الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
وزارة التعليم العالي والبحث العلمي
Ministry of Higher Education and Scientific Research
جامعة عمار ثليجي الاغواط
Amar Telidji University - Laghouat

كلية العلوم
FACULTY OF SCIENCE
قسم الإعلام الآلي
COMPUTER SCIENCE DEPARTMENT

MASTER THESIS
by MOHAMED LAMINE KOUISSI

Domain: Mathematics and Computer Science
Field: Computer Science
Option: Information Systems and Decision-making

Advancing Component-based Software Architecture: Towards Loosely-Coupled and Maintainable Components

---

Examination Committee :

Tahar Allaoui	Professor	Univ. Laghouat	President
Zahra Abdelhafidi	Professor	Univ. Laghouat	Examiner
Lakhdar Kachna	Professor	Univ. Laghouat	Examiner
Mohamed Lahcen Bensaad	Professor	Univ. Laghouat	Supervisor

Dissertation Submitted in Partial fulfillment of the requirements
for the Master degree in Computer Science

July 12, 2023

Acknowledgements

To begin, I express my gratitude and give thanks to Allah for providing me with the determination and ability to complete this project. I extend my sincerest appreciation to my supervisor, Mr. Bensaad Lahcen, for his unwavering support, guidance, and dedication throughout this endeavor. Additionally, I am thankful to my teachers who have imparted the knowledge necessary for the accomplishment of this work.

I offer a special acknowledgment to my family, comprising of my exceptional mother, father, sister, and brother who have been my pillars of strength through the ups and downs. Finally, I express my thanks to my friends in the Masters 2 class.

I dedicate this work to all of you.

ملخص

تلعب هندسة البرمجيات دورًا حاسمًا في نجاح المؤسسات والشركات وتمكينها من النمو والتوسع، مما يتيح لها مواجهة زيادة الطلبات وحجم البيانات بكفاءة، دون التأثير على الأداء أو تجربة المستخدم. قد تحول البنية البرمجية غير الملائمة دون نمو وتوسع المؤسسات، وتؤدي إلى ضعف أداء النظام وصعوبة الصيانة. تلبي الحلول الحالية في هندسة البرمجيات احتياجات المؤسسات الكبيرة في الغالب، مما يجعلها غير معقولة ومعقدة من الناحية المالية بالنسبة للمؤسسات الصغيرة والمتوسطة. وفي ضوء هذه التحديات، تقدم هذه الدراسة هيكلًا مبنيًا على المكونات كنهج مثالي لتحقيق بنية برمجية قابلة للصيانة والتطوير بتكلفة أقل. يساعد هذا النهج المعماري على التغلب على قيود التصميمات التقليدية ويمكن المؤسسات من التكيف والتوسع بكفاءة. وللتحقق من فعالية الهيكل المقترح، تم تطوير تطبيق تجارة إلكترونية يبرز مزايا هذه البنية في إدارة التعقيدات وأعباء العمل الثقيلة المرتبطة بهذا المجال. يهدف هذا العرض التجريبي إلى توضيح فوائد وجدوى الهيكل المقترح المبني على المكونات.

الكلمات المفتاحية: البرمجيات ، تطبيقات التجارة الإلكترونية ، هندسة البرمجيات ، هندسة البرمجيات القائمة على المكونات ، هندسة البرمجيات القابلة للتوسع.

Abstract

Software engineering plays a crucial role in the success of organizations and companies, enabling them to grow and expand effectively, while efficiently handling increasing user demands and data volumes without compromising performance or user experience. However, inadequate software architecture can hinder the growth and scalability of organizations, leading to system inefficiency and maintenance challenges. Current software engineering solutions mostly cater to larger enterprises, making them financially impractical and complex for small and medium-sized businesses. In light of these challenges, this study proposes a component-based structure as an ideal approach to achieve cost-effective, maintainable, and scalable software architecture. This architectural approach overcomes the limitations of traditional designs and enables efficient adaptation and expansion for organizations. To validate the effectiveness of the proposed structure, an e-commerce application was developed, highlighting the advantages of this architecture in managing complexities and heavy workloads associated with this domain. This proof-of-concept aims to demonstrate the feasibility and benefits of the proposed component-based structure.

Keywords: Software, E-commerce Applications, Software Architecture, Component-based Architecture, Scalable Software Architecture.

Contents

List of Figures

Introduction

Software architecture refers to the underlying structure and design of a software system. It encompasses the organization of components, the interaction between modules, and the overall behavior of the system. In the context of organizations and businesses, software architecture plays a crucial role in scaling their operations, particularly for those that rely on software as an e-commerce platform.

One key aspect of software architecture is its ability to enable scalability. As businesses grow, their e-commerce platforms need to handle increased user traffic, transactions, and data volume. An effective software architecture ensures that the system can accommodate these growing demands without sacrificing performance or user experience.

The reliance of organizations and businesses on software as their primary source of income makes software architecture a critical factor in their success. However, the presence of bad software architecture can have detrimental effects on these entities. It can impede their growth, hinder scalability, and lead to inefficiencies in system performance and maintenance. Additionally, the current solutions available for software architecture often cater to big companies, posing challenges for smaller organizations due to the high cost and complexity associated with implementation and maintenance.

One of the main issues arising from bad software architecture is its impact on scalability. As businesses strive to expand their operations and accommodate increasing user demands, an inadequate architecture can limit their ability to scale effectively. The lack of flexibility and modularity within the system can hinder the integration of new features and functionalities, preventing businesses from adapting to evolving market requirements.

Moreover, the current solutions for software architecture predominantly target larger enterprises, further exacerbating the challenges faced by smaller organizations. The high costs associated with acquiring and implementing complex software architectures make them financially inaccessible for many businesses. Additionally, the expertise required to maintain and manage such architectures may not be readily available, particularly for organizations without dedicated IT departments.

To address these challenges, there is a need for a high-effective software architecture that is maintainable, scalable, and easy to implement. A proposed solution is the adoption of a component-based architecture. This approach allows for easier maintenance and scalability of applications as businesses grow and evolve. By employing a modular and flexible architecture, businesses can incorporate new features and functionality into their e-commerce platforms without the need for a complete system overhaul. With a modular approach, businesses can easily maintain, upgrade, and customize specific components without disrupting the entire system. This flexibility enables businesses to adapt to changing market demands and customer needs,

facilitating growth and innovation.

Furthermore, a component-based approach offers the advantage of empowering businesses without a dedicated IT department to develop and maintain their own e-commerce platform. This opens up opportunities for smaller organizations to exercise greater control and independence over their digital presence.

To validate the effectiveness of the proposed architecture, an e-commerce application will be implemented as a demonstration. The choice of an e-commerce platform is particularly relevant as it represents a software type that often faces challenges arising from high complexity and heavy loads. By showcasing the benefits of the proposed architecture within this specific context, the aim is to emphasize its efficacy in addressing the identified problems and providing a scalable and maintainable solution.

Thesis Outline

Background and Related Works: In this chapter, we provide a comprehensive introduction to Software Architecture and e-commerce web applications. We discuss the challenges associated with poorly designed e-commerce web applications and present relevant works in the field, highlighting their limitations and drawbacks.

Modular Architecture: This chapter we introduces our own modular architecture. We delve into the various components of our architecture and explain how it addresses the challenges faced by businesses due to inadequate software design. By leveraging our modular architecture, businesses can overcome these issues effectively.

Conceptual Design: In this section, we provide detailed insights into the functionalities of our platform. We also discuss the tools and techniques employed during the design and development phase. To enhance understanding, we incorporate diagrams that further elucidate the conceptualization of our work.

Implementation: This chapter focuses on the implementation details of our system. We describe the specific tools and platforms chosen for the implementation and provide the reasoning behind these choices. Additionally, we explain the functionality of our system and include screenshots to enhance comprehension and illustrate the system's capabilities.

General Conclusion: Finally, we conclude the thesis by summarizing the project's different stages and outlining potential future directions for its further development.

Chapter 1

Background And Related Works

1.1 Background

1.1.1 Software Architecture

Software architecture serves as the organizational framework for a system, encompassing its components, their interactions, the surrounding environment, and the underlying design principles. It also takes into account the system's potential evolution over time.

The primary objective of software architecture is to fulfill specific missions or objectives while ensuring compatibility with other tools and devices. The behavior and structure of the software have a profound impact on critical decisions, necessitating careful construction and rendering to achieve optimal outcomes.

By revealing the system's structure while concealing certain implementation details, software architecture enables a focus on relationships and the interactions among elements and components. This alignment with the principles of software engineering ensures a comprehensive approach to system design and development.[5]

1.1.2 E-commerce Web Application

An e-commerce web application is a software application that allows businesses to sell their products or services online through a website. It typically includes features such as a product catalog, shopping cart, payment gateway, and order management system. E-commerce web applications can be customized to meet the specific needs of a business, and they can be developed using various programming languages and frameworks. They are designed to be user-friendly and provide a seamless shopping experience for customers, while also offering efficient and effective tools for businesses to manage their online sales.

1.1.3 Challenges of Poorly Designed E-commerce Web Applications

E-commerce web applications that are poorly designed can face several challenges that hinder their performance and profitability. One major challenge is a high bounce rate, where visitors quickly leave the site without making a purchase. This can be caused by a slow loading time, a confusing layout, or a lack of user-friendly features.

Another challenge is a lack of scalability, where the website cannot handle an increase in traffic or sales. This can lead to website crashes, slow loading times, and frustrated customers who may turn to a competitor's website instead. Poorly designed e-commerce web applications can also lead to high maintenance costs, as developers may struggle to make changes or fix issues due to a confusing or disorganized codebase.

1.2 Related Works

1.2.1 Monolithic Architecture

"A monolithic architecture is a traditional model of a software program, which is built as a unified unit that is self-contained and independent from other applications. The word "monolith" is often attributed to something significant and glacial, which is not far from the truth of a monolith architecture for software design. A monolithic architecture is a singular, large computing network with one code base that couples all of the business concerns together. Making a change to this sort of application requires updating the entire stack by accessing the code base and building and deploying an updated version of the service-side interface. This makes updates restrictive and time-consuming."[10]

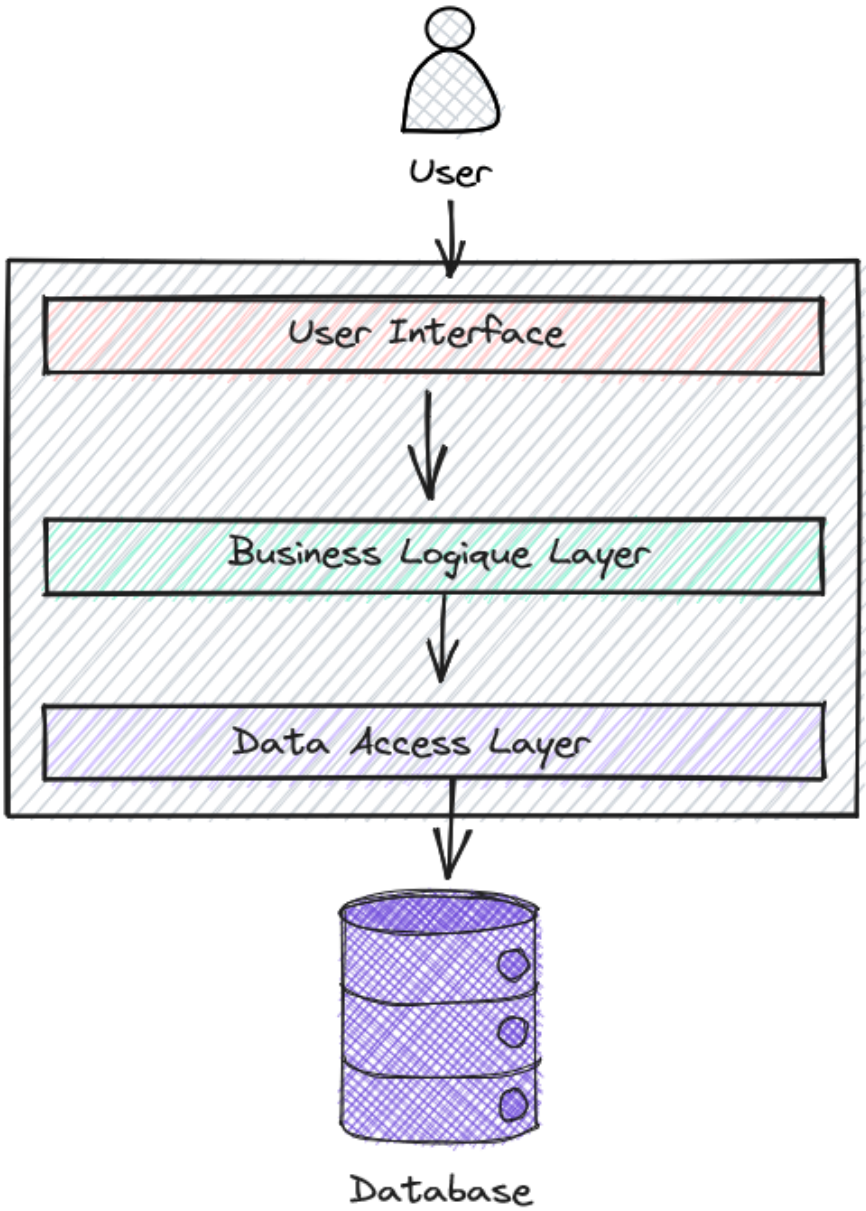

Figure 1.1: Monolithic Architecture Diagram

1.2.2 The Clean Architecture

The Clean Architecture, proposed by Robert C. Martin, is an architectural approach that aims to achieve separation of concerns in software systems. It draws inspiration from various other architectural patterns, such as Hexagonal Architecture, Onion Architecture, Screaming Architecture, DCI, and BCE.[18]

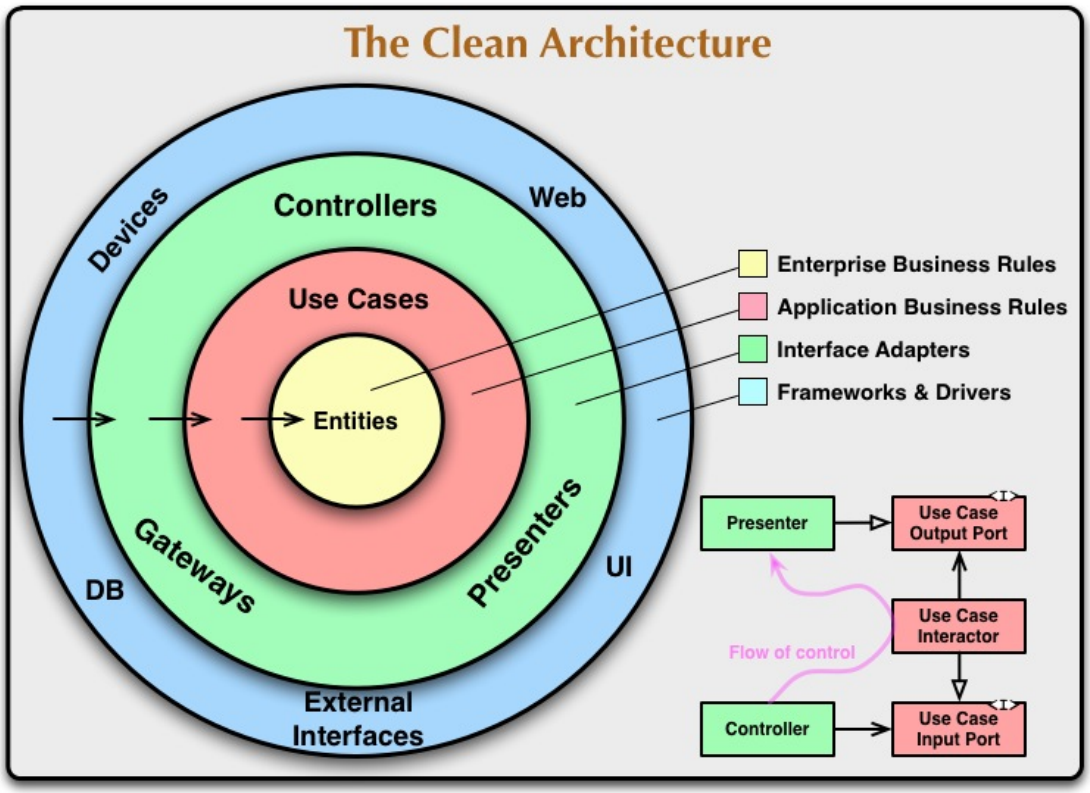

Figure 1.2: The Clean Architecture Diagram

1.2.3 Microservices Architecture

"Microservices are an architectural and organizational approach to software development where software is composed of small independent services that communicate over well-defined APIs. These services are owned by small, self-contained teams."[25] Some of the disadvantages of this architecture are the high cost of implementing and maintaining communication between microservices, the complexity of the infrastructure, and the high costs of network connections.

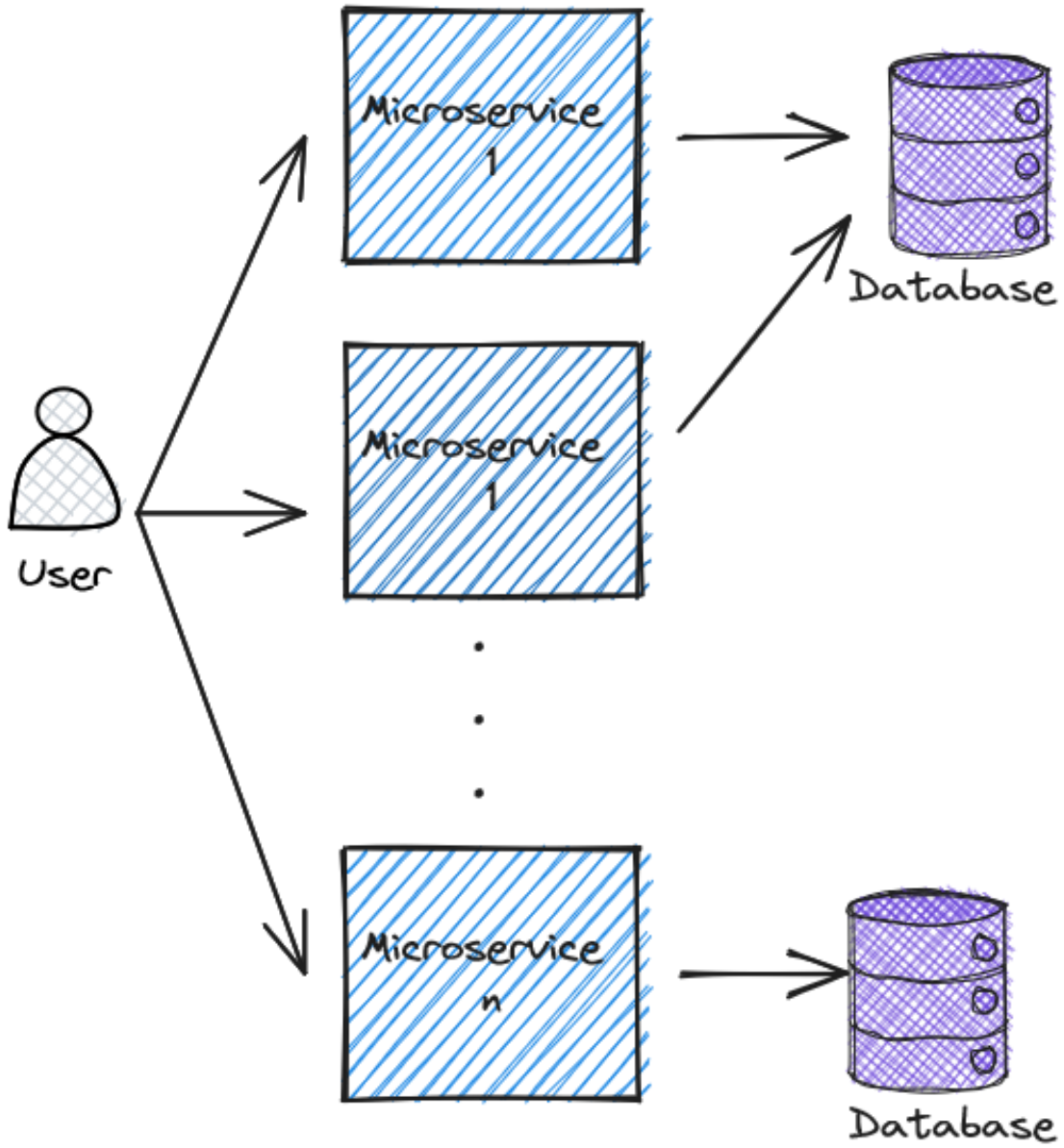

Figure 1.3: Microservices Architecture Diagram

1.2.4 condor.dz

Condor.dz is the official e-commerce website of Condor Electronics, an Algerian electronics and home appliance brand. The website serves as an online catalog

where customers can browse and view products from the brand, but they cannot purchase them directly on the website.

Drawbacks of condor.dz

- Slow response time: One of the major drawbacks of condor.dz is its slow response time which can lead to a frustrating user experience and potential loss of customers.
- Lack of functionalities: condor.dz may have limited functionalities, which can restrict users from accessing certain features or services. This can limit the overall user experience and hinder users from fully utilizing the platform.
- The website may have slow loading speeds due to high traffic, which could negatively impact the user experience.

Figure 1.4: Landing page of Condor e-commerce web application

1.2.5 tassiliairlines.dz

Tassili Airlines is an Algerian airline that operates both domestic and international flights. Its e-commerce website, tassiliairlines.dz, allows customers to search and book flights, manage their bookings, and access other travel services.

Drawbacks of tassiliairlines.dz

- Poor user interface design that can make navigation and booking a bit challenging.
- Slow performance and page loading speed, which can lead to a frustrating user experience.
- Lack of functionalities.

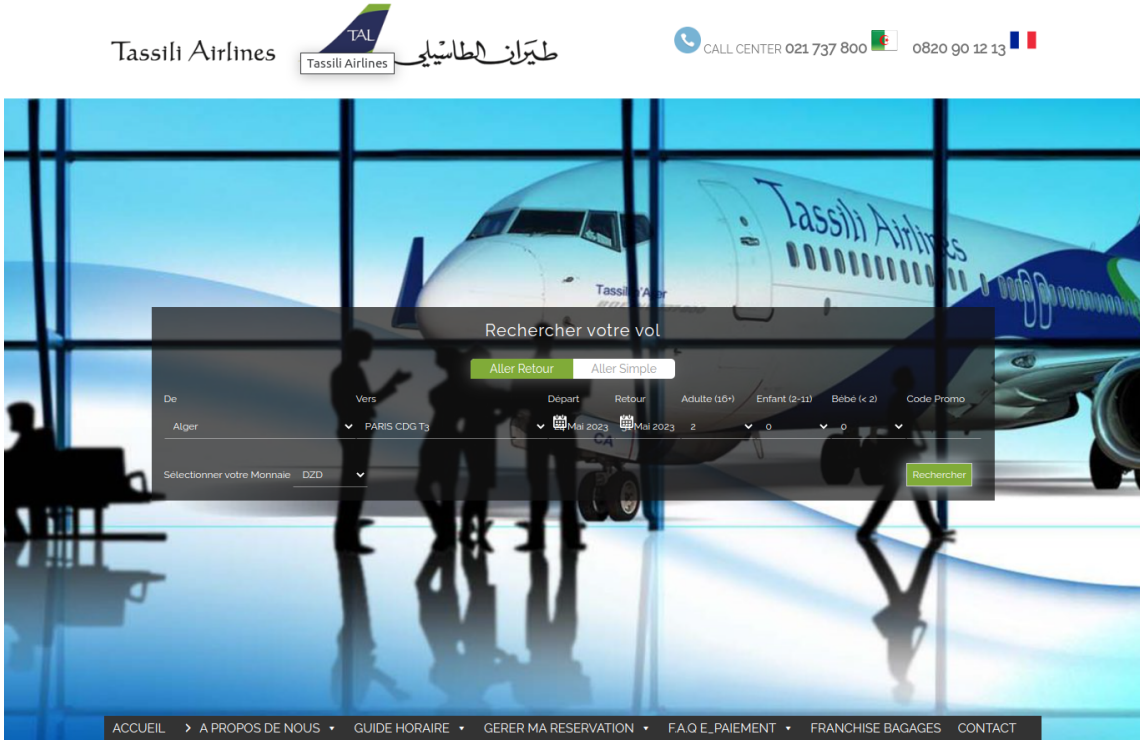

Figure 1.5: Landing page of Tassili Airlines e-commerce web application

1.3 conclusion

Conclusion: This chapter presents a thorough examination of Software Architecture and its application in E-commerce web applications. We have highlighted the challenges faced by poorly designed E-commerce web applications, emphasizing the importance of robust design and efficient execution. Furthermore, we have explored various Software Architecture approaches and discussed their limitations. In addition, we have explored existing E-commerce web applications, highlighting the negative impact of performance bottlenecks and missing essential functionalities on business revenue and reputation. By addressing these aspects, we emphasize the critical role of a well-designed and effectively executed architecture in the success of E-commerce ventures.

Chapter 2

Modular Architecture

2.1 Architectural Patterns

Architectural patterns are general, reusable solutions to commonly occurring problems in architectural patterns that encompass comprehensive and adaptable solutions that address frequently encountered challenges in software architecture. They exert a profound and far-reaching influence on the underlying codebase. Their impact manifests both horizontally and vertically. Horizontal impact pertains to the structural organization of code within distinct layers, while vertical impact encompasses the intricate flow of requests as they traverse from outer layers to inner layers and vice versa. Prominent among these architectural patterns are MVC (Model-View-Controller), MVP (Model-View-Presenter), and MVVM (Model-View-ViewModel). These patterns are widely employed in software development due to their effectiveness in tackling common architectural issues.[3]

2.1.1 Terminology: Foundational Concepts for Exploring Each Pattern

Model: "stores data and communicates directly with the database. The model is the part that represents your data and application logic. It defines the business rules that manage data handling, modification, or processing."[9]

The model encapsulates both the data and the business logic of the application. It represents and manages the underlying data, defining rules for its manipulation, processing, and interaction with the database.[9]

View: "displays the model's data and is responsible for the data's representation in the user interface.

ViewModel: "is exclusive to the MVVM pattern. This is an abstraction of the view layer and also acts as a wrapper for the model data."[9]

Controller: "is the component that integrates the view and model."[9]

Presenter: "is a component that only exists in the MVP model. The presenter gets the input from the view component and processes the data with the help of the model."[9]

2.1.2 Model-View-Controller Architectural Pattern

The Model-View-Controller (MVC) architectural pattern, introduced in the 1970s, is widely used in web applications. It promotes the principle of Separation of Concerns (SoC), simplifying testing, maintenance, and development. In MVC, the model is independent and separate from the view and controller. Changes from the view or controller are communicated to the model's observers. The controller acts as an intermediary, facilitating the connections between the model and the appropriate view.[9]

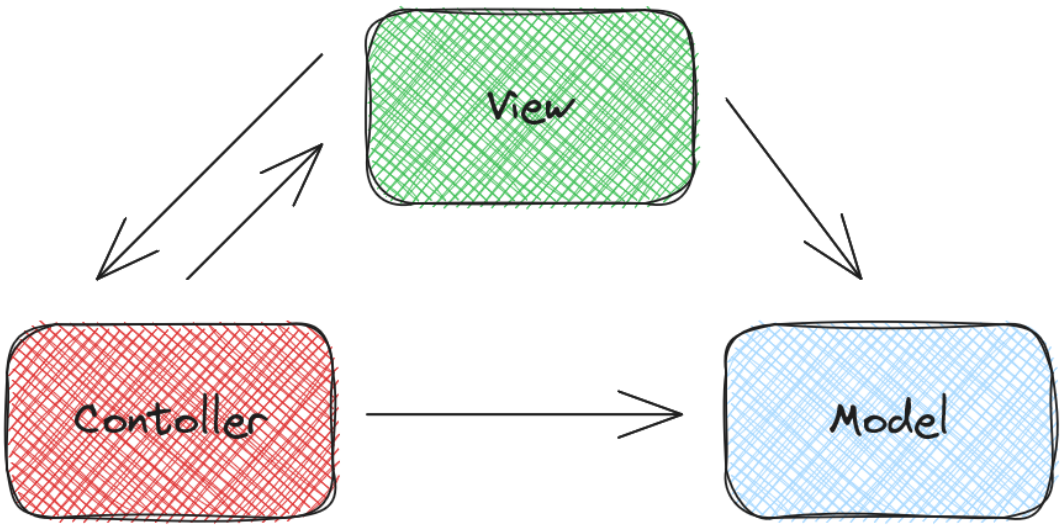

Figure 2.1: Model-View-Controller Architectural Pattern

2.1.3 Model-View-Presenter Architectural Pattern

The Model-View-Presenter (MVP) pattern is an architectural design pattern that extends the concepts of the Model-View-Controller (MVC) pattern by introducing a separate presenter component instead of a controller. MVP offers benefits such as improved support for mocking the view during testing. In MVP, the presenter acts as an intermediary, handling the presentation logic and allowing for easy isolation and mocking of the view. The key feature of MVP is the preservation of independence between the view and the presenter, promoting loose coupling and facilitating maintainability and extensibility in the application's architecture.[9]

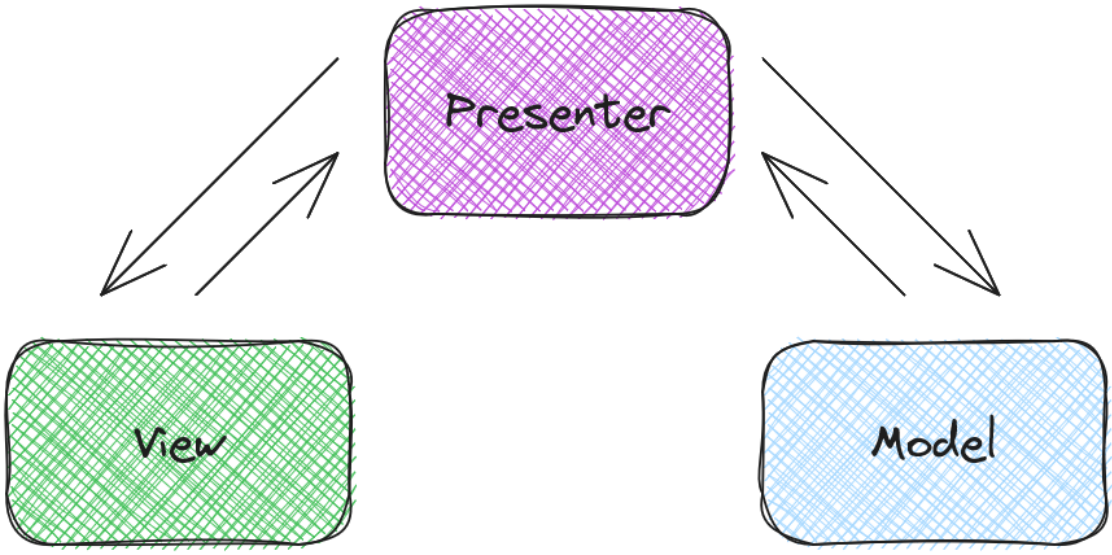

Figure 2.2: Model-View-Presenter Architectural Pattern

2.1.4 Model-View-ViewModel Architectural Pattern

The Model-View-ViewModel (MVVM) architectural pattern emphasizes the separation of domain logic and the presentation layer in an application. It enables bidirectional data binding between the view and the viewmodel, ensuring smooth synchronization of data updates. MVVM achieves decoupling between the view and the model, ensuring that changes in one do not directly impact the other. Instead, the viewmodel acts as an intermediary, managing their interaction. A notable benefit of MVVM is the ability to conduct unit testing on the logic independently from the view, thanks to the viewmodel. This promotes enhanced testability and maintainability of the application.[9]

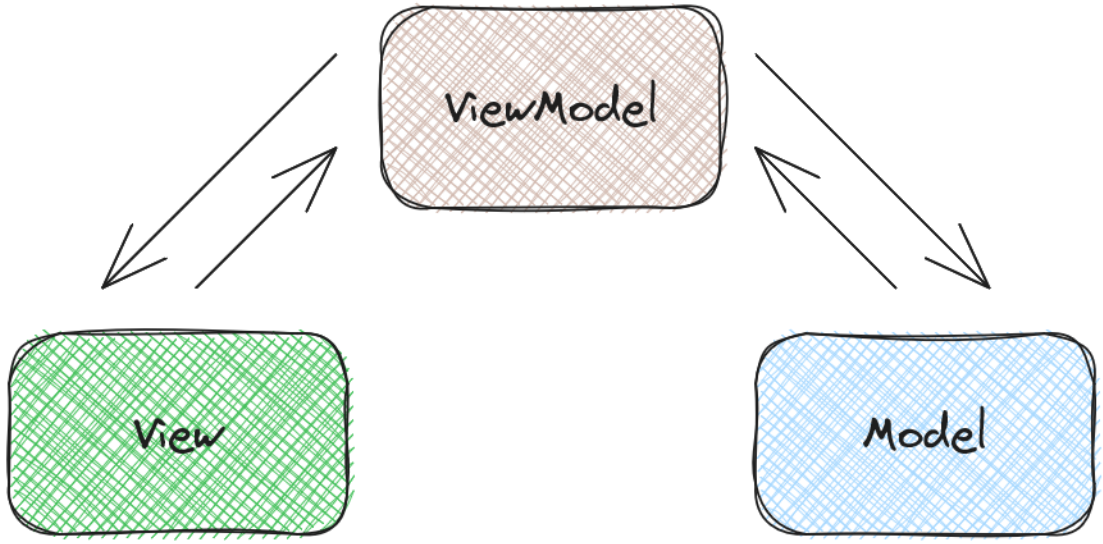

Figure 2.3: Model-View-ViewModel Architectural Pattern

2.2 SOLID Principles

"SOLID is an acronym for the first five object-oriented design (OOD) principles by Robert C. Martin (also known as Uncle Bob). These principles establish practices that lend to developing software with considerations for maintaining and extending as the project grows. Adopting these practices can also contribute to avoiding code smells, refactoring code, and Agile or Adaptive software development."[20] SOLID stands for:

- S - Single-responsibility Principle
- L - Liskov Substitution Principle
- I - Interface Segregation Principle
- D - Dependency Inversion Principle

2.2.1 The Single Responsibility Principle

this principle states that a class should only have one responsibility. Furthermore, it should only have one reason to change."[19]

2.2.2 Open-Closed Principle

"the open-closed principle. Simply put, classes should be open for extension but closed for modification. In doing so, we stop ourselves from modifying existing code and causing potential new bugs"[19]

2.2.3 Liskov Substitution Principle

"Liskov substitution, which is arguably the most complex of the five principles. Simply put, if class A is a subtype of class B, we should be able to replace B with A without disrupting the behavior of our program."[19]

2.2.4 Interface Segregation Principle

"The I in SOLID stands for interface segregation, and it simply means that larger interfaces should be split into smaller ones. By doing so, we can ensure that implementing classes only need to be concerned about the methods that are of interest to them."[19]

2.2.5 Dependency Inversion Principle

"The principle of dependency inversion refers to the decoupling of software modules. This way, instead of high-level modules depending on low-level modules, both will depend on abstractions."[19]

2.3 Module Level Design

To architect our application modules with high maintainability, scalability, and clean structure, we have combined The Clean Architecture by Robert C. Martin, SOLID Principles, and various architectural patterns.

Drawing inspiration from The Clean Architecture, we have separated our application into distinct layers, such as entities, use cases. This separation promotes loose coupling and allows each layer to focus on its specific responsibilities while maintaining independence from external dependencies.

Furthermore, we have embraced SOLID Principles to ensure our modules are designed with a strong foundation. Each module adheres to the Single Responsibility Principle, which guarantees that every module has a clear and specific purpose. This enhances maintainability by making it easier to understand, modify, and test individual modules.

Additionally, we employ the Open-Closed Principle, encouraging modules to be open for extension but closed for modification. By designing modules that can be extended without requiring changes to their existing code, we ensure scalability and minimize the risk of introducing bugs or unintended side effects.

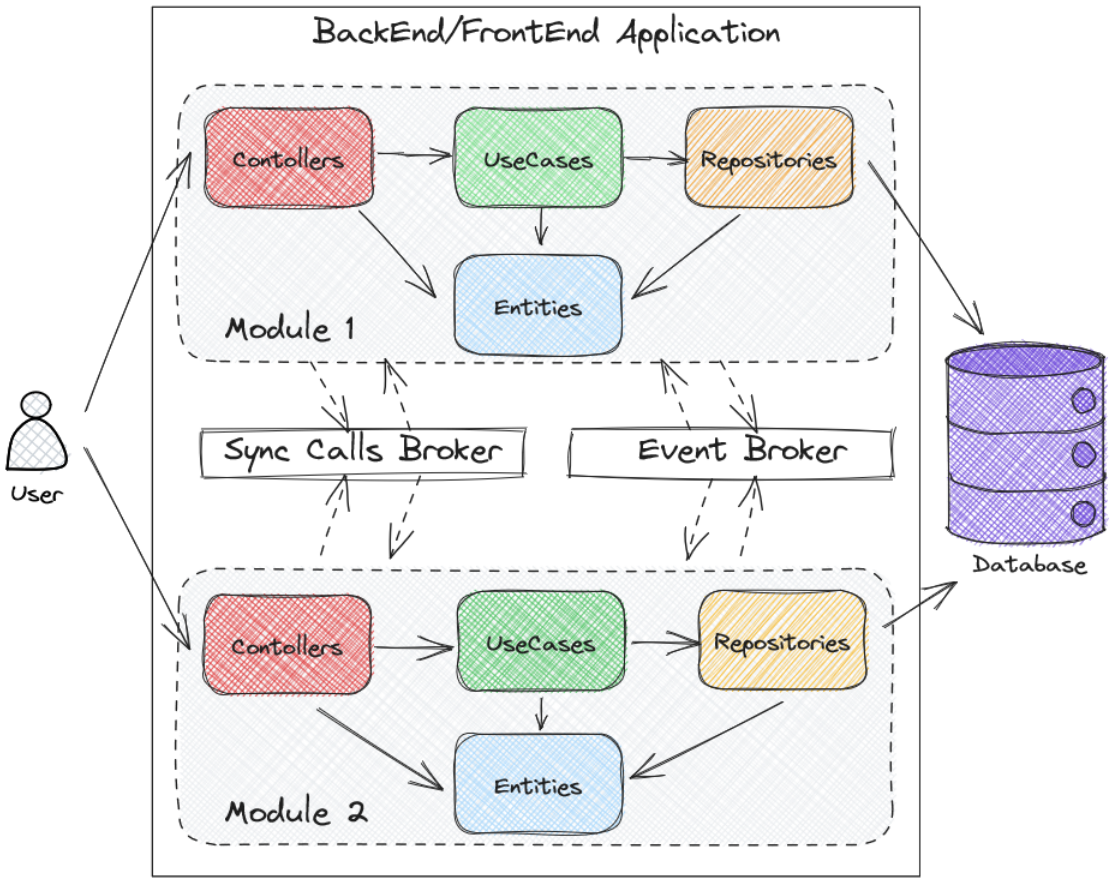

Figure 2.4: Module Level Diagram

To further enhance the architecture, we have incorporated various architectural patterns that align with our application's needs. For example, we may use the Repository pattern to abstract data access and decouple it from business logic. This promotes code reusability and flexibility, making it easier to switch between different data sources or databases.

By combining these principles and patterns, our application modules benefit from maintainability, scalability, and clean structure. Changes or enhancements can be made to individual modules without affecting the entire application, making it easier to maintain and evolve over time. The architecture's clear separation of concerns and adherence to SOLID Principles ensure a clean and robust design, while the use of appropriate architectural patterns adds further structure and flexibility to our application.

2.3.1 Entities

These encapsulate enterprise-wide business rules and represent the most general and high-level concepts in the system. They can be implemented as objects with methods or as a collection of data structures and functions. Entities should remain unaffected by changes to specific applications or external factors like UI or security.[18]

2.3.2 Controllers

"controller captures the user requests. First, it will validate the data present inside the request. If anything is invalid, it returns an error response. If everything is valid inside the request, it will call the usecase layer to perform an operation."[26]

2.3.3 Use Cases

This layer contains application-specific business rules and orchestrates the flow of data to and from entities. Use cases leverage enterprise-wide business rules defined in the entities to accomplish specific goals. Changes to the operation of the application may impact the use cases, but not the entities or external factors.[18]

2.3.4 Repositories

"The repository is the dependency of the usecase. The usecase layer asks the repository to perform database operations.

The repository layer is free to choose any database, in fact, it can call any other independent services based on the requirement.

In the project, the repository layer makes the database query for performing operations asked by the usecase layer."[26]

2.3.5 Sync Calls Broker

The Sync Calls Broker acts as an intermediary between different modules within the application, facilitating communication and coordination between them.

When a source module requires a specific functionality offered by a destination module, it sends a message to the broker requesting that functionality. The broker then validates the request message to ensure that the destination module indeed offers the requested functionality. Once the validation is successful, the broker redirects the message to the corresponding destination module.

The destination module processes the message and generates a result. The result is then sent back to the broker, which in turn sends it back to the source module that made the initial request.

This synchronous communication pattern helps establish a reliable and controlled interaction between modules within the application, ensuring that the requested functionality is available and that the results are returned appropriately.

2.3.6 Event Broker

The event broker acts as an internal component that serves as an event hub. It facilitates communication and coordination between different modules or components within the system, which produces and consumes events.

When an event occurs in the source module that holds value for the destination module in the future, the event broker captures and stores that event. It acts as a repository or buffer for events, ensuring they are available for consumption by interested modules.

Later, when the destination module requires a specific event, it can simply request that event from the broker. The broker retrieves the requested event from its storage and delivers it to the destination module.

This asynchronous communication pattern, based on events and an event-driven system, allows modules or components to operate independently and asynchronously. It decouples the sender and receiver, enabling loose coupling and flexibility in the system's architecture.

2.4 Conclusion

This chapter provides a comprehensive exploration of Architectural patterns, highlighting their distinctions and unique characteristics. Additionally, we delve into the SOLID principles and their relevance in developing loosely coupled applications. Moreover, we introduce our modular architecture as a solution to the challenges faced by poorly designed applications.

Chapter 3

Conceptual Design

3.1 System Architecture Overview

Before delving deeper, it is essential to grasp the high-level perspective of our system's design and the specific actor(user) types that engage with it. Our web application is divided into two distinct components: the backend web app, which is accessible by backend actors (users interacting with the backend web app), and the frontend web app, which is accessible by frontendy actors (users interacting with the frontend web app). Both of these applications are interconnected with a Relational Database Management System (RDMS) and a cloud storage services, facilitating the storage of static data such as images and videos for optimized performance. It is important to note that these two applications are entirely separate from each other, maintaining their individual autonomy and functionalities.

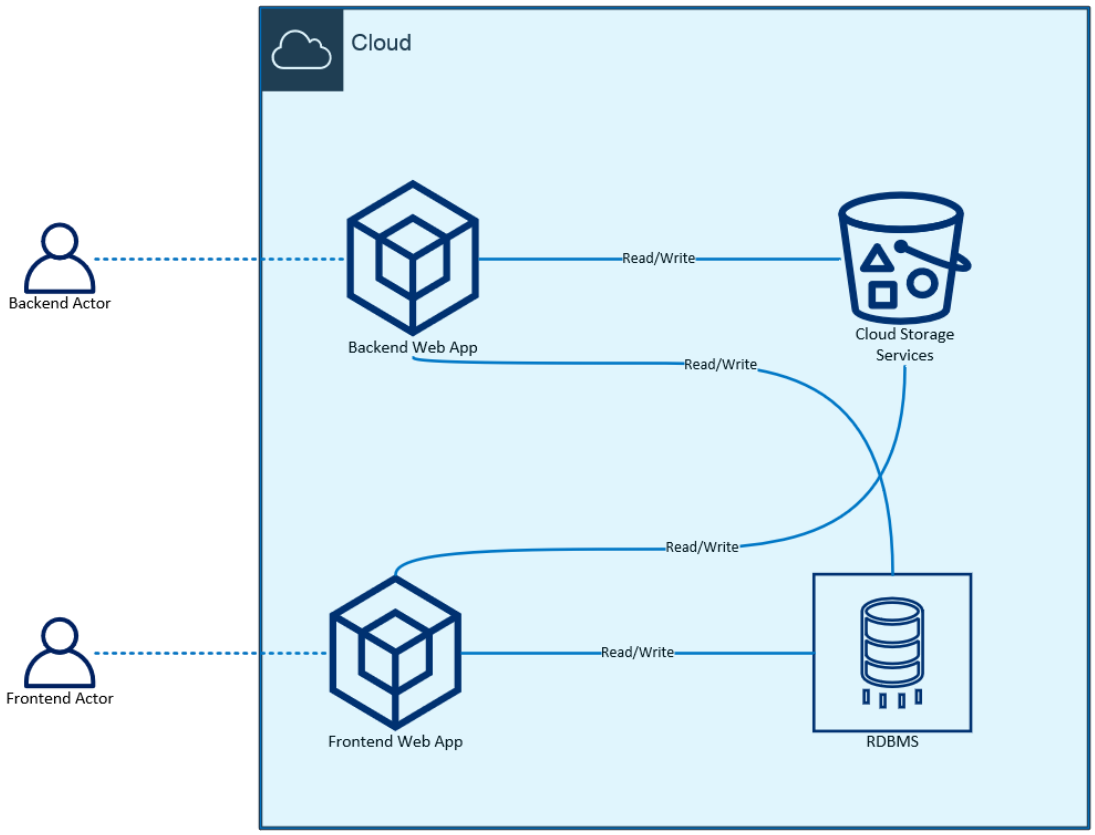

Figure 3.1: System Architecture Overview

3.2 Key Actors

The actors within our system can be categorized into two distinct groups: backend actors and frontend actors.

Backend actors consist of the following roles: editor, salesperson, assistant, shipper, and admin. These individuals primarily engage with the backend functionalities of our backend web app, contributing to activities related to content management, sales, assistance, shipping, and administrative tasks. For example, the editor is

responsible for managing and updating content, while the salesperson handles customer transactions and inquiries. The assistant supports various backend operations, and the shipper manages the shipping logistics. The admin oversees the overall system administration and ensures smooth functioning.

Frontend Actors on the other hand, include two main roles: visitor and customer. Visitors are individuals who access our system's frontend web app. They can browse through products, categories, and brands, gathering information about the available offerings. However, since visitors do not have an account, they are unable to make purchases or access personalized features.

Customers, on the other hand, have registered accounts within our frontend web app. As customers, they can engage in activities such as browsing products, categories, and brands, similar to visitors. Additionally, customers have the privilege of making purchases using their registered accounts. They can select items, add them to their shopping cart, and proceed with the checkout process. Furthermore, customers can enjoy the benefits of accessing personalized features tailored to their preferences and previous interactions, enhancing their overall experience within the system.

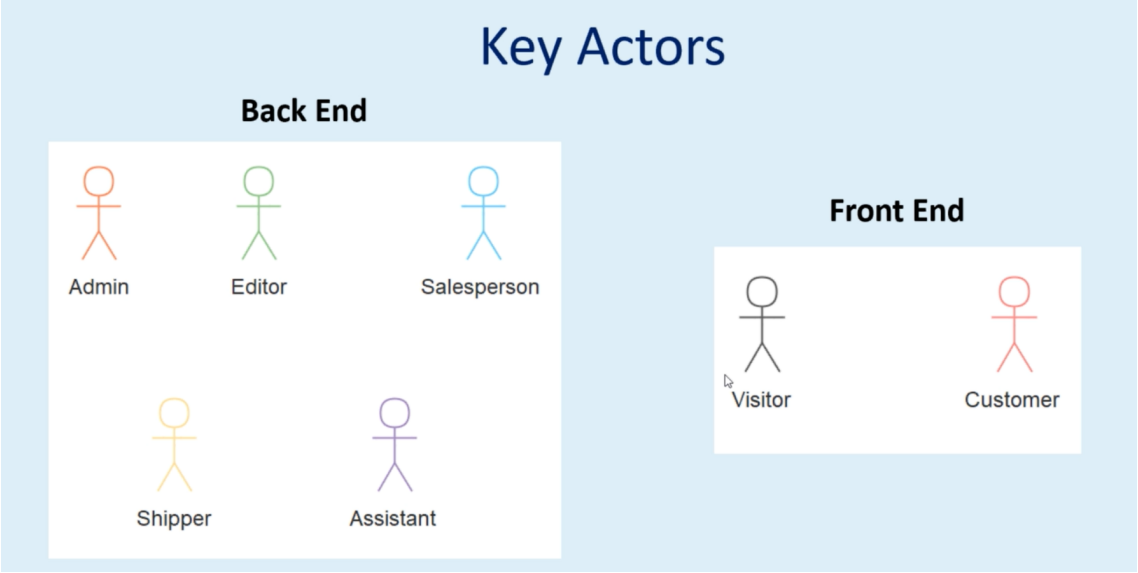

Figure 3.2: Key Actors

3.3 Backend Application Modules

In this section, we provide an overview of the modular structure employed in the Backend application of our system. The modular design facilitates the organization and separation of functionalities into discrete components, enhancing the system's flexibility, scalability, and maintainability. This section outlines the key modules implemented in the Backend application.

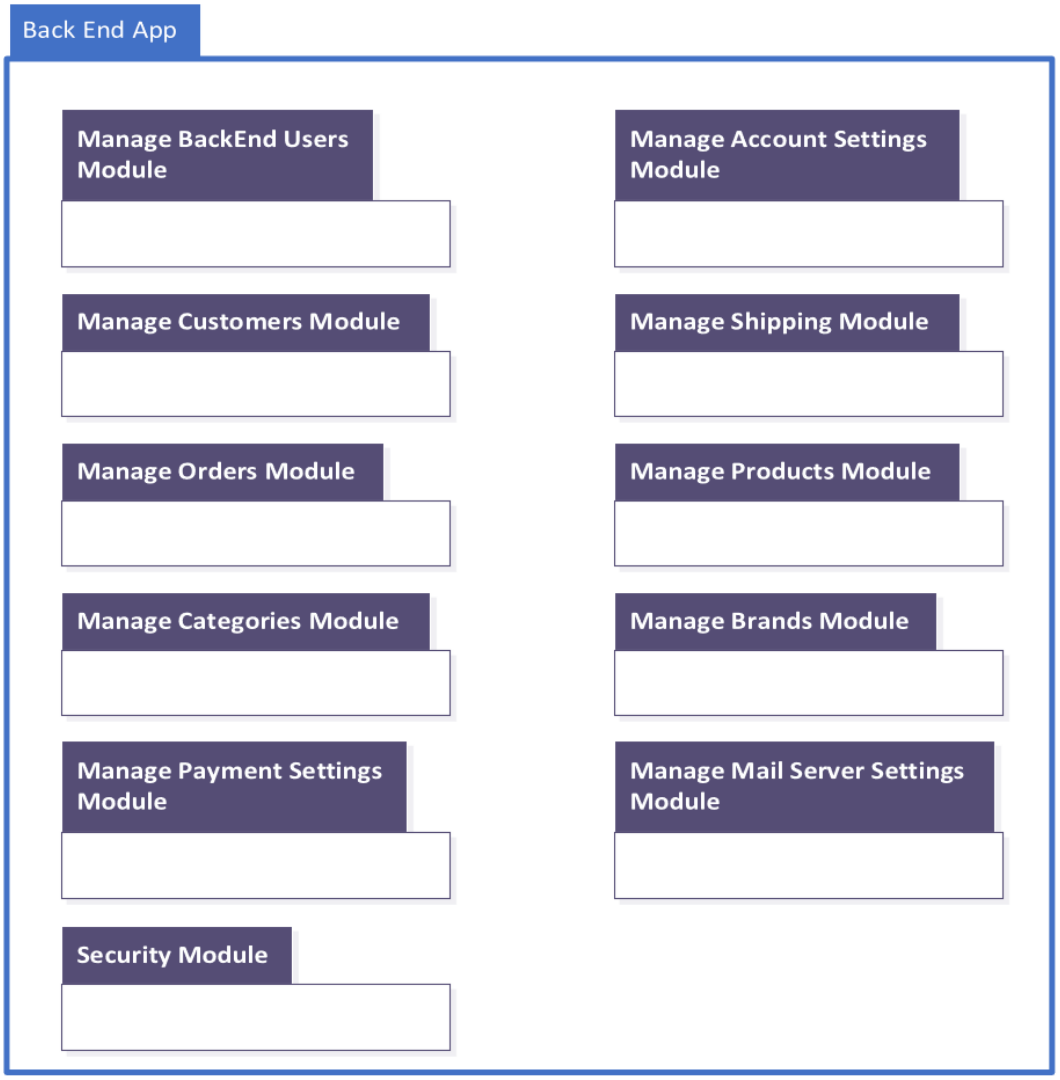

Figure 3.3: The Modular Design of the Backend Application

3.4 Frontend Application Modules

In this section, we provide an overview of the modular structure employed in the Frontend application of our system. The modular design facilitates the organization and separation of functionalities into discrete components, enhancing the system's flexibility, scalability, and maintainability. This section outlines the key modules implemented in the Frontend application.

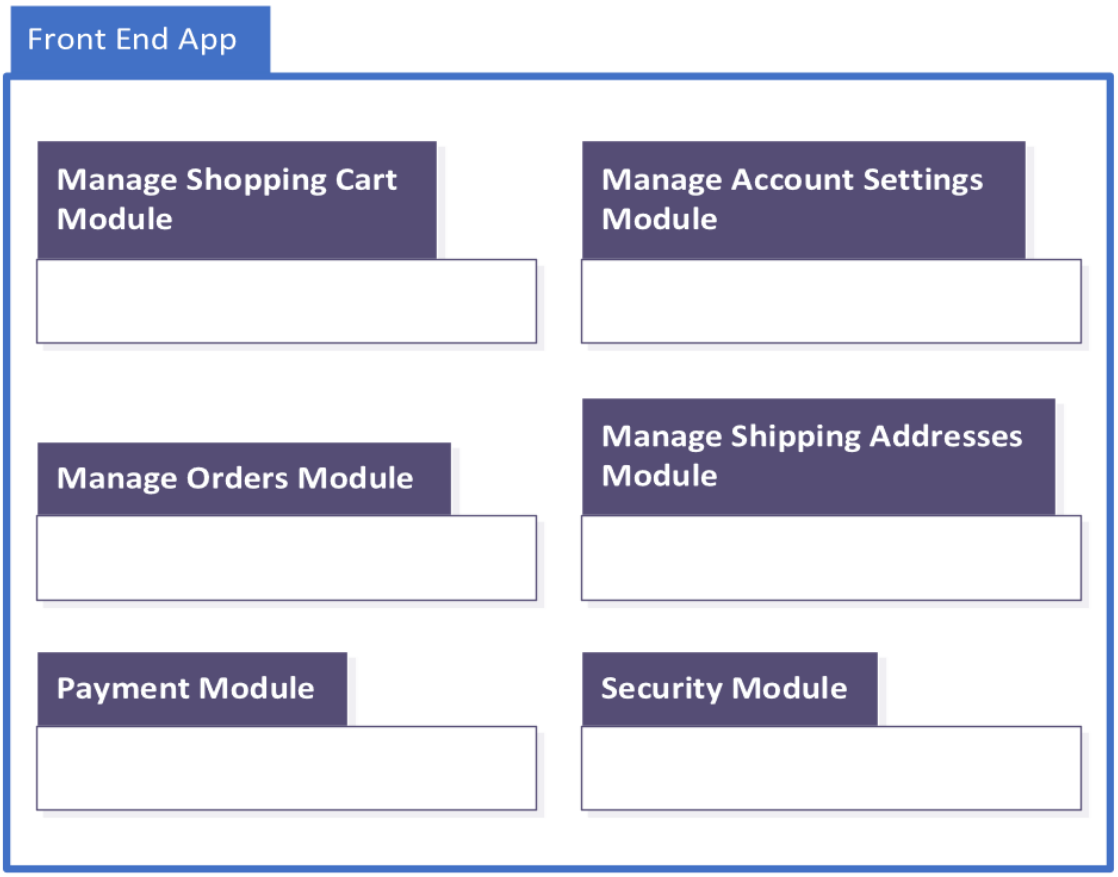

Figure 3.4: The Modular Design of the Frontend Application

3.5 System Blueprint

In the forthcoming sections, we present the blueprint of our work in the form of conceptual diagrams. For this purpose, we opted to utilize the Unified Modeling Language (UML) as it offers a comprehensive set of diagrams that are highly suitable for our project. One of the primary reasons for selecting UML is its efficacy in facilitating the translation of these diagrams into executable code, making the development process more streamlined and efficient in our particular case. By leveraging UML, we can effectively bridge the gap between the visual representation of our system and its practical implementation through code.

3.6 Unified Modeling Language

The creation of the Unified Modeling Language (UML) aimed to establish a universally accepted and comprehensive visual modeling language that is semantically and syntactically robust, specifically designed for the architecture, design, and implementation of intricate software systems.[17]

UML encompasses two primary categories of diagrams: structural diagrams and behavioral diagrams. Structural diagrams focus on illustrating the static elements and relationships within a system, providing insights into its overall composition. On the other hand, behavioral diagrams are employed to depict the dynamic aspects of a system, enabling the representation of its interactive and evolving behaviors over time.[17]

3.6.1 Structural UML diagrams

Class Diagram The Class Diagram is a fundamental UML diagram used in object-oriented solutions. It represents classes, their attributes, operations, and relationships within a system. Class diagrams are especially helpful for visualizing large systems by grouping classes together.[17]

Component Diagram Shows structural relationships among system elements, used in complex systems with multiple components. Emphasizes component interaction through interfaces.[17]

Composite Structure Diagram Reveals internal structure of a class, displaying components, relationships, and collaborations within it.[17]

Deployment Diagram Illustrates hardware and software configuration of a system, especially useful for showing deployment across multiple machines with different configurations.[17]

Object Diagram Depicts object relationships using real-world examples, providing a snapshot of the system at a specific moment.[17]

Package Diagram Highlights package dependencies in a system, representing different levels of architecture and communication mechanisms between them.[17]

3.6.2 Behavioral UML diagrams

Use Case Diagram A Use Case Diagram is utilized to represent a specific functionality of a system, illustrating the relationships and interactions between functionalities and their internal/external controllers (actors).[17]

Timing Diagram A Timing Diagram is employed to depict the behavior of objects within a defined time frame. It captures the interactions among objects during that particular time frame, offering simplicity for diagrams involving a single object and more complexity for diagrams involving multiple objects.[17]

State Diagram A State Diagram is akin to an activity diagram and serves to describe the behavior of objects that exhibit varying behaviors depending on their current state.[17]

Sequence Diagram A Sequence Diagram showcases the interactions between objects and their chronological order of occurrence. It represents the object interactions within a specific scenario.[17]

Communication Diagram A Communication Diagram, similar to sequence diagrams, focuses on the exchange of messages between objects. It presents the same information that can be conveyed through a sequence diagram but with a different emphasis on object-to-object communication.[17]

Activity Diagrams Activity Diagrams visually illustrate business or operational workflows to demonstrate the activities of different parts or components within a system. They serve as an alternative to State Machine diagrams.[17]

Interaction Overview Diagram An Interaction Overview Diagram presents the sequence in which the seven types of interaction diagrams act, providing an overview of their interactions.[17]

3.7 Functional Requirements

Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks[1]

We specifically utilize Use Case Diagrams within the UML framework to document our functional requirements. These diagrams capture the interactions between various actors and the system, illustrating the specific functionalities that need to be implemented to satisfy the user requirements effectively.[17]

3.7.1 Use Case Diagrams

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors.[16] An effective use case diagram can help your team discuss and represent:

- Scenarios in which your system or application interacts with people, organizations, or external systems.[16]
- Goals that your system or application helps those entities (known as actors) achieve.[16]
- The scope of your system.[16]

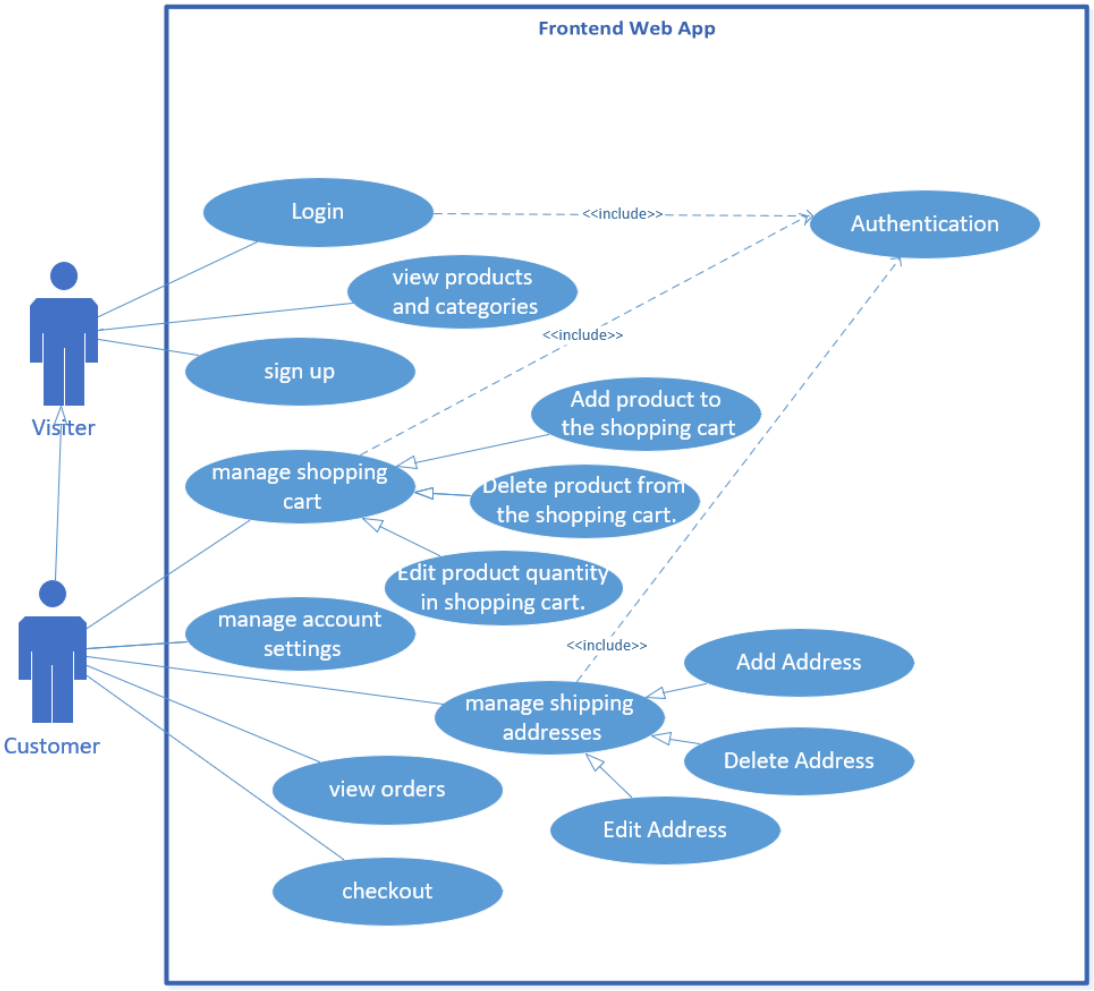

Figure 3.5: Use Case Diagram For Frontend Web Application

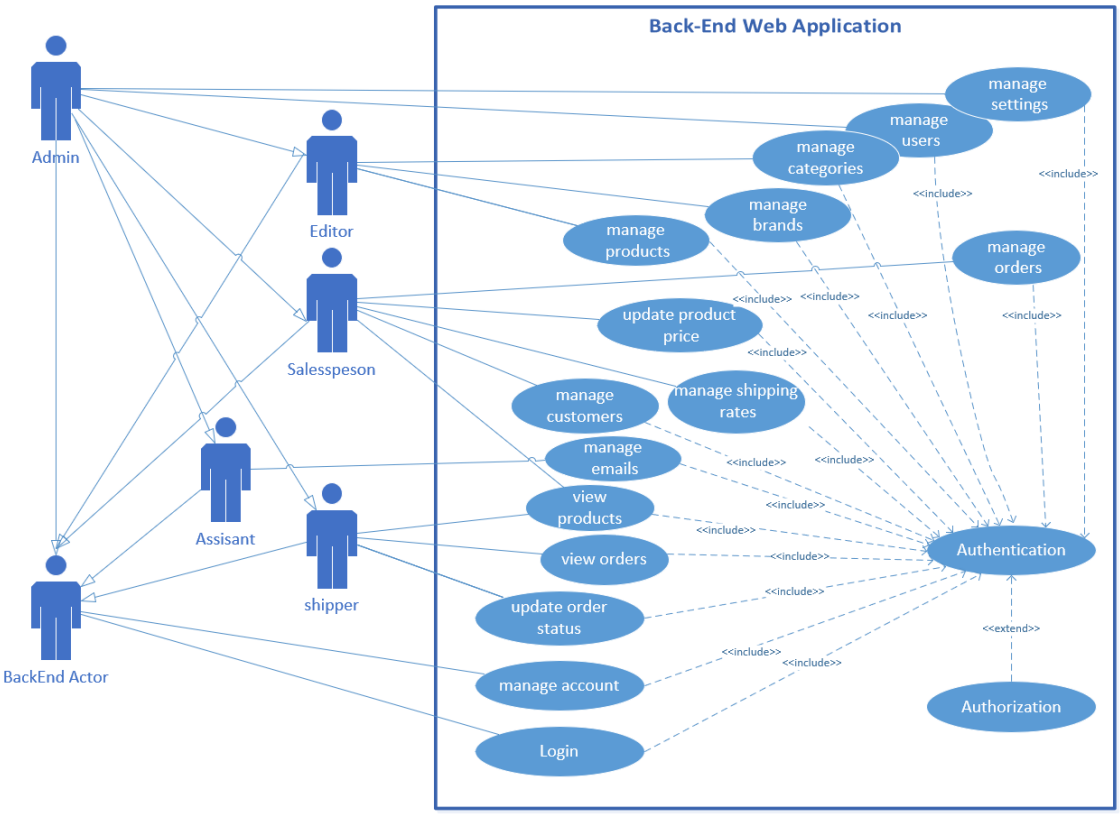

Figure 3.6: Use Case Diagram For Backendend Web Application

3.8 Class Diagrame

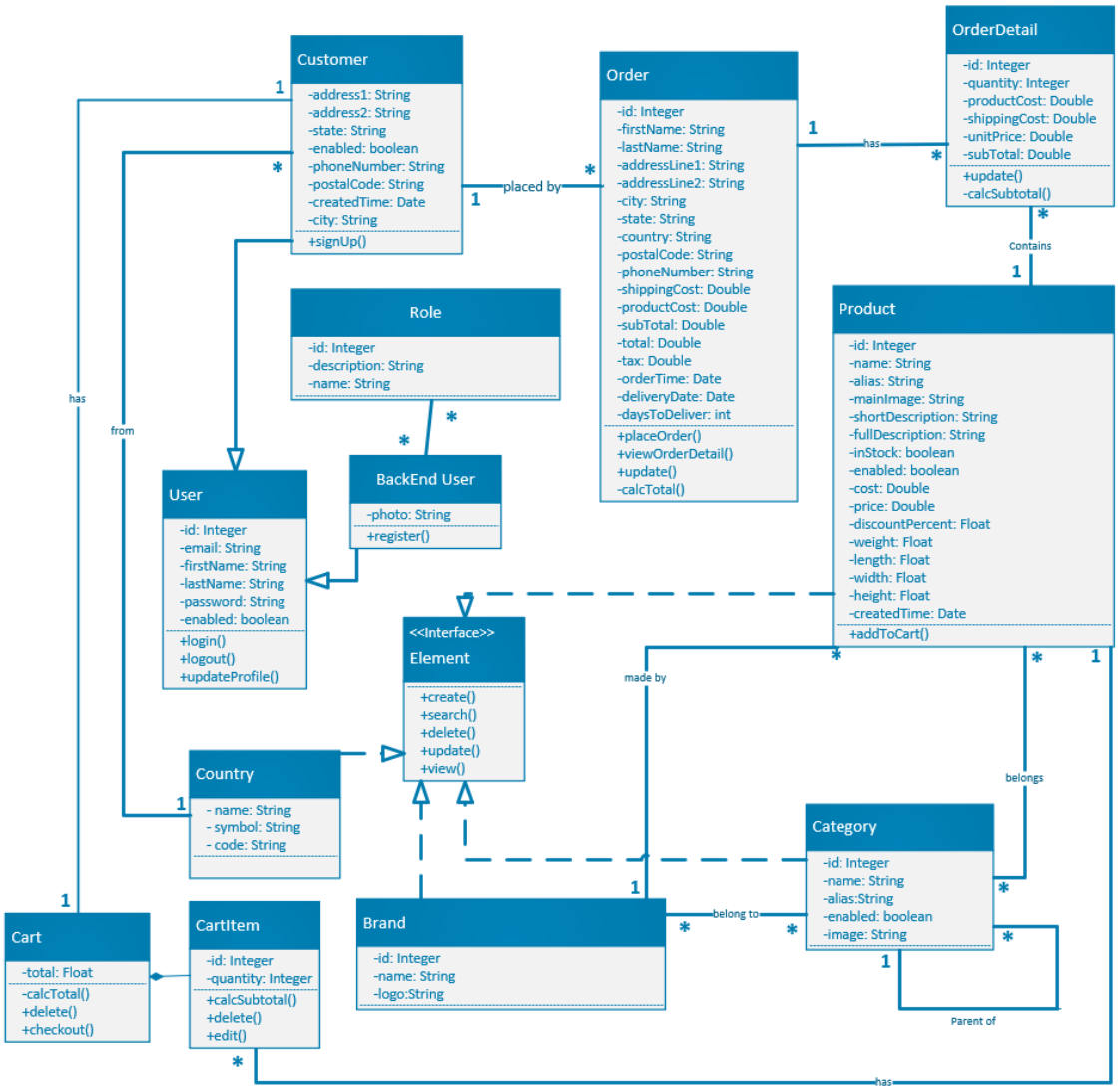

Figure 3.7: Class Diagram

3.9 Sequence Diagrames

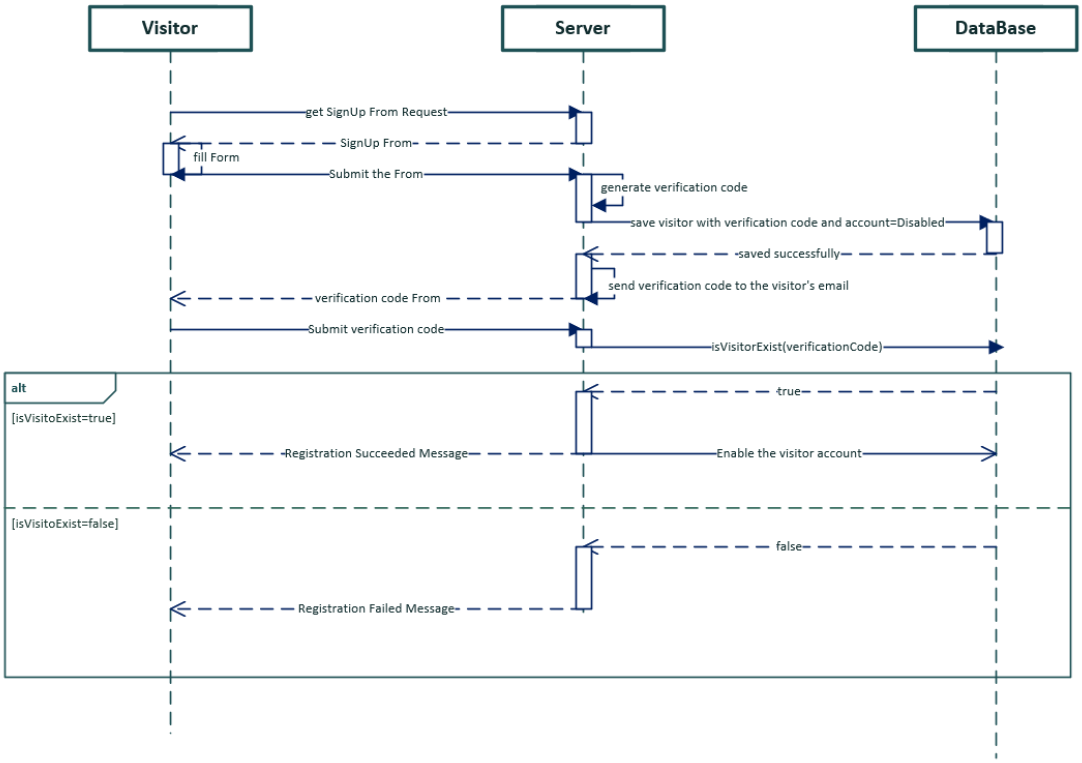

Figure 3.8: Sign Up

Sign Up The Sign Up process involves the visitor requesting and completing a form to provide their credentials. Once the form is filled out, it is submitted to the server. In response, a verification code is sent to the visitor's email address as a means of validating their email. The visitor then receives a form from the server to enter the verification code to activate their account. if the verification code is correct the server creates an account for the visitor and informs him that the Sign Up succeeded.

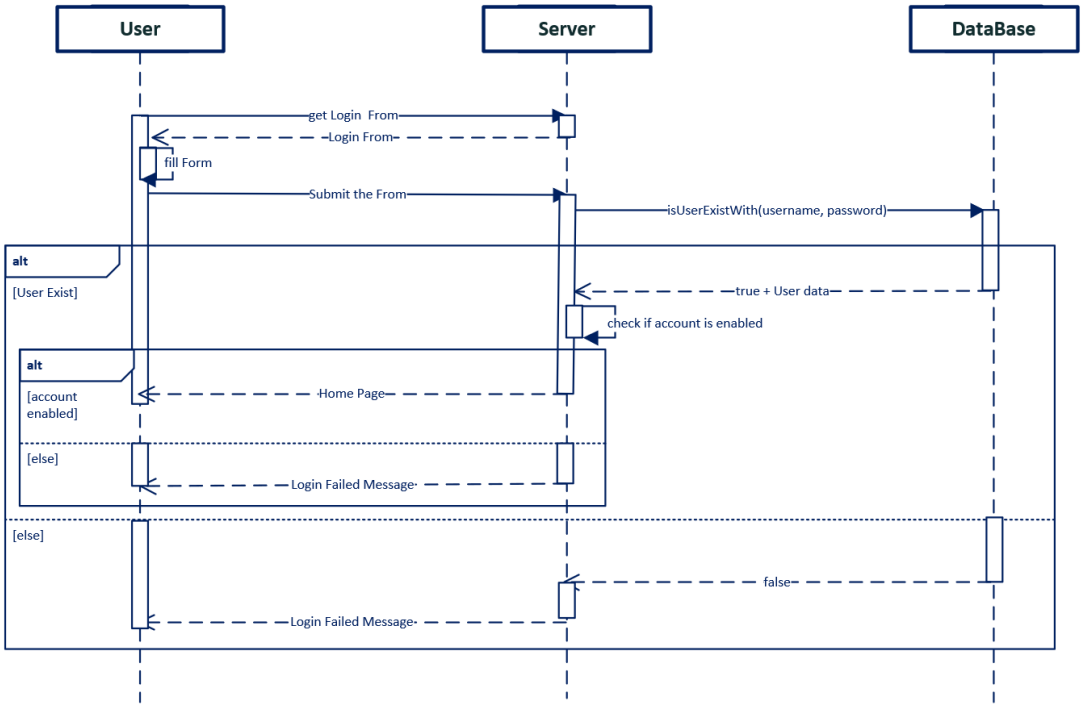

Figure 3.9: Login

Login To log in, the user (customer or backend actor) first requests the login form. The user then fills in the email and password fields and submits the form to the server. The server checks if the user exists with the email and password provided. If the user does not exist, the server denies access. If the user exists, the server checks if the user is activated. If the user is activated, the server grants access to the user's account. If the user is not activated, the server denies access and prompts them.

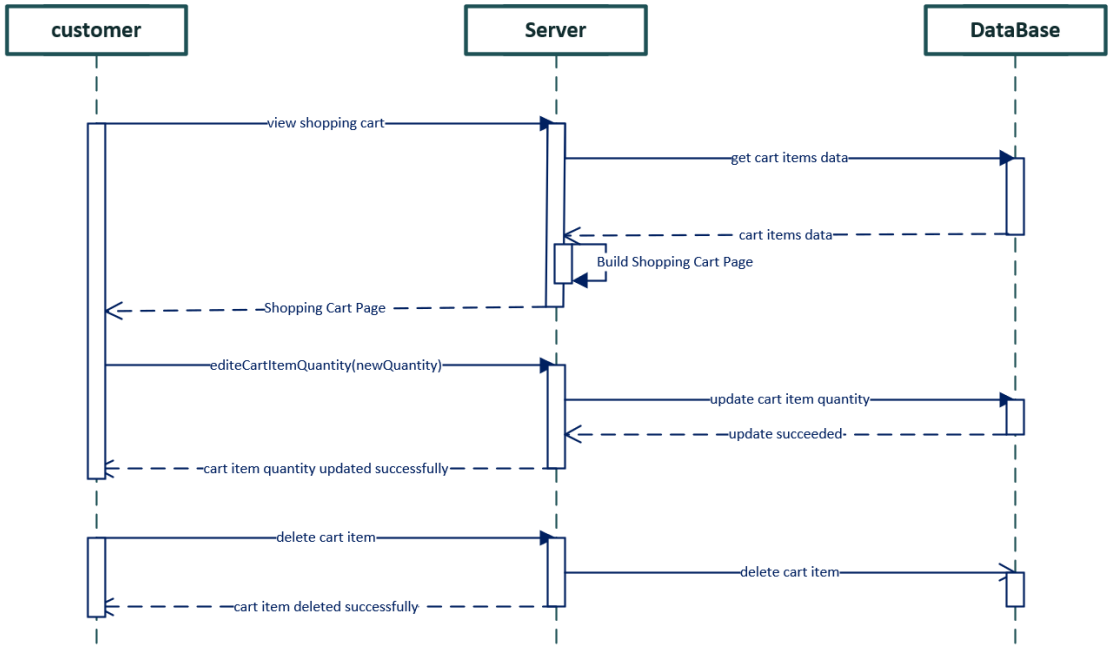

Figure 3.10: View, Update and Delete Cart Items

View, Update and Delete Cart Items Customers can view their shopping cart details, including the total price, items, and quantity of each item, by requesting the shopping cart page. They can also edit the quantity of any cart item or delete it completely. To edit or delete a cart item, the customer must access the shopping cart page first.

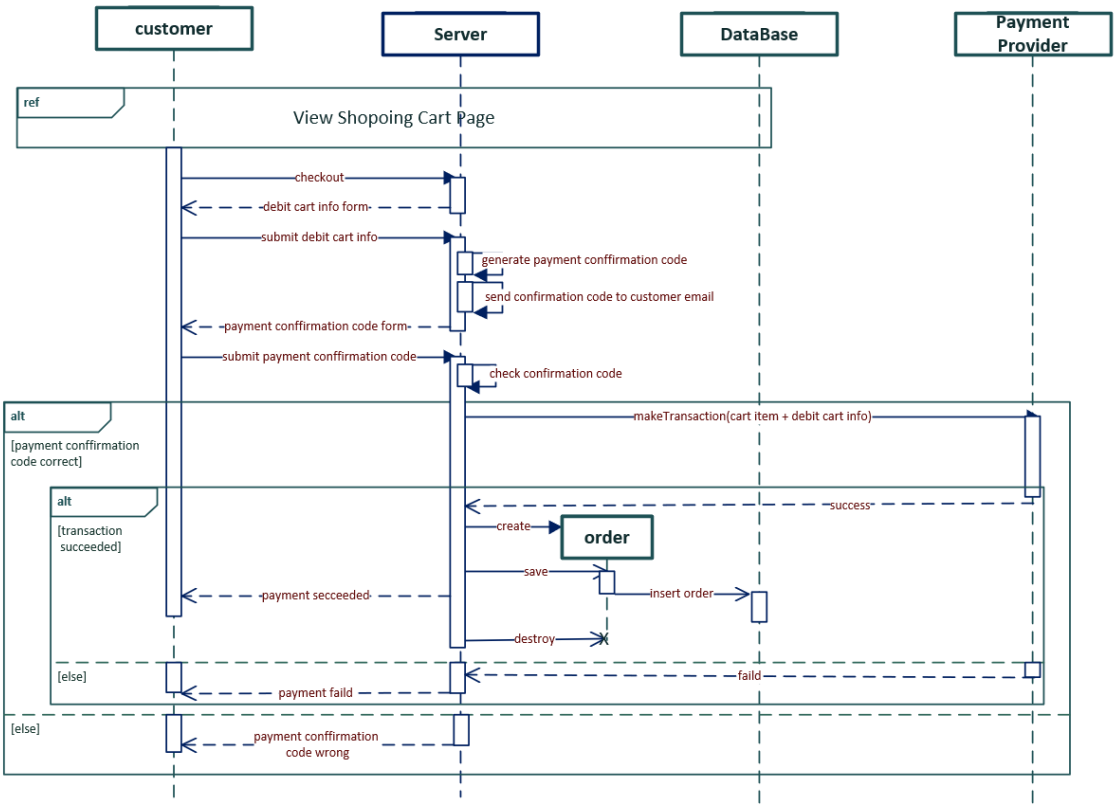

Figure 3.11: Checkout

Checkout In order for customers to proceed with the checkout process, they need to first access the shopping cart page. Once on the shopping cart page, they can request to proceed with the checkout. Upon the customer's checkout request, the server prompts them to provide their credit card information. The customer then sends their credit card information to the server, which responds by sending a confirmation code to the customer's email. The server requires the customer to provide this confirmation code as a means to verify the account owner. If the confirmation process is successful, the server forwards the credit card information to the payment provider for verification and transaction processing. In the event that the credit card information is incorrect, the payment provider notifies the server, which then informs the customer. However, if the credit card information is valid, the payment provider proceeds with the transaction, notify the server of its success, and the server creates an order. Finally, the server informs the customer that the checkout process has been successfully completed.

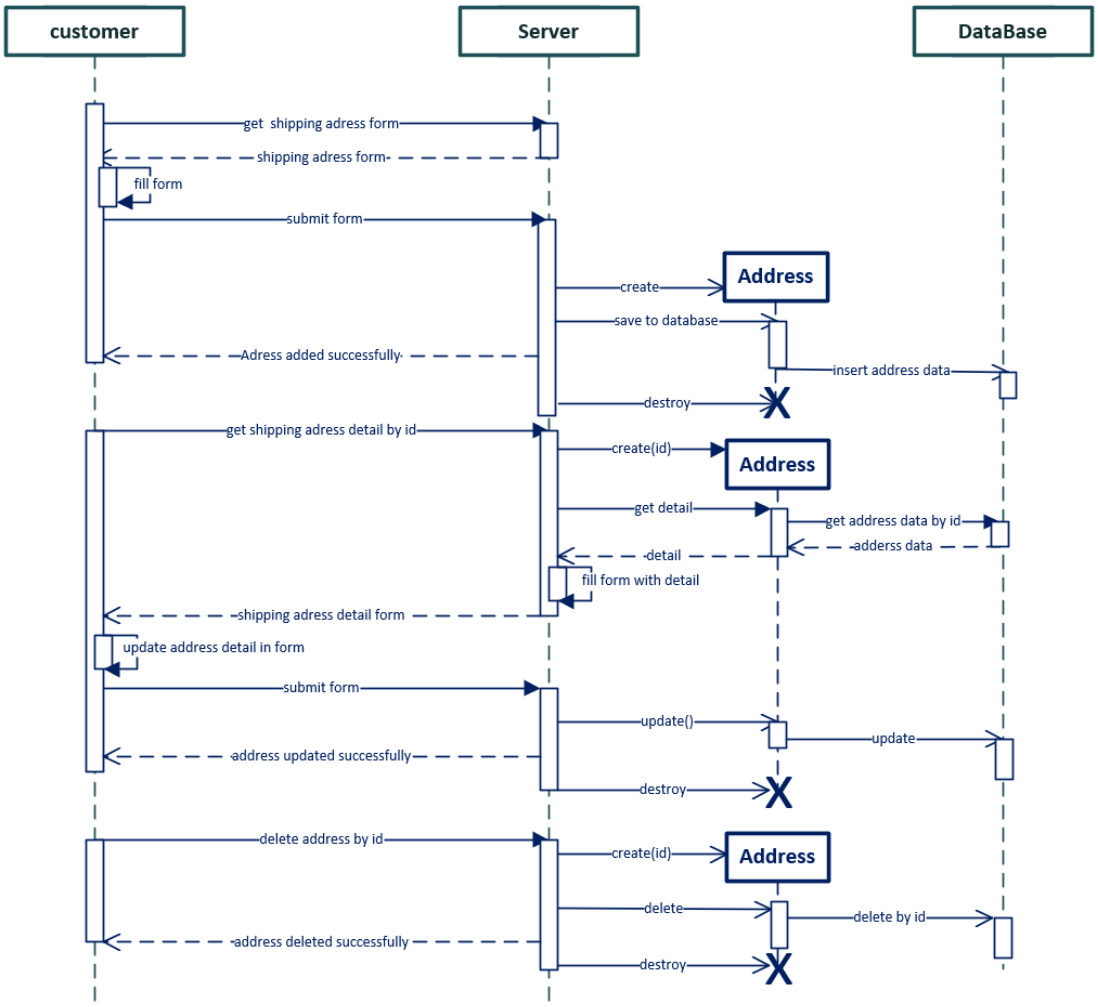

Figure 3.12: Manage Shipping Addresses

Manage Shipping Addresses The customer has the option to have multiple shipping addresses associated with their account. In order to view the available shipping addresses, the customer needs to request the Shipping Addresses page. To make any changes or add new shipping addresses, the customer can request the corresponding form for editing or adding purposes.

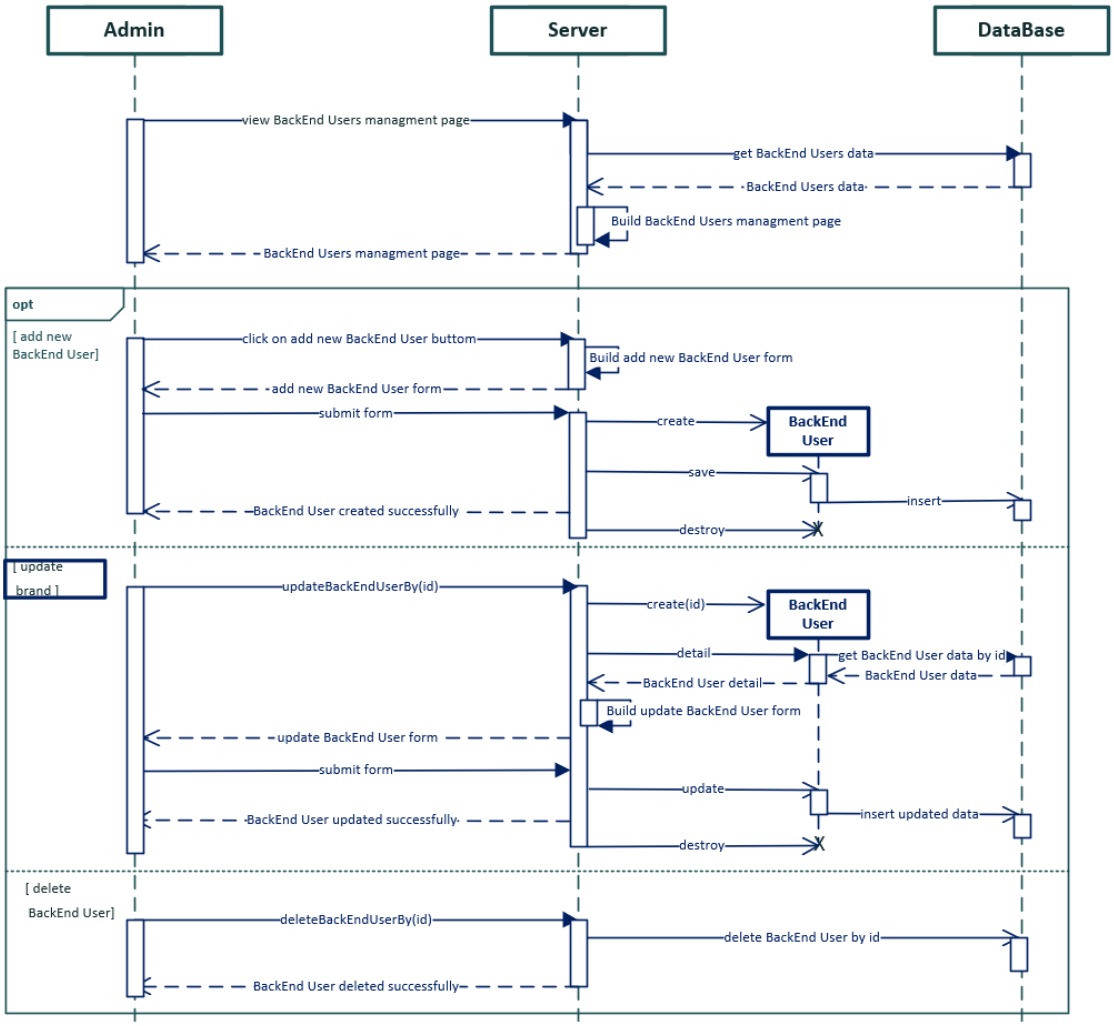

Figure 3.13: :Manage BackEnd Actors

Manage BackEnd Actors The admin holds the responsibility of registering new backend actors, as well as editing, activating, deactivating, and deleting their accounts.

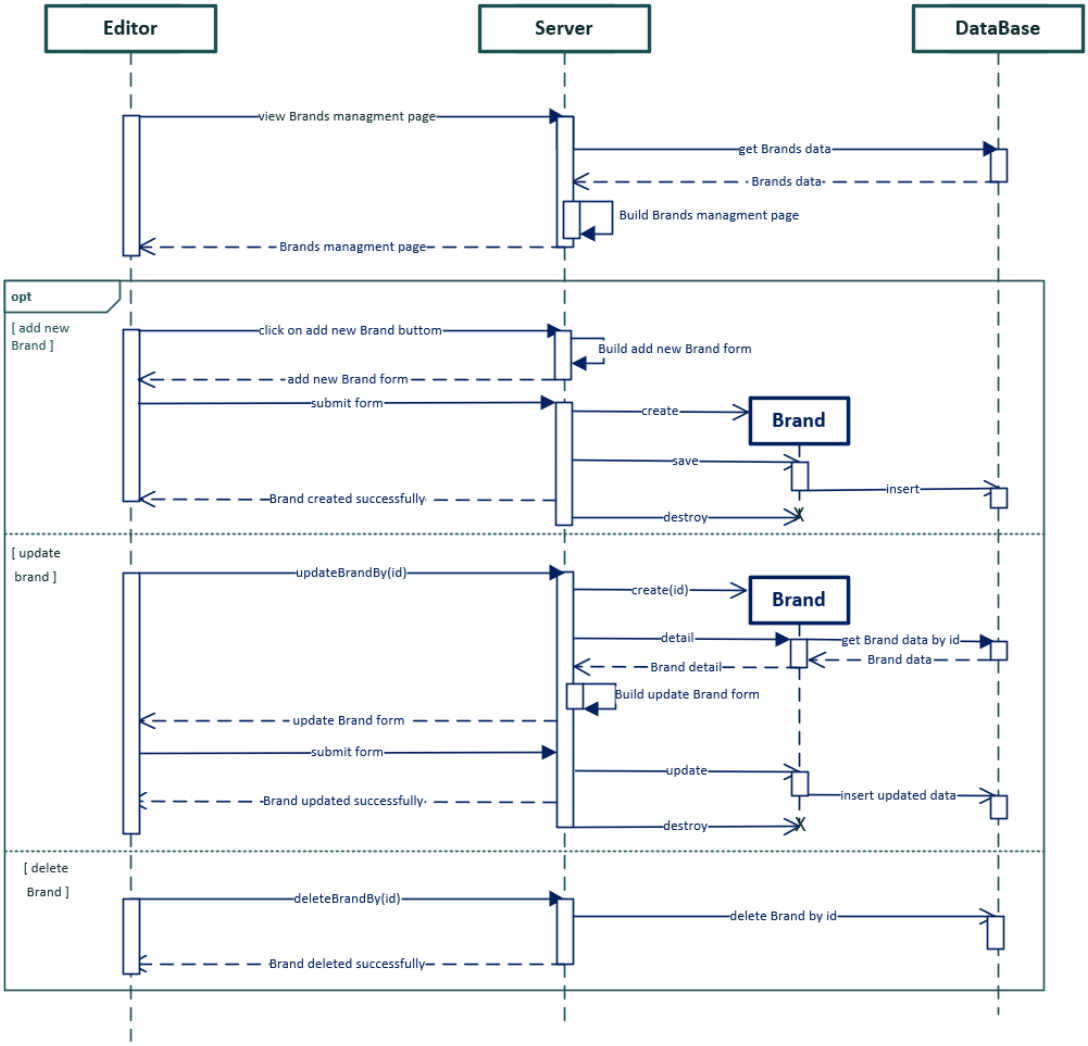

Figure 3.14: Manage Brands

Manage Brands The editor's responsibility includes creating, editing, enabling, disabling, and deleting brands.

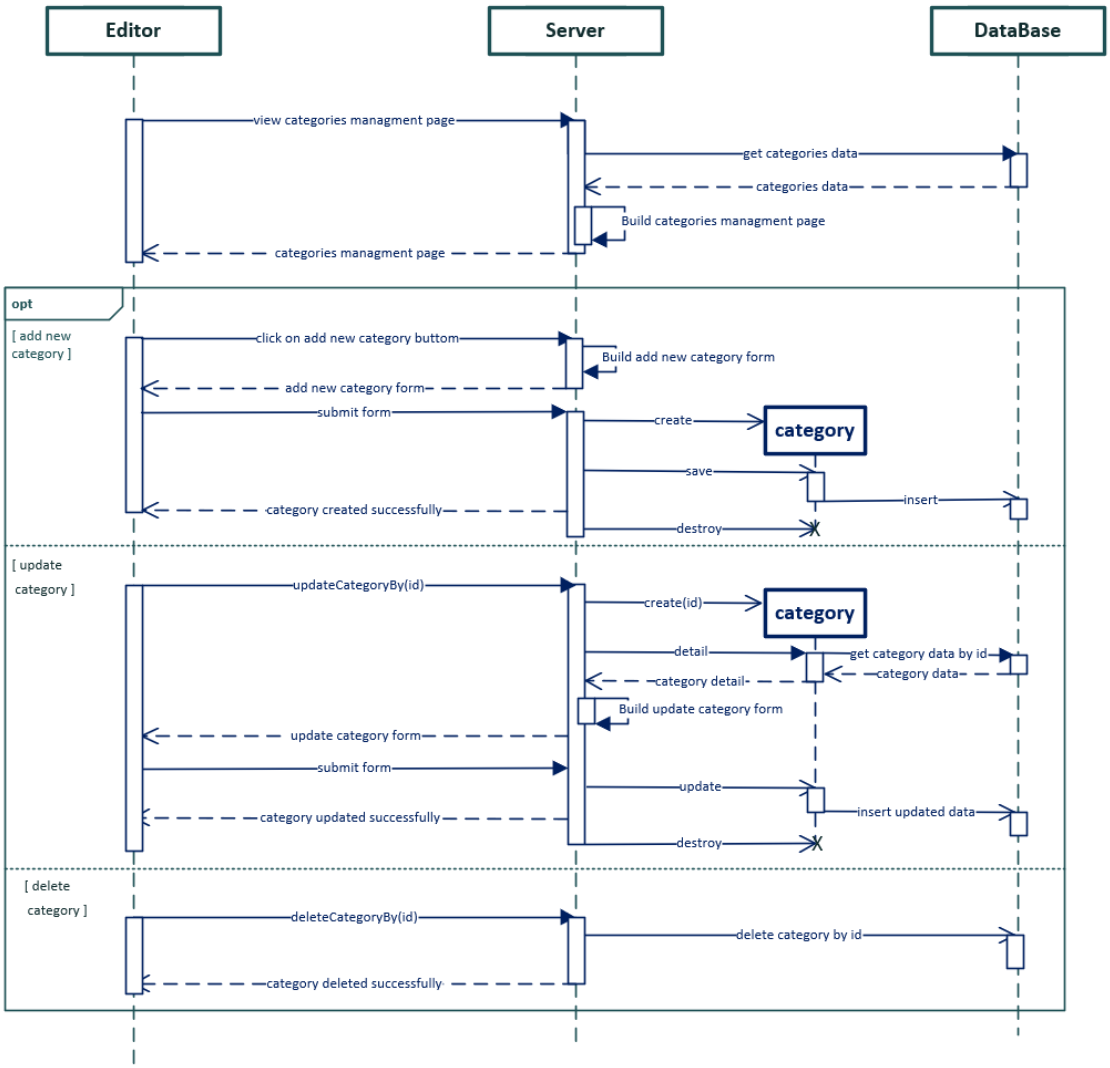

Figure 3.15: Manage Categories

Manage Categories The editor's responsibilities extend to creating, editing, enabling, disabling, and deleting categories as well.

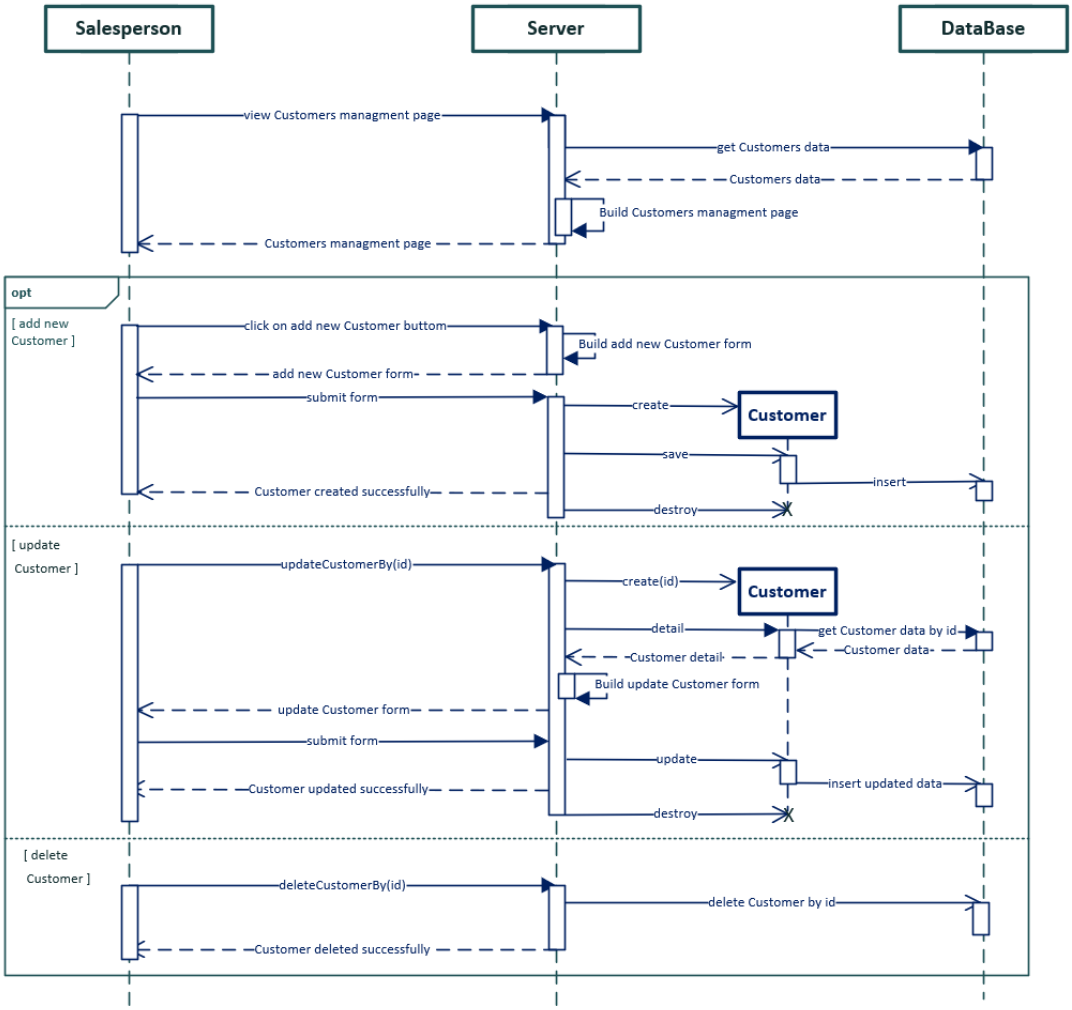

Figure 3.16: Manage Customers

Manage Customers The salesperson is entrusted with the tasks of creating, editing, activating, deactivating, and deleting customers.

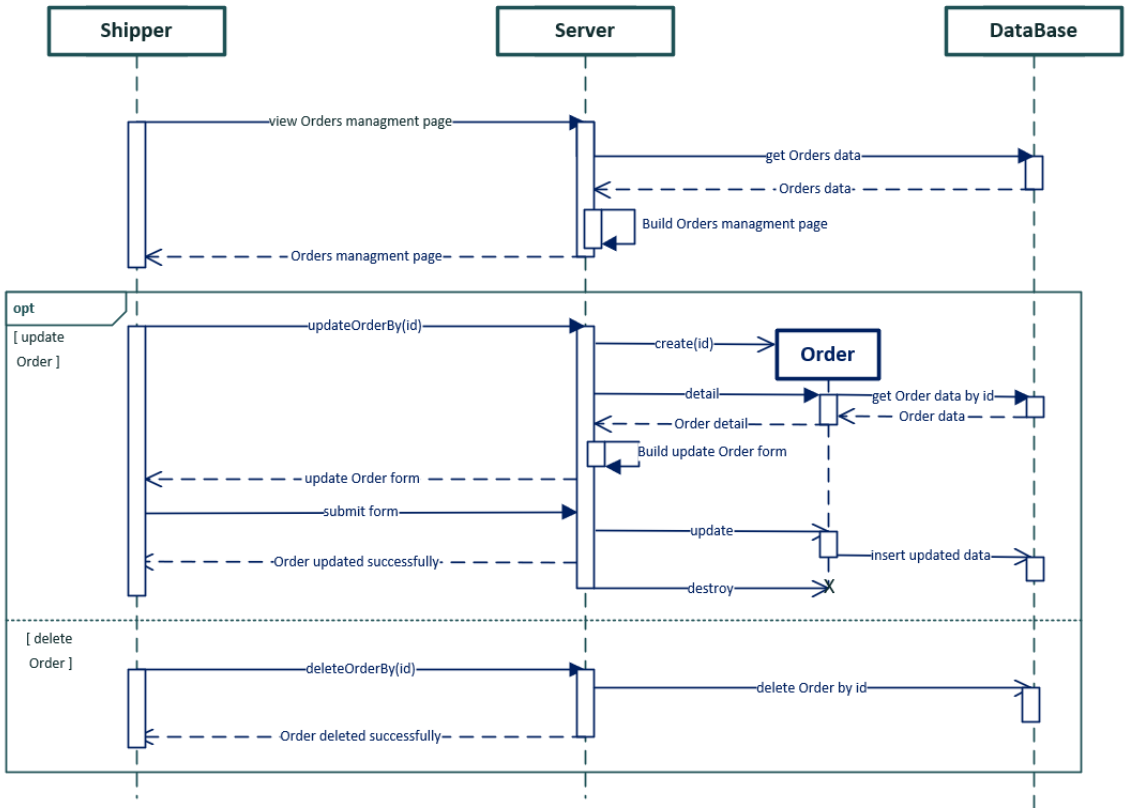

Figure 3.17: Manage Orders

Manage Orders The shipper's responsibilities include editing and deleting orders.

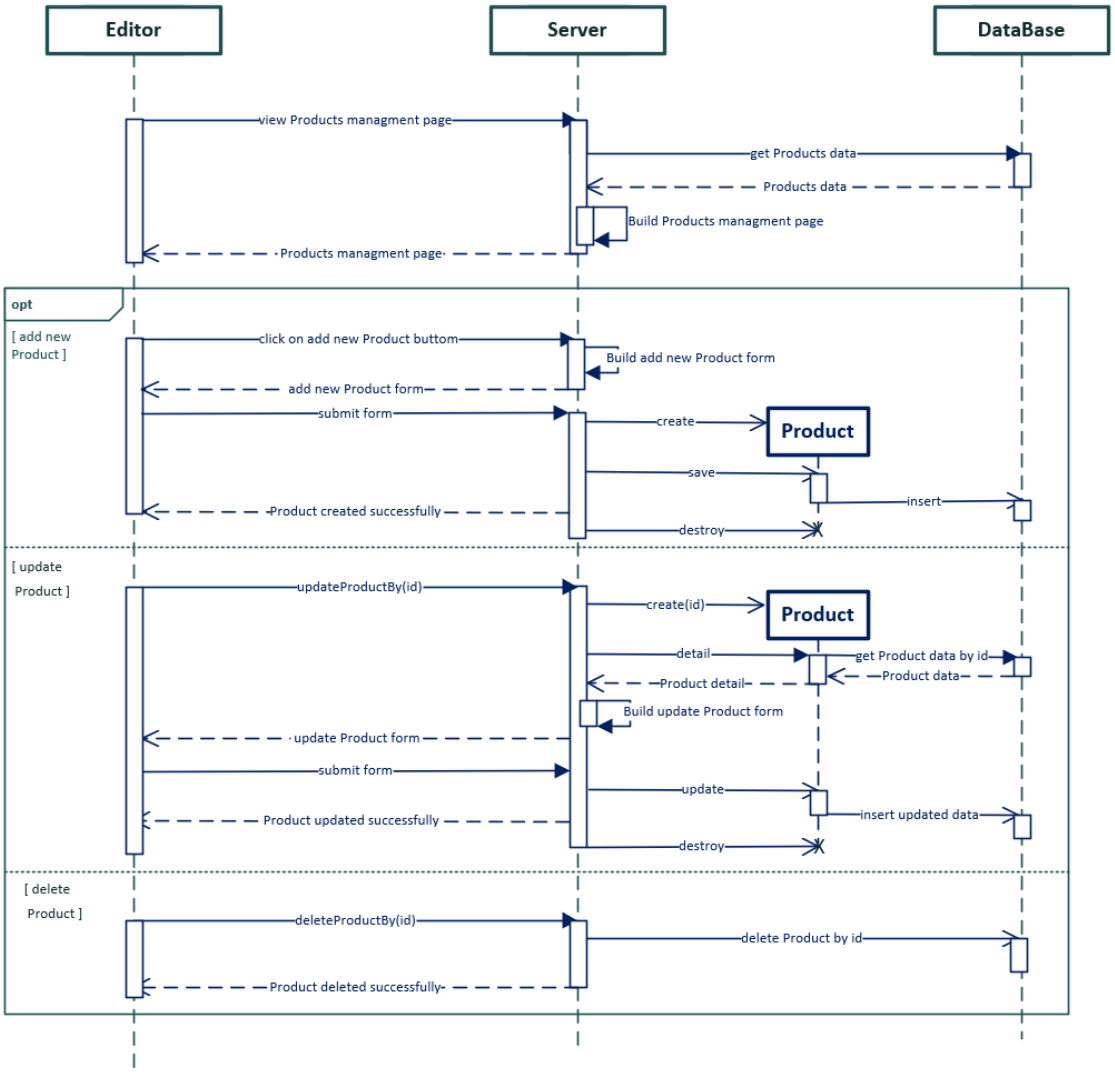

Figure 3.18: Manage Products

Manage Products The editor's responsibilities encompass creating, editing, enabling and disabling, and deleting products.

3.10 Conclusion

In this chapter, we have provided a comprehensive conceptual overview of the design of an e-commerce application. We began by examining the system from a high-level perspective, identifying the key actors involved. Utilizing UML as a descriptive tool, we further elaborated on the main functionalities of our system, delving into the modules and discussing the objects within the system. We explored how these objects interact with each other and with the system's actors, shedding light on the intricate dynamics and relationships within the system.

Chapter 4

Implementation

In this chapter, we will talk about everything related to the development, testing, and deployment of our work:

1. **The Hardware used for the development, testing, and deployment of the project**
2. **The software and technologies used for developing, testing, and deploying this project.**
3. **System Functionality and Corresponding User Interfaces**

4.1 Hardware Configuration

4.1.1 Development

Local Machine Properties

- Processor : 11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz 4 Corse 8 Logical Processors
- RAM : 8GB DDR4 @ 3200Mhz
- Disk: 500GB SSD
- GPU: Intel(R) Iris(R) Xe Graphics, 4GB
- Display Screen : Intel(R) Iris(R) Xe Graphics, 4GB

4.1.2 Testing

Lenovo ThinkBook Laptop Properties

- Processor: Intel Core i5-1135G7
- RAM: 16GB
- Disk: 512GB SSD
- GPU: Intel Iris Xe Graphics
- Display Screen: 15.6” Full HD (1920x1080)

Samsung Galaxy S8 Plus

- Processor: Qualcomm Snapdragon 835
- RAM: 4GB
- Disk: 64GB
- GPU: Adreno 540
- Display Screen: 6.2” Super AMOLED (2960x1440)

IPhone SE Properties

- Processor: Apple A13 Bionic
- RAM: 3GB
- Disk: 64GB
- GPU: Apple GPU (4-core)
- Display Screen: 4.7” Retina HD (1334x750)

IPad Pro Properties

- Processor: Apple M1
- RAM: 8GB/16GB/256GB/512GB/1TB/2TB
- Disk: 128GB/256GB/512GB/1TB/2TB
- GPU: Apple A13 Bionic (4-core)
- Display Screen: 11” Liquid Retina (2360x1640) or 12.9” Liquid Retina XDR (2732x2048)

4.1.3 Deployment

Heroku Hosting Server Properties

- Processor: Intel Xeon E5-2680 v4
- RAM: 32GB
- Disk: 1TB SSD
- GPU: NVIDIA Tesla K40

Amazon S3 Server Properties

- Processor: Intel Xeon E5-2680 v4
- RAM: 32GB
- Disk: 10TB SSD
- GPU: NVIDIA Tesla K80

4.2 Software and Technologies

4.2.1 Development

The Ubuntu Operating System

For the development environment, we opted for Ubuntu 22.04.2 as the operating system.

Ubuntu is a renowned Linux-based operating system, known for its open-source nature and availability at no cost.

Ubuntu offers a range of editions tailored for different use cases, including desktop workstations, secure connected devices, and serving as a reference for OpenStack and cloud platforms. It benefits from a thriving community and comes pre-installed on computers from leading manufacturers. Ubuntu aims to provide users with a superior computing experience while fostering collaboration in the realm of free software development.[29]

The following are key advantages of Ubuntu that influenced our decision to select it as the operating system for the development environment:

- "Using Ubuntu Desktop provides a common platform for development, test, and production environments."[30]
- "Long term support (LTS) releases are delivered every 2-years, with 5 years of standard support extended to 10 years with an Ubuntu Pro Desktop subscription."[30]
- "New Developer editions of Ubuntu Desktop are released every 6-months, with 9-months of support and access to the latest kernel, libraries, and new upstream OS features."[30]
- "With a Snap Store of over 7,000 applications in addition to the Ubuntu repository, it's easy to tailor Ubuntu desktop to your needs."[30]
- "Development tools include Visual Studio Code, IntelliJ, GitKraken, Unity and Blender."[30]
- "Office productivity tools include Slack, Microsoft Teams, Dropbox, and Zoom as well as the Microsoft-compatible LibreOffice."[30]

Java Programming Language

"The Java Programming Language is a general-purpose, concurrent, strongly typed, class-based object-oriented language. It is normally compiled to the bytecode instruction set and binary format defined in the Java Virtual Machine Specification."[21] Advantages of using Java for developing enterprise applications:

- Interoperability: Java's core philosophy of interoperability enables seamless communication and integration between different devices and platforms.[12]
- Scalability: Java offers platform scalability, allowing applications to be adapted and run on various devices with different resources, from desktop to mobile.[12]
- Reusability and Modular Design: Java's object-oriented architecture enables the creation of modular programs and reusable code, reducing development cycles and extending the lifespan of enterprise applications.[12]
- Adaptability: Java excels in adapting to new use cases, such as being an ideal platform for the Internet of Things (IoT) applications.[12]
- Extensive Ecosystem: Java benefits from a vast ecosystem of developers who constantly develop and share libraries specifically designed for enterprise and IoT application development.[12]

Integrated Development Environment IntelliJ IDEA

"IntelliJ IDEA is an Integrated Development Environment (IDE) for JVM languages designed to maximize developer productivity. It does the routine and repetitive tasks for you by providing clever code completion, static code analysis, and refactorings, and lets you focus on the bright side of software development, making it not only productive but also an enjoyable experience.it adds support for a variety of server-side and front-end frameworks, application servers, integration with database and profiling tools, and more."[14]

Structured Query Language

"Structured Query Language (SQL) is a standardized programming language that is used to manage relational databases and perform various operations on the data in them. Initially created in the 1970s, SQL is regularly used not only by database administrators but also by developers writing data integration scripts and data analysts looking to set up and run analytical queries."[15]

Database Management System MySQL

MySQL, a renowned open-source relational database management system, adopts a table-based structure to store data, organized into rows and columns. Utilizing Structured Query Language (SQL), users can effectively define, manipulate, control, and query the stored data. MySQL stands as the predominant open-source database system globally and is an integral component of the widely adopted LAMP technology stack. This stack encompasses a Linux-based operating system, Apache web server, MySQL database, and PHP for data processing. Leveraged across diverse applications, websites, and services, MySQL empowers seamless storage and retrieval of data, contributing to its extensive popularity and versatility.[7]

MySQL serves as the backbone for numerous highly accessed applications such as Facebook, Twitter, Netflix, Uber, Airbnb, Shopify, and Booking.com. Its open-source nature has facilitated the incorporation of a wide array of features, which have been developed collaboratively with users over a span of more than 25 years. As a result, MySQL offers comprehensive support for various programming languages and is highly likely to be compatible with your preferred application or programming language, enhancing its versatility and applicability.[22]

Spring Boot JAVA Framework

The Java Spring Framework is a widely adopted, open-source, enterprise-grade framework utilized for the development of resilient applications that operate on the Java Virtual Machine (JVM). In conjunction with this framework, Java Spring Boot serves as a valuable tool that streamlines the process of creating web applications and microservices. Its core functionalities encompass autoconfiguration, an opinionated approach to configuration, and the capability to establish standalone applications. The seamless integration of these features empowers developers to efficiently establish Spring-based applications, minimizing the need for extensive configuration and setup, thereby augmenting overall productivity and operational effectiveness.[13]

Template Engine Thymeleaf

Thymeleaf is a contemporary server-side Java template engine designed for web and standalone environments. Its primary objective is to enhance the development workflow by providing elegant and natural templates that are not only compatible with browsers but also function as static prototypes, enabling effective collaboration within development teams. With its seamless integration with Spring Framework, extensive tool integrations, and the ability to customize functionality, Thymeleaf is highly suited for modern HTML5 JVM web development. It supports the processing of HTML, XML, text, JavaScript, and CSS files, distinguishing itself from other

template engines by enabling templates to be used as prototypes that can be viewed as static files.[4][27]

4.2.2 Testing

Postman

Postman is a user-friendly API platform that enables the efficient creation and testing of APIs. It simplifies the entire API lifecycle by providing intuitive tools to build, execute, and analyze API tests. With its streamlined interface and collaborative features, Postman empowers users to create superior APIs more swiftly. Whether you're a non-technical user or a developer, Postman helps you enhance the quality and performance of your APIs, making it an essential tool for API testing and development. .[24][23]

JUnit

JUnit, an open-source framework for Java, enables developers to conduct automated unit tests, guaranteeing code integrity and functionality. It serves as a vital tool for constructing and executing tests, and preventing code breakages. By re-running test cases, JUnit ensures software robustness and error-free performance.[8]

Chrome DevTools

We utilized Chrome DevTools, an integrated set of web developer tools within the Google Chrome browser, to enhance our web application testing process. Chrome DevTools enabled us to make real-time edits to web pages, swiftly diagnose issues, and thereby enhance the efficiency of website development. By leveraging the capabilities of Chrome DevTools, we were able to build superior websites and accelerate our development workflow.[6]

Mockito

Mockito is an influential JAVA-based mocking framework and library meticulously crafted for the purpose of conducting rigorous unit testing of JAVA applications. With its profound capabilities, it empowers developers to simulate interfaces by incorporating mock functionalities, thereby facilitating the creation of sophisticated mock interfaces. This seamless integration of mock interfaces into unit tests augments the overall testing process, enabling comprehensive evaluation and validation of the application's functionality and behavior.[28]

4.2.3 Deployment

Heroku

Heroku is a container-based cloud Platform as a Service that enables developers to deploy and scale applications easily. With its elegant and flexible platform, Heroku simplifies the path to market for developers. It is a fully managed service, freeing

developers from server and infrastructure maintenance. Heroku offers a comprehensive experience with various services, tools, and workflows designed to enhance developer productivity.[11]

We opted to employ the Heroku Platform for hosting both our Backend and Frontend Applications, as well as our MySQL server.

Amazon Simple Storage Service

Amazon S3 is a renowned object storage service known for its scalability, data availability, security, and performance. It enables organizations of all sizes and industries to securely store and manage large volumes of data for various applications, including data lakes, websites, mobile apps, backup and restore, archival storage, enterprise applications, IoT devices, and big data analytics. With comprehensive management features, users can optimize data access, organization, and configuration according to their specific business and compliance requirements.[2]

We utilized Amazon S3 as our storage solution for storing a variety of unstructured and semi-structured data, such as images, HTML files, CSS files, JavaScript files, and more.

4.3 System Functionality and User Interfaces

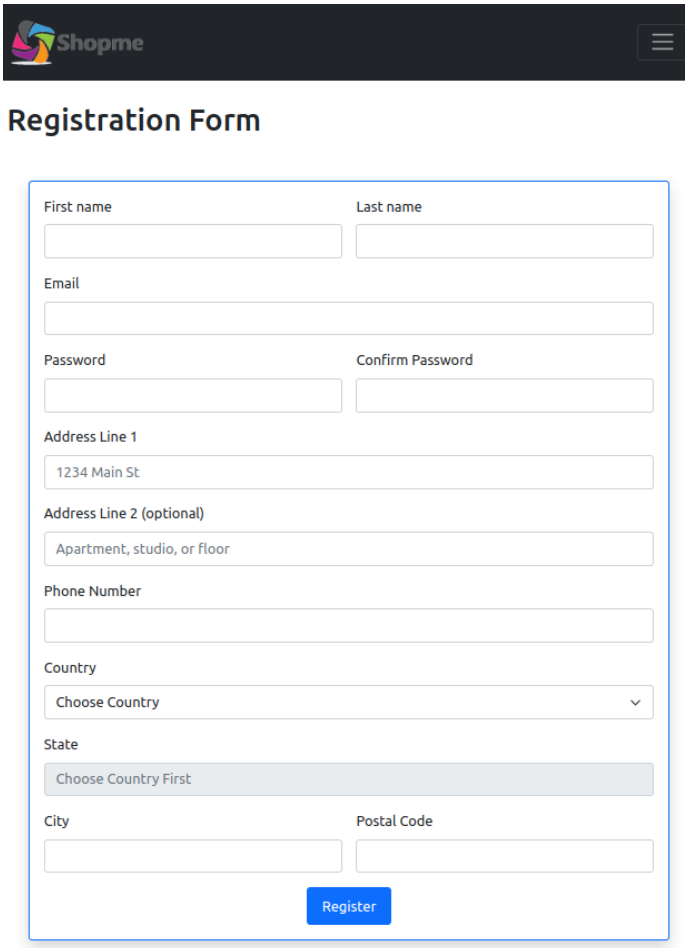

Figure 4.1: Customer Registration Form

Customer Registration Form serves as the means for visitors to register and create their customer accounts. By filling out the required fields and providing necessary information such as name, email, password, and additional details, users can successfully register and gain access to exclusive customer features and benefits.

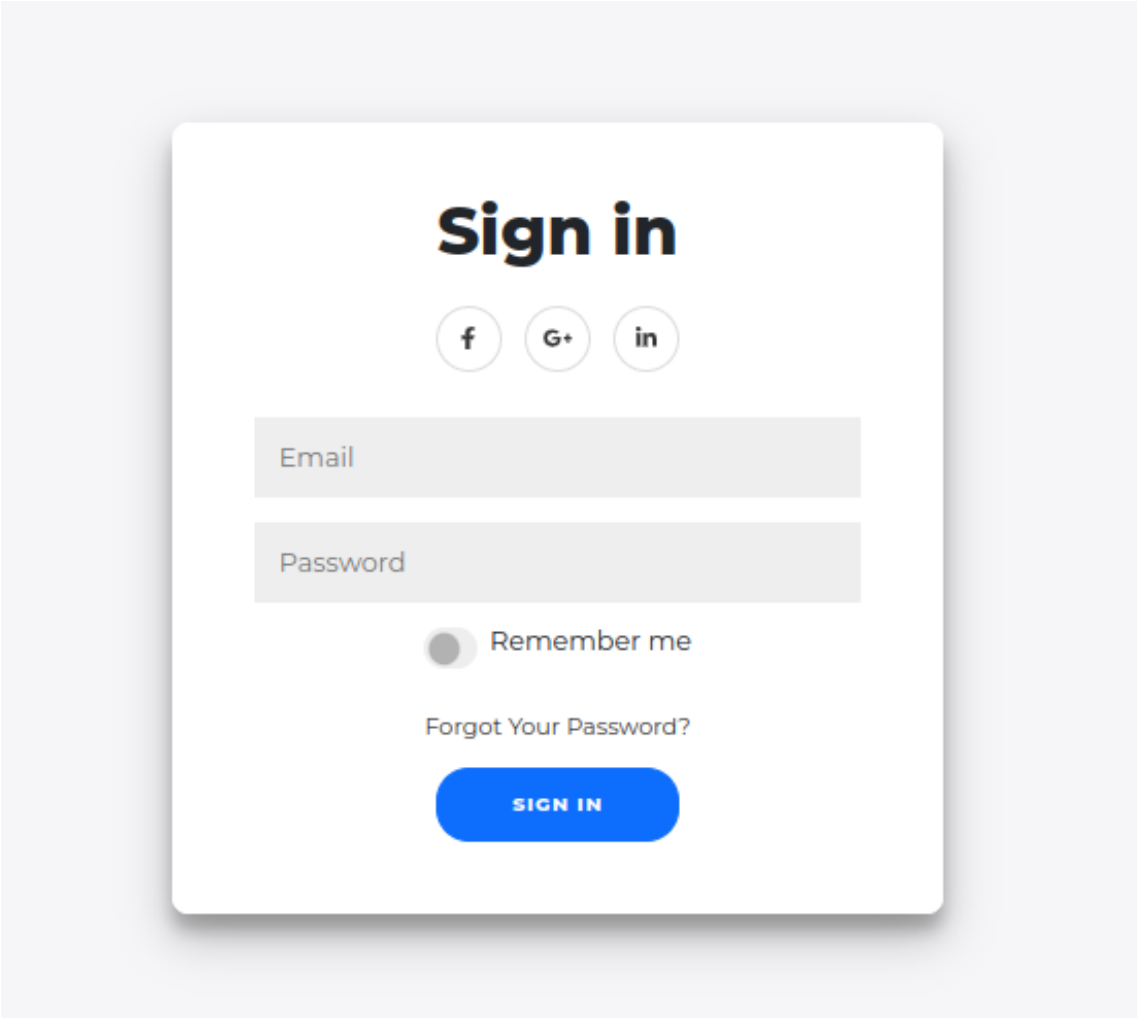

Figure 4.2: The Visitor Sign-In Form

The Visitor Sign-In Form serves as the gateway for visitors to transition into becoming customers. It offers the option to either provide necessary credentials acquired during the Sign-Up process, or conveniently access the system using their Facebook, Gmail, or LinkedIn accounts. As a result, customers gain access to exclusive functionalities that are not available to visitors

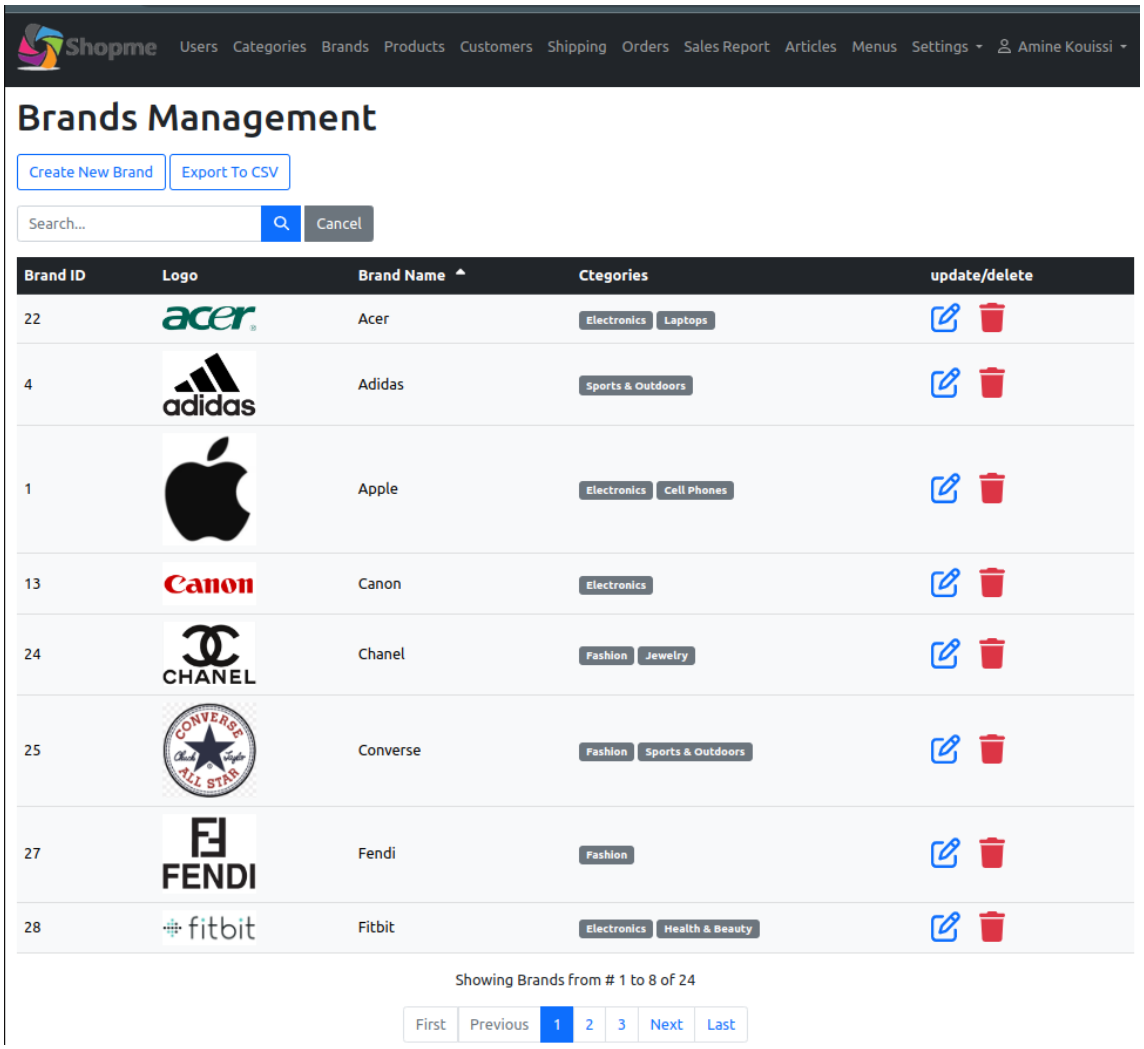

Figure 4.3: Brands Listing Page

The Brands Listing Page fulfills the task of showcasing all available brands while granting users the capability to search, delete, add, and modify them. Moreover, it empowers users to effortlessly export the brands list to a CSV file.

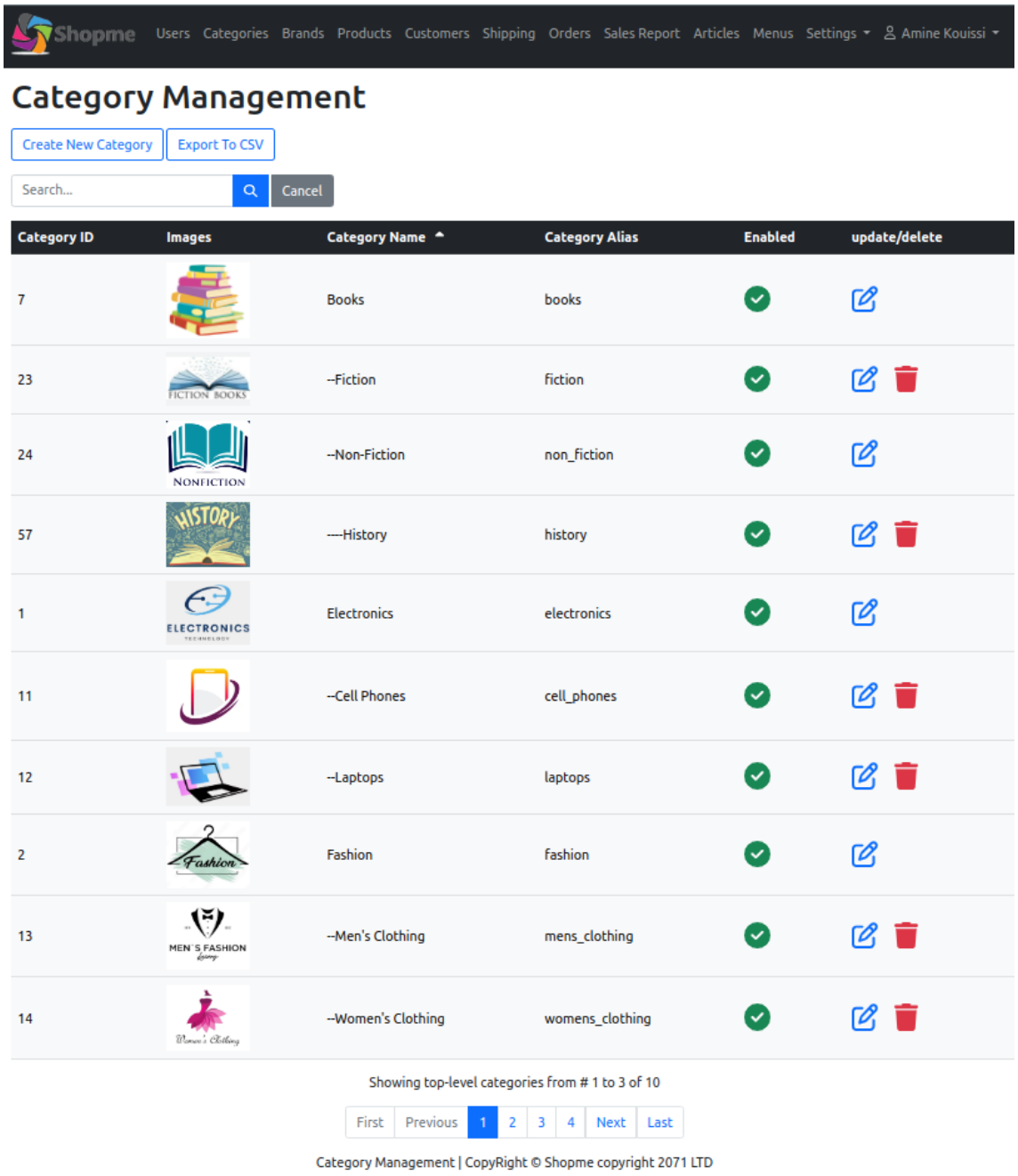

Figure 4.4: Categories Listing Page

The Categories Listing Page fulfills the task of showcasing all available categories while granting users the capability to search, delete, add, and modify them. Moreover, it empowers users to effortlessly export the categories list to a CSV file.

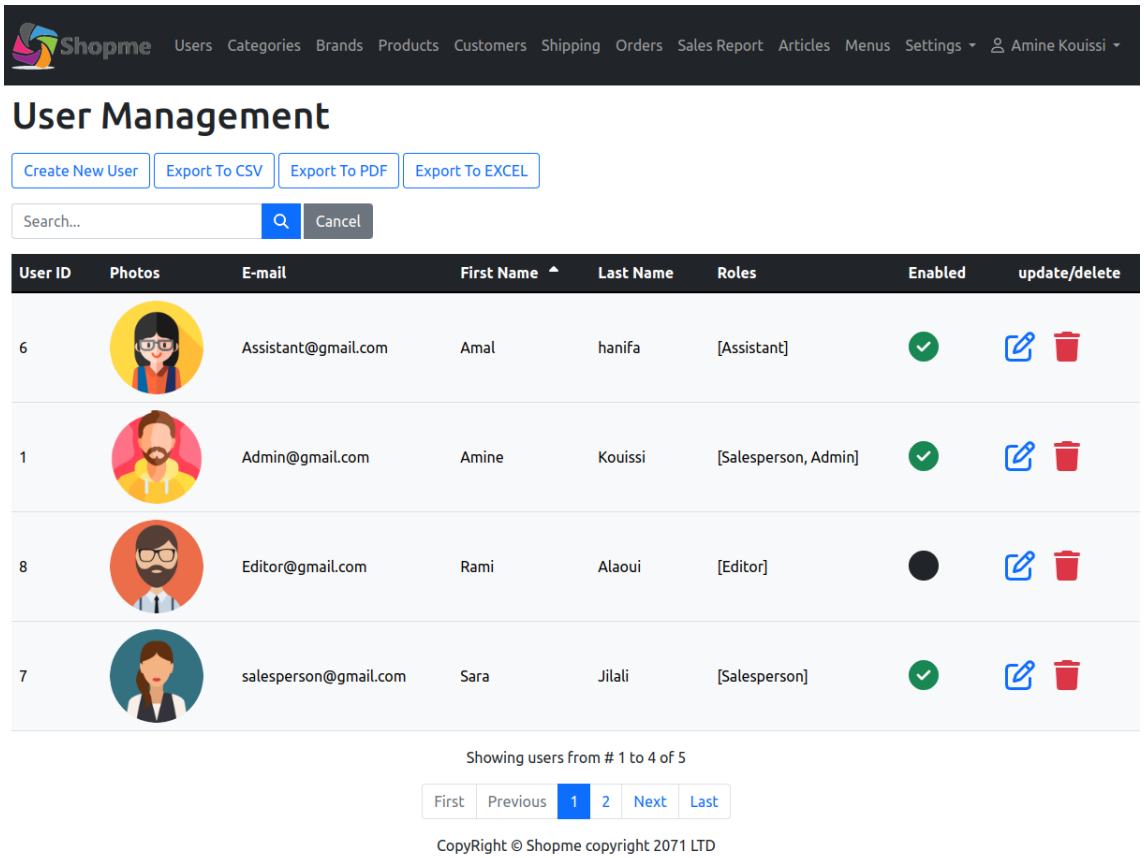

Figure 4.5: The Backend Users Listing Page

The Backend Users Listing Page exclusively accessible to administrators showcases a comprehensive list of backend users including editors, salespersons, assistants, and shippers. This page offers a range of powerful functionalities such as searching, deleting, adding, activating/deactivating, and modifying users. Additionally, administrators are empowered to seamlessly export the backend users list to various file formats including CSV, Excel, and PDF.

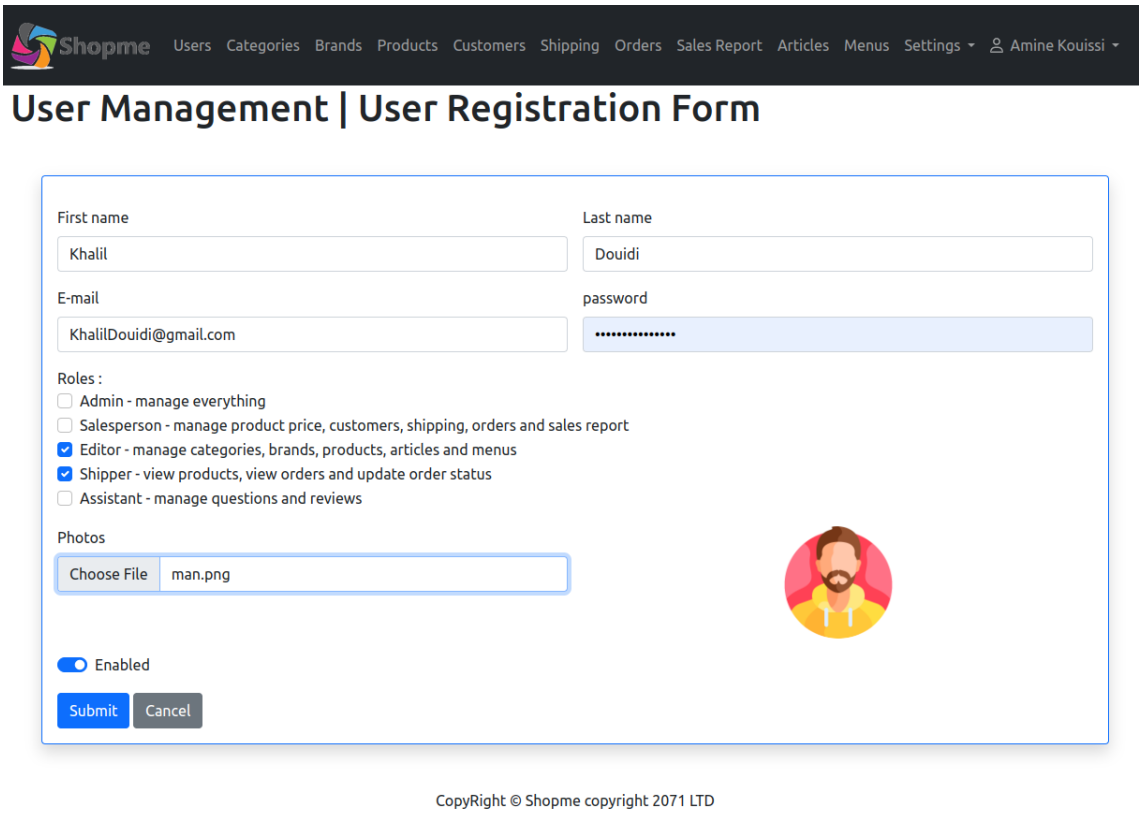

Figure 4.6: The Backend User Registration Form

The Backend User Registration Form accessible solely by administrators, provides a seamless avenue for registering new backend users. By populating the form fields with the required information and selecting the suitable roles, administrators can efficiently onboard new users.

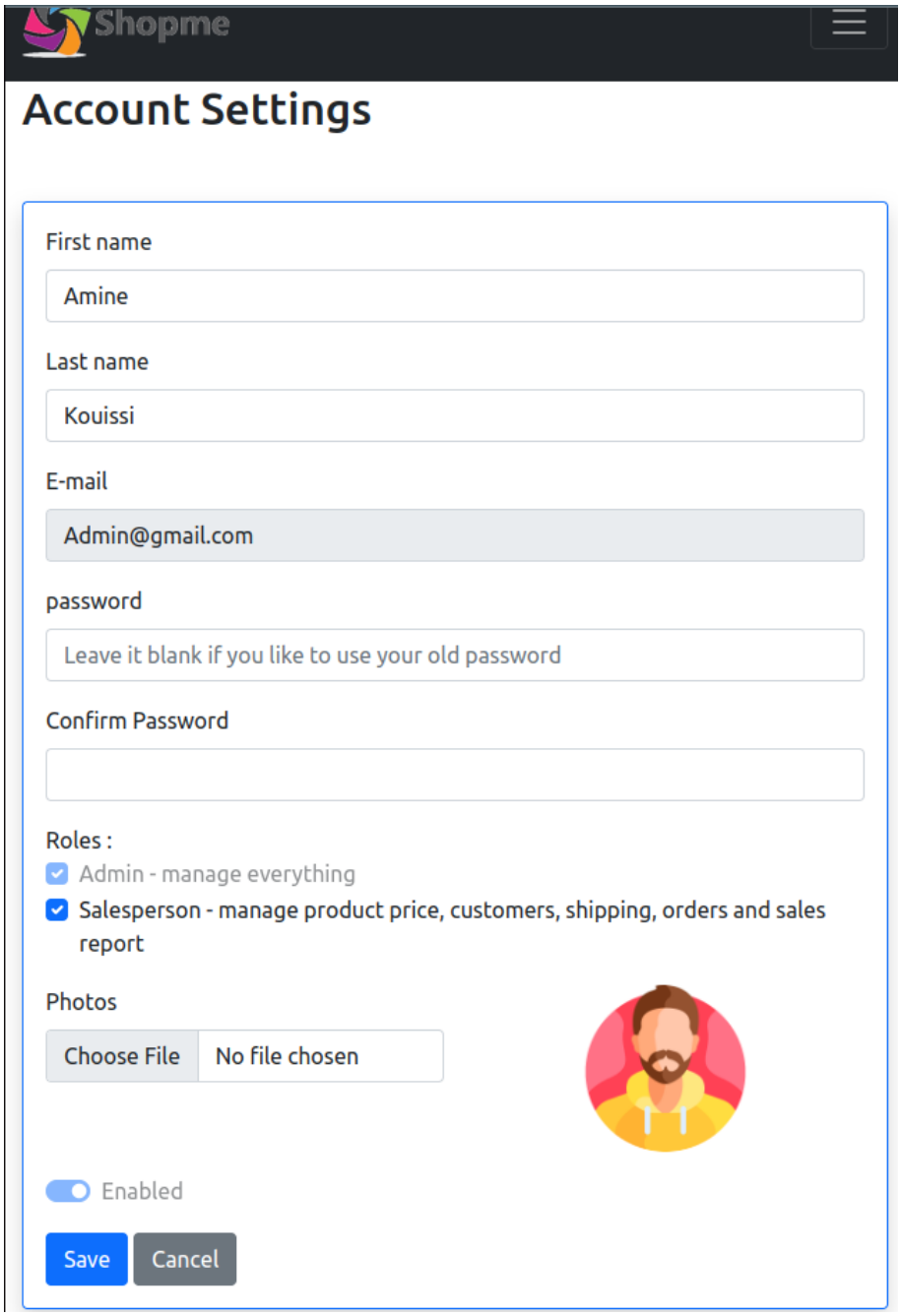

Figure 4.7: The Account Settings Page

The Account Settings Page exhibits pertinent personal information of the backend user, such as their first name, last name, email, password, and profile photo. It further enables the user to effortlessly update and modify these details as needed.

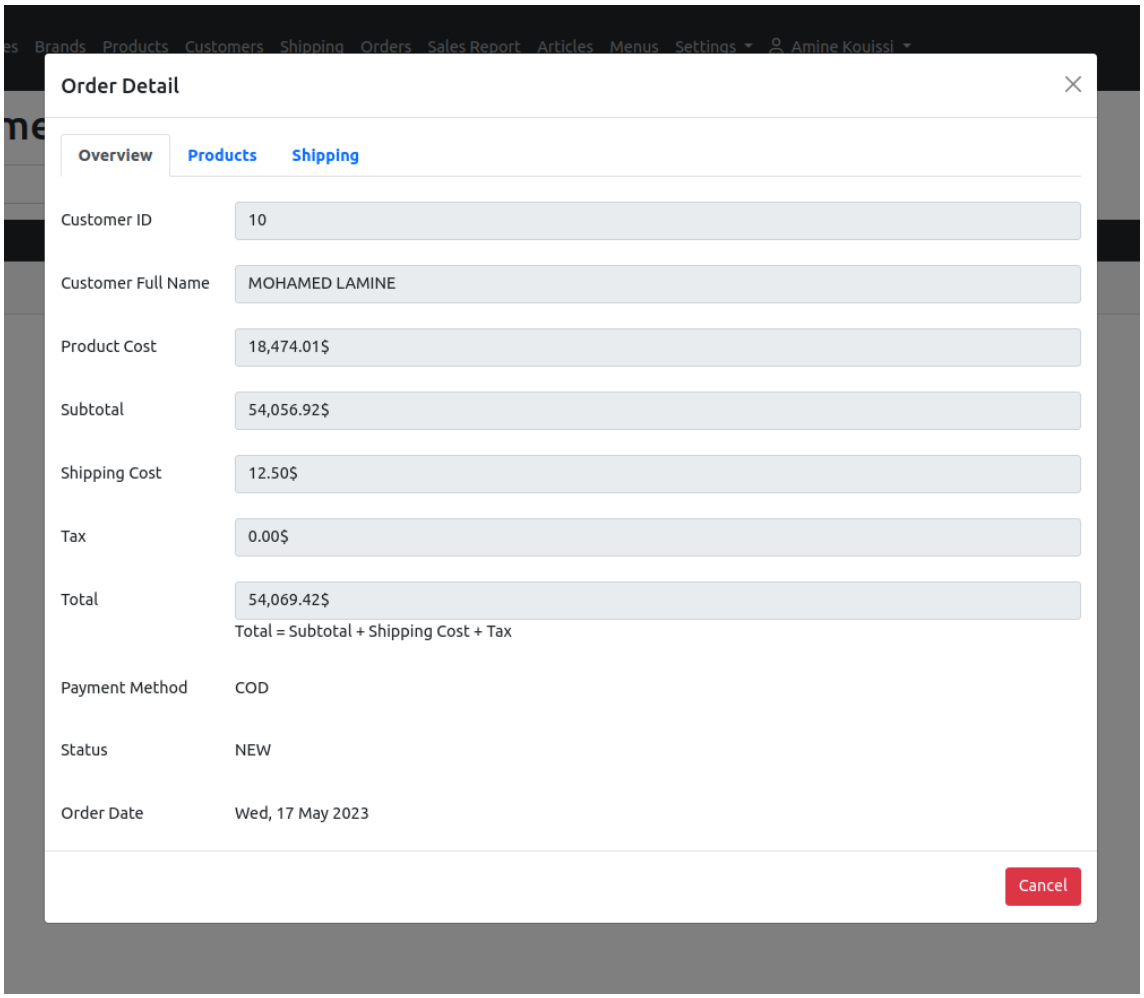

Figure 4.8: Order Detail :Overview

The Order Overview Section The Order Detail Page displays comprehensive information about a specific customer's placed order. It is organized into three distinct sections: Overview, Products, and Shipping. The Overview section presents essential details about the customer, payment method, and relevant information regarding the order as a whole, including status, cost, tax, and more.

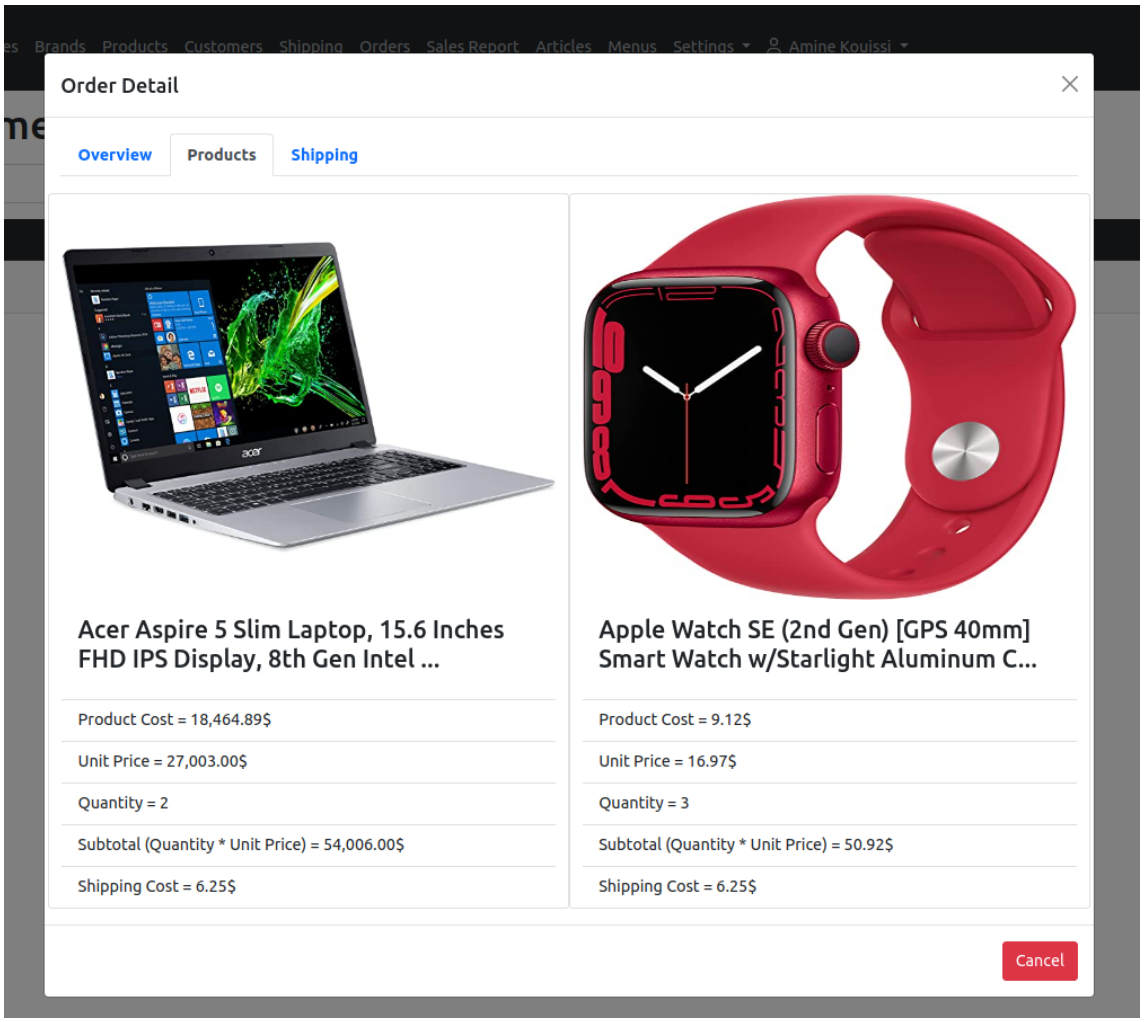

Figure 4.9: Order Detail :Products

Th Ordered Products Section Within the Order Detail Page, the Products section meticulously presents the products ordered by the customer, along with their corresponding information such as quantity, subtotal, shipping cost, and other pertinent details.

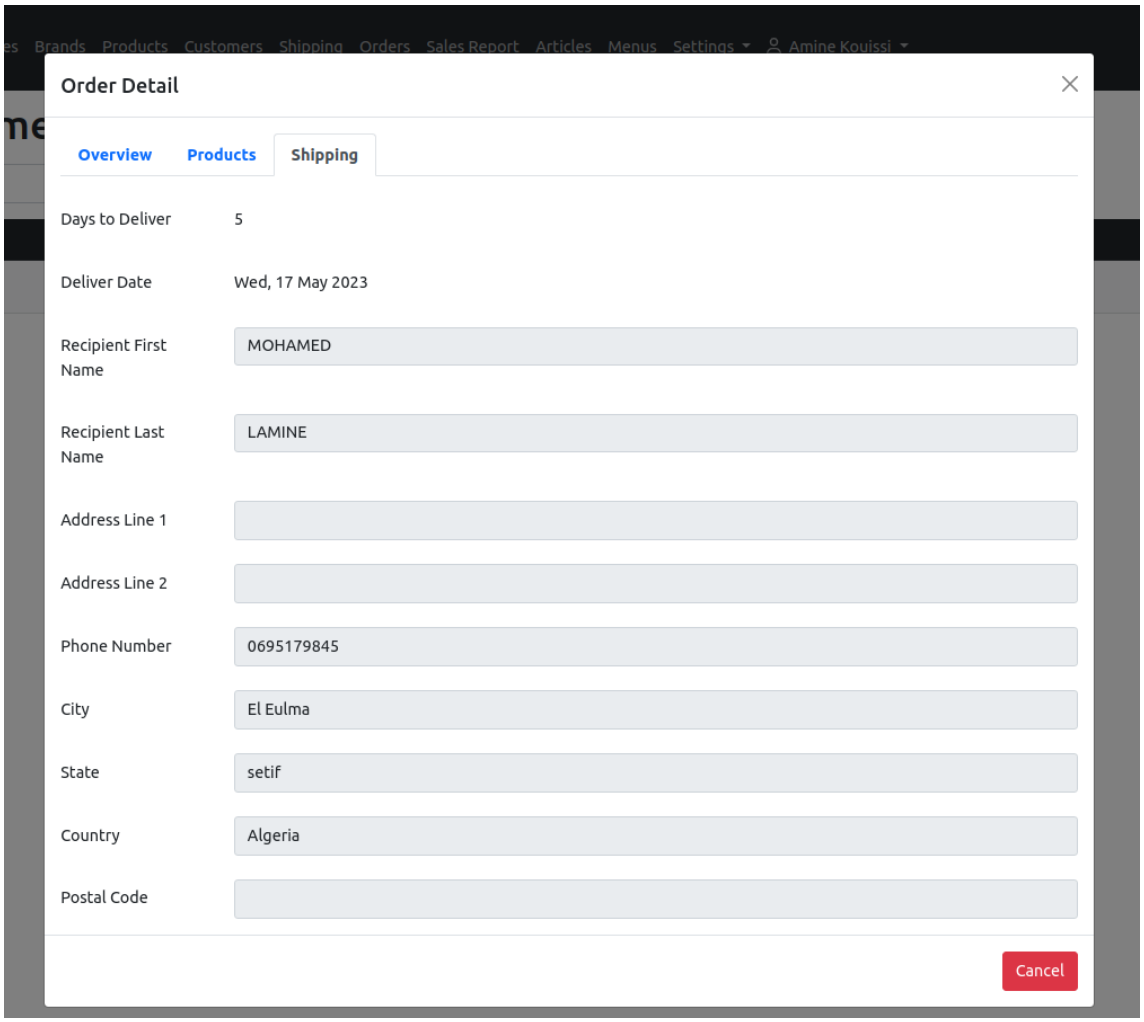

Figure 4.10: Order Detail :Shipping

The Order Shipping Information Section showcases the recipient's details, including their address, as well as the estimated number of days required for order delivery.

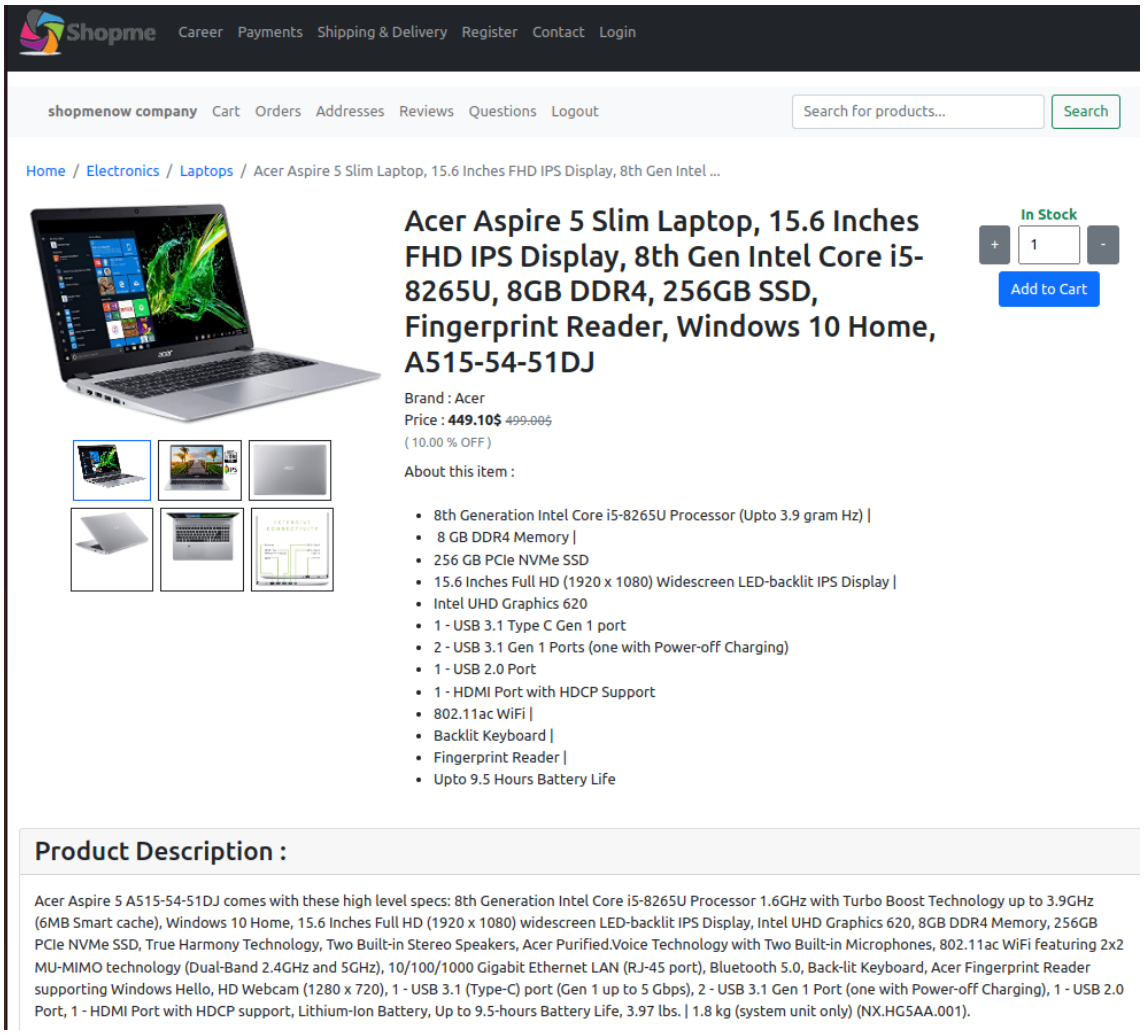

Figure 4.11: Product Detail Page

The Product Detail Page is a publicly accessible page that showcases comprehensive information about the product. It presents details such as the product name, images, features, short description, full description, and more.

4.4 Conclusion

This chapter encompasses a comprehensive discussion of the hardware and software employed for the development, testing, and deployment of our platform. Furthermore, we present a visual representation of the platform's functionalities through the inclusion of screenshots showcasing its various interfaces.

General Conclusion

This research has been driven by the challenges faced by medium businesses in effectively managing, scaling, promoting collaborative teamwork, and optimizing the performance of their software systems. In response to these challenges, we have undertaken the development of a meticulously crafted modular software architecture that prioritizes modularity and independent components. This architectural approach empowers applications to meet a diverse range of business requirements while facilitating the seamless addition, modification, or removal of features. Moreover, the architecture's high level of loose coupling between modules enables disparate development teams to operate independently and efficiently, thereby mitigating the risk of systemic failures.

Our decision to develop a comprehensive e-commerce application using this architecture was motivated by the fact that e-commerce applications often encounter the very issues that a modular software architecture effectively addresses. Through this application, we have sought to demonstrate the potential and applicability of our architectural approach.

Nevertheless, it is important to acknowledge that our modular software architecture does possess certain limitations. One such limitation is the complexity of interactions. As the number of modules increases, the management of their interactions and dependencies becomes progressively challenging, necessitating meticulous planning and design to establish well-defined interfaces and communication protocols between modules. The intricacy of these interactions can introduce overhead and potential points of failure.

Effective communication and coordination between modules represent another critical aspect that requires attention in a modular architecture. Striking the right balance between inter-module communication and minimizing dependencies is crucial for maintaining the benefits of modularity. Excessive inter-module communication or tight coupling can undermine the advantages afforded by the architecture's modular nature.

Furthermore, designing a modular architecture entails careful consideration of module boundaries, responsibilities, and interactions, adding an additional layer of complexity to the overall design process. This calls for skilled architects and developers proficient in modular design principles. Inexperienced teams may encounter challenges in effectively implementing and maintaining a modular architecture.

Future Work

Regarding future work, it is valuable to investigate alternative architectural patterns and paradigms that can complement or enhance our existing modular architecture. Furthermore, the development of programming language frameworks that streamline the adoption and utilization of this architecture holds significant potential for improving its ease of use and expanding its adoption in the industry.

What we have learned

During the course of our project, we have undergone a transformative learning experience that has profoundly influenced our understanding of software architecture. We have acquired new knowledge and skills, which have greatly shaped our perspectives in this field.

Firstly, we have delved into the exploration of various architectural patterns. Through meticulous study and analysis, we have gained insights into the intricacies and benefits of different architectural approaches, broadening our architectural repertoire.

Additionally, we have ventured into the realm of architectural paradigms, delving into new conceptual frameworks and design principles. This has enabled us to broaden our horizons and consider alternative perspectives in shaping robust and scalable software architectures.

Furthermore, we have expanded our programming language proficiency. By mastering a new programming language, we have equipped ourselves with a powerful tool for implementing innovative and efficient solutions within our software architecture.

In our pursuit of excellence, we have embraced new programming tools and technologies. These tools have significantly facilitated the development process, enabling us to enhance the functionality and performance of our platform while streamlining our workflow.

Lastly, we have honed our skills in research methodologies and organization. Through careful and systematic research, we have acquired the ability to conduct thorough investigations, gather meaningful data, and effectively organize our findings. This skill set has proved invaluable in advancing our understanding and contributing to the body of knowledge in software architecture.

Overall, this journey of learning and discovery has been instrumental in refining our perspective on software architecture, equipping us with the necessary tools and knowledge to design and develop robust, scalable, and efficient software systems.

Bibliography

[1] altexsoft.com. *Functional requirements*. 2023. URL: https://www.altexsoft.com/blog/business/functional-and-non-functional-requirements-specification-and-types/.

[2] amazon. *What is Amazon S3?* URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

[3] baeldung. *Difference Between MVC and MVP Patterns*. 202. URL: https://www.baeldung.com/mvc-vs-mvp-pattern.

[4] baeldung. *Thymeleaf*. URL: https://www.baeldung.com/spring-template-engines.

[5] castsoftware. *What Is Software Architecture?* URL: https://www.castsoftware.com/glossary/what-is-software-architecture-tools-design-definition-explanation-best.

[6] chrome. *Chrome DevTools Overview*. URL: https://developer.chrome.com/docs/devtools/overview/.

[7] Mark Drake. *What is MySQL?* 2020. URL: https://www.digitalocean.com/community/tutorials/what-is-mysql.

[8] Ishan Gaba. *What Is JUnit: An Overview of the Best Java Testing Framework*. 2023. URL: https://www.simplilearn.com/tutorials/java-tutorial/what-is-junit.

[9] Rosa Tiara Galuh. *MVC, MVP, MVVM: Which One to Choose?* 2022. URL: https://www.makeuseof.com/mvc-mvp-mvvm-which-choose/.

[10] Chandler Harris. *Microservices vs monolithic architecture*. 2023. URL: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith.

[11] heroku. *What is Heroku?* URL: https://www.heroku.com/about.

[12] IBM. *Technical benefits*. URL: https://www.ibm.com/topics/java.

[13] ibm. *What is Java Spring Boot?* URL: https://www.ibm.com/topics/java-spring-boot.

[14] jetbrains. *IntelliJ IDEA overview*. URL: https://www.jetbrains.com/help/idea/discover-intellij-idea.html#IntelliJ-IDEA-editions.

[15] Peter Loshin. *What is Structured Query Language (SQL)?* 2022. URL: https://www.techtarget.com/searchdatamanagement/definition/SQL.

[16] lucidchart.com. *Use Case Diagram*. 2023. URL: https://www.lucidchart.com/pages/uml-use-case-diagram.

[17] lucidchart.com. *What is Unified Modeling Language*. 2023. URL: https://www.lucidchart.com/pages/what-is-UML-unified-modeling-language.

[18] Robert C. Martin. *The Clean Architecture*. 2012. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html.

[19] Sam Millington. *A Solid Guide to SOLID Principles*. 2022. URL: https://www.baeldung.com/solid-principles.

[20] Samuel Oloruntoba. *SOLID: The First 5 Principles of Object Oriented Design*. 2021. URL: https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design#single-responsibility-principle.

[21] oracle. *Java Programming Language*. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html.

[22] oracle. *What is MySQL?* URL: https://www.oracle.com/mysql/what-is-mysql/.

[23] postman. *API testing*. URL: https://www.postman.com/product/tools/.

[24] postman. *What is Postman?* URL: https://www.postman.com/product/what-is-postman/.

[25] Amazon Web Services. *What are Microservices?* 2023. URL: https://aws.amazon.com/microservices/.

[26] Amit Shekhar. *Go Backend Clean Architecture*. 2023. URL: https://amitshekhar.me/blog/go-backend-clean-architecture.

[27] thymeleaf. *Thymeleaf*. URL: https://www.thymeleaf.org/.

[28] tutorialspoint. *Mockito Tutorial*. URL: https://www.tutorialspoint.com/mockito/index.htm.

[29] Ubuntu. *About the Ubuntu project*. URL: https://ubuntu.com/about.

[30] Ubuntu. *Ubuntu Desktop for developers*. URL: https://ubuntu.com/desktop/developers.