnecessary elements. The product pages will display essential information prominently—such as prices, product images, and add-to-cart buttons—while additional details can be accessed as needed via expandable sections. This approach helps in reducing clutter and focusing the user's attention on their primary tasks.

• **Consistency and Standards:** Throughout the application, consistent interface elements ensure that users do not have to wonder whether different words, situations, or actions mean the same thing. Use of common icons, standardized color schemes, and familiar layouts across pages will make navigation intuitive for all users, whether they are first-time visitors or returning customers.

• **Responsive and Accessible Design:** The interface will be responsive, ensuring that Shopsmart is accessible and usable on a variety of devices, from desktops to mobile phones. Accessibility features such as keyboard navigability, screen reader support, and sufficient contrast ratios will be integral to the design, catering to users with disabilities and enabling a broader audience to use the application comfortably.

5. Requirements Traceability

5.1 Functional Requirements That Stated In SRS Document

Requirement Description	Class	Service	Controller	Repository
Login/Logout	User	UserService	UserController	UserRepository
Signup	User	UserService	UserController	UserRepository
Show Profile	Profile	ProfileService	ProfileController	ProfileRepository
Edit Own Profile	Profile	ProfileService	ProfileController	ProfileRepository
Manage User	User	UserService	UserController	UserRepository
Search Product	Product	ProductService	ProductController	ProductRepository
Show Product	Product	ProductService	ProductController	ProductRepository
Manage Product	Product	ProductService	ProductController	ProductRepository
Rate Product	Rate	RateService	RateController	RateRepository
Add Product to Shp. Cart	ShoppingCart	ShoppingCartService	ShoppingCartController	ShoppingCartRepository

5.2 - Non-Functional Requirements That Stated In SRS Document

5.2.1 - Usability

• **Classes / Components:** UI Components, User, Profile, ShoppingCart
• **Services / Controllers:** All user-facing services and controllers (e.g., UserService, ProfileService, ShoppingCartService, UserController, ProfileController, ShoppingCartController)
• **Implementation:** The application design emphasizes intuitive navigation, clear labeling, and user feedback mechanisms to enhance usability. Accessibility features are integrated into every component, ensuring that the application is usable for all demographics, including those with disabilities.

5.2.2 - Reliability

• **Classes / Components:** User, Product, Order, SecurityConfig, AuthenticationEntryPoint
• **Services / Controllers:** All critical services and controllers that handle data transactions and user authentication (e.g., UserService, ProductService, OrderService, SecurityConfig, AuthenticationEntryPoint)
• **Implementation:** Reliability is ensured through robust error handling, transaction management, and consistent data validation across all components. The system is designed to gracefully handle failures and ensure data integrity and availability.

5.2.3 - Performance

• **Classes / Components:** Product, ShoppingCart, User, Rate
• **Services / Controllers:** Services and controllers involved in processing high volumes of data and requests (e.g., ProductService, ShoppingCartService, RateService)
• **Implementation:** Performance optimization techniques such as efficient query handling, caching strategies, and asynchronous processing can be employed in the future. On the other hand, load balancing and resource management can be configured to handle peak loads and ensure responsive user interactions in the future.

5.2.4 - Supportability

• **Classes / Components:** SecurityConfig, AuthenticationEntryPoint, JwtTokenProvider, RefreshToken
• **Services / Controllers:** All services and controllers, with a focus on those providing security and user management (e.g., SecurityConfig, AuthenticationEntryPoint)
• **Implementation:** The application is designed for easy monitoring and troubleshooting, with comprehensive logging, clear documentation, and configurable parameters (not completed yet). In the future, support features also can include the ability to update and maintain security measures without downtime.

6. Annexes

Traceability Table

Name Of Team Member	Contributions For Delivery-4
Mert Tazeoğlu	• Creation of ‘Coding Standards’ document • Creation of ‘Software Design Description’ document • Checking mistakes in of both of documents • Creation of 1 sequence diagram for 1 critical use case • Creation of ER diagram
Asım Ateş	• Creation of 4 sequence diagrams • Creation of class diagram
Kenan Gökdeniz Acet	• Creation of 2 sequence diagrams (DIDN’T)
Emre Can Şahin	• Creation of 2 sequence diagrams (DIDN’T)
Tarık Sümer	• Creation of 2 sequence diagrams (DIDN’T)