MedTech
Mediterranean
Institute of Technology

TEKERS

ENGINEERING PROGRAM

FINAL REPORT

A Scalable and Customizable Multi-Tenant E-Commerce Platform

BY

Najla Seghaier

ACADEMIC SUPERVISOR

Mrs. Asma Amdouni

INSTITUTION SUPERVISOR

Mr. Oussama Trabelsi

TEKERS

Tunis, 2022-2023

Approval

APPROVED BY

ACADEMIC SUPERVISOR

Name	Signature	Date

COMPANY SUPERVISOR

Name	Signature	Date

ACADEMIC EVALUATOR

Name	Signature	Date

Declaration

I certify that I am the author of this project and that any assistance I received in its preparation is fully acknowledged and disclosed in this project.

I have also cited any source from which I used data, ideas, or words, either quoted or paraphrased. Further, this report meets all the rules of quotation and referencing in use at MedTech, as well as adheres to the fraud policies listed in the MedTech honor code.

No portion of the work referred to in this study has been submitted in support of an application for another degree or qualification to this or any other university or institution of learning.

I also certify that this final version of my capstone project report includes the corrections and comments mentioned by the jury members and it is submitted online on Moodle.

Student Name	Signature	Date
Najla Seghaier		09/18/2023

Work Term Release

I hereby state and verify by my signature that I have reviewed this report. I hereby affirm that the report contains

☐ no confidential data/information, and I authorize it to be released.

☐ confidential data/information, and I do not authorize it to be released.

COMPANY SUPERVISOR

Name	Signature	Date

Abstract

This capstone project report documents my experience developing a multi-tenant e-commerce web platform for the startup TEKERS. The project was undertaken as part of my graduation requirements at MedTech University. The platform is designed to provide TEKERS' customers with an easy-to-use and flexible e-commerce solution that can be customized to meet their specific business needs.

The project utilized the MERN stack, which includes MongoDB, Express.js, React.js, and Node.js, to build a scalable and modular platform that can accommodate multiple tenants. The platform includes a comprehensive set of features, such as product management, shopping cart, streamlined checkout process, and efficient order management.

While the initial plan included the central management of social media posts, the project scope was adjusted due to time constraints and prioritization of other functionalities. Instead, the development focused on enhancing the platform's customization capabilities, allowing TEKERS' customers to tailor their online stores to align with their branding and customer experience preferences effectively.

The final deliverable is a scalable, customizable, and secure multi-tenant e-commerce web platform ready for deployment. The platform has been thoroughly tested to ensure its reliability and functionality.

Keywords: E-Commerce, Multi-Tenant, MERN Stack, Customization, Scalability, Security, Software Development.

Acknowledgements

This capstone project marks the culmination of my studies at MedTech University and my experience working with TEKERS. It has been a long and challenging journey, but also an immensely rewarding one, thanks to the knowledge gained and the network of great people that supported and guided me throughout the project.

First and foremost, I extend my heartfelt gratitude to the esteemed professors at MedTech University for providing me with a solid foundation in computer science and software engineering. Their dedication to fostering knowledge and encouraging critical thinking has been crucial in shaping my skills as a software engineer.

Special thanks to my academic supervisor, Mrs. Asma Amdouni, for her unwavering support and mentorship. Her valuable insights and feedback have been instrumental in shaping this project.

I am deeply grateful to my company supervisor, Mr. Oussama Trabelsi, for his continuous support, expertise, and guidance during the development of the multi-tenant e-commerce platform at TEKERS.

To the entire TEKERS team, thank you for giving me this opportunity to work on such a meaningful project. Your collaborative spirit and support have made this journey both enjoyable and rewarding.

Lastly, I want to express my immense appreciation to my family and friends for their unwavering encouragement and understanding throughout this endeavor. Your support has been the cornerstone of my determination during the challenging times of this journey.

Thank you all for being an essential part of this project and for making it a remarkable learning experience.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Abbreviation	Definition
AES	**A**dvanced **E**ncryption **S**tandard
API	**A**pplication **P**rogramming **I**nterface
BE	**B**ack-**E**nd
BSON	**B**inary J**SON**
CRUD	**C**reate, **R**ead, **U**pdate, **D**etele
CTA	**C**alls-**T**o-**A**ction
DAO	**D**ata **A**ccess **O**bject
DoS	**D**enial-**o**f-**S**ervice
DSR	**D**esign **S**cience **R**esearch
FE	**F**ront-**E**nd
GAE	**G**oogle **A**pp **E**ngine
GUI	**G**raphical **U**ser **I**nterface
HTTP	**H**ypertext **T**ransfer **P**rotocol
HTTPS	**H**ypertext **T**ransfer **P**rotocol **S**ecure
JWT	**J**SON **W**eb **T**oken
MDC	**M**ultiple **D**imensions **C**lustering
MTA	**M**ulti-**T**enant **A**rchitecture
QoS	**Q**uality **o**f **S**ervice
RBAC	**R**ole-**B**ased **A**ccess **C**ontrol
SaaS	**S**oftware **a**s **a** **S**ervice
SDLC	**S**oftware **D**evelopment **L**ife**c**ycle
SEO	**S**earch **E**ngine **O**ptimization
SLA	**S**ervice **L**evel **A**greement
SIEM	**S**ecurity and **I**nformation **E**vent **M**anagement
SOA	**S**ervice-**O**riented **A**rchitecture
SMB	**S**mall and **M**edium-**S**ized **B**usiness
TPS	**T**ransactions **P**er **S**econd
UI	**U**ser **I**nterface
URL	**U**niform **R**esource **L**ocator

1. INTRODUCTION

In today's digital age, e-commerce has evolved into an essential component of the modern business landscape. As an increasing number of businesses shift their operations online, the need for e-commerce solutions that are efficient, reliable, and scalable has grown significantly. To address this demand, businesses are turning to multi-tenant e-commerce platforms, offering the unique advantage of allowing multiple merchants to sell their products and services through a single, centralized platform. Leveraging the economies of scale benefits, these platforms represent a cost-effective approach for merchants to reach their target audience and foster growth in the competitive digital market.

1.1. Project Context

This capstone project was conducted in collaboration with TEKERS, a newly founded start-up that began its journey in 2021 as a consulting firm working on a Software as a Service (SaaS) project for a European customer. As part of its expansion and growth strategy, TEKERS aims to establish its presence in the Tunisian market by developing innovative SaaS products.

Figure 1: TEKERS Logo

TEKERS' vision is to provide technology solutions tailored to the needs of Small and Medium-sized Businesses (SMBs), empowering them with efficient and scalable e-commerce tools. In pursuit of this vision, TEKERS embarked on the development of a multi-tenant e-commerce web platform. The platform's primary objective is to offer a versatile and cost-effective solution, enabling multiple merchants to sell their products and services through a centralized platform.

The motivation behind this study arises from several factors that highlight the importance of developing a multi-tenant e-commerce platform tailored to the evolving needs of modern businesses:

1. **Empowering SMBs:** With the increasing popularity of social networks as selling platforms, a growing number of small businesses are establishing their digital presence to reach wider audiences. However, the lack of dedicated e-commerce platforms can hinder their growth potential. The project aims to provide these businesses with a cost-effective solution, leveraging the principle of economy of scale, enabling them to

transcend the limitations of social networks and enjoy the benefits of a comprehensive e-commerce platform.

2. **Versatility and Adaptability:** The platform's flexibility allows it to surpass domain-specific limitations and be applicable to a wide range of industries, accommodating the diverse needs of businesses across various sectors. By providing such a generic solution, the platform becomes a powerful resource for enterprises looking to establish a robust and effective online store without the burden of developing custom solutions from scratch.

3. **Fostering Unique Identities:** The success of an e-commerce platform lies in its ability to offer merchants the freedom to craft a distinct and memorable identity for their stores. By incorporating powerful customization options, our platform empowers merchants to personalize their stores, creating a unique shopping experience for their customers. This sense of individuality reinforces brand loyalty and sets their businesses apart in the competitive online market.

4. **Enhanced User Experience:** By developing a user-friendly and intuitive platform, our project endeavors to enhance customer satisfaction, encourage repeat purchases, and drive positive recommendations.

5. **Scalability and Growth:** As businesses evolve and expand, they require an e-commerce platform capable of scaling alongside their growth. Our project seeks to deliver a scalable solution, ensuring that businesses can seamlessly accommodate increased traffic, larger product catalogs, and higher order volumes without compromising on performance and efficiency.

By addressing these motivating factors, our multi-tenant e-commerce platform aims to revolutionize the way businesses conduct online operations in the Tunisian market, providing them with a powerful and adaptable toolset to thrive in the competitive digital market.

1.2. Problem Statement

In the dynamic landscape of modern commerce, the paradigm of business operations has been significantly redefined by the emergence of e-commerce. This transformative shift has resulted in a high level of convenience, offering shoppers an accessible and streamlined avenue to procure products and services online. Yet, as the digital sphere becomes increasingly saturated with a multitude of e-commerce websites, businesses confront different challenges in proving their online presence, capturing new markets, and nurturing customer engagement.

A paramount challenge emerging in this environment is the necessity for businesses to establish an online identity, while faced with the practical limitations of financial, temporal, and technical resources. These factors often hinder their ability to construct their own e-commerce platforms from the ground up. Consequently, many enterprises turn to social media platforms like Facebook and Instagram to establish an online niche. While this offers a semblance of digital presence, it falls short of providing an effective and comprehensive solution. Such platforms lack the structural architecture required for streamlined operations, encompassing attributes like categorization, searchability, and complex filtering mechanisms. Additionally, these platforms undermine effective order management and practical tracking of purchase history, precipitating a cycle of extended processes and elevated costs that burden both the merchant and the shopper.

The repercussions of these challenges are acutely felt by SMBs, often constrained by limited resources and expertise. Many SMBs find themselves excluded from the pursuit of developing and upholding their own e-commerce websites, hindered by the high costs involved.

In light of these issues, the focal thrust of our capstone project emerges in the form of a multi-tenant e-commerce web platform, an agile solution poised to empower businesses to effectively address their online presence. Equipped with the principle of customization, this platform seeks to furnish enterprises with the necessary suite of tools to sculpt their digital domain according to their unique vision, thereby enabling them to establish a distinctive and compelling presence in the competitive digital world.

Furthermore, scalability, security, and maintainability are ongoing issues that many businesses face when developing and maintaining their own e-commerce websites. With multi-tenancy, these issues are alleviated as businesses can take advantage of the shared infrastructure and resources of the platform. Our project will aim to address these issues and provide a secure and scalable solution for businesses.

Hence, this capstone project embarks on a journey not merely to tackle these diverse challenges but to take a step forward towards digitalizing the Tunisian commerce landscape, offering businesses a secure, scalable, and customizable solution to thrive in the digital age and catalyze the trajectory of success in the ever-evolving e-commerce domain.

1.3. Purpose of the Study

The primary goal of this capstone project is to develop a robust and adaptable multi-tenant e-commerce web platform tailored to address the unique challenges faced by businesses in the dynamic digital marketplace. By harnessing the power of modern technology and innovative design, the platform aims to provide businesses, especially SMBs, with a comprehensive and user-friendly solution for establishing and managing their online presence. This ambitious initiative seeks to empower merchants with the tools to efficiently showcase their products, engage customers, manage orders, and streamline their operations.

Moreover, this project endeavors to promote a culture of customization, allowing each business to introduce in their digital storefront a distinct identity that resonates with their target audience. By enabling merchants to tailor their online stores according to their brand values and customer preferences, the platform aspires to enhance customer experiences and foster deeper connections between businesses and their clientele.

Another objective of our study is to address the financial challenges faced by SMBs in developing their own websites. By offering a multi-tenant platform that leverages a shared infrastructure and resources, we aim to provide a more cost-effective and scalable solution that can be easily maintained by businesses of all sizes.

Furthermore, the platform's Multi-Tenant Architecture (MTA) will address the challenges of scalability, reliability, security, and maintainability that often burden individual businesses venturing into e-commerce. By implementing best practices in security and performance, the platform aims to provide a robust environment that empowers businesses to focus on growth and innovation, rather than struggling with technical complexities.

Ultimately, the purpose of this study is to contribute to the advancement of e-commerce solutions by creating a platform that bridges the gap between business aspirations and technological and financial capabilities. Through this project, we aim to empower businesses to join the digital landscape, ensuring their competitiveness and success in the challenging world of online commerce.

1.4. Research Questions

The study aims to address the challenges faced by SMBs in managing their online stores. To guide our investigation, we have formulated the following research questions:

- **What are the key design considerations and trade-offs in implementing a multi-tenant system?**
- **How can a shared platform meet the specific needs of different businesses?**
- **How can an e-commerce platform be designed and developed to meet the quality requirements of scalability, security, and maintainability?**

By addressing these research questions, we aim to provide insights and recommendations for developing a multi-tenant e-commerce platform that meets the quality requirements of scalability, security, and maintainability, while also optimizing the user experience to improve customer engagement.

1.5. Approach and Boundaries of the Study

1.5.1. Research Approach

The study adopts the design science research approach, which is a problem-solving approach aimed at developing innovative solutions to real-world problems. This approach involves iteratively designing and evaluating artifacts, such as software systems, to address specific problems or opportunities. The design science approach emphasizes the importance of rigorously evaluating the effectiveness of the proposed solution and its impact on the problem domain.

By adopting the design science approach, we aim to develop a multi-tenant e-commerce platform that meets the needs of SMBs and provides a cost-effective, scalable, secure, and easy-to-maintain solution for conducting e-commerce transactions. The platform will be designed and evaluated based on the quality requirements and best practices identified in the literature.

1.5.2. Scope

The scope of this study is to develop a multi-tenant e-commerce platform that meets the needs of SMBs. The study will focus on the design and development of the platform, taking into consideration the quality requirements of scalability, security, and maintainability, as well as the user experience and interface design. The study will not cover the marketing or business strategy aspects of e-commerce, nor will it cover the development of mobile applications or other types of software. The scope of the study is limited to the development of a web-based platform that is accessible via desktop and mobile web browsers.

1.5.3. Limitations

It's important to note that this study has several limitations that should be taken into account when interpreting the results. First, the scope of this study is limited to the development of an e-commerce platform for businesses of all sizes, with a particular focus on SMBs. While the developed platform may be applicable to larger enterprises, this has not been the primary focus of the study. Given that SMBs face unique challenges in developing their own e-commerce websites, we believe that this focus is justified.

Second, our solution does not specifically address other digital marketing areas such as email campaigns, SEO (Search Engine Optimization), or paid search. These areas may be relevant to the broader digital marketing landscape but are beyond the scope of this study.

Third, the study is limited to a specific geographical region, and the findings may not be applicable in other regions with different market conditions and regulatory frameworks. For example, differences in consumer behavior, competition, or legal requirements may affect the feasibility or success of the platform in other regions.

Despite these limitations, the study provides valuable insights into the design and development of multi-tenant e-commerce platforms, with a focus on security, scalability, and maintainability. Our creative approach to providing a generic yet customizable platform is a noteworthy contribution.

1.5.4. Delimitations

The scope of this study is delimited to the development of a multi-tenant e-commerce platform for businesses. The platform is designed to meet the specific needs of SMBs, and the study focuses exclusively on the technical development of the platform, rather than evaluating its effectiveness in the marketplace or its impact on specific businesses.

Furthermore, the study is limited to the development of the platform using a specific technical environment, which includes the use of the MERN stack. While this technical environment is widely used and well-suited to the development of e-commerce platforms, the findings of the study may not be directly applicable to other technical environments.

These delimitations are important to consider when interpreting the results of the study, as they define the specific focus of the research and its limitations.

1.5.5. Assumptions

The following assumptions were made in this study. First, it is assumed that the user has a basic understanding of web technologies and e-commerce concepts. While the platform is designed to be user-friendly and accessible to businesses of all sizes, some level of technical knowledge is required to effectively use the platform. Second, it is assumed that the business has already identified the products and services that it wishes to sell and has the necessary resources to manage and fulfill orders. Finally, it is assumed that the platform will be hosted on a reliable and secure server and that the necessary maintenance and updates will be performed on a regular basis to ensure optimal performance.

1.6. Contributions

The main technical contributions of this work encompass:

- The design of a scalable and maintainable platform architecture that can support the growth and evolution of businesses over time.
- The development of a multi-tenant e-commerce platform that allows businesses to create and manage their own online stores, with support for customizable themes, as well as products and orders management.

- The integration of various security measures to protect the platform and its users from common web-based attacks, such as Denial-of-Service (DoS) attacks.

The main scientific contributions of this work encompass:

- The application of the design science research methodology to the development of an e-commerce platform, which provides a systematic and rigorous approach to problem-solving and knowledge creation.

- The exploration of the concept of multi-tenancy in e-commerce platforms, which allows multiple businesses to share a single instance of the platform while maintaining their own independent identities and data.

- The investigation of Architectural approaches taking into consideration the intricate demands of multi-tenancy, code maintainability, and the financial limitations often faced by SMBs.

- The introduction of a novel approach that leverages the multi-tenancy concept to provide customization options, granting merchants the ability to curate distinct and personalized online storefronts, enhancing their engagement with customers and fostering brand loyalty.

Collectively, these contributions advance our understanding of e-commerce platform development, multi-tenancy application, architectural considerations, and customization possibilities. This study offers valuable insights for researchers, practitioners, and businesses aiming to thrive in the evolving digital commerce landscape.

1.7. Structure

This report is structured as follows: Section 2 provides the necessary background and literature review for the study, including relevant theoretical concepts and existing literature in the field of multi-tenant e-commerce platforms and SaaS applications in general. Section 3 outlines the methodology employed for this study. The subsequent Section 4 explains the diverse iterations undertaken during the course of this research. In Section 5, the results and findings of the study are presented, addressing each of the research questions. Section 6 then discusses the implications of the findings both on the research and industry, while also presenting potential threats to validity. Finally, Section 7 summarizes the major findings of the study and offers specific recommendations for future work in this area.

2. BACKGROUND and LITERATURE REVIEW

In this chapter, we provide a comprehensive background on the concepts and theories that underpin our project. We begin by defining key terms and concepts that are essential to understanding our study. Then, we review the existing literature to contextualize our research within the current state of the field.

2.1. Background

In this section, we will provide a theoretical background that is essential to understand our project. We will define key concepts and terms related to e-commerce and multi-tenancy and software engineering in general. By establishing this foundation, we aim to provide readers with the necessary background knowledge to engage with the rest of the report.

- **E-commerce**

E-commerce, short for electronic commerce, refers to the buying and selling of goods or services over the internet or other electronic networks. E-commerce includes a wide range of online activities such as online shopping, electronic payments, online auctions, and internet banking. E-commerce has become increasingly popular due to the convenience and speed of online transactions, as well as the global reach that the internet provides.

- **Software as a Service (SaaS)**

SaaS is a software delivery model in which a provider hosts and delivers software applications over the internet to customers on a subscription basis. The provider manages the infrastructure, security, and maintenance of the software, while customers access and use the software through a web browser or thin client. SaaS applications can range from simple productivity tools to enterprise-level systems and are designed to be scalable and customizable to meet the specific needs of each customer. This model allows businesses to access and use software without the need for expensive hardware, IT staff, or ongoing maintenance costs.

- **Multi-tenancy**

Multi-tenancy is a software architecture concept where a single instance of an application is shared by multiple organizations or tenants, while providing each tenant with a separate and isolated environment. In a multi-tenant system, each tenant has their own set of

data, configurations, and functionalities, while sharing the same underlying application and infrastructure. Multi-Tenancy is commonly used in SaaS applications, where multiple customers can access the same software application via the internet, but with their own isolated data and configurations. This approach offers benefits such as cost efficiency, scalability, and customization options for different tenants. However, it also requires careful management of security, data privacy, and performance to ensure the isolation and availability of each tenant's data and resources.

- **Software Development Lifecycle (SDLC)**

The SDLC is a structured process that outlines the stages and activities involved in developing software applications. It encompasses planning, designing, building, testing, deploying, and maintaining software products. The SDLC provides a systematic framework for managing and guiding the development process, ensuring that software is developed efficiently, meets quality standards, and aligns with the needs and objectives of the project.

- **MERN stack**

The MERN stack is a set of technologies used for building web applications. It consists of four main components: MongoDB, Express.js, React.js, and Node.js.

  - MongoDB is a document-oriented NoSQL database that stores data in JSON-like documents.

  - Express.js is a web application framework for Node.js that provides a set of features for building web applications, such as routing and middleware.

  - React.js is a JavaScript library for building user interfaces. It enables developers to build complex and interactive User Interface (UI) components using declarative syntax.

  - Node.js is a JavaScript runtime environment that allows developers to build server-side applications using JavaScript. It provides a set of Application Programming Interfaces (APIs) for building scalable and high-performance web applications.

Together, these technologies provide a full-stack solution for building modern web applications that can handle large amounts of data and traffic.

- **ESLint**

ESLint is a popular open-source JavaScript linting tool that is used to analyze and identify potential issues, errors, and coding style inconsistencies in JavaScript code. It provides developers with guidelines and suggestions to improve the quality, readability, and

maintainability of their code. ESLint can be customized to enforce specific coding standards, best practices, and coding conventions, making it a valuable tool for ensuring code quality in projects.

- **Application Programming Interface (API)**

API is a set of protocols, routines, and tools for building software applications. It specifies how software components should interact and provides a standard way for different software components to communicate with each other. APIs enable developers to create software applications that can interact with other applications, systems, or services, regardless of the underlying technology or programming language. APIs can be used to access data, functionality, or services from a remote system, and they are often used to enable integration between different software systems or to build software applications that can be used across different platforms and devices.

- **Sharding**

Sharding is a database design technique that involves horizontally partitioning data across multiple servers or nodes. Each shard holds a distinct subset of the data, allowing for efficient distribution and management of large datasets. Sharding aims to enhance performance and scalability by enabling databases to handle increased workloads and data volumes.

- **Scalability**

Scalability refers to the ability of a system, process, or organization to handle an increasing amount of work, or to grow in size, capacity, or performance, without being significantly hindered by bottlenecks or decreased efficiency. In other words, scalability refers to the ability of a system to expand and handle increased demand without affecting its performance or requiring significant changes to its underlying architecture or infrastructure. It is an important aspect of software design and architecture, particularly in cloud computing environments where applications need to be able to scale dynamically to meet changing demand. Scalability can be achieved through various techniques such as load balancing, horizontal and vertical scaling, caching, and clustering, among others.

- **Vertical scaling**

Vertical scaling, also known as scaling up, refers to the process of increasing the capacity or power of a single computer or server by adding more resources, such as RAM, CPU, or storage. This is typically done by upgrading the existing hardware components, such as replacing a CPU with a faster one or adding more memory to the system. Vertical scaling is

often used to improve the performance of applications that require more resources as their workload increases, such as databases or web servers.

- **Horizontal scaling**

Horizontal scaling refers to the process of adding more computing resources to a system to handle increasing workloads. This can be done by adding more servers, nodes, or other resources in parallel to the existing ones. Horizontal scaling is an important concept in the context of cloud computing and distributed systems, as it allows applications to scale seamlessly and dynamically to meet changing demands. Horizontal scaling is often contrasted with vertical scaling, which involves adding more resources to a single machine or server. Horizontal scaling is generally considered more cost-effective and flexible than vertical scaling, as it allows applications to scale horizontally by adding more commodity hardware, rather than investing in expensive high-end hardware.

- **Load balancing**

Load balancing is a technique used to distribute incoming network traffic across multiple servers or resources, with the goal of improving overall system performance, availability, and reliability. Load balancing enables efficient resource utilization by ensuring that no single resource is overburdened while others remain idle. In practice, load balancing can involve multiple strategies, including round-robin scheduling, geographic proximity, and real-time traffic monitoring, among others. Load balancing can be achieved through a variety of methods, including hardware-based solutions, software-based solutions, and cloud-based solutions. The ultimate goal of load balancing is to ensure that traffic is distributed evenly across resources, resulting in optimized performance and reduced downtime.

- **Caching**

Caching is a technique used in computing to store frequently accessed data or resources in a temporary, high-speed storage location, known as a cache. The purpose of caching is to accelerate data retrieval and enhance system performance by reducing the need to fetch data from slower, primary storage locations. When a request for data is made, the system first checks the cache. If the data is found in the cache, it can be retrieved more quickly than if it were to be fetched from the original source, such as a database or a remote server. Caching is particularly effective for optimizing repetitive or resource-intensive operations, resulting in improved response times, reduced network traffic, and overall smoother user experiences.

- **Security**

Security refers to the measures and practices implemented to protect systems, networks, and applications from unauthorized access, use, disclosure, disruption, modification, or destruction. It involves identifying and assessing potential risks, vulnerabilities, and threats to an organization's assets, and then implementing appropriate controls to mitigate those risks. These controls can include various technologies, such as firewalls, encryption, access controls, intrusion detection and prevention systems, and Security Information and Event Management (SIEM) systems, as well as policies, procedures, and training programs to ensure that employees and other authorized users are aware of security risks and know how to prevent and respond to security incidents. The goal of security is to ensure the confidentiality, integrity, and availability of an organization's information and systems, as well as to comply with legal and regulatory requirements and industry standards.

- **Data isolation**

Data isolation is a technique that ensures that each tenant's data is separated and protected from other tenants' data in a multi-tenant system. It represents an important consideration in multi-tenant systems to ensure that tenants' data is protected and confidential. It helps prevent data breaches, accidental or intentional data modification or deletion, and unauthorized access to sensitive data.

- **Containers**

Containers are lightweight, standalone, and executable software packages that contain everything needed to run a piece of software, including the code, runtime, libraries, and system tools. They provide a consistent and isolated environment for running applications, ensuring that they behave the same across different environments.

- **Containerization**

Containerization is the process of packaging an application and its dependencies into a single container image. This image can be executed consistently across various computing environments, from development to testing to production.

- **Image**

An image is a lightweight, stand-alone, and executable software package that includes the code and its runtime dependencies. Images are the building blocks of containers. They can be versioned and shared among developers and systems.

- **Dockerfile**

A Dockerfile is a text file that contains instructions for building a Docker image. It specifies the base image, desired software installations, configurations, and commands required to create a reproducible container image.

- **Service-Oriented Architecture (SOA)**

SOA is an architectural approach that structures an application as a collection of loosely coupled and interoperable services. Each service represents a discrete unit of functionality and can be independently developed, deployed, and maintained. SOA aims to enhance reusability, scalability, and flexibility by promoting the creation of services that communicate through well-defined interfaces.

- **Monolithic architecture**

Monolithic Architecture refers to a software design where an entire application is built as a single, interconnected unit. In this approach, all components and features of the application are tightly integrated within the same codebase. Monolithic architectures are characterized by their simplicity and centralized management but may pose challenges in terms of scalability and maintenance as the application grows.

- **Microservices architecture**

Microservices Architecture is an architectural style that decomposes an application into a collection of small, independent services. Each service is responsible for a specific functionality and can be developed, deployed, and scaled separately. Microservices architectures aim to promote modularity, scalability, and agility, allowing teams to work on different services simultaneously and enabling the application to easily adapt to changes.

2.2. Existing Related Studies

In this section, we delve into a comprehensive exploration of existing related studies. Our investigation encompasses various critical aspects, including Data Architecture, System Architecture, Customization, Security, Scalability, and Maintenance, all within the context of multi-tenancy. By analyzing these studies, we aim to gain valuable insights into established practices, trends, and innovations, ultimately informing our approach and contributing to the advancement of our platform's design and functionality.

2.2.1. Data Architecture

One of the fundamental aspects of SaaS applications is the centralized and network-based access they provide to data, which offers significant advantages over locally installed applications. As highlighted in the literature [1], SaaS applications allow customers to access their data with reduced overhead and enjoy the convenience of centralized data management. However, designing an effective data architecture for multi-tenant SaaS applications poses unique challenges. In particular, there are three distinct approaches that have been identified in the literature: "separate databases", "shared database with separate schemas", and "shared database, shared schema". Each approach presents its own trade-offs in terms of data isolation, scalability, customization, and security. In this section, we will delve into the existing related work on multi-tenant data architecture and explore these different approaches, their benefits, and their implications for SMBs in the context of e-commerce platforms.

Three Distinct Approaches

Authors Chong et al. (2006) discussed three distinct approaches to designing data architecture for multi-tenant applications: Separate Database, Shared Database with Separate Schemas, and Shared Database Shared Schema [1].

a. Separate Database:

The first approach to multi-tenant data architecture is to allocate a different database for each tenant. In this approach, each tenant has their own dedicated database, and metadata is used to associate each tenant with the appropriate database.

There are several advantages to this approach. Firstly, database security settings can be implemented to ensure that tenants cannot access or modify the data of other tenants,

whether intentionally or accidentally. This provides a high level of data privacy and isolation. Additionally, the solution can be easily extended and customized to meet the specific storage and processing requirements of each tenant. This flexibility allows for tailored solutions and optimized performance.

However, there are some trade-offs to consider. The separate databases approach comes with higher maintenance costs since each database requires its own management and upkeep. Additionally, the resource usage is higher as each server needs to accommodate multiple databases instead of a shared one. This can result in increased hardware and infrastructure costs.

In conclusion, the separate databases approach could be a valid option for premium customers who prioritize higher levels of data isolation, performance, and security. These customers may be willing to invest in the additional costs associated with maintaining separate databases for each tenant. This approach provides strong data privacy and customization capabilities, but it requires careful consideration of the associated maintenance and resource requirements.

b. Shared Database with Separate Schemas:

The second approach to multi-tenant data architecture involves using a shared database with separate schemas. In this approach, multiple tenants utilize the same database, but each tenant has their own set of tables within the database.

There are several advantages to this approach. Firstly, it is relatively straightforward to implement, as long as the number of tables required by each tenant remains manageable. This simplicity in implementation can save development time and effort. Additionally, the shared database with separate schemas approach allows for a higher number of tenants to be accommodated on each server compared to the separate databases approach. This efficient resource utilization can result in cost savings.

Furthermore, the shared database with separate schemas approach offers a lower cost of maintenance since the infrastructure and administration efforts are centralized. This centralized management reduces the overhead associated with managing multiple databases and ensures consistent maintenance practices.

However, there are trade-offs to consider. This approach provides less isolation between tenants compared to the separate databases approach since they are still sharing the same database infrastructure. While each tenant has their own set of tables, they are still operating within the same database environment. Additionally, scaling the shared database with

separate schemas approach may present challenges, especially if the number of tenants or the complexity of their data requirements increases. Careful consideration should be given to ensure that the shared database can effectively handle the growing demands of multiple tenants.

In conclusion, the shared database with separate schemas approach can be suitable for applications that require a relatively small number of database tables. It strikes a balance between resource utilization, maintenance costs, and data isolation, making it a viable option for multi-tenant applications with moderate requirements.

c. Shared Database, Shared Schema:

The final approach to multi-tenant data architecture involves using a shared database with a shared schema. In this approach, the same database and the same set of tables are used to host the data of multiple tenants. Each record in the tables is associated with the appropriate tenant using a tenant ID column.

There are several advantages to this approach. Firstly, it offers the lowest hardware costs since all tenants share the same database infrastructure. This can result in significant cost savings, especially when serving a large number of tenants. Additionally, the shared database, shared schema approach also leads to the lowest maintenance costs since there is no need to manage and maintain separate databases or schemas for each tenant.

However, there are some trade-offs to consider. This approach may require additional development effort to ensure proper isolation and security between tenants within the shared database and schema. Special care must be taken to implement robust access controls and data separation mechanisms. Additionally, if the tables within the shared database grow too large, it can lead to performance issues that affect all tenants.

In conclusion, the shared database, shared schema approach is appropriate when the application is capable of serving a large number of tenants with a small number of servers. This approach offers significant cost savings in terms of hardware and maintenance. However, it does require careful implementation to address isolation and security concerns. It is a suitable choice when customers are willing to sacrifice physical data isolation in exchange for the lower costs that this approach makes possible.

Choosing an Approach

When deciding on the appropriate multi-tenant data architecture, several factors need to be considered. These include economic considerations, security considerations, and tenant-specific requirements.

a. Economic considerations:

Economic considerations play a crucial role in determining the chosen approach. Applications optimized for a shared approach typically require a larger development effort due to the relative complexity of developing a shared architecture. This results in higher initial costs. However, these shared architectures can support more tenants per server, leading to lower ongoing operational costs. Therefore, the economic feasibility of the chosen approach needs to be assessed, considering both initial and long-term costs [1].

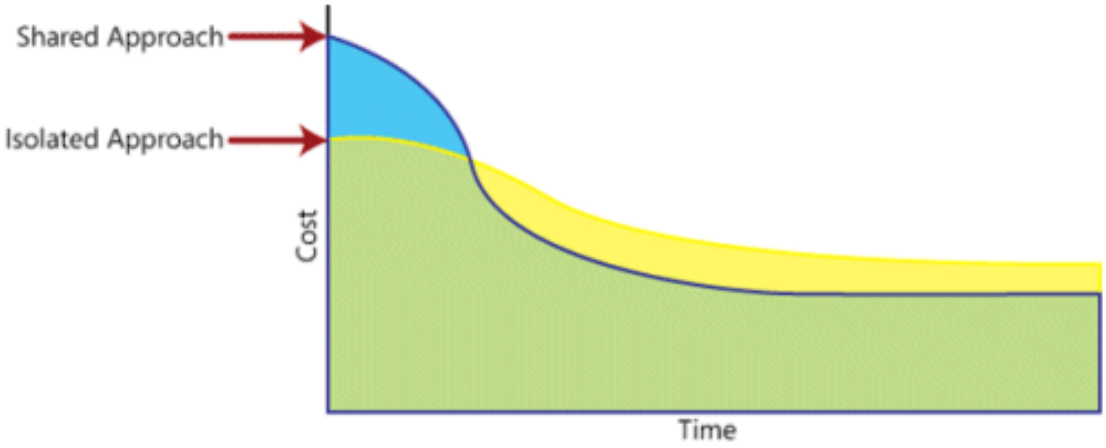

Figure 2: Cost over time for a hypothetical pair of SaaS applications; one uses a more isolated approach, while the other uses a more shared approach [1]

b. Security considerations:

Security considerations are also essential when choosing a data architecture. The separate databases approach offers a straightforward way to ensure data isolation and security. Each tenant has their own dedicated database, providing strong data privacy. However, it is important to note that a shared approach can also provide robust data safety with the use of sophisticated design patterns. Implementing such patterns may require higher development costs but can maintain data security even in a shared environment. It is crucial to evaluate the specific security requirements of the application and the sensitivity of the data involved [1].

c. Tenant considerations:

The choice of data architecture should also take into account tenant considerations. Factors related to the targeted tenants can influence the decision. Firstly, the expected number of tenants plays a role. If a large number of tenants is anticipated, a shared approach may be more cost-efficient, as it can accommodate multiple tenants on a single server. Secondly, the storage space and computational resource requirements of each tenant should be considered. If most or all tenants require significant storage space or high computational resources for their queries, a separate-database approach may be more suitable to meet their needs. Finally, the number of concurrent users expected for the application should be taken into account. If a large number of concurrent users is anticipated, a more isolated approach may be preferable to ensure optimal performance and user experience [1].

d. Performance considerations:

When selecting a data architecture approach for multi-tenant systems, it is crucial to consider the impact on performance. The performance of the system can be influenced by two main factors: tenant resource usage and the number of tenants.

Based on tenant resource usage, different isolation patterns can be employed. For larger tenants with high Transactions Per Second (TPS) requirements, the Separate Database pattern offers better isolation capabilities without significant performance overhead. On the other hand, smaller tenants with lower TPS can benefit from the Separate Schema or Shared Schema patterns, which provide noticeable performance advantages [2].

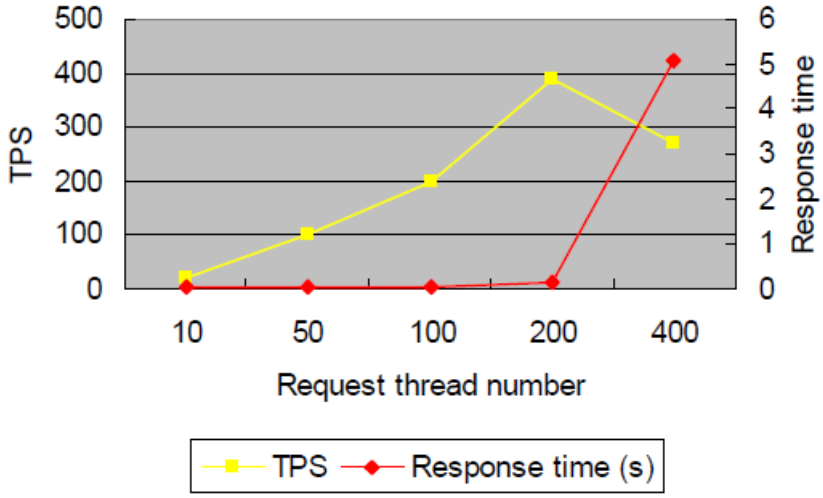

Figure 3: Curves of Performance Metrics change [2]

Considering the number of tenants, the Separate Schema pattern achieves the highest performance. However, the Separate Database pattern and the Shared Schema pattern with simple indexes may experience performance degradation as the number of tenants increases. To mitigate this, advanced indexing technologies like Multiple Dimensions Clustering (MDC) and clustered indexes can greatly enhance system performance. These techniques centralize the data store from the tenant dimension, reducing data fetching interference among tenants and eliminating the time-consuming combination of indexes step associated with simple separated indexes [2].

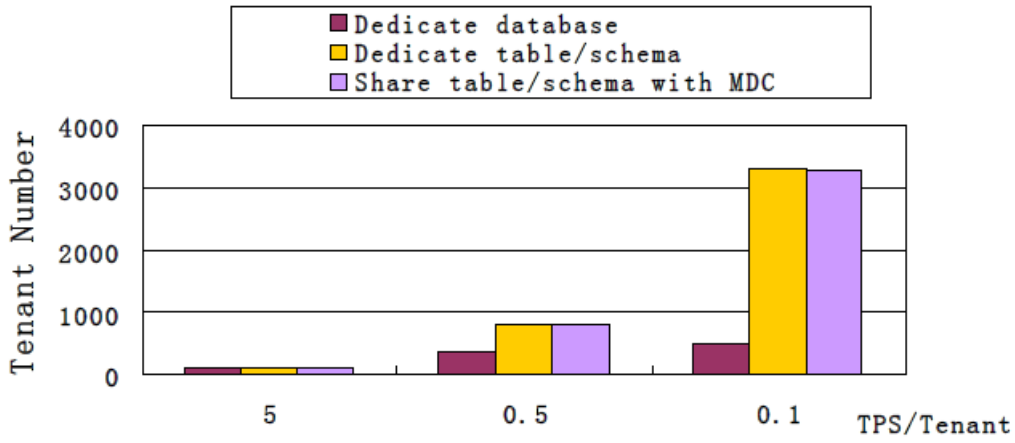

Figure 4: Performances Evaluation of Isolation Patterns: Comparison of Maximal Tenant Capacity [2]

In conclusion, choosing the appropriate data architecture for a multi-tenant application requires careful consideration of economic factors, security requirements, tenant-specific considerations, and performance objectives. It is important to weigh the costs, security implications, and specific needs of the application and its tenants to make an informed decision.

Securing a Shared Database

When implementing a shared database in a multi-tenant application, it is crucial to prioritize data security due to the simultaneous usage of the system by multiple users who require privacy and confidence [3]. Effective data management plays a critical role in addressing this challenge to safeguard a shared database, various approaches can be implemented, including application-level security, database-level security, and data encryption [4].

Application-level security involves establishing a secure connection between the application and the database. The application is responsible for enforcing access controls to ensure that tenants can only access and modify their own data. This approach offers the advantage of handling security through code, making it easier to implement. Additionally, no special

database configurations are required when a new tenant wants to create an account. Furthermore, shared database connections for multiple tenants can save on resources. However, this approach is prone to error if developers do not pay sufficient attention to security implementation.

Database-level security, on the other hand, involves assigning an identity and a secret to each tenant. Instead of the application having its own credentials, it uses the tenant's credentials to connect to the database and perform operations on their behalf. This approach offers a higher level of security and reduces the risk of human error during development. However, it has some drawbacks. Database connections cannot be persisted and reused for multiple tenants, resulting in higher resource usage. Additionally, configuring the database for each new tenant adds complexity to the customer acquisition process. Finally, this approach is less flexible compared to application-level security.

Security through data encryption is another approach to securing a shared database. Each tenant's data is encrypted using a secret key. When logging in, users need to provide their tenant key, which is then used to decrypt their data. This approach provides the highest level of security since even if an intruder gains access to the database, they will not be able to decipher the data. However, this approach relies on tenants to safeguard their encryption keys. Furthermore, certain queries like sorting can become challenging without access to the plaintext data, requiring the development of specialized algorithms.

In conclusion, securing a shared database in a multi-tenant application requires careful consideration of the different security approaches available. Application-level security offers ease of implementation and resource savings but requires attention to detail. Database-level security provides a higher level of security but introduces complexities and limitations. Data encryption ensures the highest level of security but relies on tenants to protect their keys and may present challenges for certain types of queries. The chosen approach should align with the specific security requirements and considerations of the application and its tenants.

2.2.2. System Architecture

The system architecture of a multi-tenant application plays a crucial role in ensuring efficient resource management, process sharing, and performance isolation. In this section, we will explore the benefits of multi-tenancy in resource management, delve into the four affinity models for process sharing, and discuss the importance of performance isolation. By understanding the system architecture considerations, we can design and implement a robust and scalable infrastructure that optimizes resource utilization, enhances process efficiency, and delivers reliable performance for each tenant in a multi-tenant environment.

Resource Management Benefits of Multi-Tenancy

One of the significant advantages of multi-tenancy is the efficient sharing of resources, which leads to significant cost benefits [5]. Research suggests that sharing resources at the level of application instances offers the highest efficiency, as it minimizes overhead and enables fine-grained workload and resource management [6]. This level of granularity allows for effective management of requests and threads, optimizing resource utilization and enhancing overall system performance. By leveraging multi-tenancy, organizations can achieve greater cost-effectiveness and resource efficiency in their systems.

Process Sharing

In multi-tenant systems, process sharing plays a crucial role in optimizing resource utilization and enhancing overall system performance. There are four affinity models that govern how requests are handled within the system: non-affine applications, server-affine applications, cluster-affine applications, and inter-cluster-affine applications [7].

- **Non-affine applications:**

In non-affine applications, incoming requests can be handled by any application instance without considering the tenant from which the request originates. This model offers advantages such as easy scalability, efficient resource utilization, and straightforward load balancing. However, it is not suitable for stateful applications that store user sessions or contexts, and local caching becomes challenging due to the unpredictable nature of request handling.

- **Server-affine applications:**

Server-affine applications ensure that requests from a specific tenant are always handled by the same application instance, although one instance can handle multiple tenants. This model enables efficient local caching and is well-suited for stateful applications. However, load

balancing becomes more challenging, limiting scalability to horizontal scaling only, and high availability through replication becomes harder to achieve.

- **Cluster-affine applications:**

Cluster-affine applications serve the users of a particular tenant using a fixed subgroup of application instances, with multiple tenants being served by one subgroup. This model strikes a balance between server-affine and non-affine applications. It allows for both horizontal and vertical scaling, provides availability through process replication within the same cluster, and allows for caching within the cluster, which can be in the same geographic region or data center. However, it may result in less resource usage efficiency compared to non-affine applications, as some nodes within a cluster may be idle while others struggle to handle the load and require scaling. Additionally, caching efficiency may be lower compared to server-affine applications.

- **Inter-cluster-affine applications:**

Inter-cluster-affine applications are similar to cluster-affine applications, but an application instance can be part of multiple groups. This model offers more flexible clustering, leading to more efficient resource usage compared to cluster-affine applications. However, handling stateful applications and caching can be more challenging compared to cluster-affine applications.

By understanding and implementing these process-sharing affinity models, organizations can optimize resource management, scalability, caching efficiency, and overall system performance based on the specific requirements of their multi-tenant applications.

Performance Isolation

Ensuring performance isolation is crucial in multi-tenant systems to address the concerns of potential cloud users regarding performance guarantees. Resource isolation plays a key role in achieving performance isolation, and various approaches have been proposed to address this challenge.

The lack of performance guarantees has been identified as a significant obstacle for potential cloud users [8]. Without proper isolation, tenants may experience performance degradation due to resource contention from other tenants sharing the same infrastructure. To mitigate this, performance isolation techniques are employed [7].

One notable approach, as proposed by Lin et al. [9] (2009), focuses on achieving performance isolation within a single MTA. This approach aims to allocate resources in a way

that prevents one tenant from negatively impacting the performance of other tenants. By effectively managing resource allocation and limiting resource contention, this approach ensures that each tenant receives the performance they require without being affected by the activities of other tenants.

By implementing performance isolation mechanisms, multi-tenant systems can provide consistent and predictable performance to their tenants, addressing concerns and increasing confidence in the reliability and performance of the cloud infrastructure.

2.2.3. Customization

Customization plays a pivotal role in meeting the diverse needs and preferences of tenants in multi-tenant applications. It enables tenants to tailor various aspects of the application to align with their specific business requirements, user experience expectations, and branding. In this section, we will explore the different dimensions of customization in multi-tenant applications, ranging from the need for customization and types of customizations based on the business domain to the techniques and architectural considerations involved. We will also delve into the tradeoffs associated with customizability, including the balance between ease of use and technical complexity. By understanding the intricacies of customization, we can design and implement multi-tenant applications that offer flexibility and adaptability to cater to the unique requirements of each tenant.

The Need for Customization:

In the context of multi-tenant SaaS systems, there is a need for customization to meet the unique requirements and business logic of individual users [10]. While these systems typically provide a shared application layer structure, they often only cater to the common needs of the majority of users [11]. However, in the market, information system users often have specific characteristics aligned with their own business logic, necessitating a certain degree of customization from SaaS systems based on customer demands [11]. Meeting individual business needs through user configuration and secondary development is considered one of the key elements in promoting the broader adoption of the SaaS model [12].

Types of Customizations based on the business domain:

In the realm of SaaS customization, various industry partners provide different approaches to meet the specific needs of users. The paper [13] highlights examples of customization capabilities offered by prominent platforms:

1. Google App Engine (GAE) [14]: GAE allows customization at the program layer through a deployment description file. Users can modify parameters such as servlets, Uniform Resource Locator (URL) paths, JSPs, and security methods. Additionally, GAE supports service layer customization, enabling users to define service names, domains, and control billing budgets. However, customization options for workflows and data are not available within the GAE framework.

2. Amazon EC2 [15]: Similar to GAE, Amazon EC2 offers customization capabilities at the program layer, allowing users to modify deployment settings. However, EC2 does not provide customization options for the workflow layer, focusing primarily on program-level customization.

3. Salesforce.com [16]: Salesforce.com provides a flexible customization framework that enables users to customize the user interface, workflows, and data within their framework.

These examples illustrate that customization capabilities can vary across platforms, with each offering specific customization options at different layers of the application stack. While some platforms focus on program-level customization, others provide more extensive customization possibilities for workflows, data, and user interfaces.

In the case of our project, our customization goals will be similar to the Salesforce example as it will focus on offering a personalized UI and data models and will be developed with the option for easily integrating customized workflows in the future.

Customization Assets:

In paper [17], the authors distinguished between five assets that can be customized to meet specific tenant requirements:

GUI Customization: This involves customizing the Graphical User Interface (GUI) of the application. It includes changes to elements such as logos, themes, layouts, fonts, and interaction features. GUI customization is the most basic and visible form of customization.

Workflow Customization: Tenants can customize the workflow of the application to control the business process logic. They can create their own workflow templates using existing type services or select from a repository of predefined templates.

Service Customization: Service customization involves two main steps:

- Service Selection: Tenants choose a concrete service implementation and bind it with the service type used in the workflow template.
- Service Configuration: Tenants configure the properties of the selected service to achieve the desired behavior and performance.

Data Customization: Data customization in SaaS applications involves tailoring various aspects of data management to meet tenant-specific requirements. This includes customization of data storage, encryption, compression, and schema enhancement. Different multi-tenant data architectures, such as separate databases, shared databases, and separated schemas with shared databases, offer varying levels of isolation, sharing, and scalability to accommodate these customization needs [1].

QoS Customization: Quality of Service (QoS) requirements are typically defined through Service Level Agreements (SLAs) between tenants and SaaS application providers. QoS customization involves allocating the appropriate amount of resources and making choices regarding services, workflow customization, and even data architectures. It is a deeper level of customization that impacts the overall performance and availability of the application.

The Different Actors in SaaS Customization:

Various parties can be involved in the customization of SaaS applications based on its level of complexity [17]:

SaaS application developers/designers: They aim to provide high customizability by understanding common requirements within a domain and designing an extensible framework. Developers/designers can perform initial customizations to reduce the effort required by tenants.

SaaS application tenants: Tenants can customize their applications based on pre-configured templates provided by developers/designers. For example, in an online retail SaaS application, customized templates for different domains (electronics, clothing, grocery, etc.) can be offered, allowing tenants to further tailor their applications.

SaaS application consultants: For complex applications like enterprise ERP, HRM, CRM, tenants may hire specialized consultants to handle customization tasks, reducing the need for in-house training and expediting time-to-market.

SaaS application users: Users within each tenant organization can have some level of customization according to their personal preferences. However, their customization options are generally limited to superficial changes and are constrained by the choices taken by developers/designers and tenants themselves.

Customization at Runtime in Multi-tenant Applications

In the context of multi-tenant applications, the concept of customization extends to enabling individual tenants to design and personalize various parts of the application. This customization is facilitated by multi-tenant aware applications, which possess the ability to dynamically adjust their behavior during runtime execution without requiring a redeployment of the entire application [18].

By incorporating multi-tenant awareness into the application architecture, tenants gain the flexibility to tailor specific aspects of the application to meet their unique requirements. This customization can encompass various elements, such as user interfaces, workflows, data models, and business logic. The application adapts its behavior based on the preferences and configurations set by each tenant, enabling a more tailored and personalized experience [19].

The ability to customize the application at runtime without the need for redeployment provides tenants with agility and efficiency in managing their individualized requirements. They can fine-tune the application to align with their specific business processes, branding, and user experience preferences [19].

By empowering tenants with customization capabilities, multi-tenant applications can better cater to the diverse needs and preferences of their user base, enhancing overall user satisfaction and driving broader adoption.

Customization Techniques:

Customization in SaaS applications can be performed in multiple ways, depending on the level of flexibility and control provided by the application [20]:

Source Code: In an extensible SaaS application, customization can involve modifying the source code by adding new functionalities or altering existing ones. This requires development skills and access to the underlying codebase. Salesforce, for example, offers VisualForce for creating custom user interfaces and Apex for adding custom business logic [21]. Modifying the

source code provides the highest level of customization but also requires expertise and thorough testing to ensure the stability and compatibility of the application.

Composition: Another approach to customization is composition, where existing components or services are combined to create new workflows or functionalities. SaaS platforms often provide marketplaces or app exchanges where users can find and integrate pre-built applications or services into their own SaaS environment. This allows for rapid customization without the need for extensive coding. Salesforce's AppExchange, for instance, offers a wide range of applications that users can utilize to compose new workflows and enhance their applications [21].

Configuration: Configuration-based customization involves adjusting various parameters or settings within the application to tailor its behavior to specific needs. This can include customizing user interfaces, automating business processes, or modifying default settings. Configurable options are typically exposed through configuration files or settings panels provided by the SaaS application. For example, in Salesforce, users can customize user interfaces and automate business logic using out-of-the-box configurations and workflow tools without directly modifying the source code [21]. This approach offers a balance between flexibility and ease of use.

Leveraging concepts from product-line engineering and multi-tenancy patterns:

Research [22] suggests that concepts from product-line engineering offer valuable insights for enabling customization in multi-tenant SaaS systems. By adopting these concepts, organizations can define variation points that allow different variants to be linked together. Additionally, the variability model derived from product-line engineering can serve as a guiding framework for SaaS customization, ensuring that the customization process is systematic and efficient.

Furthermore, the packaging and deployment of the SaaS solution can benefit from the application of multi-tenancy patterns [23]. These patterns assist in distinguishing between components that are shared among all tenants and those that are specific to certain tenants. By leveraging these patterns, organizations can effectively manage the deployment of their multi-tenant SaaS applications, ensuring that the appropriate components are deployed to each tenant based on their specific requirements.

Together, these approaches based on product-line engineering and multi-tenancy patterns form the foundational elements for supporting variability within a multi-tenant SaaS application

architecture. They provide the necessary framework and guidance to accommodate customization and ensure that the SaaS solution can effectively adapt to the diverse needs of its tenants [24].

SOA Architecture for Customization:

Service-Oriented Architecture plays a crucial role in making customization possible and maintainable in SaaS applications [25]. SOA facilitates the conversion of business processes into a set of interconnected services or repeatable business tasks, which can be accessed over the network as needed. By combining these services, specific business processes can be achieved, allowing the business logic to quickly adapt to changing conditions and requirements [26]. Therefore, SOA architecture serves as a suitable infrastructure for building customizable SaaS applications [27].

Tradeoff Between Customizability and Ease of Use:

The ease of customization can vary depending on the chosen method and the level of support provided by the SaaS application [20]:

Manual: Manual customization requires tenants to manually make decisions and adjustments at each customization point. This approach provides the highest level of control and flexibility but may involve more effort and expertise.

Automated: Automated customization aims to simplify the customization process by automatically generating the required changes based on tenants' input or predefined rules. However, complete automation may not always meet all tenants' specific requirements, and customization results may need further fine-tuning.

Guided: Guided customization combines automated suggestions with manual decision-making. Automated customization tools provide a few top matching choices for each customization point, allowing tenants to review and make decisions based on their judgment. This approach reduces manual work significantly while still providing tenants with control over the customization process. It helps streamline customization efforts and minimizes errors introduced by fully automated approaches.

It is important to note that the level of customization and ease may vary depending on the specific SaaS application and the extent to which it supports customization capabilities.

Tradeoff Between Customizability and Technical Complexity:

In multi-tenant systems, the ability to handle different configurations for each tenant regarding the user interface, functional/non-functional behavior, and referenced services is crucial [28].

To differentiate the level of customization in multi-tenant systems, we can consider two ideas. One approach involves using a separate Meta-Data Manager to provide customization information and dynamically adapt the application [29]. This results in a configurable application that allows for tenant-specific behavior or appearance through configuration settings, without the need for tenant-specific code. Consequently, each tenant can access the same code base, while their configurations remain independent of one another.

On the other hand, an application that allows tenant-specific code extensions provides the most powerful way to adapt to customer needs. However, implementing such customizations in a multi-tenant environment presents significant technical challenges. Mietzner et al. (2008) proposed an SOA based approach for multi-tenant applications, which enables tenants to replace or extend specific portions of the application's code without impacting other tenants [30]. This approach allows for granular customization while maintaining the integrity and isolation of tenant-specific code within the shared application infrastructure. Furthermore, their work demonstrates that extensive modifications, including tenant-specific developments, can be incorporated into the system based on established patterns [31].

By understanding these customization options, multi-tenant systems can offer varying degrees of flexibility and adaptability to meet the unique requirements of each tenant.

2.2.4. Security

In multi-tenant systems, robust security measures are crucial due to the shared nature of computer hardware and infrastructure. It is essential to understand the security challenges inherent in such environments and explore effective countermeasures to ensure the confidentiality, integrity, and availability of data, thereby establishing a secure environment for all tenants.

Security Issues in Multi-Tenant Systems

Security is a critical aspect to consider in multi-tenant systems due to the shared nature of computer hardware among multiple tenants. The fundamental security issue stems from the very premise of multi-tenancy, where tenants coexist on the same infrastructure [32]. This

shared environment poses risks, such as malicious tenants launching attacks towards co-resident tenants within the same cloud data center, potentially compromising data confidentiality and integrity [33]. Furthermore, the breakdown of barriers between tenants can lead to unauthorized access to sensitive data or interference with other tenants' applications, especially in cases where tenants are competitors operating within the same environment. Service providers bear the responsibility of implementing robust security measures to prevent unauthorized access, ensure data isolation, and protect the applications and data of individual tenants [34]. Implementing strong security controls, including access controls, encryption, and monitoring, as well as conducting regular security audits and vulnerability assessments, is crucial to addressing these security concerns in multi-tenant systems and maintaining a secure environment for all tenants.

Attack Examples

Security attacks in multi-tenant systems can take various forms, posing risks to the overall stability and security of the shared environment. One example is the potential for an overload caused by one tenant to adversely impact the performance of other tenants, leading to service degradation or disruptions [35]. Additionally, misconfigurations within the underlying infrastructure can introduce vulnerabilities, potentially allowing one tenant to gain unauthorized access to another tenant's data or resources. These misconfigurations may arise due to the complexities of managing shared resources and the need to maintain strong isolation between tenants [32].

Countermeasures

Countermeasures play a crucial role in mitigating security risks in multi-tenant systems. One key approach is the implementation of logical authentication and access controls. Authentication and authorization serve as the foundation for application security [32]. Additionally, Role-Based Access Control (RBAC) is a widely adopted mechanism in multi-tenant environments [34]. RBAC involves two phases: in the first phase, users are assigned to specific roles, and in the second phase, privileges, such as access to resources, are assigned to roles rather than individual users. This enables dynamic assignment, re-assignment, and revocation of privileges without altering the underlying permissions, allowing for flexible and granular control over user access. By leveraging RBAC, multi-tenant systems can enhance security by ensuring that users acquire and accumulate permissions based on their role memberships, while maintaining strong isolation between tenants and their respective data and resources.

2.2.5. Scalability

Scalability is a critical aspect to consider in multi-tenant SaaS applications, where multiple tenants share the same application and data-store. The shared nature of the infrastructure introduces unique challenges in achieving scalable performance. The paper "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?" (2010) highlights some important points regarding scalability in multi-tenant environments [36].

In the context of multi-tenant applications, scalability becomes a more pressing concern compared to single-tenant applications. While in the SMB segment, a tenant typically does not require more than one application and database server, there are other factors to consider. For instance, it may be necessary to consolidate all the data for a particular tenant onto the same server to optimize frequently accessed database queries. These constraints significantly impact the scalability options for both the application and its datastore.

To ensure scalability in multi-tenant SaaS applications, it is essential to adopt strategies that allow for efficient resource allocation and management. Techniques such as horizontal scaling, where additional server instances are added to handle increasing load, can be employed. Additionally, employing load balancing mechanisms to distribute incoming requests evenly across multiple servers can help maintain optimal performance levels.

Furthermore, the use of technologies such as caching, and data partitioning can enhance scalability by reducing the load on the shared datastore and improving response times for tenant-specific data retrieval.

By carefully considering the unique scalability requirements of multi-tenant SaaS applications and implementing appropriate scaling strategies, organizations can ensure that their applications can handle growing numbers of tenants and increasing demands without compromising performance or user experience [36].

2.2.6. Maintenance

Maintenance poses a significant challenge in multi-tenant systems, particularly in adapting the software system to changing requirements and deploying updates [37]. Introducing multi-tenancy into a software system adds complexity that can impact the maintenance process.

To minimize the impact on code complexity, it is essential to separate the implementation of multi-tenant components from the single-tenant logic as much as possible. Failure to do so can result in maintenance becoming a nightmare for several reasons [36]:

a) **Reeducation of developers**: Mixing multi-tenant code with single-tenant code requires reeducating all developers about multi-tenancy, increasing the learning curve and potential for errors.

b) **Increased code complexity**: Mixing multi-tenant code with single-tenant code makes it more challenging to track and manage where multi-tenant functionality is introduced, leading to increased code complexity and decreased maintainability.

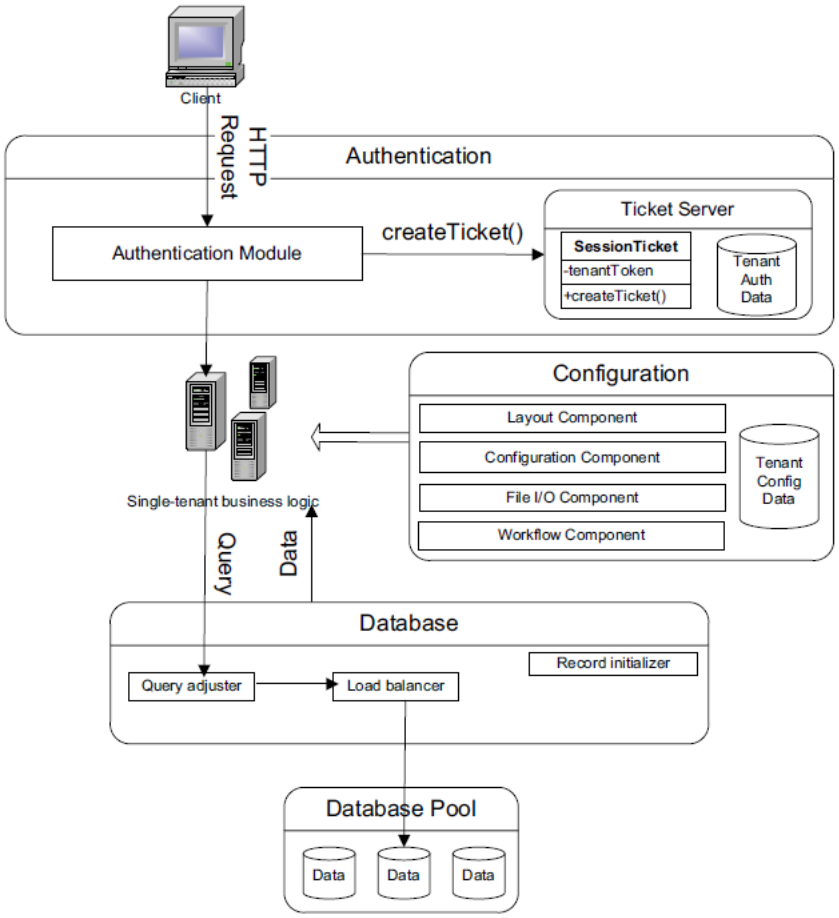

Figure 5: Architectural overview for multi-tenancy [36]

While implementing multi-tenancy can increase code complexity, the quality of the implementation plays a crucial role. In a non-layered software architecture, the introduction of multi-tenancy can lead to code scattering and maintenance difficulties. Conversely, in a layered architecture, it is possible to implement multi-tenancy as a relatively isolated cross-cutting concern with minimal effort, enabling easier maintenance of the overall application [36].

3. METHODOLOGY

In this chapter, we describe the methodology used to conduct our research, which is based on the Design Science Research (DSR) approach. The DSR approach is a problem-solving methodology that aims to develop innovative and effective solutions to practical problems. We will follow the design science research process as outlined by Philipp Offerman, Olga Levina, Marten Schönherr, and Udo Bub, as well as the regulative cycle developed by Roel Wieringa. By adopting this methodology, we aim to provide a rigorous and systematic approach to the development of our e-commerce platform.

3.1. Design Science Research Methodology

For this project, we have adopted the Design Science Research (DSR) approach. DSR is a problem-solving methodology that focuses on systematically creating and evaluating innovative solutions to complex problems through the creation of artifacts.

The DSR approach is well-suited for our project because we aim to design and develop a solution to a real-world problem in the e-commerce industry. Additionally, it provides a structured framework for developing and evaluating creative solutions to practical problems. DSR also emphasizes the importance of knowledge creation and theory-building, which is an important aspect of our project. By developing a multi-tenant e-commerce platform, we are contributing to the knowledge base of software engineering, particularly in the area of multi-tenancy and e-commerce. By following the DSR approach, we can ensure that our knowledge creation and theory-building efforts are grounded in sound design principles and empirical evaluation.

To guide the development and evaluation of our solution, we will follow the Regulative Cycle proposed by Roel Wieringa. The Regulative Cycle is a process model that supports the iterative development of solutions in DSR. It consists of five stages: problem investigation, solution design, design validation, solution implementation, and solution evaluation. Each stage is followed by an evaluation phase, where the solution is tested against the criteria established in the previous stage. The results of the evaluation are then used to refine the solution in the next iteration of the cycle.

The iterative nature of DSR allows for continuous improvement and refinement of the solution based on feedback from users and stakeholders. The DSR approach is particularly suited for our project as it enables us to create and evaluate a solution that addresses the specific needs and requirements of SMBs in the e-commerce industry. Through the iterative cycles, we can

refine and improve the solution to ensure it meets the needs of our target users and provides a viable and effective solution to the problem at hand.

In the following subsections, we will provide a detailed description of each stage of the Regulative Cycle and the activities that will be carried out in each iteration. We will also outline the principles and guidelines of DSR, as described by Alan R. Hevner, which will be followed throughout the project to ensure the rigor and validity of our research contributions.

3.2. Design Science Research Iterations

To guide the development of the proposed multi-tenant web-based e-commerce platform, we followed the Design Science Research (DSR) model, which is a problem-solving methodology that combines theory-building with the creation of a practical artifact (Wieringa, 2009). As mentioned previously, the DSR model consists of a regulative cycle with five phases, namely problem investigation, solution design, design validation, solution implementation, and solution evaluation (figure 6). We iterated through this cycle multiple times to improve the quality of our proposed platform.

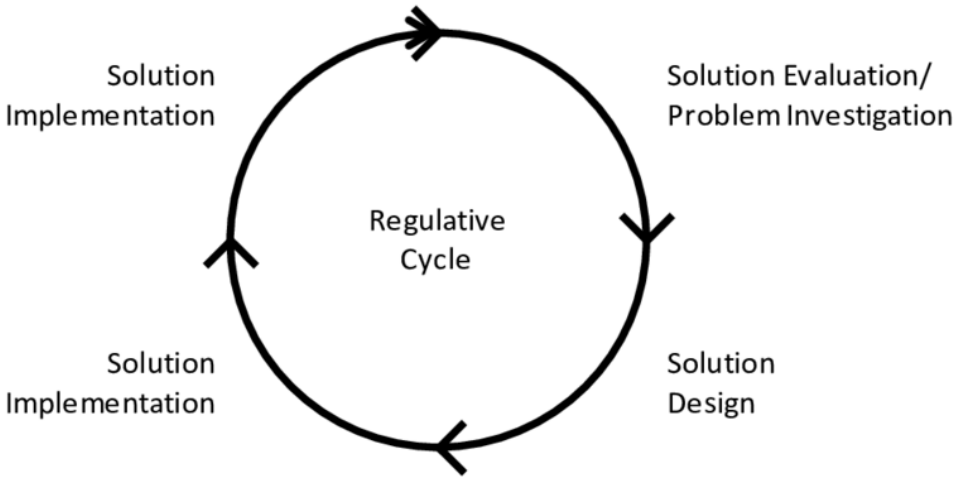

Figure 6: The Regulative Cycle of the DSR Methodology [38]

3.1.1. Problem Investigation

The Problem Investigation phase was a critical first step in our project, where we conducted a thorough analysis of the challenges faced by SMBs in managing their social media activity and developing their own e-commerce websites.

We began by conducting a comprehensive literature review of the latest research and industry reports to gain an understanding of the most pressing issues facing SMBs in the e-commerce domain.

Next, we conducted interviews with a range of stakeholders, including SMB owners and managers, industry experts, and e-commerce platform providers, to gain insights into their experiences and perspectives. These discussions allowed us to identify key pain points, such as a lack of resources, expertise, and time, which hindered SMBs' ability to develop their own e-commerce websites.

We also examined existing e-commerce platforms and solutions to understand their strengths and weaknesses. This analysis highlighted the limitations and uncertainties of the current solutions and allowed us to identify opportunities for innovation and improvement.

Overall, the Problem Investigation phase provided us with a comprehensive understanding of the challenges and limitations faced by SMBs in the e-commerce domain. This knowledge served as the foundation for our subsequent solution design and implementation efforts.

3.1.2. Solution Design

During the Solution Design phase, we leveraged the insights gained from the Problem Investigation phase to propose a multi-tenant web-based e-commerce platform that would cater to the needs of SMBs. We followed an iterative approach, where each iteration built on the knowledge and feedback gained from the previous one. So, the solution varied from one iteration to another, as we gained extra knowledge and feedback from stakeholders during the evaluation of the previous iteration. Through close collaboration with technical experts, managers, and other stakeholders, we ensured that the proposed solution would be effective, feasible, and aligned with the project goals.

To prioritize features based on their importance to the end-user, we utilized an agile development approach that involved stakeholders in every step of the design process. We carefully considered the needs of SMBs and made sure to avoid unnecessary features that

would add complexity and hinder user adoption. Our design focused on ensuring a seamless user experience, with a user-friendly interface and efficient workflows.

We also paid close attention to the technical feasibility of the proposed solution, taking into account factors such as performance, scalability, and security. Additionally, we engaged in in-depth research to ensure that the proposed solution was developed using best practices and standards in software development.

Throughout the Solution Design phase, we documented our design decisions and changes, making sure to keep a clear record of any modifications made to the design. This allowed us to evaluate the design and make improvements in subsequent iterations.

Overall, the Solution Design phase was a critical part of our project, allowing us to propose a solution that would meet the needs of SMBs and be technically feasible. By involving stakeholders in the design process and prioritizing user needs, we ensured that the proposed solution would be effective and user-friendly.

3.1.3. Design Validation

During the design validation phase, we conducted an assessment of the proposed solution to determine its ability to meet the needs of SMBs while also being scalable, reliable, and secure. To ensure the validity of our evaluation, we identified potential validity threats and took steps to address them.

Through our validation process, we identified limitations and uncertainties in the solution design and used these insights to refine and improve the proposed solution in subsequent iterations. By addressing potential validity threats and ensuring the rigor and validity of our evaluation, we were able to make sound design decisions that resulted in a solution that met the needs of SMBs in the e-commerce industry.

3.1.4. Solution Implementation

During the solution implementation phase, we developed and implemented the web-based multi-tenant e-commerce platform proposed in the solution design phase. Our implementation followed the Scrum method, which allowed us to work flexibly and proactively address design problems as they arose. The implementation process varied from one iteration to another, depending on the scope of the proposed solution. We collaborated with the

engineering team, managers, and other stakeholders to ensure that the implementation was effective, efficient, and aligned with the proposed solution.

We paid close attention to the technical feasibility of our proposed solution while addressing the important quality issues of performance, scalability, and security. We made sure to use best practices and standards in software development. We also considered the practicality of the solution, taking into account factors such as ease of use, maintenance, and cost-effectiveness. Each iteration involved making improvements and refinements to the implementation based on feedback from users and stakeholders, as well as our own observations and assessments. Throughout the implementation process, we documented our decisions and changes, making sure to keep a clear record of any modifications made to the design.

Overall, the solution implementation phase was a critical part of our project, enabling us to turn our proposed design into a tangible, functional solution. By working closely with stakeholders and technical experts, and by keeping a keen eye on both the technical feasibility and practicality of our proposed solution, we were able to develop an effective, efficient, and user-friendly multi-tenant web-based e-commerce platform.

3.1.5. Solution Evaluation

In the solution evaluation phase, we thoroughly examined the implementation of our proposed multi-tenant web-based e-commerce platform. This phase aimed to gather crucial data about the system's performance, identify areas for potential improvement, and assess alignment with stakeholder expectations.

We conducted rigorous unit testing and engaged in stakeholder interviews to gather valuable feedback that guided platform enhancements. Through this iterative process, we refined the platform, ensuring it met the specific needs of SMBs while maintaining scalability, reliability, and security. This rigorous evaluation process resulted in a robust and effective solution tailored to address the identified challenges.

3.3. Principles and Guidelines

To ensure the quality of our DSR artifacts, we adhere to the seven guidelines proposed by A. R. Hevner et al. [39]. These guidelines provide a framework for conducting research that are focused on the creation and evaluation of technology-based solutions to important business problems. The seven guidelines are described in the following table:

Table 1: Design Science Research Guidelines [39]

Guideline	Description
Guideline 1: Design as an Artifact	We should produce feasible and viable artifacts in the form of constructs, models, techniques, or instantiations via our DSR efforts. In other words, we aimed to create a practical solution that could be implemented in real-world settings.
Guideline 2: Problem Relevance	The DSR aims to create technology-based solutions that address real, crucial, and relevant business issues. This guideline ensured that our research was grounded in practical problems faced by SMBs.
Guideline 3: Design Evaluation	We should rigorously prove the utility, quality, and efficacy of our design artifacts through well-executed evaluation methods. We rigorously tested our artifacts to ensure they met the needs of SMBs and effectively addressed the challenges identified in the problem investigation phase.
Guideline 4: Research Contributions	The DSR should contribute to the areas of the design artifact, design foundations, and/or design processes in a clear and verifiable manner. We made sure that our research contributed to the development of practical and innovative solutions for SMBs in the e-commerce domain.
Guideline 5: Research Rigor	We should rely on the use of rigorous methods in both the construction and evaluation of the design artifact. We followed strict research methods and procedures to ensure the validity and reliability of our findings.

Guideline 6: Design as a Search Process	We should make use of available resources to achieve desired outcomes while adhering to the laws of the problem environment in our search for an effective artifact. We were aware of the limitations and constraints of the problem environment and worked within these limitations to develop practical solutions.
Guideline 7: Communication of Research	We should present our DSR effectively to both technology-oriented and management-oriented audiences. We made sure to present our research findings and artifacts in a clear and accessible manner to both technical and non-technical stakeholders.

By following these principles, we have ensured that our project meets the highest standards of DSR and that our artifacts have been rigorously designed, evaluated, and communicated, providing clear contributions to the field of design science research.

4. ITERATIONS

In this chapter, we present the outcomes of our work through a series of iterations following the regulative cycle approach. Each iteration represents a distinct phase in the development and enhancement of our multi-tenant e-commerce application. Through these iterative cycles, we aimed to continuously improve the system's design, functionality, and performance while addressing the specific needs and challenges of a multi-tenant environment.

4.1. Iteration 1: Requirements Engineering

Iteration 1 marks the initial phase of our multi-tenant e-commerce platform development, where we lay the foundation for the project's success. This iteration is dedicated to comprehensive problem investigation and solution design, aiming to identify functional requirements and quality attributes that cater to the needs of merchants and shoppers. We adopt the 4+1 View Model framework to gain a holistic perspective and ensure coherence across diverse architectural aspects. By employing best practices and research-driven approaches, our goal is to create a scalable, efficient, and user-friendly e-commerce platform that aligns with stakeholders' expectations and the business domain's demands.

4.1.1. Problem Investigation

A well-conducted requirements engineering process is essential for the success of the project. Research indicates that a significant number of project failures can be attributed to inadequate requirements engineering, which leads to misunderstandings, misalignments, and ultimately, unsatisfactory solutions [40]. By investing time and effort in thoroughly understanding the problem and gathering accurate requirements, we aim to avoid these pitfalls and lay the groundwork for a successful project outcome.

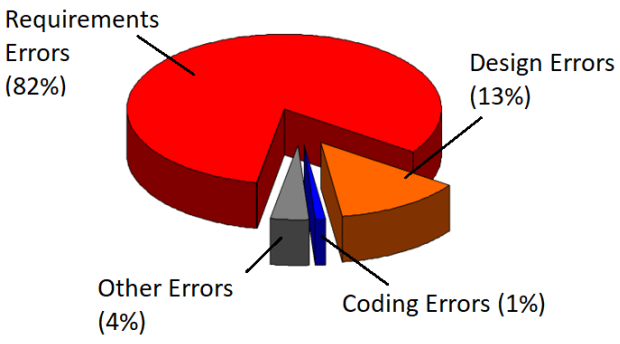

Figure 7: The Source of Software Errors [40]

To begin the problem investigation, we need to gain a comprehensive understanding of the e-commerce business domain and the specific challenges faced by both the merchants and the shoppers. This involves exploring the features and functionalities that are crucial for an e-commerce platform to effectively serve the needs of all parties involved. By analyzing existing e-commerce platforms provided by some of the leading shops in Tunisia, we can gather valuable insights and best practices that can guide our solution design.

Furthermore, we need to consider the scalability aspect of the multi-tenant environment. As more merchants and shoppers join the platform, the system should be able to handle the growing number of users, increasing traffic, and expanding product catalogs without compromising performance or user experience. Scalability is a critical factor in ensuring the long-term success and sustainability of the platform, and addressing this aspect from the outset will be a key focus of our problem investigation.

In addition to understanding the functional requirements, it is equally important to identify and prioritize the quality attributes that the e-commerce platform should exhibit. These quality attributes, such as reliability, security, usability, and performance, contribute to the overall success and acceptance of the platform. By investigating the specific quality attributes that are crucial for our project, we can align our solution design and implementation with the desired outcomes.

By conducting a comprehensive problem investigation, we aim to uncover the underlying challenges, requirements, and quality attributes that need to be addressed in our solution. This investigation will serve as a solid foundation for the subsequent phases, guiding our solution design and implementation to meet the specific needs of both the merchants and the shoppers in the multi-tenant e-commerce platform.

4.1.2. Solution Design

In the Solution Design phase, we will explore the techniques employed for requirements gathering, detail the functional requirements for both merchants and shoppers, and discuss the quality attributes that shape our solution design.

Requirements Gathering

In order to develop a successful and user-oriented multi-tenant e-commerce platform, a thorough understanding of the needs and expectations of our target users was paramount. We embarked on a comprehensive requirements-gathering process, engaging actively with

various stakeholders, including potential tenants and end users. By conducting brainstorming sessions, interviews, and market research, we aimed to identify and capture their unique requirements, preferences, and pain points. Below we present the requirements gathering techniques that we adopted.

a. Brainstorming:

We conducted brainstorming sessions to elicit and explore the desired functionalities of our scalable multi-tenant e-commerce application. This technique allowed us to generate innovative ideas and solutions based on the needs and preferences of our target customers. By brainstorming, we aimed to capture a comprehensive list of features and capabilities that would make our e-commerce app competitive and appealing to SMBs.

b. Market Research:

To gain a deeper understanding of the e-commerce landscape, we conducted extensive market research. This involved studying existing e-commerce platforms, analyzing their strengths and weaknesses, and identifying emerging trends and end users' preferences. By staying informed about the market dynamics, we ensured that our app would incorporate the latest industry standards and address the evolving needs of online businesses.

c. Stakeholder Interviews:

We conducted structured and non-structured interviews with key stakeholders, including business owners, and the main prospect customers. By engaging with stakeholders, we developed a deep understanding of their expectations, pain points, and desired features in an e-commerce platform. We also gained valuable insights into their unique requirements, operational workflows, and priorities. This information guided the development of our app to meet the needs of our target audience effectively.

This requirements-gathering phase served as the foundation for designing a solution that would meet the expectations of our target customers.

Functional Requirements

In the process of developing our e-commerce platform, it is essential to identify and define the functional requirements that will drive its design and implementation. These requirements outline the specific functionalities and capabilities that the platform must possess to meet the needs of both the Merchants (tenants) and the Shoppers (end users). By thoroughly analyzing the discovered user needs and business processes, we aim to formulate

clear, consistent, and complete functional requirements that will serve as the foundation for our platform's development.

By focusing on these requirements, we can create a robust and user-centric system that fulfills the diverse needs of the Merchants and the Shoppers, ultimately driving the success and adoption of our e-commerce solution.

The functional requirements are organized based on the primary actors of the system, namely the Merchants and the Shoppers. This approach allows for a focused examination of the unique requirements and expectations of each actor.

Merchant's Functional Requirements:

- **User Registration and Authentication:**
  - Create a trial account.
  - Login using username and password.
  - Logout.
- **Profile management:**
  - Edit merchant's personal profile information (name, profile picture, etc.).
  - Edit merchant's credentials.
- **Store Management**
  - Edit store's information (logo, name, description).
  - Manage store's admins (list, add, edit, delete).
- **Products Management:**
  - List, add, edit, and delete products.
  - Set product discount.
  - Categorize and tag products for easy search and navigation.
- **Categories Management:** List, add, edit, and delete categories.
- **Tags Management:** List, add, edit, and delete tags.
- **Coupons Management:** List, add, edit, and delete coupons.
- **Orders Management:**
  - View and manage shoppers' orders, including order status (for unconfirmed orders).
  - Process refunds and returns. *
  - Generate order invoices. *
- **Inventory Management:**
  - Track stock levels and inventory availability.
  - Receive notifications for low stock or out-of-stock items. *

    - Manage stock replenishment and reordering (track store's reordering of inventory items getting out of stock from suppliers). *
- **Store Customization**
    - Customize the store's branding, theme, and content.
    - Set up navigation menus.
    - Configure store policies, payment options, and shipping options.
    - View available home page layouts (default, customized).
    - Create custom home page layout:
        - Set up home page structure and product showcases.
        - Set up store banners and promotional content.
    - Select a home page layout.
- **Analytics and Reporting: ***
    - Generate sales reports and inventory reports.
    - Track product popularity.
    - Analyze customer behavior and preferences.

Note: The star (*) refers to requirements that were identified but not implemented.

Shopper's Functional Requirements:

- **User Registration and Authentication:**
    - Create an account.
    - Login using username and password.
    - Logout.
- **Profile Management:**
    - Edit personal profile information (name, profile picture, etc.).
    - Edit login credentials.
    - Manage shipping and billing addresses (add, edit, and delete).
- **Product Search and Navigation:**
    - View products by category, or tag.
    - Search for products based on keywords.
    - Sort products list by pertinence, price, or alphabetical order.
    - Filter products list by product attributes, price interval, brands, and tags (new, popular, on sale, etc.).
    - View detailed product information, including description, images, and reviews.
- **Comparison Tool Management:**
    - View a comparison table based on product attributes.
    - Add products to comparison tool.
    - Remove products from comparison tool.

- **Wishlist Management**
  - View wishlist.
  - Save products to wishlist.
  - Remove products from wishlist.
- **Order Management:**
  - View orders list
  - Edit unconfirmed orders.
  - Delete unconfirmed orders.
  - Get email notifications for order status updates.
- **Shopping Cart Management:**
  - Add products to the cart, update quantities, and remove items.
  - Apply discount coupon codes during checkout.
  - Choose from multiple payment options and provide shipping details.

By addressing the functional requirements in this phase, we lay the groundwork for subsequent design and implementation activities. These requirements serve as a guidepost for developing the core features and functionalities of the e-commerce platform, facilitating effective communication and alignment among the engineering team, stakeholders, and end users.

Quality Attributes

The success and acceptance of our e-commerce platform depends not only on its functionality but also on its ability to exhibit high-quality characteristics. The quality attributes define the non-functional requirements that shape the architecture, behavior, and performance of the system. These attributes play a vital role in creating a robust, reliable, and user-friendly platform that meets the expectations of both Merchants and Shoppers.

In this phase, we identify and prioritize the key quality attributes that our e-commerce platform should exhibit. Each attribute represents a critical aspect of the system's behavior and functionality, addressing specific concerns related to performance, reliability, security, usability, maintainability, and interoperability.

1. Performance:
   - Scalability: The system should be able to handle a growing number of tenants, increasing traffic, and expanding product catalogs without compromising performance.
   - Responsive and fast UI: The application should provide a smooth and seamless user experience with quick response times, especially during product browsing, search, and checkout processes.

- Efficient data processing: The application should efficiently handle data retrieval, processing, and storage to minimize response times and ensure smooth operations.

2. Security:

    - Authentication and authorization: The application should provide secure user authentication and authorization mechanisms to ensure that only authorized users can access and perform actions within the system.
    - Data privacy: The system should protect sensitive user information, such as personal and payment data, using encryption and secure communication protocols.
    - Protection against cross-tenant data leakage: The application should ensure that data from one tenant is securely isolated and cannot be accessed or manipulated by other tenants.

3. Maintainability:

    - Modular and decoupled architecture: The application should be designed with a modular and decoupled architecture, allowing for easier maintenance, updates, and future enhancements.
    - Code readability and maintainability: The codebase should follow best practices, coding standards, and be well-documented to facilitate ongoing maintenance and future development.
    - Version control: The system should have proper version control practices to streamline development activities.

4. Reliability:

    - High availability: The system should be highly available to ensure uninterrupted access for both merchants and shoppers, minimizing downtime and system failures.
    - Fault tolerance: The application should be designed to handle errors gracefully, recover from failures, and prevent data loss or corruption.

5. Usability:

    - Intuitive UI: The application should have a user-friendly interface with clear navigation, well-organized layouts, and intuitive controls for ease of use.

- Multilingual and localization support: The application should support multiple languages and provide localization features to accommodate users from different regions.

6. Interoperability:

    - Seamless communication and data exchange: The system needs to be able to interact and integrate with external systems, such as third-party payment gateways, shipping providers, or inventory management systems. This can involve implementing standard protocols, formats, or APIs that enable smooth integration and interoperability between the application and external systems.

4.1.3. Design Validation

In the Design Validation phase, our focus shifted towards reviewing and validating the proposed solution design before proceeding with the actual implementation. This critical phase ensures that the identified requirements effectively address the needs of both Merchants and Shoppers while adhering to the desired quality attributes, providing a solid foundation for the development of a scalable and efficient multi-tenant e-commerce platform.

To achieve this, the primary validation technique employed was gathering feedback and conducting reviews with stakeholders, including Merchants, Shoppers, and the engineering team. Involving stakeholders in the validation process is of utmost importance as their input helps in identifying potential issues, improving user satisfaction, and ensuring that the final solution is well-informed and meets the diverse needs of all involved parties. Therefore, we engaged in regular review and feedback sessions with stakeholders to present the solution design and gather their input. During these sessions, we also addressed any concerns or questions raised by stakeholders and incorporated their suggestions into the design where feasible.

Thanks to this collaborative approach, we gained confidence in the viability, feasibility, and effectiveness of our solution design. We were also able to address any areas for improvement before moving forward with the subsequent phases.

4.1.4. Solution Implementation

In the implementation phase of iteration 1, our objective was to transform the solution design into a comprehensive blueprint that lays the groundwork for the development of a robust and scalable multi-tenant e-commerce platform. To accomplish this, we focused on translating the identified functional requirements and quality attributes into concrete representations, such as use case diagrams and quality tactics. Additionally, we made critical technological choices that will set the foundation for ensuring the desired quality attributes.

To guide us throughout the various iterations of the development process, we will employ the 4+1 View Model, a widely recognized framework that provides a holistic perspective of the system. This model presents the software architecture from multiple viewpoints, each catering to the specific interests and concerns of different stakeholders involved in the project. By utilizing the 4+1 View Model, we can capture essential details of the system while ensuring cohesion and consistency across diverse architectural aspects.

Figure 8: The 4+1 Architectural View Model [41]

Use Case View

In this section, we will leverage the Use Case View to provide a comprehensive description of how the system behaves and interacts with its users and external entities.

The system will cater to two main actors: Merchants and Shoppers. As a result, the system will be composed of two distinct subsystems, each addressing the specific needs and interactions of these actors:

- ***Back-Office Subsystem***: The Back-Office subsystem is a crucial component of our multi-tenant e-commerce platform, designed to cater to the needs of Merchants. This subsystem serves as a centralized hub, providing Merchants with essential tools and functionalities to effectively manage their online stores. Through a user-friendly interface, Merchants can efficiently handle various aspects of their stores, such as store data management, inventory control, and order processing. Additionally, the platform offers customization options, empowering Merchants to tailor their stores to align with their unique branding and customer experience preference. The use cases for the Back-Office Subsystem are represented in Figure 9 below.

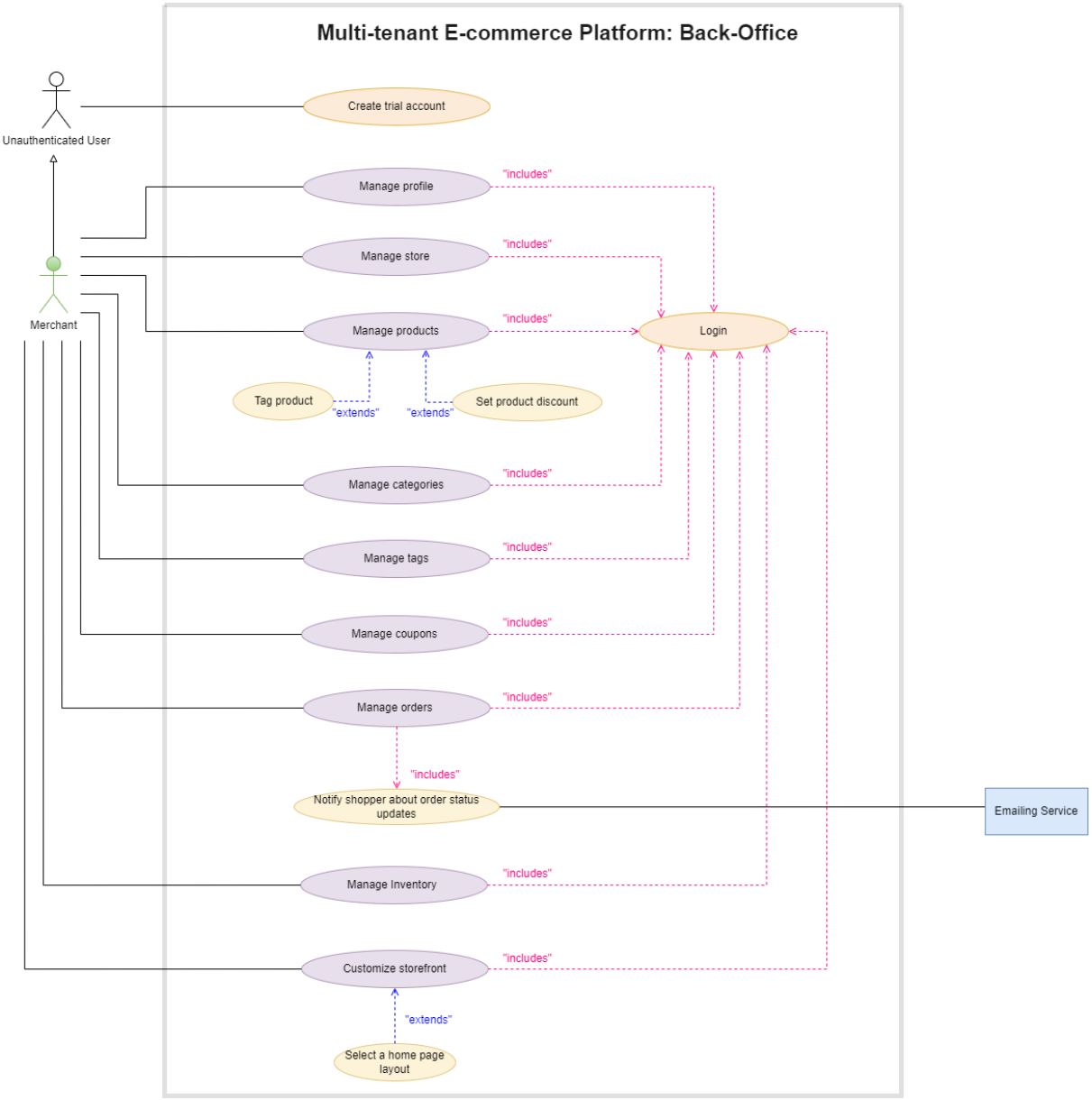

Figure 9: Use Case Diagram for the Back-Office subsystem

- ***Front-Office Subsystem***: The Front-Office subsystem represents the user-facing interface of our multi-tenant e-commerce platform, catering to the needs of Shoppers. This subsystem provides Shoppers with a seamless and intuitive shopping experience, allowing them to browse products, create accounts, and place orders with ease. Unauthenticated Shoppers can explore the platform as guests, getting a glimpse of the available products and offerings. Upon registration and login, Shoppers gain access to additional features, such as wishlist management, which allows them to keep track of desired products for future purchases. The platform's product browsing capabilities facilitate efficient product discovery, ensuring Shoppers can easily find and select the items they desire. The different use cases of the Front-Office Subsystem are presented in figure 10 below.

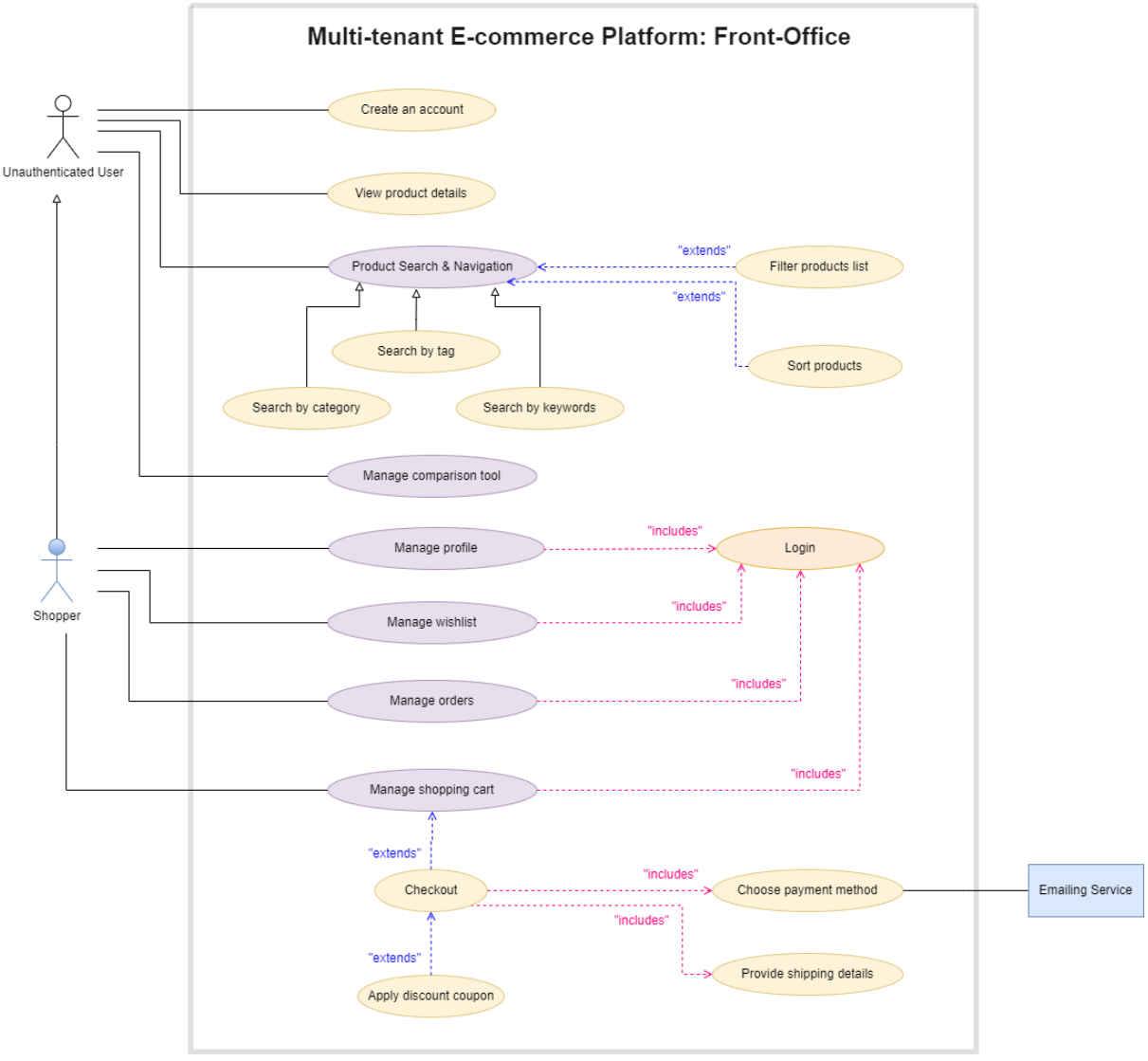

Figure 10: Use Case Diagram for the Front-Office subsystem

Quality Tactics

To ensure that our platform exhibits the identified quality attributes, we will define specific quality tactics. These tactics are concrete approaches that address each quality attribute's goals and requirements. By implementing these tactics, we can proactively manage performance, reliability, security, usability, maintainability, and interoperability aspects throughout the development process.

a. Performance

To ensure optimal performance and responsiveness of our multi-tenant e-commerce platform, we have identified specific quality tactics that address key performance goals and requirements.

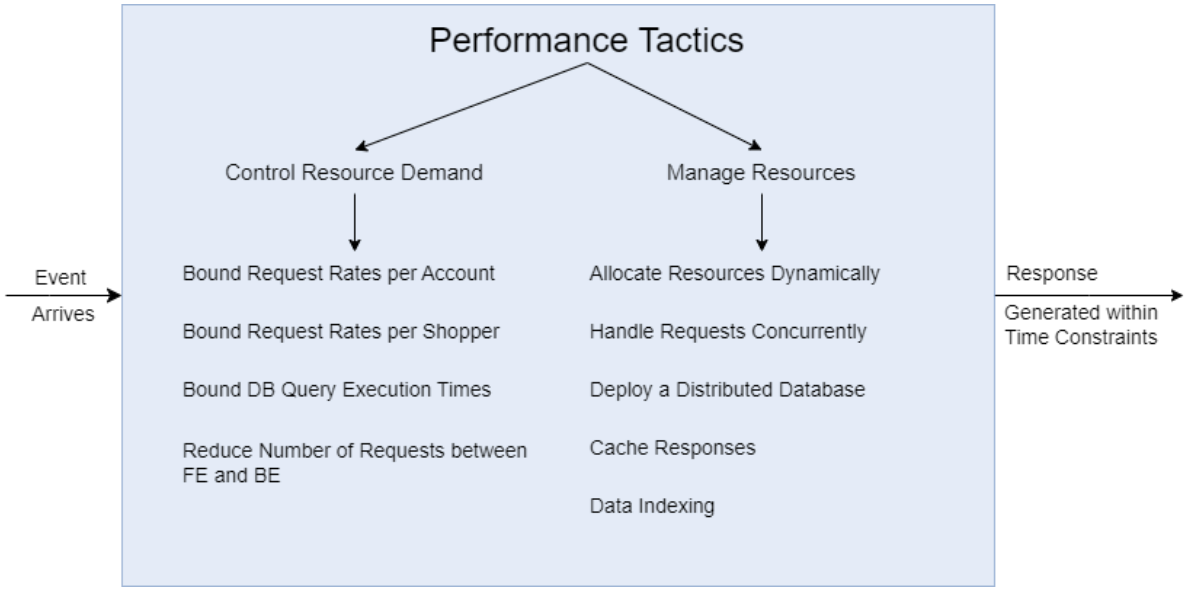

Figure 11: Performance Quality Tactics

Control Resource Demand:

- Bound Request Rates per Account: By setting limits on the number of requests allowed per account, we can prevent any single tenant from overloading the system and affecting others' performance.
- Bound Request Rates per Shopper: Limiting the number of requests per shopper helps distribute the workload evenly, ensuring a smooth experience for all users.
- Bound Database Query Execution Times: We will implement strategies to optimize database queries and set limits on their execution times to prevent long-running queries from impacting overall system responsiveness.

- Reduce Number of Requests between FE and BE: Minimizing the number of requests between the FE (Front-End) and BE (Back-End) components reduces communication overhead and improves system performance.

Manage Resources:

- Allocate Resources Dynamically: To handle varying workloads, we will dynamically allocate resources, scaling up or down as needed to meet demand efficiently.

- Handle Requests Concurrently: Implementing concurrent request handling allows the system to process multiple requests simultaneously, improving response times and overall system throughput.

- Deploy a Distributed Database: A distributed database architecture enables data to be stored and accessed efficiently across multiple nodes, enhancing data retrieval speed and system performance.

- Cache Responses: Caching frequently requested data and responses reduces the need for repeated processing, resulting in faster access times for users.

- Data Indexing: Implementing efficient data indexing techniques optimizes data retrieval, ensuring that queries are executed more rapidly.

By adopting these performance quality tactics, we aim to provide a high-performance e-commerce platform that meets the demands of our tenants and shoppers, delivering a seamless and satisfying user experience.

b. Availability

Ensuring high availability of our multi-tenant e-commerce platform is paramount to provide uninterrupted access for merchants and shoppers. To achieve this, we have devised specific availability tactics to handle faults, recover from failures, and prevent potential issues.

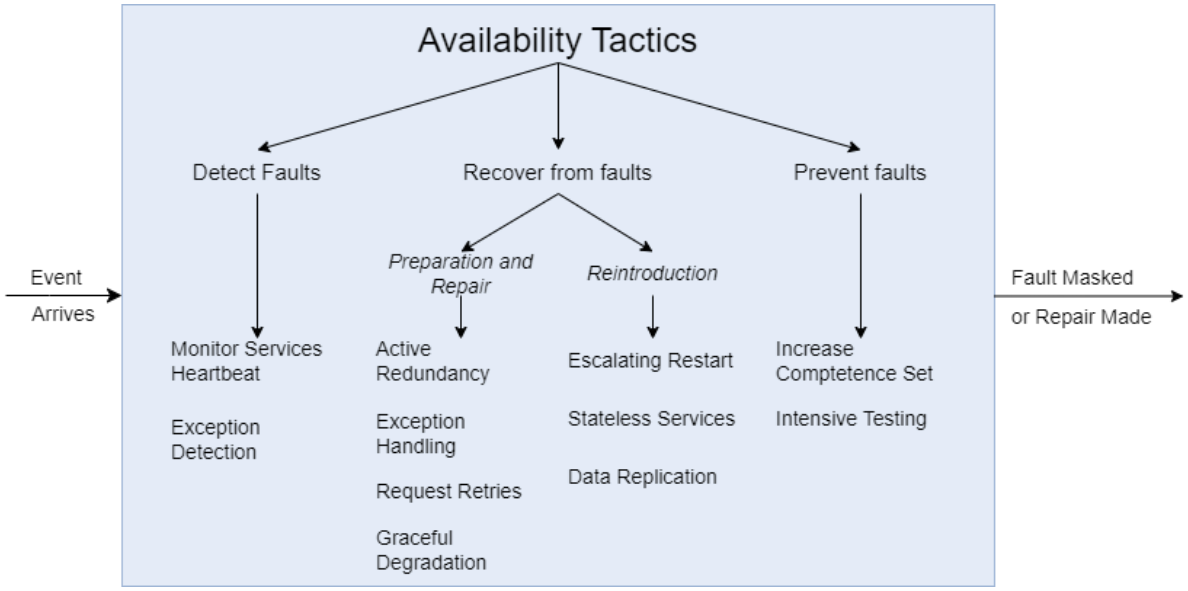

Figure 12: Availability Quality Tactics

Detect Faults:

- Monitor Services Heartbeat: By continuously monitoring the heartbeat of services, we can promptly detect any anomalies or service failures, allowing us to take timely corrective actions.
- Exception Detection: Implementing robust exception detection mechanisms helps us identify potential issues in the system, enabling swift response to mitigate or address faults.

Recover from Faults:

- Preparation and Repair:
    - Active Redundancy: Employing active redundancy by having backup instances ready to take over in case of a failure helps ensure seamless service continuity.
    - Exception Handling: Implementing comprehensive exception handling protocols enables the system to recover gracefully from unexpected errors, minimizing downtime.
    - Request Retries: Automatic retries for failed requests can help recover from transient failures, reducing the impact on users and ensuring service availability.

- Graceful Degradation: In situations of high load or system stress, implementing graceful degradation allows the platform to prioritize critical functions, maintaining essential services during periods of instability.

- Reintroduction:

    - Escalating Restart: Gradually restarting services with an escalation strategy minimizes potential service disruptions during the recovery process.

    - Stateless Services: Using stateless service architecture helps simplify the recovery process by allowing service instances to be recreated without relying on stored state information.

    - Data Replication: Replicating critical data across multiple locations ensures data availability even in the event of hardware failures.

Prevent Faults:

- Increase Competence Set: By continuously improving and expanding the expertise of our development and operations teams, we enhance the platform's stability and resilience.

- Intensive Testing: Rigorous and comprehensive testing, including load testing, stress testing, and fault injection, helps identify potential weak points and strengthens the platform's ability to handle various scenarios.

Through the strategic implementation of these availability tactics, we are dedicated to providing a robust and highly available e-commerce platform, ensuring a reliable and seamless shopping experience for all users.

c. Security

Security is of paramount importance in our multi-tenant e-commerce platform to protect sensitive data, maintain confidentiality, and ensure a secure shopping environment for merchants and shoppers. To achieve robust security, we have implemented a set of security tactics, including detection, resistance, reaction, and recovery measures.

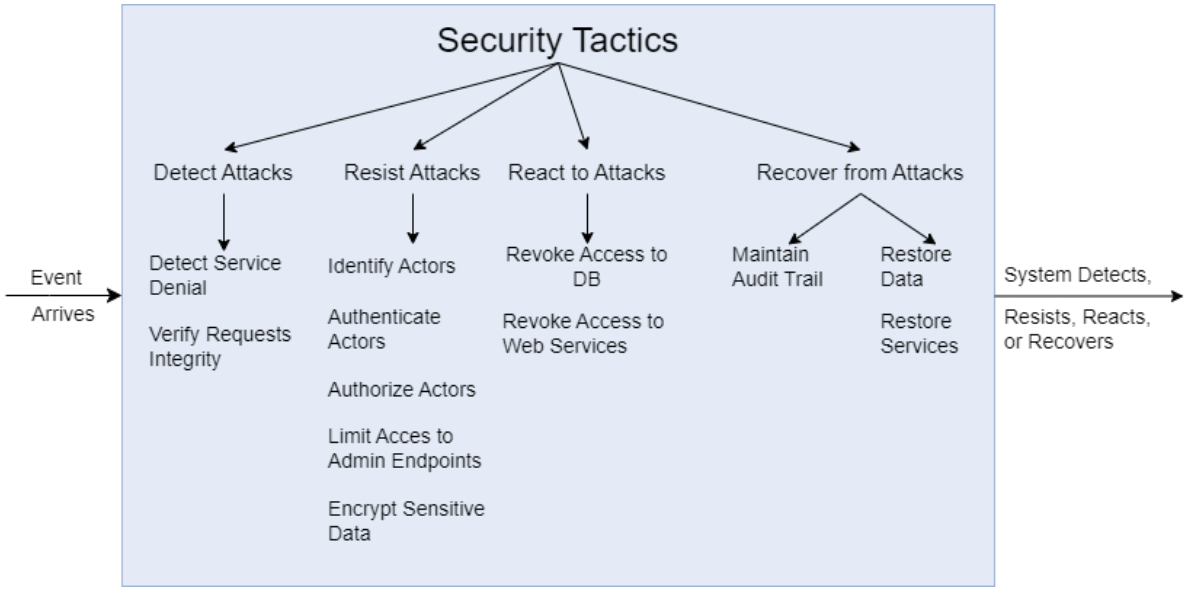

Figure 13: Security Quality Tactics

Detect Attacks:

- Detect Service Denial: Employing techniques to detect DoS attacks helps identify and mitigate attempts to overwhelm the platform's resources, ensuring continuous service availability.
- Verify Requests Integrity: Implementing request integrity checks, such as digital signatures and validation, helps detect and prevent tampering with data during transmission.

Resist Attacks:

- Identify Actors: Thoroughly identifying and authenticating users and external entities helps prevent unauthorized access to the platform, ensuring that only legitimate actors can interact with the system.
- Authenticate Actors: Strong user authentication mechanisms, including multi-factor authentication, enhance security and protect against unauthorized access.
- Authorize Actors: Implementing role-based access control allows us to define specific permissions for different user roles, limiting access to sensitive functionalities.
- Limit Access to Admin Endpoints: Restricting access to administrative endpoints and functions to authorized personnel prevents unauthorized changes to critical settings and configurations.

- Encrypt Sensitive Data: Utilizing encryption techniques for storing and transmitting sensitive data ensures that even if data is compromised, it remains unreadable and unusable to unauthorized parties.

React to Attacks:

- Revoke Access to Database: In the event of a security breach, promptly revoking access to the database prevents further unauthorized access and potential data leakage.

- Revoke Access to Web Services: Revoking access to web services helps contain the impact of an attack and prevents unauthorized control over the platform.

Recover from Attacks:

- Maintain Audit Trail: Implementing an audit trail allows us to track and log user activities, facilitating the identification and investigation of security incidents.

- Restore Data: Regularly backing up critical data and implementing data recovery processes enables us to restore data to a clean state in case of data breaches.

- Restore Services: In the event of an attack affecting service availability, restoring services quickly and efficiently ensures minimal disruption to merchants and shoppers.

Through the implementation of these security tactics, we aim to provide a secure and trusted e-commerce platform, safeguarding user data, and maintaining a resilient defense against potential security threats.

d. Maintainability

Maintainability is a crucial aspect of our multi-tenant e-commerce platform, ensuring that the system can be easily updated, enhanced, and adapted to meet future needs without compromising its stability or performance. To achieve maintainability, we have employed a set of tactics focused on reducing module size, increasing cohesion, and minimizing coupling.

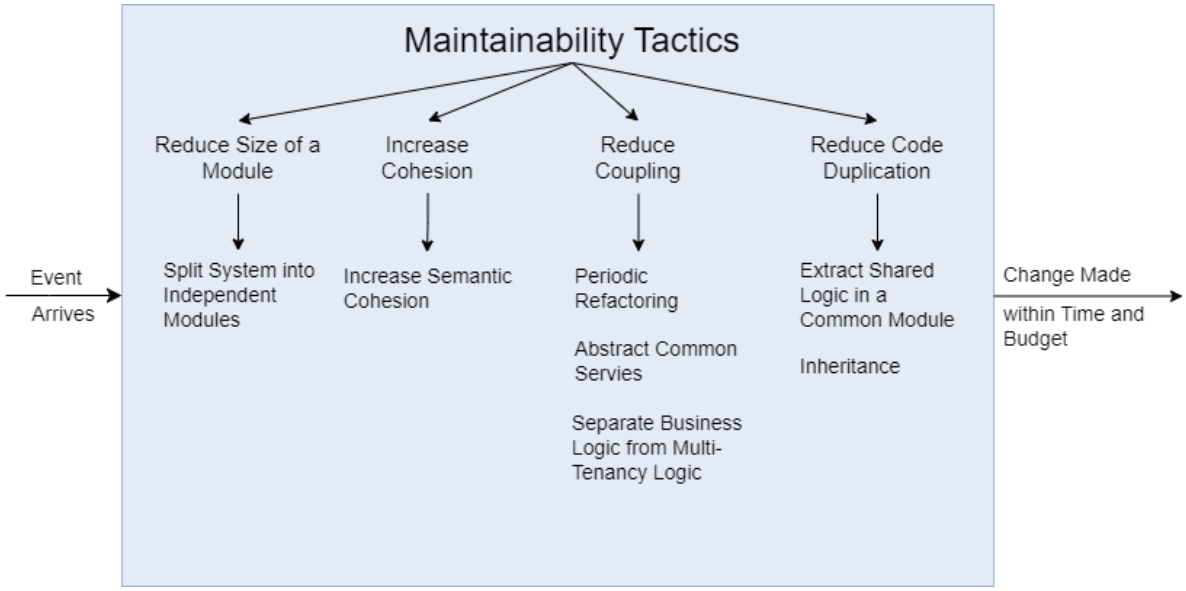

Figure 14: Maintainability Quality Tactics

Reduce Size of a Module:

- Split System into Independent Modules: Breaking down the system into smaller, independent modules reduces complexity and allows for more straightforward updates and modifications in isolated areas, enhancing maintainability.

Increase Cohesion:

- Increase Semantic Cohesion: Enhancing semantic cohesion within modules ensures that each module's components are closely related and focused on performing a specific set of related tasks, making it easier to comprehend, update, and maintain.

Reduce Coupling:

- Periodic Refactoring: Regularly reviewing and refactoring the codebase allows us to identify and eliminate unnecessary dependencies between modules, reducing the risk of unintended side effects during future updates.

- Abstract Common Services: Abstracting common functionalities into separate services allows for greater code reuse and flexibility, making future modifications more straightforward and reducing the impact of changes across the system.

- Separate Business Logic from Multi-Tenancy Logic: By decoupling the business logic from the multi-tenancy logic, we can achieve a more modular architecture, simplifying maintenance and updates to each aspect independently.

Reduce Code Duplication:

- Extract Shared Logic in a Common Module: Identifying and extracting shared functionalities into a common module minimizes redundant code, making it easier to maintain and update the shared logic across the platform.

- Inheritance: Utilizing inheritance appropriately allows us to inherit properties and behaviors from parent classes, reducing code duplication and promoting maintainability through a more structured and organized codebase.

By implementing these maintainability tactics, our e-commerce platform can evolve and adapt efficiently, ensuring that future changes and enhancements can be seamlessly integrated while preserving the platform's overall stability and performance.

e. Usability

To deliver a user-friendly and intuitive interface for our multi-tenant e-commerce platform, we have devised a set of usability tactics based on Nielsen's Ten Usability Heuristics and other best practices. These tactics aim to ensure that the system is easy to use, efficient, and satisfying for both merchants and shoppers.

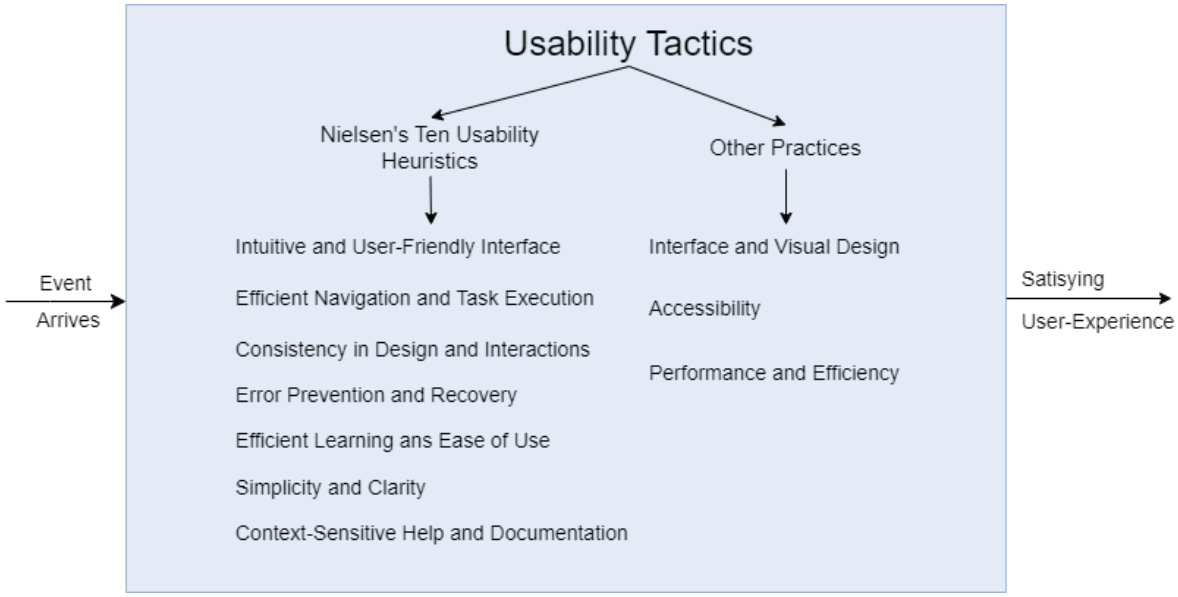

Figure 15: Usability Quality Tactics

Nielson's Ten Principles:

- **Intuitive and User-Friendly Interface:**

1. Match between System and the Real World: Design an interface that reflects the mental model of users, using language and concepts familiar to them.

- **Efficient Navigation and Task Execution:**

2. Visibility of System Status: Provide clear and real-time feedback on system status.

3. User Control and Freedom: Enable users to undo, redo, and exit undesirable states without negative consequences.

- **Consistency in Design and Interactions:**

4. Consistency and Standards: Maintain consistency in design and interactions, following established design conventions and standards.

- **Error Prevention and Recovery:**

5. Error Prevention: Implement design elements that prevent errors.

6. Help Users Recognize, Diagnose, and Recover from Errors: Provide guidance and assistance to users in case of errors.

- **Efficient Learning and Ease of Use:**

7. Recognition rather than Recall: Minimize the need for users to memorize information by presenting necessary options and actions explicitly.

8. Flexibility and Efficiency of Use: Provide shortcuts and efficient methods for experienced users.

- **Simplicity and Clarity:**

9. Aesthetic and Minimalist Design: Keep the interface simple, free from unnecessary elements, and aesthetically pleasing.

- **Context-Sensitive Help and Documentation:**

10. Help and Documentation: Offer context-sensitive help and documentation that is easily accessible when users need it.

Other practices:

- **User-Centered Design:**
    - User-Centric Design**:** Adopt a user-centric design approach by involving end-users, conducting usability testing, and gathering feedback throughout the development process. Understanding user needs and preferences helps create a platform that aligns with user expectations and behaviors.
    - Personalization and Customization**:** Incorporate personalization features based on user preferences and past behavior. Customizing product recommendations, offers, and content enhances user engagement and encourages repeat visits.
- **Interface and Visual Design:**
    - Intuitive Navigation: Design an intuitive and straightforward navigation system that allows users to easily find products, navigate between different sections, and complete tasks with minimal effort. Clear and consistent menus, categories, and search functionalities contribute to a seamless browsing experience.
    - Clear Calls-to-Action (CTAs)**:** Use visually distinct and prominent CTAs throughout the platform to guide users through the shopping journey. Well-designed CTAs prompt users to take specific actions, such as adding items to the cart or completing a purchase.
- **Accessibility:**
    - Multilingual Support**:** Provide multilingual support to cater to users from diverse linguistic backgrounds. Offering content and interfaces in multiple languages increases accessibility and widens the platform's reach.
- **Performance and Efficiency:**
    - Efficient Data Processing: Optimize data retrieval, storage, and manipulation within the e-commerce platform. By employing techniques such as data indexing, caching, and asynchronous processing, the system can ensure faster response times and efficient handling of large datasets. This tactic plays a crucial role in maintaining a seamless user experience and supporting the platform's scalability as the amount of data grows.

- Performance Optimization**:** Optimize the platform's performance to minimize loading times and reduce latency. A fast and responsive platform enhances user satisfaction and encourages continued usage.

By implementing these usability tactics, our e-commerce platform will deliver an engaging and seamless user experience. These tactics, combined with other quality attributes, will contribute to the overall success and satisfaction of both merchants and shoppers, fostering long-term engagement and success for our multi-tenant e-commerce platform.

f. Interoperability

Interoperability plays a crucial role in our multi-tenant e-commerce platform, enabling seamless communication and data exchange with third-party services and systems. To achieve a high level of interoperability, we have employed specific tactics focused on locating third-party services and managing interfaces.

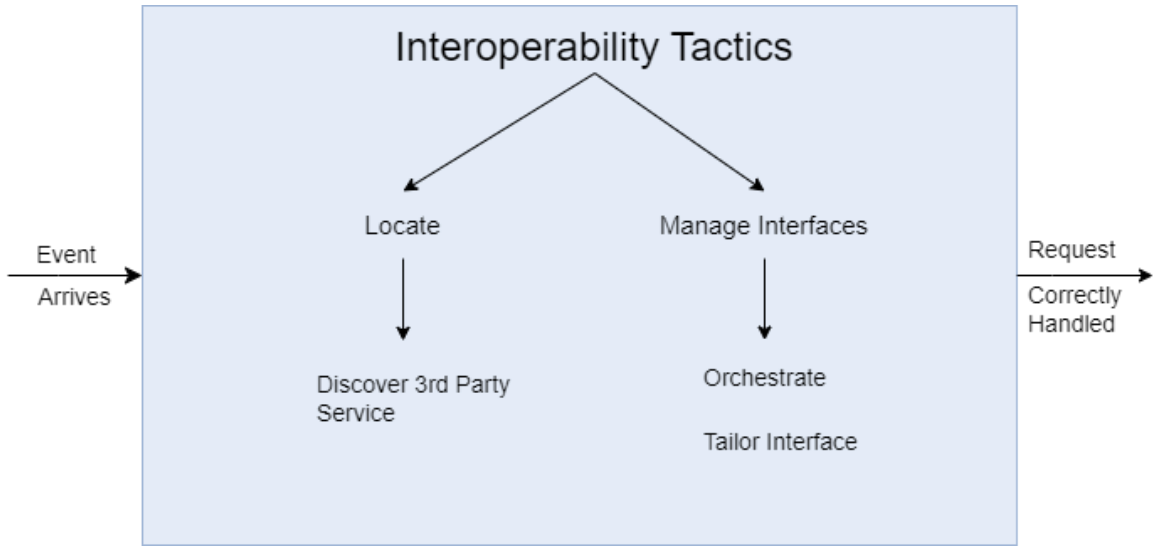

Figure 16: Interoperability Quality Tactics

Locate:

- Discover 3rd Party Service: Actively locating and discovering relevant third-party services is vital for integrating external functionalities into our platform. By identifying and understanding these services, we can effectively establish connections and exchange data.

Manage Interfaces:

- Orchestrate: Orchestration involves managing the flow of data and interactions between different services and systems. By implementing a robust orchestration

mechanism, we ensure that data is exchanged efficiently and cohesively, enhancing interoperability.

- Tailor Interface: Adapting the interface of our platform to suit the specific requirements of integrated third-party services ensures smooth communication and interaction. By tailoring the interface, we enhance compatibility and minimize potential conflicts during integration.

Through these interoperability tactics, our e-commerce platform can interact seamlessly with external systems, enabling efficient data exchange and providing enhanced functionalities for both merchants and shoppers.

Ensuring Quality Attributes Through Technological Choices

In the process of crafting a robust multi-tenant e-commerce platform, the selection of appropriate technologies becomes a crucial step that profoundly impacts its performance, scalability, reliability, usability, maintainability, and interoperability. This section delves into the rationale behind our technology selections and sheds light on how each choice reinforces specific quality attributes.

Back-End: Node.js, Express.js, Docker

Node.js:

- Node.js is a server-side JavaScript runtime that allows us to build scalable and efficient applications. It uses an event-driven, non-blocking I/O model, making it particularly suitable for handling input/output heavy tasks and asynchronous operations.
- Why Node.js:
    a. Input/Output heavy tasks: Node.js excels at handling tasks that involve extensive I/O operations, such as reading and writing to databases or interacting with external APIs, due to its non-blocking nature. This ensures that the platform can efficiently manage multiple requests without getting blocked by long-running tasks. As a result, the platform maintains its responsiveness even during periods of peak loads and increased traffic. This directly contributes to enhancing the platform's **performance** and **availability** quality attributes.
    b. No complex calculations: As our e-commerce platform is primarily focused on managing data, product listings, and handling user interactions, it does not involve heavy computational calculations. As a result, Node.js's limitation in handling complex calculations does not adversely affect the platform's performance. This factor solidifies Node.js as a well-suited choice for our backend development.

Express.js:

- Express.js is a popular and lightweight web application framework for Node.js, providing essential features and utilities for building web APIs and handling Hypertext Transfer Protocol (HTTP) requests and responses effectively.
- Why Express.js:
    a. General purpose and flexible: Express.js is a versatile framework that can cater to a wide range of applications, allowing us to tailor our BE logic to suit the specific needs of our multi-tenant e-commerce platform.
    b. Provides facilities for creating web APIs: Our platform's requirement for seamless communication between the FE and BE led us to choose Express.js for web API development. Express.js is known for creating robust and well-structured web APIs, allowing for smooth data exchange and interaction across various subsystems. This design approach not only supports seamless integration but also aids in maintaining a clear and organized codebase. Therefore, Express.js enhances the **maintainability** and **interoperability** quality attributes of our platform.
    c. Widely used, large community, and rich documentation: Express.js boasts a large and active community of developers, which results in comprehensive documentation and a wealth of available resources. This extensive support network ensures that we can efficiently troubleshoot issues and find solutions for any challenges that may arise during the development process. Additionally, this popularity means that skilled developers familiar with Express.js can be readily engaged, accelerating the implementation process and promoting productivity.

Docker:

- Docker is a versatile platform designed to create, deploy, and manage applications efficiently using containers. It provides a standardized environment for applications, ensuring consistency across various stages of the SDLC.
- Why Docker:
    a. Streamlined Deployment and Consistency: Docker's containerization approach ensures that applications run consistently across different environments, from development to production. This uniformity minimizes the chances of unexpected issues arising due to environment disparities. As a result, the platform's deployment process is streamlined, enhancing the **maintainability** and **stability** of the application.
    b. Scalability and Resource Efficiency: Docker's container-based architecture allows for effortless scaling of individual components, which contributes to horizontal **scalability**. This dynamic scaling capability supports the platform's **performance**

during times of increased traffic or demand. Additionally, the efficient use of resources within containers aids in optimizing resource consumption and maintaining high-performance levels.

c. Isolation and Security: Docker ensures application components run within isolated containers, preventing conflicts and minimizing potential security vulnerabilities. This isolation enhances **security** by adding an extra layer of protection and reducing the attack surface.
d. Rapid Development and Testing: The ability to create consistent development, testing, and production environments with Docker accelerates the SDLC. Developers can work in identical environments as the production setup, reducing the chances of issues arising during deployment. This efficient development and testing process enhances the overall development workflow, which positively impacts the platform's **maintainability** and **reliability**.
e. Continuous Integration and Delivery (CI/CD): Docker's containerization aligns seamlessly with CI/CD pipelines, facilitating automated testing and deployment processes. The platform's CI/CD integration ensures consistent and reliable releases, enhancing the platform's **reliability** quality attribute.

Database: MongoDB

- MongoDB is a NoSQL, document-oriented database that provides a flexible and scalable solution for storing and managing data in a non-structured format. It stores data in BSON (Binary JSON) format, allowing for dynamic schema definition, making it well-suited for handling the diverse data needs of our multi-tenant e-commerce platform.
- Why MongoDB:
    - Flexible data storage: MongoDB's document-based model allows us to store data in a schema-less manner, accommodating changes and variations in data structures over time. This **flexibility** aligns with the dynamic nature of an e-commerce platform, where product data, user information, and other entities may evolve and expand over the platform's lifecycle, thus contributing to the platform **maintainability**.
    - Distributed and easy to scale with sharding: MongoDB supports horizontal scaling through sharding, enabling us to distribute data across multiple servers. This capability ensures that the database can handle a growing number of tenants, products, and user interactions without compromising **performance** or responsiveness, contributing to the platform's **scalability**.

- Simple to use with the possibility to use advanced queries: MongoDB offers a simple to use query language, making it easy for developers to work with and extract data. Moreover, it provides advanced query capabilities, such as indexing and aggregation, which enhance data retrieval **performance** and support complex data analysis tasks.
- Support for data replication: MongoDB has built-in support for data replication, allowing us to maintain multiple copies of the database across different servers. This replication ensures data redundancy and high **availability**, contributing to the platform's **reliability** and **fault tolerance**, and ensuring that critical data is safeguarded and accessible even in the event of hardware failures or network issues.

Front-End: React, Redux Toolkit, Redux Saga, and Material UI

- React: React is a popular JavaScript library for building user interfaces. It allows us to create dynamic and interactive components that efficiently update and render only the necessary parts of the UI when data changes. React's virtual DOM enables smooth and fast rendering, contributing to the platform's responsive UI and improved **performance**.
- Redux Toolkit: Redux Toolkit is a set of tools and conventions that simplify the management of the application's state using Redux, a predictable state container. By using Redux Toolkit, we can efficiently manage the state of our e-commerce platform, enabling seamless data flow and ensuring consistent application behavior across different components. This organization of state contributes to the platform's **maintainability** and reduces potential bugs caused by state-related issues.
- Redux Saga: Redux Saga is a middleware library for handling side effects in Redux applications, such as asynchronous data fetching and API calls. By using Redux Saga, we can manage complex asynchronous operations in a more organized and readable manner. This capability ensures smooth handling of data retrieval and communication with the BE, supporting the code **maintainability** and the platform's **performance** and responsiveness.
- Material UI: Material UI, a sophisticated React UI framework built on Google's Material Design principles, enhances various quality attributes of our platform's user interface. Its ready-to-use components, designed with **usability** and consistency in mind, accelerate development while ensuring a visually appealing and user-friendly experience. Moreover, the responsiveness of Material UI components contributes to a seamless experience across devices, reinforcing the platform's commitment to providing a high-quality user interface.

Overall, the strategic selection of Node.js, Express.js, Docker, and MongoDB in our BE architecture and database design significantly bolsters our platform's performance, scalability, reliability, security, and maintainability. Additionally, the combination of React, Redux Toolkit, Redux Saga, and Material UI forms a powerful FE technology stack that aligns with our objective of creating a powerful, user-friendly, and performant e-commerce platform that meets the requirements and quality attributes identified in the previous phases.

4.1.5. Solution Evaluation

The solution implementation in Iteration 1 was subject to a comprehensive evaluation process to assess its alignment with the identified functional requirements and quality attributes. Stakeholder feedback played a pivotal role in this evaluation, as their insights were invaluable in assessing the effectiveness and accuracy of the proposed solution.

During the evaluation phase, meetings were conducted with key stakeholders, including merchants, shoppers, and the engineering team. These collaborative sessions provided an opportunity for stakeholders to express their views and opinions on the implemented solution. By engaging stakeholders in the evaluation process, we gained a deeper understanding of their specific needs and expectations, enabling us to fine-tune the solution to better cater to their requirements.

The use case diagram and the quality attribute tactics developed during the solution implementation phase were thoroughly examined during the evaluation. Stakeholders verified that the use cases accurately represented the system's behavior and interactions, addressing their respective roles and needs within the platform. Additionally, the quality attribute tactics were scrutinized to ensure they aligned with the desired performance, reliability, security, usability, maintainability, and interoperability aspects of the platform.

The evaluation process yielded positive results, with the solution demonstrating a strong correspondence to the stakeholders' real needs and expectations. The feedback from stakeholders helped us identify areas of strength and opportunities for further improvement as we progress with subsequent iterations.

Overall, the evaluation phase reaffirmed the effectiveness of our approach in addressing the challenges identified during the problem investigation phase. The valuable insights obtained during the evaluation phase guided us in making data-driven decisions and iterative improvements as we progressed through subsequent development iterations.

4.2. Iteration 2: Architectural Design

In this iteration, our focus shifts to the architectural aspect of the multi-tenant e-commerce platform. Building upon the use case view discussed in the previous phase, we will delve into the comprehensive exploration of the 4+1 architectural model, which encompasses four distinct views: process, logical, development, and physical. Each of these views offers unique insights into the system, catering to the diverse needs and interests of stakeholders involved in the development process. By thoroughly examining these perspectives, we lay the groundwork for a well-structured and cohesive architecture that aligns with the project's goals and quality attributes. This iteration plays a pivotal role in shaping the foundation of our e-commerce platform and ensuring its seamless integration, robustness, and efficient functionality across all aspects of development.

4.2.1. Problem Investigation

In this iteration, we shift our focus to the critical aspect of architectural design in the development of our multi-tenant e-commerce platform. While requirements engineering is undeniably essential, the architecture plays a pivotal role in driving the success of the entire project. A well-structured architecture is the backbone of the system, and any shortcomings in this phase can lead to significant financial losses and render the project output useless.

To ensure a successful outcome, meticulous attention should be given to carefully structuring the problem and deriving architectural requirements from the identified functional requirements and quality attributes. By aligning the architecture with the desired performance, reliability, security, usability, maintainability, and interoperability aspects outlined in the previous iteration, we lay the foundation for a robust and efficient system that meets the needs of both merchants and shoppers.

An essential consideration in this iteration is selecting the most suitable architectural style for our multi-tenant e-commerce platform. The right choice will ensure the correct static organization of the software in its development environment, promoting modularity and reusability while accommodating the system's evolving needs.

Additionally, we recognize that the runtime behavior of the system can be complex and challenging to understand fully. Misunderstandings or misinterpretations of interactions between components can lead to costly mistakes. Therefore, we will diligently analyze and validate the runtime behavior to ensure its correctness and efficiency.

Furthermore, the deployment phase presents its own set of challenges, as errors and mistakes during the system infrastructure build can have severe consequences.

By addressing these architectural considerations with precision and foresight, we aim to create an e-commerce platform that delivers the needed performance, reliability, security, usability, maintainability, and interoperability, positioning it for success in the competitive digital landscape.

4.2.2. Solution Design

In this iteration, we address the challenges identified earlier by delving into the four views of the 4+1 view model. This comprehensive approach provides valuable insights into the system from different perspectives, enabling us to devise effective solutions. In this section, we will focus on explaining the logical and process views, while the development and physical views will be thoroughly examined during the solution implementation phase. By exploring each view, we aim to establish a cohesive and well-structured architectural framework that aligns with our project's objectives and quality attributes, ensuring the successful development of our multi-tenant e-commerce platform.

Logical View

The logical view is dedicated to representing the inherent static organization and structure of a software system. It emphasizes the relationships, dependencies, and interactions among different entities within the system, while also providing a high-level abstraction of its overall functionality. To visualize this perspective, we have created a class diagram, which offers a graphical depiction of the logical view, showcasing the key classes and their associations.

Before delving into the intricacies of the class diagram, it is essential to consider the crucial aspect of data storage strategy. Developing a multi-tenant e-commerce platform necessitates a thoughtful approach to effectively manage data for multiple merchants while ensuring cost efficiency and scalability. In our multi-tenant e-commerce platform, we explored the three possible data storage approaches existing in the state-of-art:

Table 2: Data Storage Approaches in Multi-tenant Systems

Approach	Description	Pros	Cons
Separate Database for Each Tenant	Hosts a dedicated database for each store, ensuring high flexibility and performance. However, the implementation complexity and cost inefficiency make it impractical for our platform, where the target customers are small businesses with relatively low data and traffic requirements.	- High flexibility and performance for each store.	- Implementation complexity and cost inefficiency.
Shared Database with Separate Collections	Each store's data is stored in separate collections within the database (e.g., Product_store1, Product_store2). While more cost-effective than the first option, this approach is technically challenging and less scalable due to the limited number of allowed collections, especially in databases like MongoDB.	- More cost-effective than a separate database for each store.	- Technically challenging and less scalable due to limited collection limits.
Shared Database with Shared Collections	Utilizes a common database with shared collections for all stores. Each store-specific entity (e.g., products) includes a "storeId" field to associate it with the respective store. Strikes a balance between cost-effectiveness and ease of implementation. Overcomes scalability constraints associated with collection limits.	- **Cost**-effective and easier to implement than the separate database approach. - Overcomes **scalability** constraints associated with collection limits. - Efficiently manages data for multiple tenants without excessive overhead.	- Not as flexible as a separate database for each store.

Given the project's objectives and considerations, we have decided to adopt the third solution.

Our class diagram below reflects the chosen approach, showcasing the "Store" class with a composition relationship with other classes. This relationship ensures that the existence of all other classes is dependent on the "Store" class, meaning that if the "Store" class is destroyed, all related entities will also be removed.

Additionally, to accommodate the shared data structure for shoppers and merchants, we use a single "User" class for both with the attribute "role" to differentiate them during authentication and authorization. Shoppers and merchants share all attributes except for "shopperInfo". Leveraging the flexibility of NoSQL databases, we can accommodate this distinction, allowing a "User" to have zero or one "shopperInfo" object, depending on their role.

To ensure that product updates do not impact old purchasing orders, we decided to duplicate some of the product's information in the purchasing order using the attribute "products" of type "ProductInfo". This guarantees that merchants have access to the original product information even if the product's details change over time. For instance, if the price of a product increases, having access to the original price in past orders allows merchants to handle potential returns more effectively.

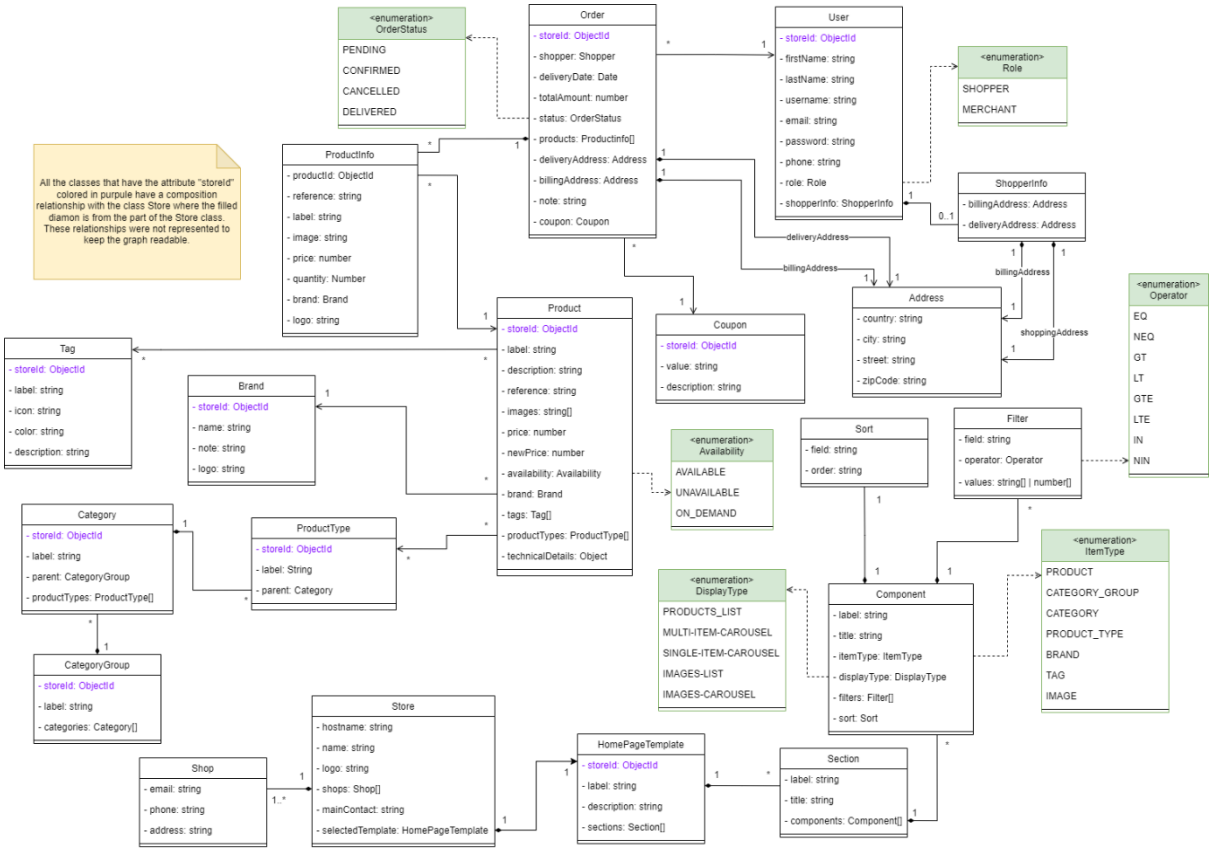

Figure 17: Data Storage Class Diagram

The logical view, coupled with the data storage strategy and corresponding class diagram, sets a solid foundation for our multi-tenant e-commerce platform's development phase. With a clear understanding of the static organization and data storage mechanisms, we are well-equipped to proceed with the implementation, bringing our platform to life and delivering a seamless experience to both merchants and shoppers.

Process View

In the context of the 4+1 view model, the process view centers on capturing the dynamic behavior and runtime aspects of the software system. It offers valuable insights into how the system operates and how various processes are executed to accomplish the system's functionalities effectively.

Sequence diagrams will be utilized to illustrate the chronological flow of interactions between various system elements, showcasing the dynamic exchange of messages and actions during the execution of critical use cases. This will offer valuable insights into the sequence of events, aiding in the identification of potential bottlenecks and areas for optimization.

In addition to sequence diagrams, we will employ an activity diagram to model the workflow and procedural logic of the most important use case, which is the order functionality workflow. This activity diagram will provide a comprehensive overview of the tasks, decisions, and branching scenarios that occur within the system during the execution of the use case. By visualizing these activities, we can gain a deeper understanding of the system's behavior, ensuring efficient task execution and improved user experience.

Product Creation Scenario (Merchant)

Adding products is a critical use case for merchants, and we illustrate it through a white-box sequence diagram. The process initiates when the Merchant decides to add a product, triggering the creation of a Product Form Popup. The Merchant then proceeds to insert all relevant product information, such as name, description, price, and quantity.

Optionally, the Merchant has the convenience of creating a new tag, brand, or product type directly from the product page if the required ones are not already available. This feature streamlines the user experience, eliminating the need to navigate to separate pages for creation.

Following this, the Merchant can proceed to tag the product and then selects its brand and product type from existing options. The tag, brand, and product type references allow for better organization and searchability of products.

Upon successful completion, the product is created in the database, and its details are added to the product list on the FE, which is then rendered to the Merchant.

Note: To maintain diagram clarity, we have created references to the processes of tag creation, brand creation, and product type creation, as they are important steps in the overall process of adding a product.

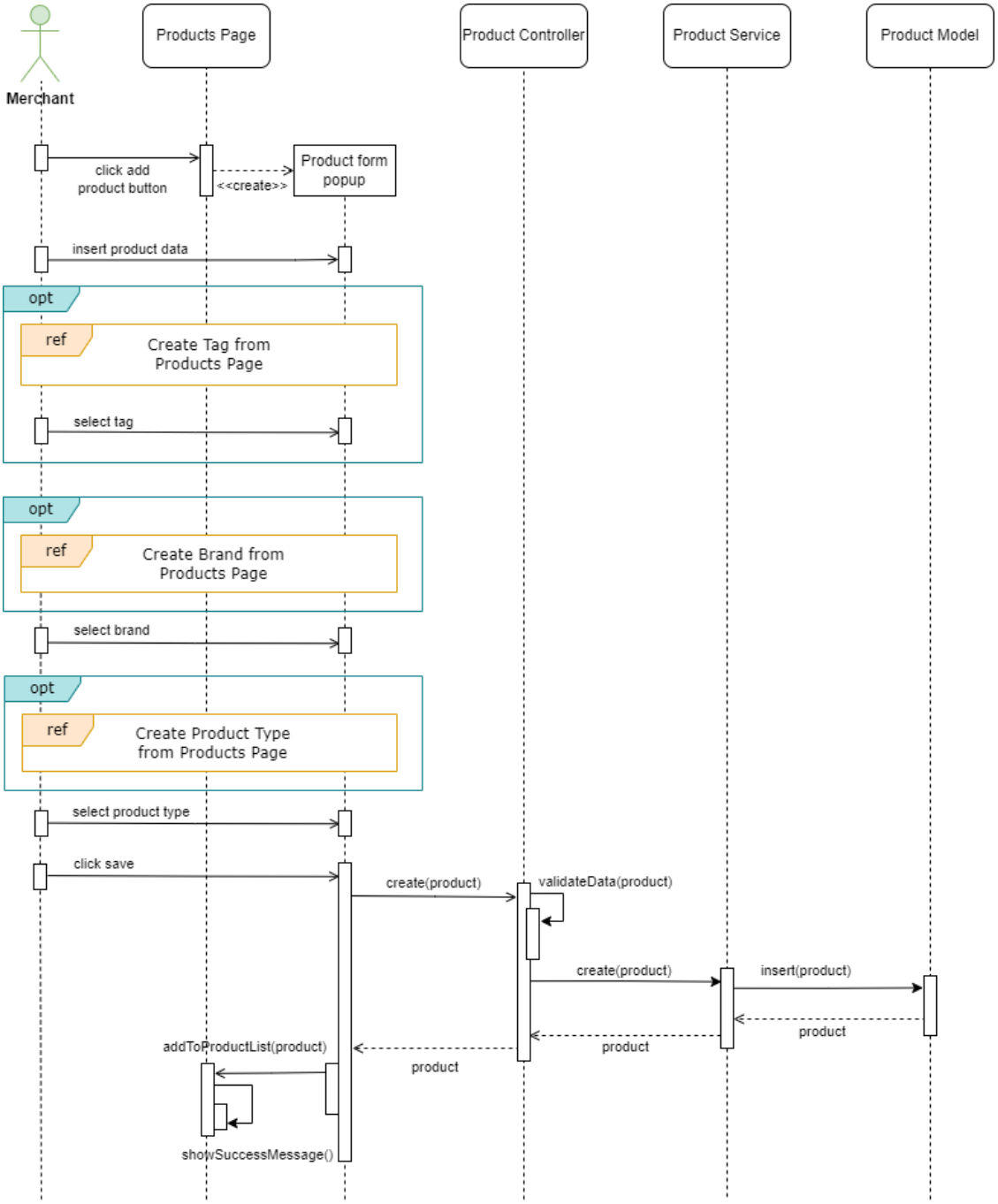

Figure 18: Create Product Sequence Diagram

Tag and Brand Creation from the Products Page (Merchant)

The tag and brand creation processes share a similar workflow. When the Merchant chooses to add a new Tag or Brand, a corresponding Form Popup appears, allowing the Merchant to input the required information. Upon clicking "save", the Controller validates the input, and the Service takes charge of creating the Tag/Brand. Subsequently, the newly created Tag/Brand is added to the database. Finally, the Tag/Brand is appended to their respective lists in the FE, making them accessible for the Merchant in both the Products Page and the Tags/Brands Page.

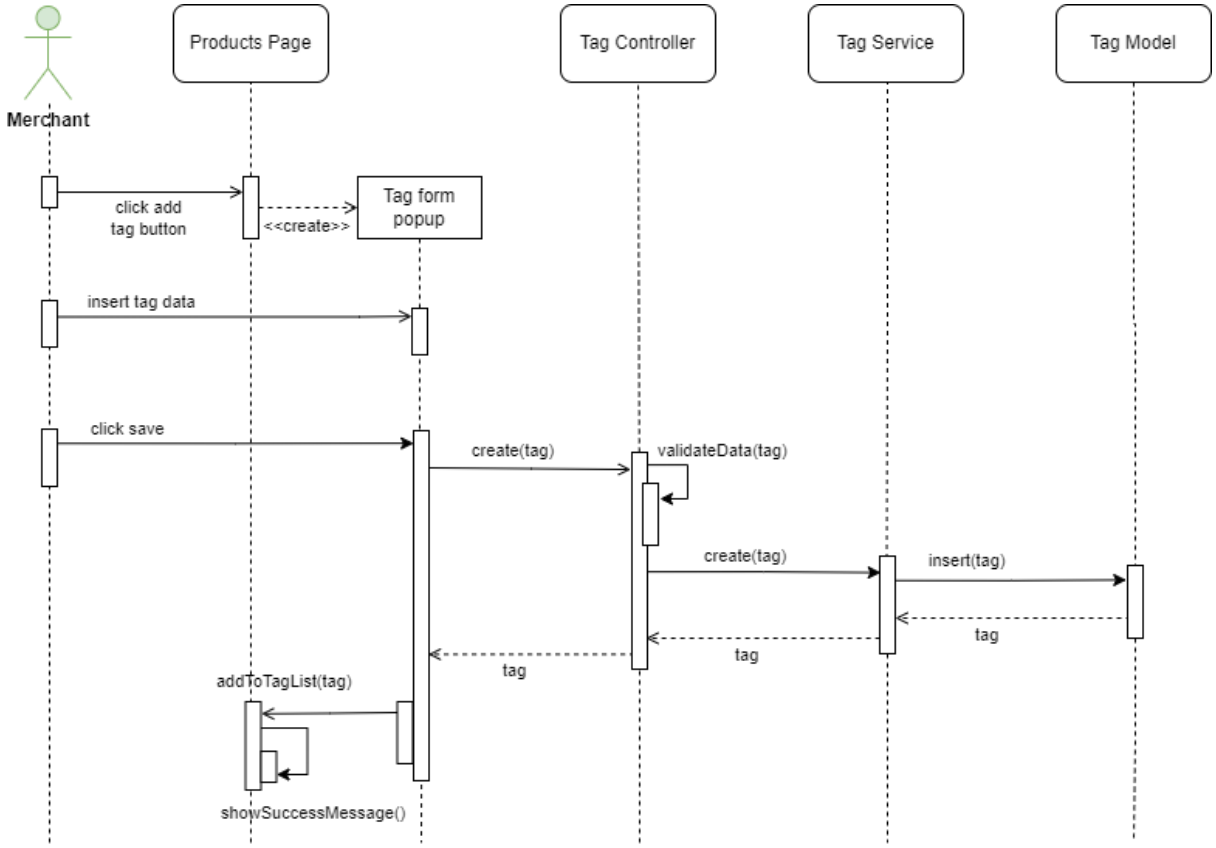

Figure 19: Create Tag from Products Page Sequence Diagram

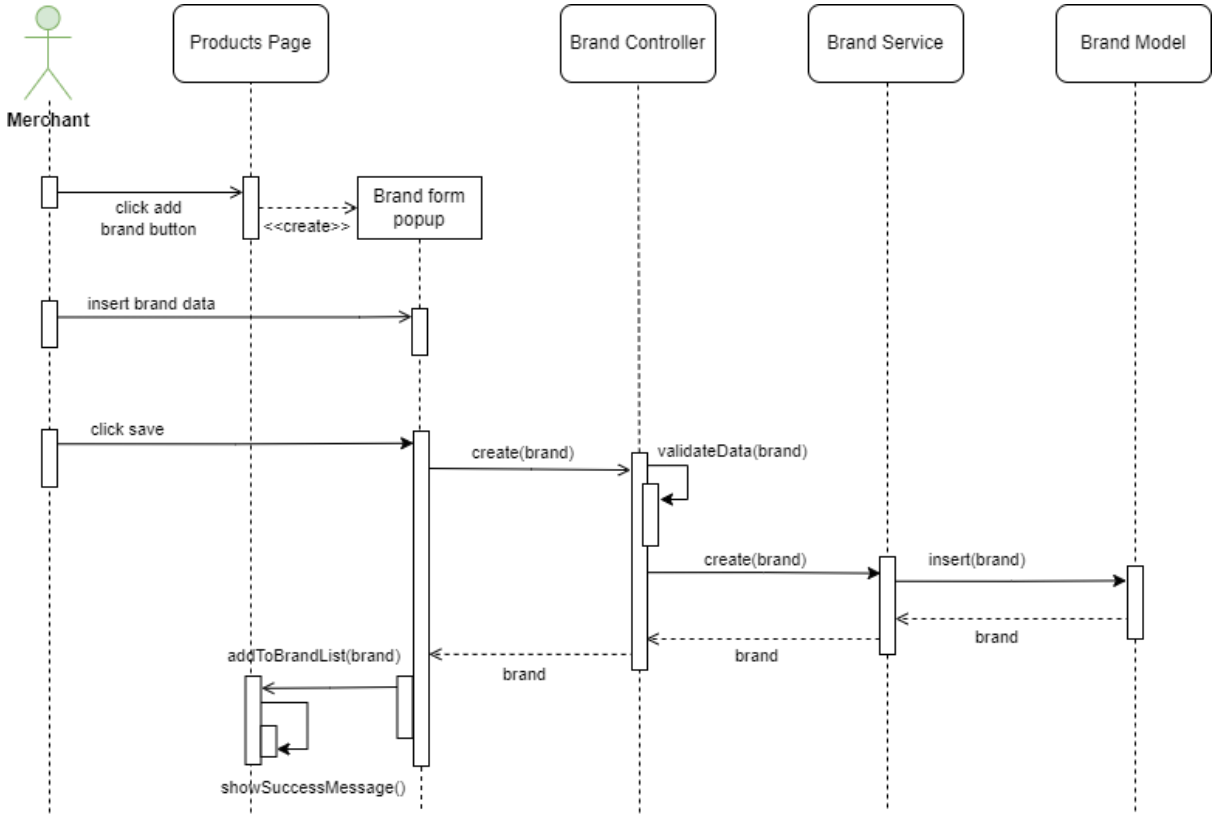

Figure 20: Create Brand from Products Page Sequence Diagram

Product Type Creation from the Products Page:

To create a Product Type, the Merchant begins by inserting the relevant information for the Product Type, such as its name, description, and attributes. After providing the necessary details, the Merchant proceeds to select the associated Category Group. If the required Category Group does not exist, they have the option to create a new one. Once the Category Group is selected or created, the Merchant proceeds to choose the relevant Category under which the Product Type will be categorized. Similarly, if the Category does not exist, they can create a new Category. Finally, the Merchant clicks 'save' to initiate the creation process.

The Product Type Controller validates the input, and the Product Type Service handles the actual creation of the Product Type. The new Product Type is then added to the database and becomes available within the platform, enabling the Merchant to assign it to relevant products.

Note: As the creation of Category Group and Category is included as part of the Product Type Creation process, and they are similar to the Tag and Brand Creation processes, we chose to represent this scenario using a black-box sequence diagram. This approach helps keep the diagram clear and focused on the main interactions without unnecessary complexity.

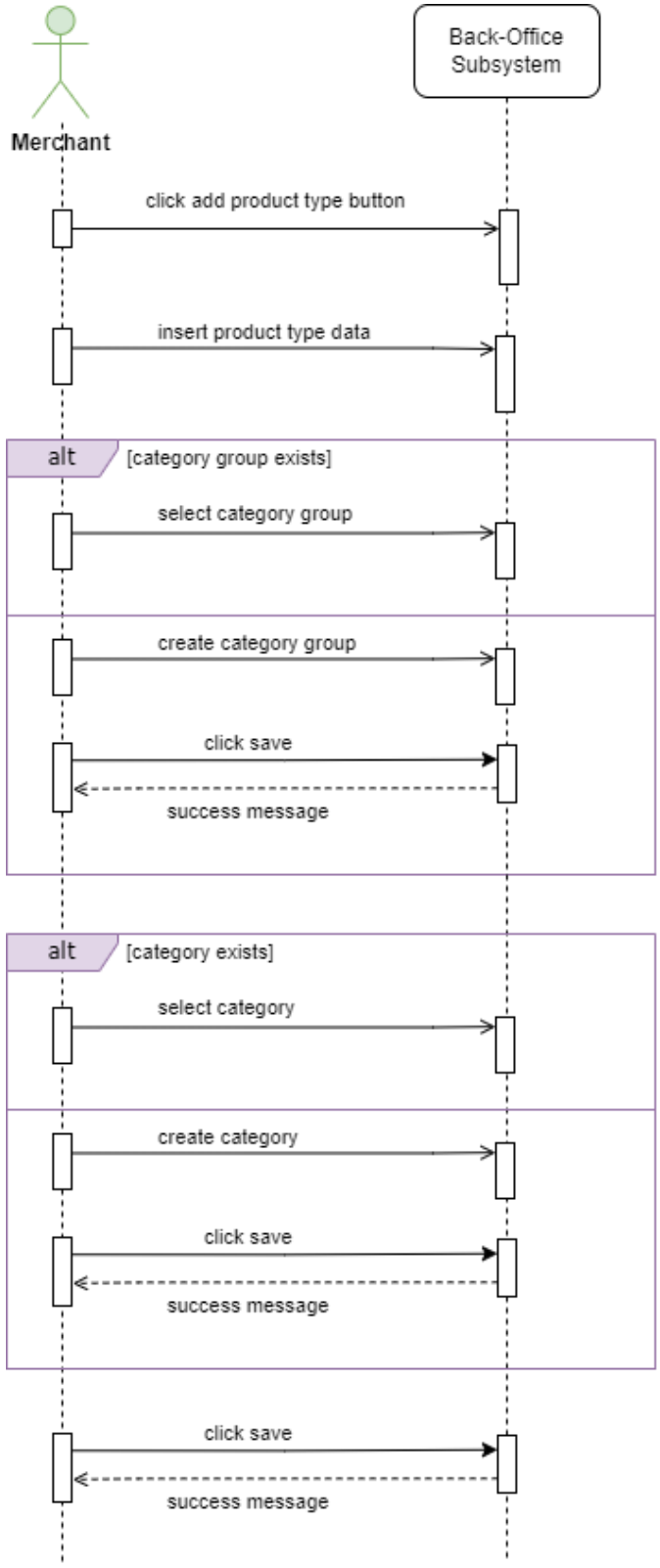

Figure 21: Create Product Type from Products Page Sequence Diagram

Product Purchase Process (Shopper)

The Product Purchase Process is a key use case for shoppers, encompassing the steps involved in selecting and purchasing products from the e-commerce platform. It begins with the shopper's search and navigation to find the desired product and concludes with the successful completion of the order. The following activity diagram demonstrates the flow of actions and decision points within the process.

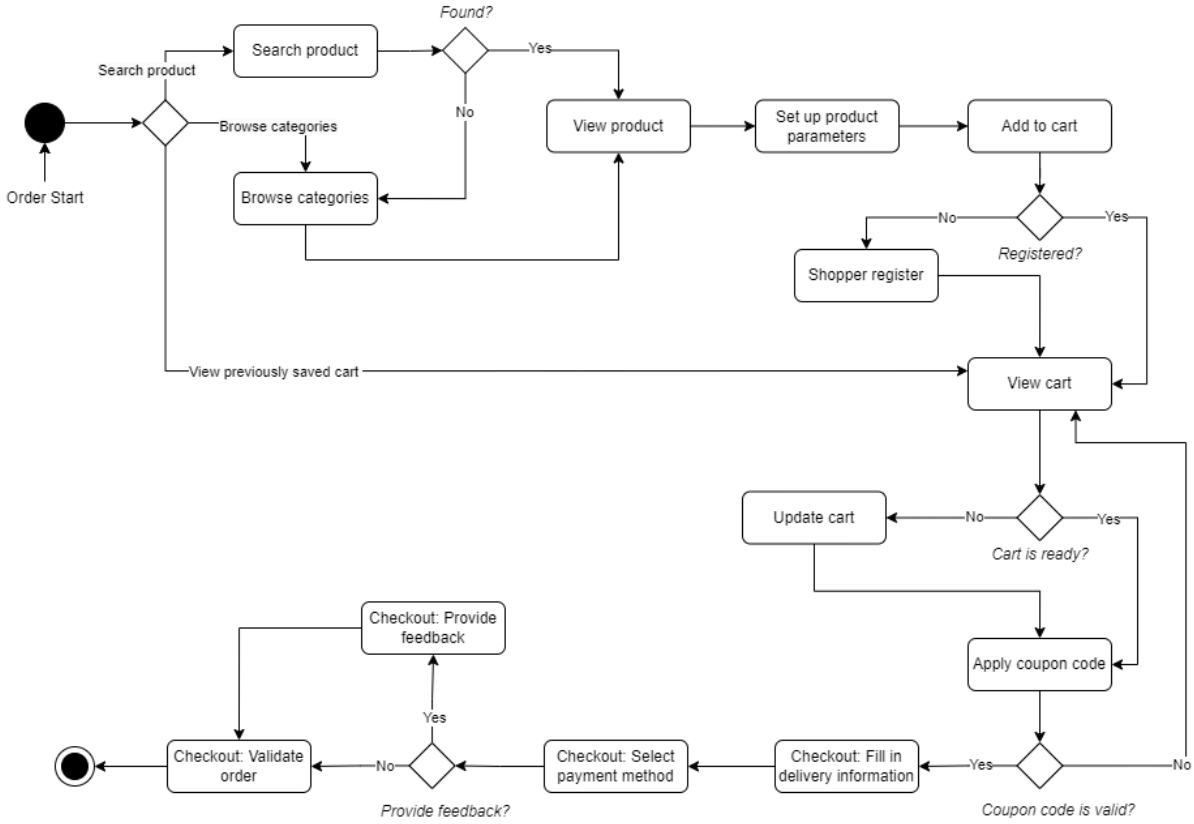

Figure 22: Shopper's Product Purchase Diagram

By combining the process view's sequence and activity diagrams, we aim to attain a holistic understanding of the system's runtime behavior, ensuring its optimal performance and seamless functionality. This comprehensive insight will guide us in making informed architectural decisions and refining the platform's design to meet the needs and expectations of both merchants and shoppers.

4.2.3. Design Validation

In the Design Validation phase, we ensured that the proposed architectural design effectively addresses the identified requirements and quality attributes and is in alignment with the needs and expectations of the stakeholders. This crucial phase plays a vital role in

confirming the validity and feasibility of our architectural decisions before proceeding with the actual implementation.

To validate the logical view, we thoroughly examined the class diagram representing the logical organization and structure of our multi-tenant e-commerce platform. By engaging stakeholders in review sessions, we aimed to gather valuable insights and feedback. This collaborative approach ensures that the logical view aligns with the system's functionality and accurately captures the relationships and interactions among various system entities.

Similarly, for the process view, we utilized sequence diagrams and activity diagrams to model the dynamic behavior and runtime elements of the system. These diagrams provided a clear depiction of how the system functions and how processes are carried out to achieve various functionalities. By involving stakeholders in review and feedback sessions, we could validate the accuracy and efficiency of these process representations.

During the Design Validation phase, we addressed any concerns, discrepancies, or potential improvements identified by stakeholders, fine-tuning our architectural design as needed. This iterative approach ensures that the architectural decisions are well-informed and align with the desired quality attributes, fostering confidence in the viability and effectiveness of the overall system design.

4.2.4. Solution Implementation

In the following section, we will delve into the comprehensive exploration of the last two views in the 4+1 architectural model: the development view and the physical view. These perspectives play a crucial role in shaping the implementation and deployment aspects of our multi-tenant e-commerce platform. By thoroughly examining these views, we can establish a well-structured and efficient development environment while ensuring optimal utilization of hardware resources and seamless integration into the physical infrastructure. The discussion of these views will provide valuable insights into the technical aspects of our solution, consolidating the foundation for a robust and high-performing e-commerce platform.

Development View

The development view of our multi-tenant e-commerce platform centers on the choice of an appropriate architectural style that guides the organization of the system's components and functionalities. After careful consideration, we have opted for the three-tier architecture, which offers a structured and modular approach to design, enhancing maintainability, scalability, and flexibility.

Three-Tier Architecture:

The three-tier architecture divides our system into three distinct layers, each responsible for specific functionalities. These layers are:

1. **Presentation Layer**: The topmost layer of the architecture is the presentation layer, responsible for managing the user interface and handling user interactions. It ensures a seamless and responsive user experience by presenting information to users and capturing their inputs.

2. **Business Logic Layer:** Situated in the middle, the business logic layer encapsulates the core functionality and business rules of our e-commerce platform. It processes user requests from the presentation layer, interacts with the data layer, and performs essential operations such as order processing, inventory management, and user authentication.

3. **Data Layer:** The bottom layer, the data layer, handles data storage and retrieval operations. It communicates with the business logic layer to provide access to the database, allowing for efficient data management and seamless integration with external data sources.

The adoption of the three-tier architecture ensures a well-structured development environment, allowing for parallel development of different layers and fostering collaboration among engineering team members. This organization simplifies code maintenance and updates, enhances code reusability, and promotes efficient debugging and testing practices.

To illustrate the development view and the organization of these layers, we employ a layer diagram. This diagram visually represents the relationships and dependencies between the three tiers, facilitating a clear understanding of the system's architecture. It also highlights the structured organization of the system into distinct layers, each responsible for specific functionalities and data processing. By employing a layered architecture, we achieve a clear separation of concerns, promoting modularity and scalability.

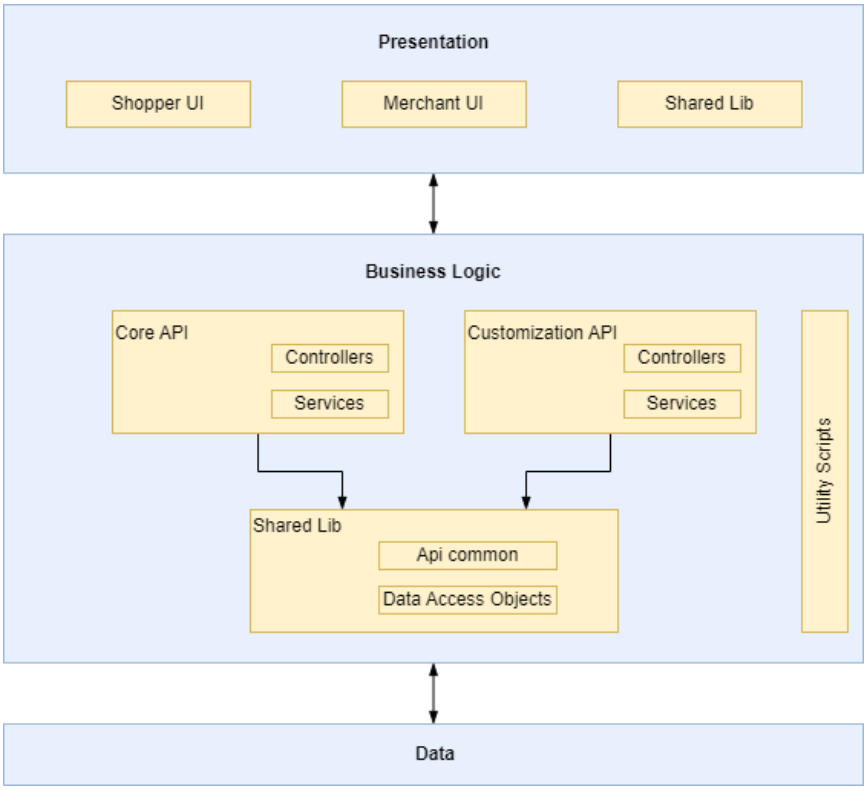

Figure 23: Three-Tier Layer Diagram

The layer diagram demonstrates how the presentation layer, encompassing the intuitive interfaces for both shoppers and merchants, interacts with the business logic layer. Within the business logic layer, two essential APIs, the Core API and Customization API, orchestrate the platform's functionality through their respective controllers and services. These APIs interact with shared libraries that hold common functionality and Data Access Objects (DAOs), ensuring reusability and maintainability. At the core of the system, the data layer holds the data repository, which the business logic layer accesses to retrieve and update information. This separation of concerns facilitates future enhancements or modifications to individual layers without disrupting the entire system, enabling us to adapt and evolve the platform to meet changing requirements and business needs.

Closer examination of each layer reveals:

❖ **Presentation Layer:**

- The Presentation Layer encompasses the user interfaces for both the store and admin-app sections of the application, providing distinct interfaces for merchants and platform administrators.
- Within the Presentation Layer, we have the following components:
    - **Shopper UI:** This interface represents the user interface provided for the shoppers, allowing them to browse products, place orders, and manage their accounts.

- **Merchant UI:** This interface is designed for merchants, providing them with tools and functionalities to manage their respective stores, including product listings, order processing, and store customization options.
- **Shared Lib:** The Shared Lib component contains shared code and functionalities that are utilized by both the Store UI and Admin-app UI, promoting code reusability and consistency across different UIs.

❖ **Business Logic Layer:**

- Within the Business Logic Layer, we adopted the SOA architectural style by creating two main sets of APIs, each catering to specific aspects of the platform following the single responsibility principle:
  - **Core API:**
    - **Controllers:** The Controllers within the Core API are responsible for handling incoming requests from the UIs, orchestrating the flow of data and interactions between different components.
    - **Services:** Services in the Core API encapsulate business logic and provide essential functionalities, such as order processing, user authentication, and product management.
  - **Customization API:**
    - **Controllers:** The Controllers in the Customization API manage requests related to store customization, enabling merchants to personalize their store's appearance and settings.
    - **Services:** Services within the Customization API implement the business logic required for handling store-specific customization requests and configurations.
  - **Shared Lib:**
    - **API common:** The API common module in the Shared Lib contains shared code and functionalities that are utilized by both the Core API and Customization API, promoting code reusability and consistency.
    - **Data Access Objects:** The Data Access Objects are responsible for interacting with the data layer, facilitating communication between the Business Logic Layer and Data Layer.

❖ **Data Layer:**

- The Data Layer is the bottommost layer of our multi-tenant e-commerce platform, responsible for managing the persistent storage and retrieval of data. It stores information related to products, users, orders, and other entities, ensuring data integrity and security.

By visualizing the relationships and interactions between these layers, the layer diagram underscores the structured organization of the multi-tenant e-commerce platform, facilitating a comprehensive understanding of its architecture and supporting the project's successful implementation.

Physical View

The physical view of our multi-tenant e-commerce platform emphasizes the deployment aspect, illustrating how the system components are distributed across hardware resources. This view provides insights into the physical infrastructure required to support the execution and operation of our e-commerce platform.

To visualize the physical view, we utilize a deployment diagram, which presents the mapping of software artifacts onto the hardware nodes in the deployment environment.

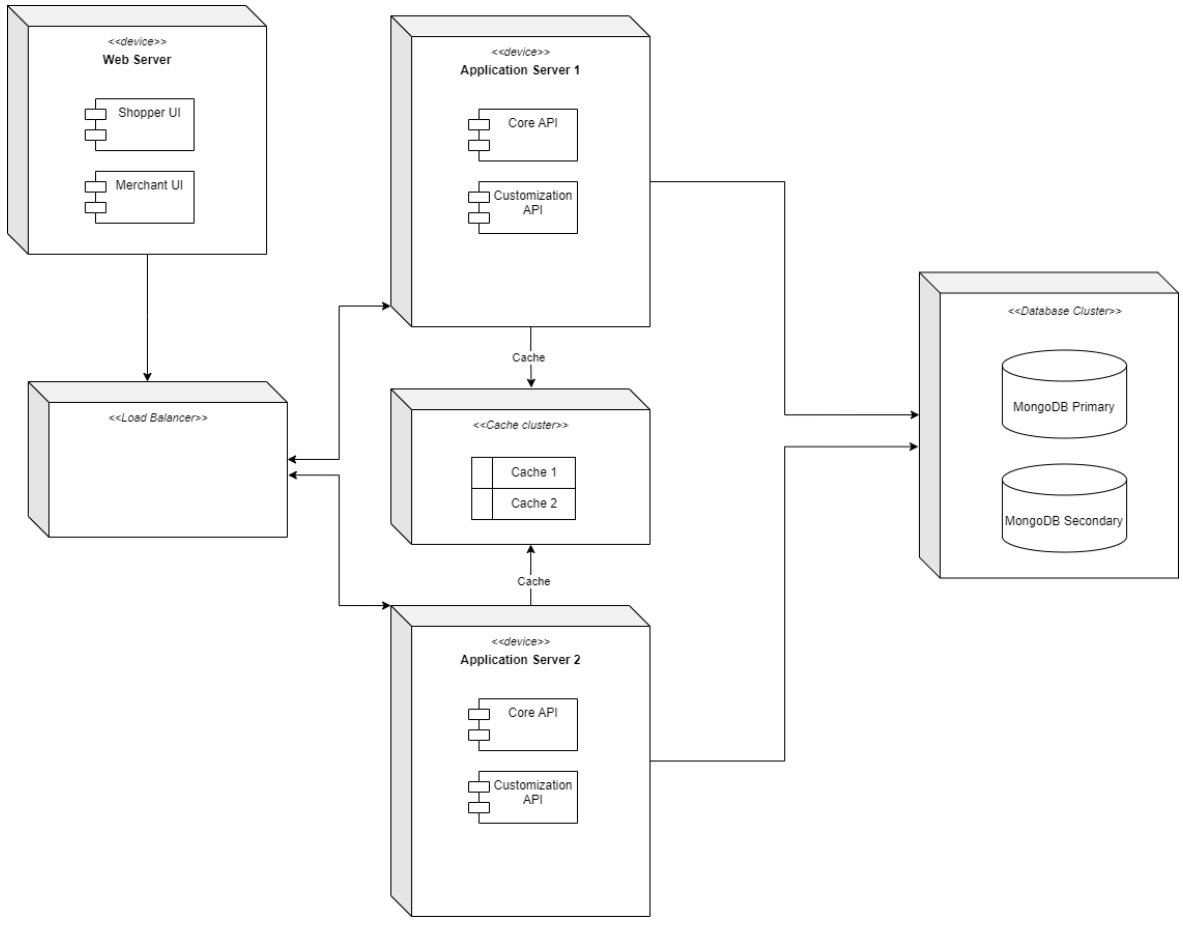

Figure 24: Deployment Diagram

In the deployment diagram, each node represents a hardware resource, such as a server, or a database. The software artifacts, including the application components and databases, are represented as deployment targets that reside within these nodes.

For our multi-tenant e-commerce platform, the deployment diagram showcases the distribution of components across different nodes. The presentation layer is deployed on web servers, ensuring efficient handling of user interface interactions and quick response times. The business logic layer, comprising the core functionality of the platform, is deployed on application servers, enabling seamless processing of user requests and interactions. Lastly, the data layer, responsible for data storage and retrieval, is deployed on dedicated database servers, ensuring efficient and reliable data management.

Load Balancing: To ensure optimal performance and scalability, our deployment strategy includes the use of active load balancing techniques. Load balancers are employed to distribute incoming traffic evenly across multiple web servers, reducing the risk of overload and ensuring a consistently responsive user experience.

Caching: To further enhance performance, we have implemented a caching strategy within our deployment environment. By caching frequently accessed and rarely updated data, such as store information and the home page content, we significantly reduce response times and alleviate database load. This results in a highly responsive user experience, enabling swift access to product details, seamless category browsing, and efficient platform navigation.

High Availability and Redundancy: Another crucial aspect of our physical view is the implementation of high availability and redundancy measures. Redundant hardware resources and failover mechanisms are utilized to ensure continuous operation even in the event of hardware failures or network issues, minimizing downtime and maximizing system reliability.

Scalability: The physical view also incorporates the potential for scalability to accommodate the growing demands of our platform. Our deployment environment is designed with the flexibility to add additional nodes and resources as the platform's user base expands, ensuring that our e-commerce platform can handle increased traffic and growth.

In conclusion, the physical view, represented through the deployment diagram, provides valuable insights into the underlying hardware infrastructure supporting our multi-tenant e-commerce platform. By carefully designing and optimizing the physical deployment, we aim to create a reliable, performant, and scalable platform that aligns with the quality attributes established in earlier iterations. The physical view complements the other architectural views,

solidifying the foundation of our e-commerce platform and ensuring its success in the dynamic and competitive digital landscape.

4.2.5. Solution Evaluation

Building upon the work completed during the Solution Implementation phase, the Solution Evaluation plays a pivotal role in validating the effectiveness and accuracy of our architectural design and implementation for the multi-tenant e-commerce platform. During multiple stakeholder meetings, we conducted thorough evaluations to ensure alignment with the project's true requirements.

Our evaluation process involved comprehensive assessments of the development and physical views, meticulously scrutinizing the layer and deployment diagrams. These visual representations serve as the backbone of our platform's development, illustrating the robustness of our architectural choices and their adherence to the identified needs and quality attributes.

By subjecting the solution to stakeholder examination and obtaining valuable feedback, we have honed the platform to closely align with the real-world requirements of our merchants and shoppers. This process of continuous evaluation and improvement ensures that the multi-tenant e-commerce platform is well-equipped to cater to the diverse and evolving needs of its users, establishing a strong foundation for the subsequent phases of development and deployment.

4.3. Iteration 3: Multi-Tenant E-Commerce Platform Implementation and Containerization

In this iteration, our focus shifts towards the development phase of the multi-tenant e-commerce platform, a critical step in transforming our architectural design into a robust and operational system. Building upon the foundation laid in the previous iterations, where we explored the use case view of the 4+1 architecture and delved into the four architectural views of the platform, our primary objective is to transform the architectural design into a fully functional and dynamic system.

During this phase, we will concentrate on actual implementation, translating the architectural decisions into concrete software components and code. Our development process will involve creating, integrating, and refining the various modules that constitute the multi-tenant

e-commerce platform. By leveraging insights gained from the architectural views, we will ensure a cohesive and well-organized system that aligns precisely with the project's goals and quality attributes.

Throughout the development process, we will prioritize adherence to best practices, industry standards, and coding conventions. This approach will ensure that the codebase is robust, maintainable, and scalable. Rigorous testing and validation will be conducted to guarantee that the platform meets the identified functional requirements and quality attributes.

The objective of this iteration is to deliver a high-performing, secure, user-friendly, and reliable e-commerce platform that meets the diverse needs of both merchants and shoppers.

4.3.1. Problem Investigation

In this final iteration, we tackle the challenge of effectively translating the carefully designed multi-tenant e-commerce platform into a tangible and fully functional reality. While the requirements specification and architectural decisions lay the foundation, the actual implementation is the pivotal step that brings the entire project to fruition.

The problem at hand is the need to bridge the gap between theoretical concepts and practical execution. Without successful implementation, the project remains incomplete, and the platform's envisioned potential remains unrealized. This poses a significant risk, as all the planning, design, and effort invested in the earlier phases may go to waste if the implementation falls short.

Our task is to navigate the complexities of coding, module development, and system integration to bring the platform to life. This involves converting the logical and development views into working components, ensuring seamless interactions, and building a cohesive system that operates as envisioned.

To overcome this challenge, we must carefully analyze and translate the design elements into functional code, addressing potential roadblocks and ensuring that the platform's functionalities meet the specified requirements. Rigorous testing and validation will be vital to resolve any issues, and we must be prepared to make adjustments and refinements as necessary to achieve a high-quality and reliable system.

Ultimately, successful implementation is not just about technical expertise; it also requires collaboration, coordination, and alignment within the engineering team. Clear

communication and efficient teamwork are critical to navigate the intricacies of coding, debugging, and ensuring the platform's overall integrity.

By confronting and resolving the challenges posed by implementation, we aim to deliver a fully operational, scalable, and user-friendly multi-tenant e-commerce platform. The successful resolution of this problem marks the realization of our project's vision and establishes the foundation for a competitive digital marketplace.

4.3.2. Solution Design

In the Solution Design phase of this iteration, our focus will be on developing a detailed low-level diagram that serves as a comprehensive translation of the code structure organization. By creating this detailed representation, we aim to enhance the understanding of the system's implementation and ensure its alignment with the architectural design and requirements.

We have created detailed package diagrams for both the BE and the FE components of our multi-tenant e-commerce platform. These diagrams serve as low-level illustrations, providing clarity and organization to the development view. By focusing on separate diagrams for BE and FE, we ensure a clear understanding of the system's architecture, facilitating efficient collaboration and seamless implementation. These diagrams establish a well-structured environment that promotes code reusability, maintainability, and scalability, crucial for the successful development of our platform.

Back-End Development View

The BE package diagram showcases the organization of modules within the BE architecture of our multi-tenant e-commerce platform. Below, we provide a detailed description of each package:

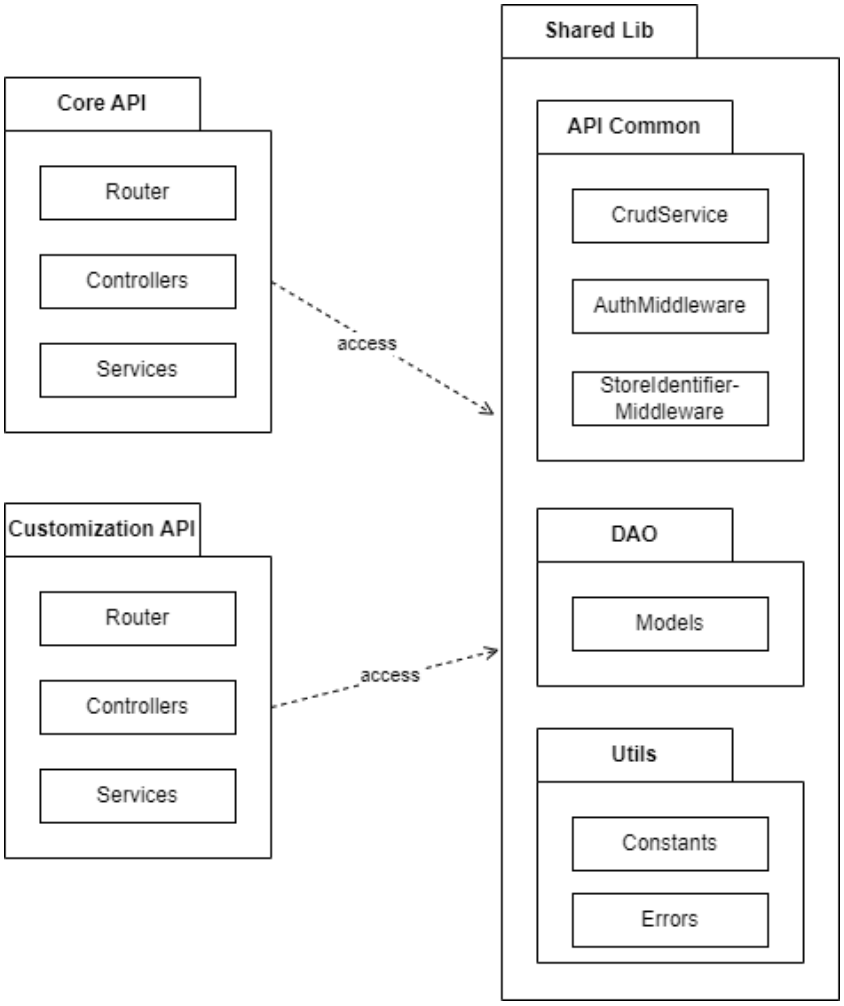

Figure 25: BE Package Diagram

Core API & Customization API:

- **Router:** This component is responsible for connecting the URLs to their corresponding controllers, handling incoming requests, and routing them to the appropriate controller for further processing.
- **Controllers:** The Controllers package encompasses modules that handle validations for the respective models. It ensures that the data passed to the BE adheres to the specified rules and constraints before processing.
- **Services:** The Services package hosts the essential logic for each model. It contains modules responsible for processing business rules, executing operations, and orchestrating interactions between various components within the BE.

Shared Lib:

- **API Common:** This package contains shared code and functionalities utilized by both the Core API and Customization API. It fosters code reusability and consistency across different APIs, reducing code duplication and simplifying maintenance.
    - CrudService: The API Common package includes a generic class called "CrudService", designed to implement basic CRUD (Create, Read, Update, Delete) operations for any model in the BE. By using this generic service, we avoid duplicating code for similar operations across various models, promoting a more efficient and maintainable codebase.
    - AuthMiddleware: This middleware component ensures the authentication of users accessing the APIs, enhancing security and privacy within the system.
    - StoreIdentifierMiddleware: The StoreIdentifierMiddleware plays a key role in identifying the store associated with incoming requests, allowing the BE to serve the appropriate data for each store within the multi-tenant environment.
- **DAO:**
    - Models: Each Model represents a specific entity within our system, such as "Store", "Product", "Order", and more. By defining the schema, data types, and validations for each entity, Mongoose Models ensure data integrity and consistency when storing and retrieving information. These models act as a bridge between our application's logic and the database, allowing efficient data management and retrieval.
- **Utils**
    - Constants: The Constants module within the Utils package provides a central location to define and store constant values and configurations used throughout the BE. This promotes a standardized approach to configuration and ensures that any changes can be made in one place, reducing the risk of errors and inconsistencies.
    - Errors: The Errors module within the Utils package handles error management and provides a set of custom error classes to handle various types of errors gracefully. By using custom error classes, we improve error reporting and make it easier to handle and communicate errors to users and developers alike.

The BE package diagram establishes a structured and well-organized development environment, ensuring a consistent implementation of functionalities and promoting code reuse. By adhering to this design, our BE architecture is poised for efficient maintenance, scalability, and easy incorporation of new features as our platform evolves to meet the needs of our growing user base.

Front-End Development View

The FE package diagram illustrates the organization of modules within the FE architecture of our multi-tenant e-commerce platform. It consists of the following packages:

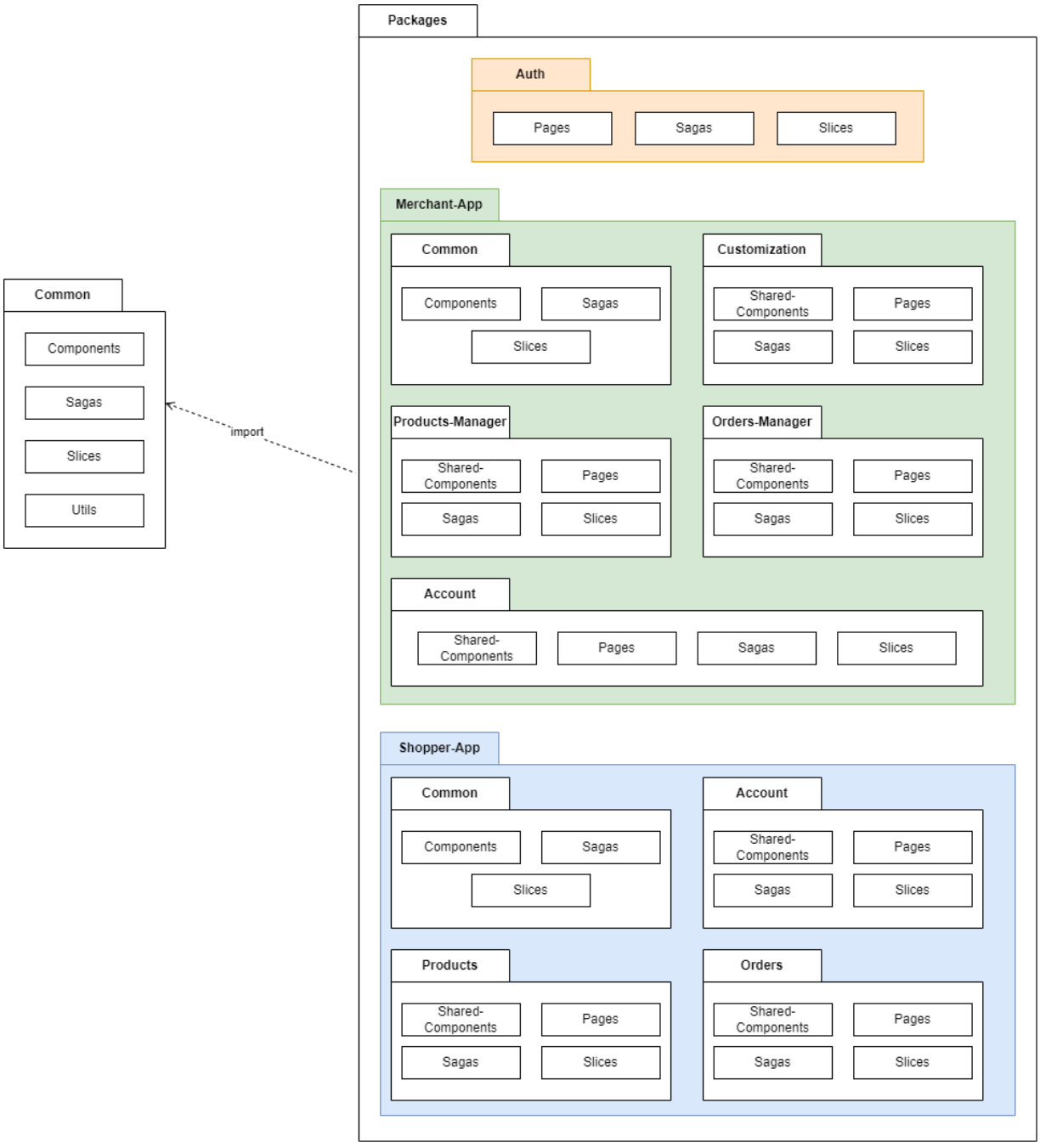

Figure 26: FE Package Diagram

1. **Common:**
    a. **Components:** Reusable components shared between the Merchant-App and Shopper-App for consistent user interface elements.

b. **Sagas:** Handlers and requests for asynchronous actions, ensuring smooth interactions with the BE.
c. **Slices:** Manage state changes using Redux Tookit for seamless data flow throughout the application.
d. **Utils:** Contains utility functions used throughout the FE for common tasks.

2. **Merchant-App:**
    a. **Customization:** Components and logic for store customization, enabling merchants to personalize their stores.
    b. **Products-Manager:** Functionalities for managing products, including listing, adding, editing, and deleting products. Additionally, it is responsible for managing product related entities, such as categories, brands, and tags management.
    c. **Orders-Manager:** Features for handling order management, such as editing, confirming, and rejecting orders.
    d. **Account:** Pages and logic for managing merchant account settings and profile information.
3. **Shopper-App:**
    a. **Account:** Pages and logic for managing shopper account settings, addresses, and user profile.
    b. **Products:** Pages and features for browsing, searching, and filtering products.
    c. **Orders:** Pages and functionalities for tracking and managing shopper orders.

The FE package diagram provides a clear structure, ensuring code reusability and maintainability across the different sections of our e-commerce platform.

These low-level diagrams serve as valuable tools for developers, facilitating efficient coding and implementation. They outline the structure of the code providing a clear blueprint for the development team to follow during the development process. Additionally, these diagrams act as a reference point for future code maintenance and modifications, ensuring that any changes made to the system remain consistent with the established design.

4.3.3. Design Validation

In the Design Validation phase, our primary focus was to rigorously assess and validate the detailed low-level diagrams, ensuring that they accurately translate the project's design into a concrete implementation. This critical step bridges the gap between the architectural design and the actual development of the multi-tenant e-commerce platform.

To achieve this, we engaged in close collaboration with the engineering team. Regular review and feedback sessions were conducted, providing a platform to present the detailed low-level diagrams and gather valuable insights from stakeholders. By incorporating their feedback and addressing any concerns or questions, we were able to refine and improve the design to better align with the expectations and needs of all involved parties.

During the validation process, special attention was given to verifying that the diagrams effectively capture the identified functional requirements and quality attributes to ensure a well-informed and holistic validation process.

Additionally, this phase serves as an opportunity to identify and rectify any potential issues or shortcomings before proceeding with the actual implementation. Therefore, we were able to iron out any issues, and consequently to fine-tune the design and build a solid foundation for the development phase.

By conducting a comprehensive Design Validation, we aim to establish a high level of confidence in the viability, efficiency, and effectiveness of our implementation approach.

4.3.4. Solution Implementation

With a comprehensive understanding of the research objectives, design principles, and architectural considerations established, the following section delves into the practical realization of the multi-tenant e-commerce platform. This section provides a visual representation of the different capabilities that cater to both merchants and shoppers within the e-commerce ecosystem. We organized it into distinct parts:

- **Merchant Application and Shopper Application:** Each part encapsulates the important aspects of the platform's functionalities, allowing readers to navigate through the interfaces that enable merchants in managing their online stores and shoppers in seamless product exploration and purchase.
- **Containerization:** This is a critical aspect of our implementation, leveraging Docker to ensure efficient deployment and scaling of various platform components.
- **Security:** Robust security practices have been integrated to safeguard user data. We employ encryption, authentication, access controls, and other measures to ensure the confidentiality and integrity of sensitive information.

Merchant Application

The Merchant Application of our multi-tenant e-commerce platform presents a range of essential features, each contributing to the comprehensive toolkit offered to merchants. This section provides a visual tour of these functionalities, thoughtfully organized by module. To maintain clarity and conciseness, we spotlight only the list and creation pages for each module as the emphasis is on the interface interaction and display. The editing process closely mirrors creation, and the deletion functionality, while vital, offers limited visual insights.

Products Manager

This module is dedicated to managing inventory and related entities. This encompasses the management of products as well as categories, brands, and tags.

❖ Products List

This interface enables the management of products through functionalities such as listing, editing, creation of new products, and deletion of single or multiple products. Users are also provided with sorting and filtering options for the displayed products.

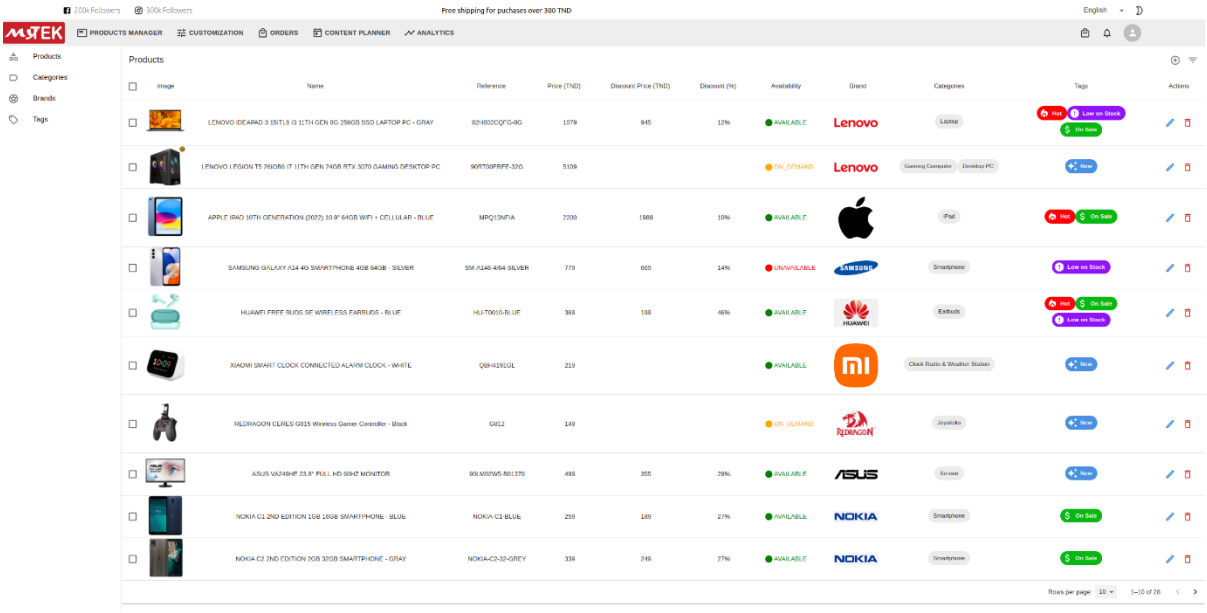

Figure 27: Products List Interface

❖ Product Creation

The product creation interface facilitates the process of adding new products to the system by specifying their details. Initially, the product's name and reference are provided. Optionally, a description can be added to enhance the product's information. Additionally, merchants have the ability to assign availability status to the product, indicating whether the

product is currently in stock or not. The interface also accommodates the application of a discount, where the discounted price and percentage can be automatically calculated and displayed alongside the original price once one of them is provided. Brand selection, tag assignment, and product type designation are among the subsequent details to be configured. If any of these elements are not present in the system, they can be added directly from this interface. Upon selecting a specific product type, the system auto-populates the relevant characteristics associated with that product type. These characteristics contribute to streamlined product filtering. Furthermore, multiple images of the product can be uploaded to provide comprehensive visual information to potential shoppers.

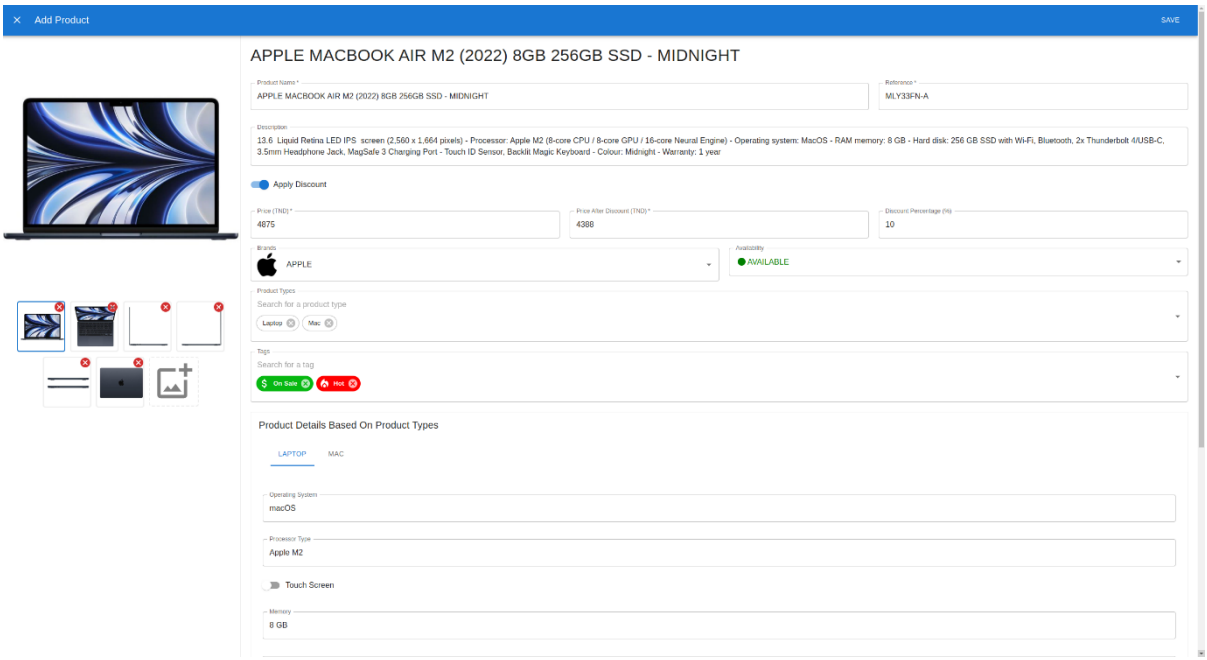

Figure 28: Product Creation Form

Adding New Entities from the Product Creation Interface:

As previously mentioned, the product creation interface not only facilitates the addition of new products but also provides the convenience of adding new brands, tags, and product types if they are not yet present in the system. In the case of adding a new product type, selecting the relevant category group and category becomes necessary. The system allows us to choose these categories and groups from the displayed lists or add new ones, as illustrated in the figure below. This comprehensive feature enables the addition of new brands, tags, category groups, categories, and product types all from within the product creation interface. This consolidated approach minimizes the need for navigating through various pages and reduces interruptions in the workflow.

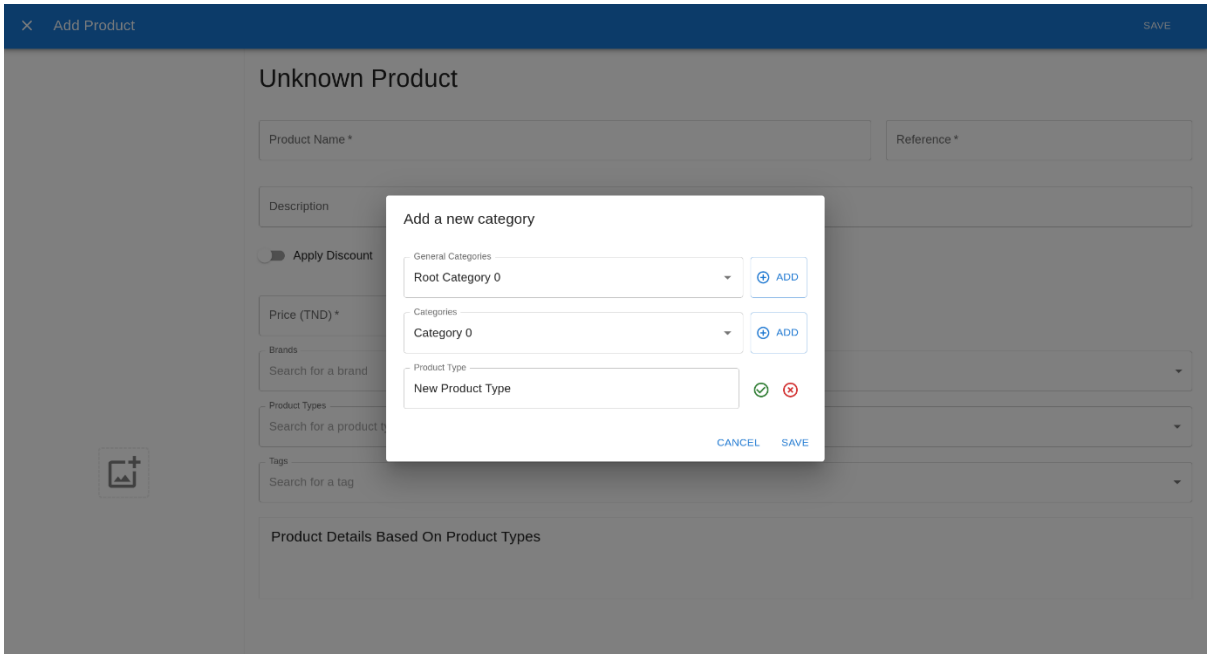

Figure 29. Adding New Categories from the Product Creation Form

❖ Categories List

Within the categories page (figure 29), a comprehensive interface grants access to the listings of category groups, categories, product types, and the details of these product types. Upon landing on the page, the initial view presents the list of category groups. Subsequently, upon selecting a category group, the associated categories get displayed. Further selection of a specific category reveals its corresponding product types. Finally, upon choosing a product type, its attributes are presented within the Product Type Details table.

The interface provides the flexibility to conduct searches within the Category Groups list, Categories list, and Product Types list. Additionally, new entities can be introduced, and existing ones can edited or deleted across these categories.

The Product Type Details table encompasses the columns of Name, Note, Type, Filter Option, and Actions. The 'Filter Option' signifies whether a particular attribute will be accessible as a filtering option on the product search page. The specification of 'Type' (string, number, boolean) dictates the attribute's potential values and presentation in the filtering form. Available actions include editing and deleting attributes. The deletion functionality allows for the selection of multiple product attributes to be deleted simultaneously with a single click. Furthermore, the addition of new attributes to the product type details is supported (as seen in figure 30). Also, the product type details can be filtered by name, type, and filter option.

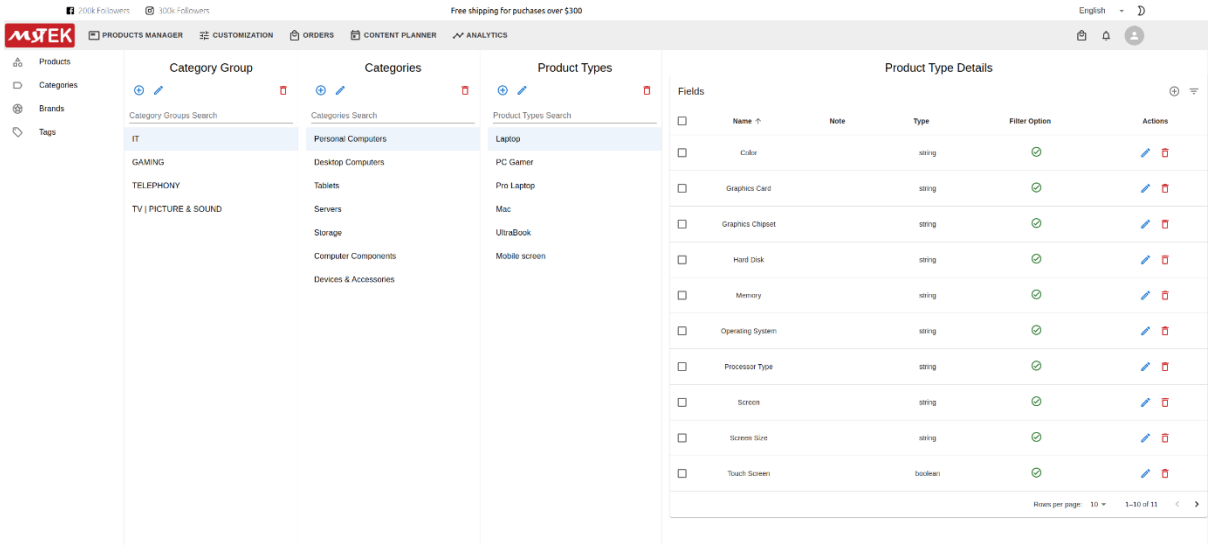

Figure 30: Categories List Interface

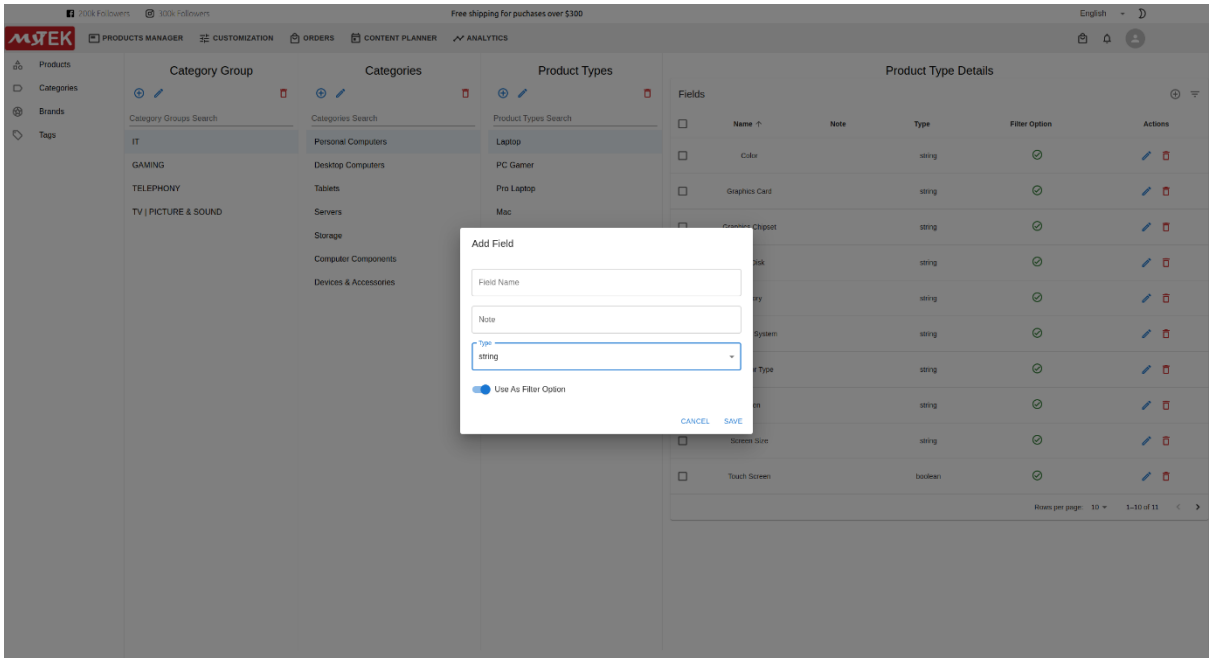

Figure 31: Specifying the Product Type Attributes

❖ Product Type Creation

The creation process involves the following steps: initiating a category group, followed by its corresponding categories, and finally the creation of product types within those categories. However, for the sake of brevity, only the product type creation interface will be illustrated, omitting the detailed depiction of category group and category creation. The product type creation process is straightforward: by providing the product type label and a representative picture, followed by clicking 'Save', the product type can be successfully generated. This process remains consistent for category group and category creation as well.

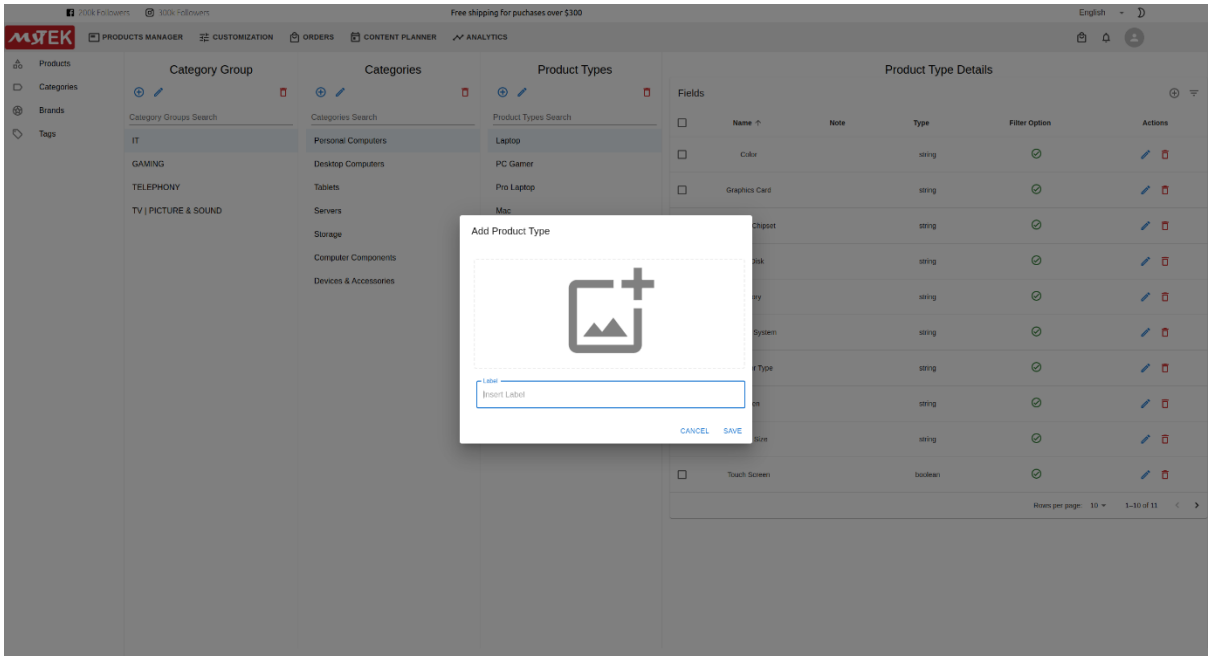

Figure 32: Product Type Creation Form

❖ Tags List

The Tags page displays a list of created tags, organized by label, description, preview, and actions. New tags can be added, and the displayed list of tags can be filtered by label. Through this interface, merchants will also be able to edit or delete the existing tags.

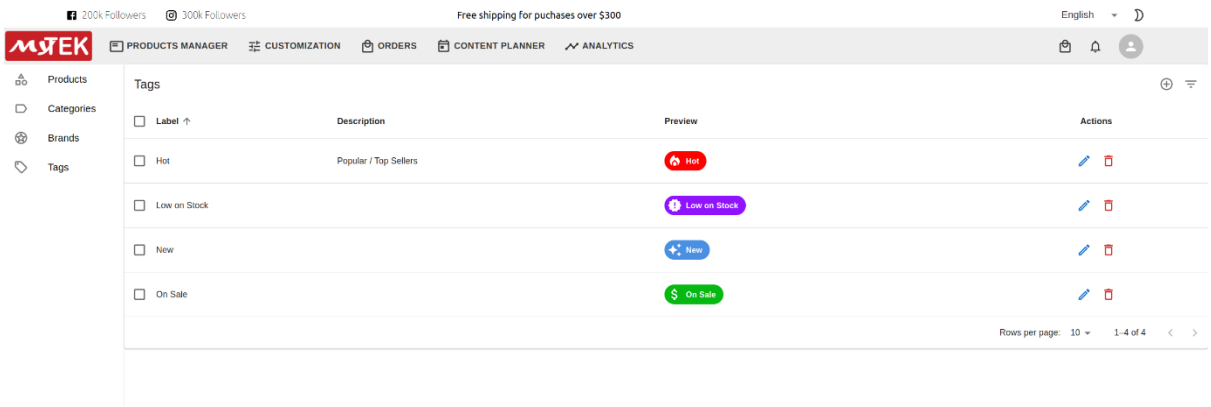

Figure 33: Tags List Interface

❖ Tag Creation

The process of creating a tag involves inserting the tag's label, selecting an associated icon, and choosing a color. These specifications contribute to the generation of a preview. Additionally, merchants have the option to provide a description for the tag.

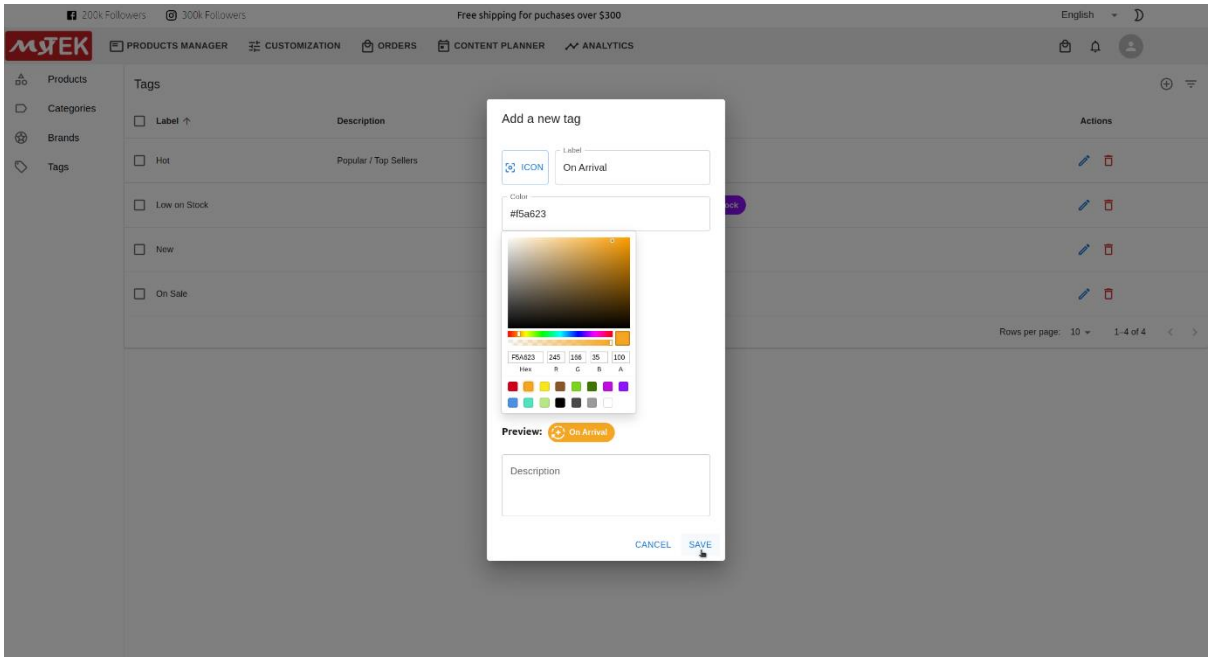

Figure 34: Tag Creation Form

❖ Brands List

The brands page displays a list of brands that have been created within the system, presenting their logos, names, and any accompanying notes added by the merchant during the brand creation process (figure 34). It provides the capability to add new brands and allows filtering of the displayed brand list by name. Additionally, the page facilitates the editing or deletion of existing brands. Multiple selection and deletion is also possible across all the application's lists, designed to optimize efficiency as exemplified in figure 35.

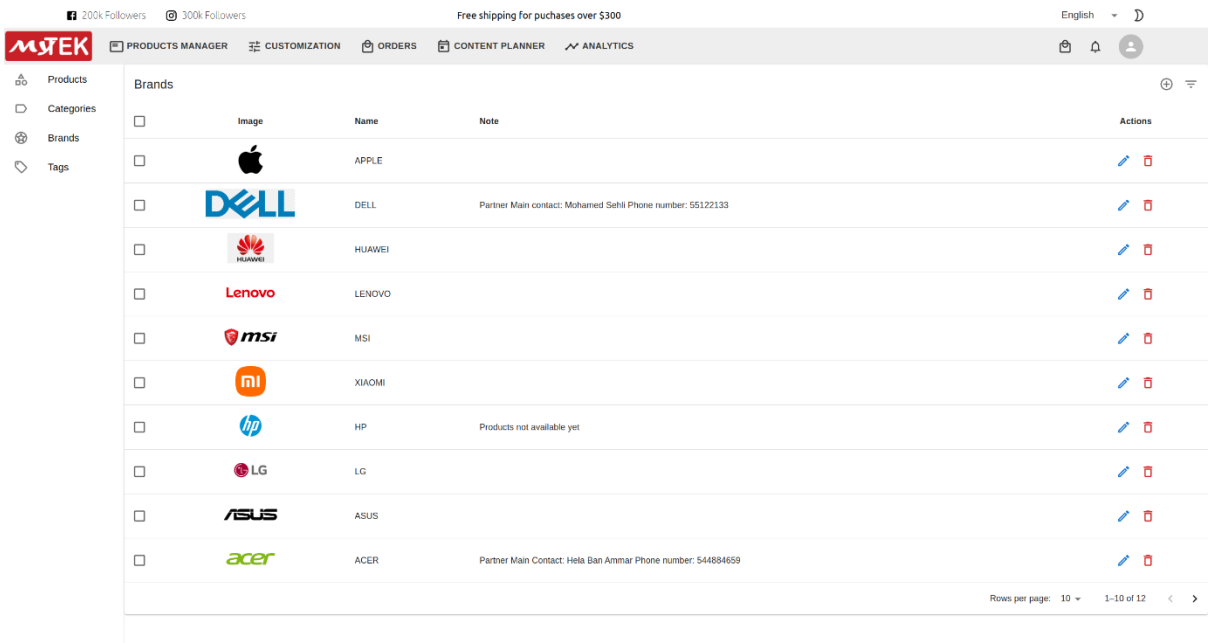

Figure 35: Brands List Interface

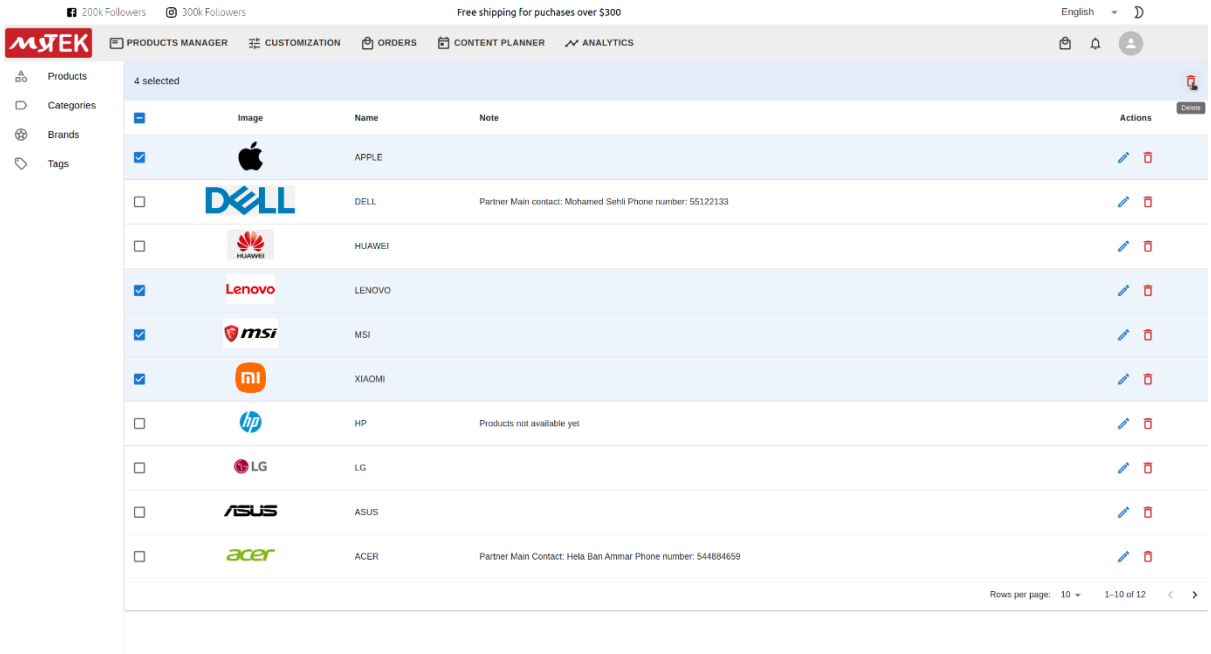

Figure 36: Bulk Deletion

❖ Brand Creation

Clicking the 'Add' button on the brands page triggers the appearance of the 'Add Brand' form. This form prompts for the specification of the brand's name, logo upload, and optionally, the addition of a note (figure 36). Upon completing these inputs and selecting 'Save', the brand is successfully generated and becomes visible within the brands list.

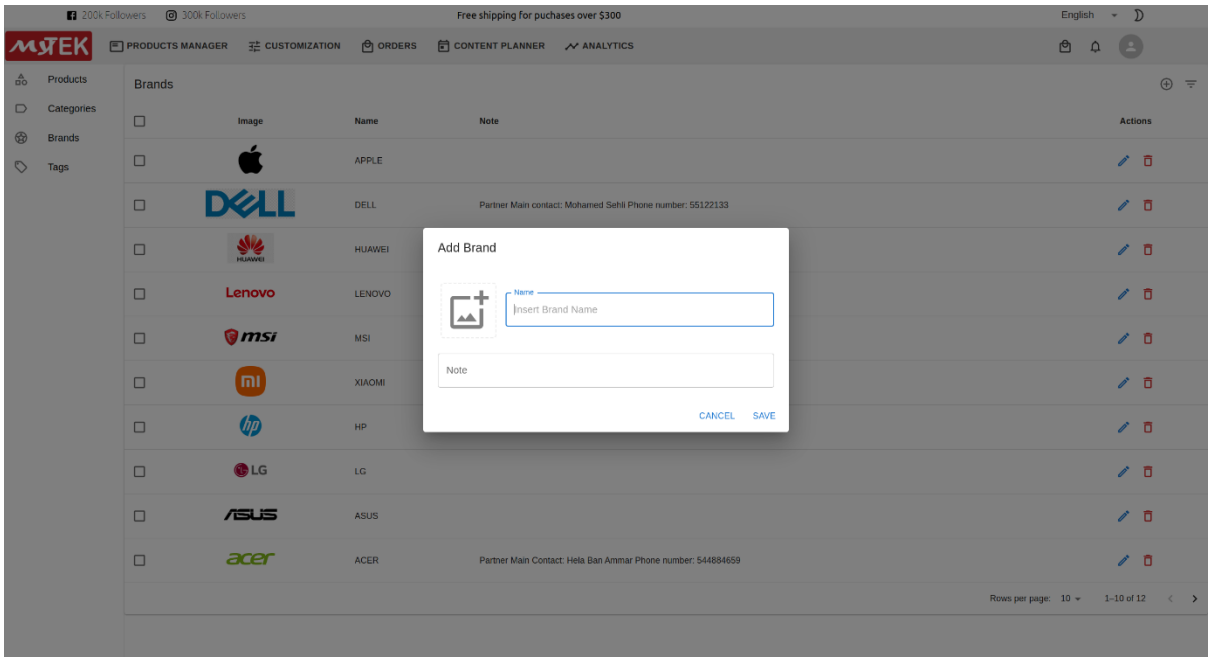

Figure 37: Brand Creation Form

Orders Manager

The Orders Manager module is designed to facilitate access to all the orders created by shoppers and their subsequent processing.

❖ Orders List

The interface below showcases a comprehensive list of orders, each accompanied by relevant information such as shopper details (full name, phone number, email), estimated delivery date, addresses, total order price, and its current status. Merchants are equipped with quick actions like order confirmation, cancellation, and editing, all accessible with a single click. Moreover, merchants have the ability to sort the displayed orders based on various attributes and apply filters by shopper or status for enhanced convenience.

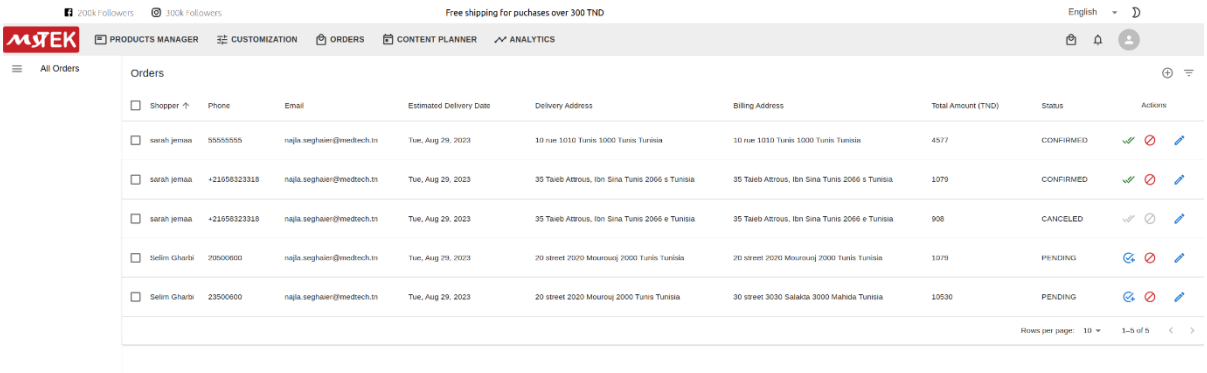

Figure 38: Orders List Interface

❖ Order Details

The Order Details interface offers a comprehensive view of an individual order, providing essential information and enabling effective management. Key components of this interface include:

- **Order Reference**: The unique identifier for the order, facilitating easy reference and tracking.
- **Shopper Details and Customer Note**: This section displays the shopper's relevant information, including full name, contact details, and any customer notes they may have provided during the order creation process.
- **Order History**: A chronological record of the order's journey, documenting status updates and key events from creation to fulfillment.

- **Order Information**: Vital details about the order itself, encompassing the total order amount, delivery and billing addresses, selected delivery date, and current status. Merchants have the ability to modify the status from 'Pending' to 'Confirmed' or 'Cancelled' based on the order's progress.

- **Product List**: A comprehensive list of products within the order, each accompanied by relevant information such as product name, reference, price, quantity, and current availability. Merchants can modify the product quantity, view detailed product information, or remove products from the order.

- **Add Products**: Merchants can easily expand the order by adding new products, enhancing flexibility and customization.

- **Sorting and Filtering**: The interface offers functionality to sort and filter the listed products based on attributes such as price, quantity, and availability, enabling merchants to efficiently manage the order's contents.

The Order Details interface serves as a comprehensive control center, empowering merchants to manage each order efficiently and ensuring a streamlined order processing experience.

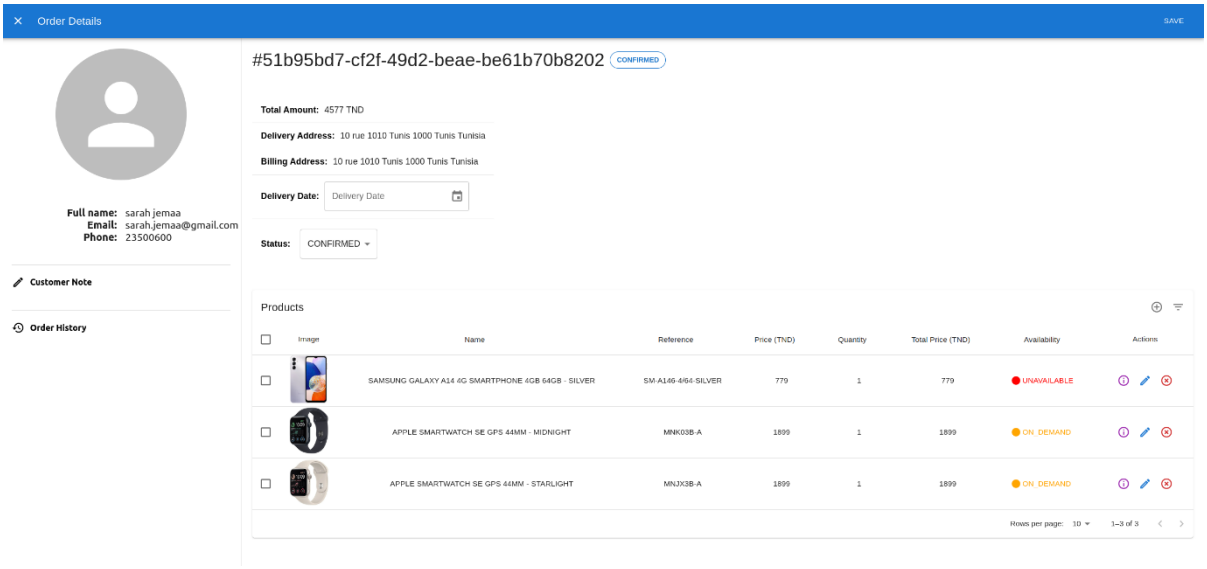

Figure 39: Order Details Interface

Customization Manager

This module presents the customization features offered to merchants, enabling them to create personalized home page templates and manage their store profile pages. These tools empower merchants to infuse their brand identity into their digital storefronts, enhancing customer interactions and engagement.

❖ Home Page Templates List

Figure 40 illustrates the list of home page templates available to merchants, providing options for customizing their storefront's appearance. Templates can either be the platform's default choices or ones previously created by the merchant. Each template is accompanied by essential details including its name, description, creation date, latest update date, and whether it's currently selected as the home page template. Practical actions, such as previewing, editing, or deleting a template, can be taken. The preview action provides a glimpse into how the template would look, helping merchants in their selection process. Additionally, merchants can enrich their choices by adding new templates using the plus button. The flexibility extends further with the ability to filter templates based on name, creation date, or update date.

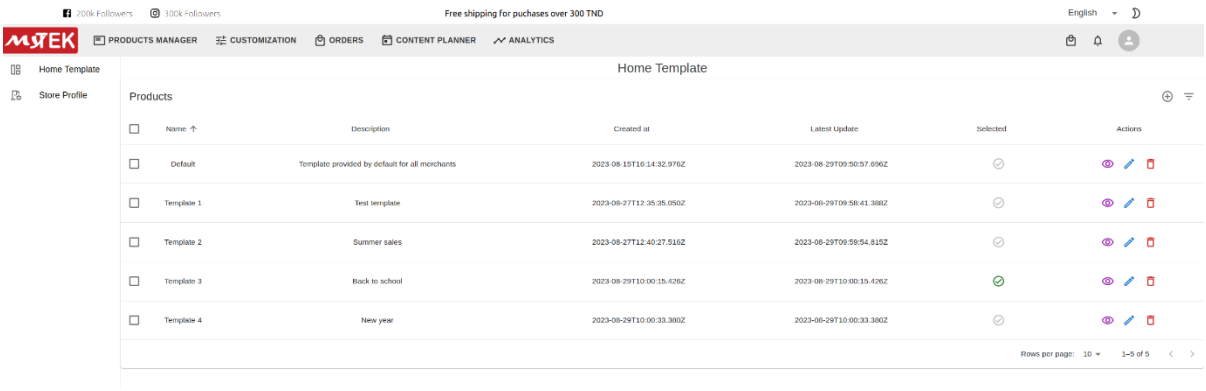

Figure 40: Templates List Interface

❖ Template Creation

The Home Page Template Creation Form presents an intricate yet user-friendly interface for crafting dynamic and engaging home page templates tailored to merchant preferences. The form is thoughtfully divided into distinct segments, each contributing to the composition of a comprehensive and visually appealing layout, as illustrated in figure 41.

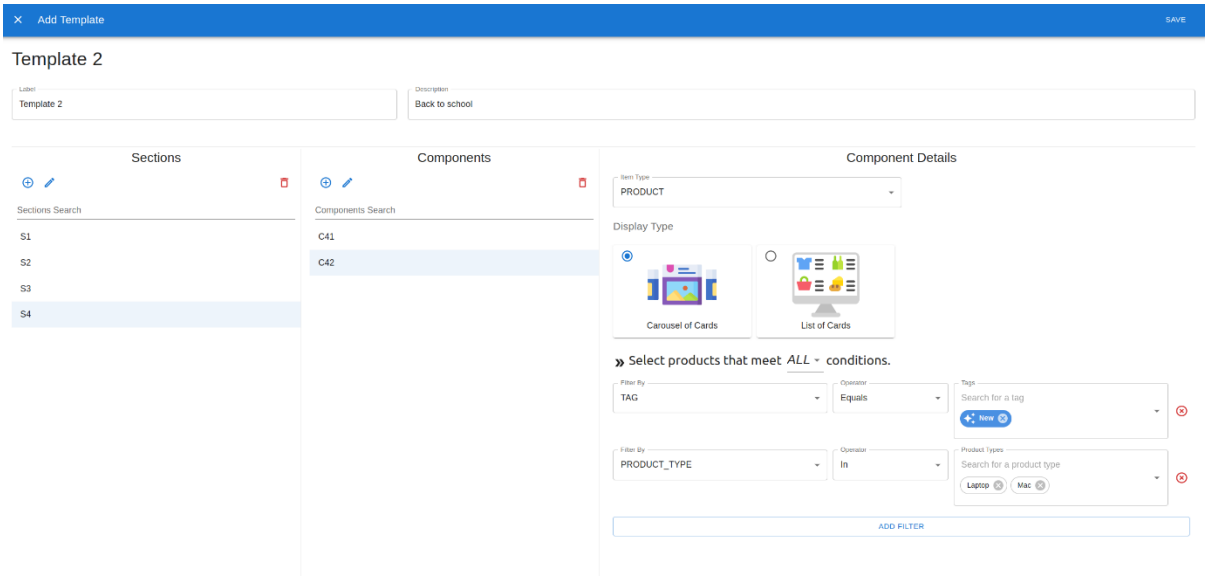

Figure 41: Template Creation Form

At the start of the form, users are prompted to provide a Label and Description for the template. This introductory information establishes the template's purpose and characteristics. The creation process consists of three main parts, effectively guiding users through the design process:

1. Sections:

The initial section centers on the configuration of individual sections. Here, users can add new sections, as well as edit or delete existing ones. This flexible approach allows for continuous refinement and adjustment. An intuitive filtering mechanism further enhances ease of use. Each section requires a Label, serving as an identifier within the template. Optional Title and Description fields can be specified. The Title corresponds to the text that will be visible on the actual home page, while the Description offers an opportunity for additional context.

2. Components within Sections:

Within each section, users have the ability to incorporate a variety of components. Just like sections, components can be created, edited, or deleted. These building blocks contribute to the overall structure and content of the home page. To define a component, users need to specify a Label, similar to section labels, and optionally a Title and a Description.

3. Component Details:

The third and pivotal part focuses on the specific details that govern how components are presented on the home page. Each component requires careful consideration of its "Item Type", which acts as a foundation for content composition. The available item types include

“PRODUCT”, “CATEGORY GROUP”, “CATEGORY”, “PRODUCT TYPE”, “BRAND”, and “IMAGE”. For instance, selecting “PRODUCT” as the item type implies that the component will feature products, while choosing “BRAND” implies a focus on brands.

For components involving images, users can choose from two Display Types: “Carousel of Images” or “List of Images”. Meanwhile, other item types, such as “CATEGORY” or “PRODUCT TYPE”, are supported by “Carousel of Cards” or “List of Cards” display options. Each display type is visually represented by an icon, enhancing clarity for merchants.

For “IMAGE” item type, after selecting the display type, users only need to upload the image(s), and the view will subsequently incorporate these images. In the case of “PRODUCT TYPE”, the items presented are determined by the selected operator, such as “In”, or “Not In”. This selection is followed by the designation of applicable product types. A similar approach applies to “CATEGORY GROUP”, “CATEGORY”, and “BRAND” item types.

As for "PRODUCT" item type, conditional specifications are of paramount importance. Users can apply diverse filters to determine the products that populate a component. Filter attributes can be “category group”, “category”, “product type”, “price”, “brand”, or “tag”. For each filter, users select an operator, such as “Equals”, “Not Equals”, “In”, “Not In”, “Greater Than”, “Greater or Equal”, etc., that aligns with the specific filtering criteria.

Once all these elements are carefully configured, users can proceed to save the template, resulting in the successful creation of a personalized home page template. The Home Page Template Creation Form truly empowers merchants to craft engaging and meaningful content, resonating with their target audience and elevating the overall shopping experience.

Some examples of these elements’ output are reflected in the template preview of figure 42. In this example, a part of a template layout is shown, providing insights into the potential composition and arrangement of components and sections.

Figure 42: Home Page Template Preview

In the preview, we can observe a first section composed of two distinct components. The first component is a carousel of images, providing a visually engaging display. The second component is a list of images, featuring two pictures that complement the section's visual appeal.

Continuing to the template preview, we encounter a second section dedicated to product cards, also comprising two distinct components. The first component showcases products that have been filtered by the tag “Hot”, aligning with the component's descriptive title.

Meanwhile, the second component products are filtered by the “On Sale” tag, featuring products currently available at discounted prices.

❖ **Store Profile**

The Store Profile Page within the customization feature provides merchants with the ability to personalize their storefronts, ensuring that each merchant can showcase their unique brand identity and critical information. This customizable store page is a pivotal component of our multi-tenant e-commerce platform, granting merchants the autonomy to create a distinct digital presence within the shared platform.

The interface starts with essential fields, allowing merchants to input their store's name and upload their store logo, thereby establishing a strong visual representation of their brand.

Central to the store page's functionality is the “Shops” section, which enables merchants to effortlessly manage multiple physical locations. The user-friendly design streamlines the addition of new shops through the "ADD SHOP" button, offering flexibility and accommodating various business needs. Within this section, merchants can further refine their store's information by editing or deleting existing shops, ensuring that the merchant's store representation remains current and accurate.

For each shop, merchants specify a comprehensive set of information, including the shop's name, email address, phone contact, and physical address. This data allows shoppers to engage with the merchant on various levels. Notably, the store page features the option to specify opening and closing hours for each day of the week. Merchants can tailor this information to reflect the store's operational hours accurately, enabling shoppers to plan their visits with confidence.

By allowing merchants to configure store-specific details, our platform reinforces the merchant's brand identity and strengthens their connection with potential customers. Through the interplay of visual elements like logos and comprehensive data like opening hours, the store profile page fosters an environment of trust and engagement, ultimately enhancing the overall shopping experience.

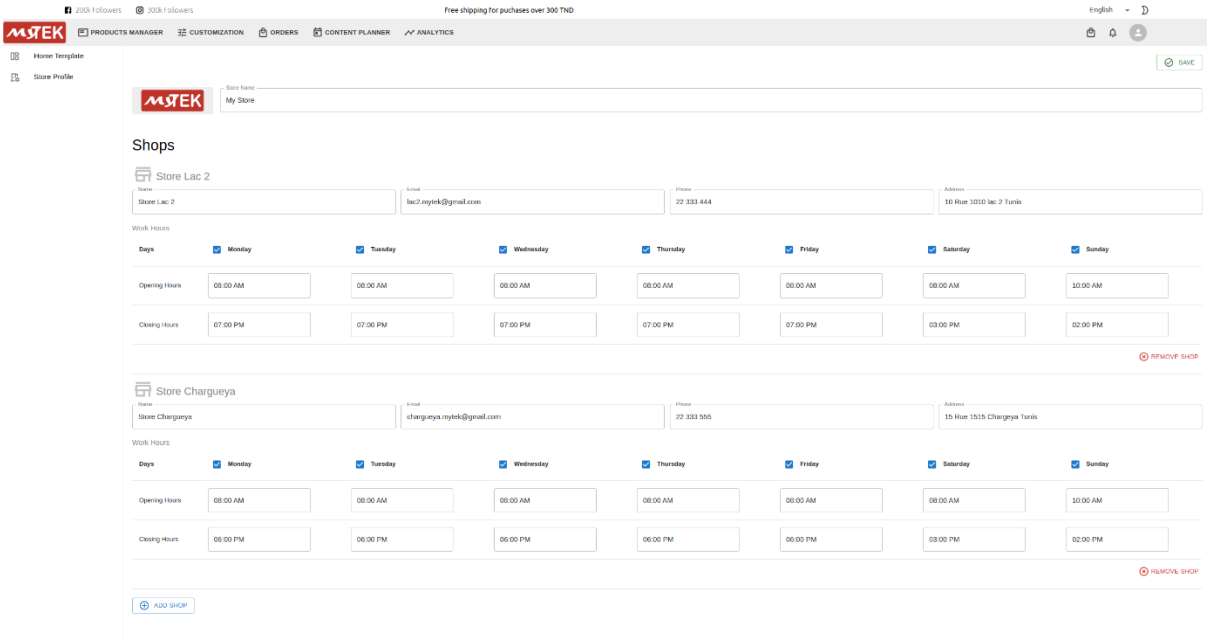

Figure 43: Store Profile Interface

Shopper Application

As users explore the Shopper Application within our multi-tenant e-commerce platform, they'll discover a suite of essential features, thoughtfully crafted to enrich their shopping experience at every step. This section provides an insight into these functionalities, focusing on the core aspects that define the user journey. We'll begin by showcasing the store's home page, offering an inviting starting point for users. Moving forward, we'll delve into the product search functionality. Further along, we'll explore the intuitive shopping cart interface, highlighting its convenience. Lastly, we'll guide you through the streamlined checkout page, completing the comprehensive shopper experience.

❖ Home Page

The shopper's home page introduces the template curated by the merchant (as depicted in figure 42) that serves as an inviting gateway to diverse offerings. Upon arrival, shoppers are greeted with a range of meticulously designed sections. These can encompass a products carousel featuring captivating images for promotional advertisements, inviting visitors to explore further. Additionally, the home page contains carousels highlighting the hottest products, a selection of items on sale, and dynamic displays showcasing various product types or store categories.

Navigation is made intuitive as shoppers effortlessly explore the menu, which offers access to different category groups, nested categories, and specific product types within those

categories. On the right side of the menu, conveniently accessible buttons provide quick entry points to essential actions. These include the search function for easy product discovery, access to the shopper's wishlist, entry to the shopping cart for managing selected items, and a direct link to the user's account for personalized interactions.

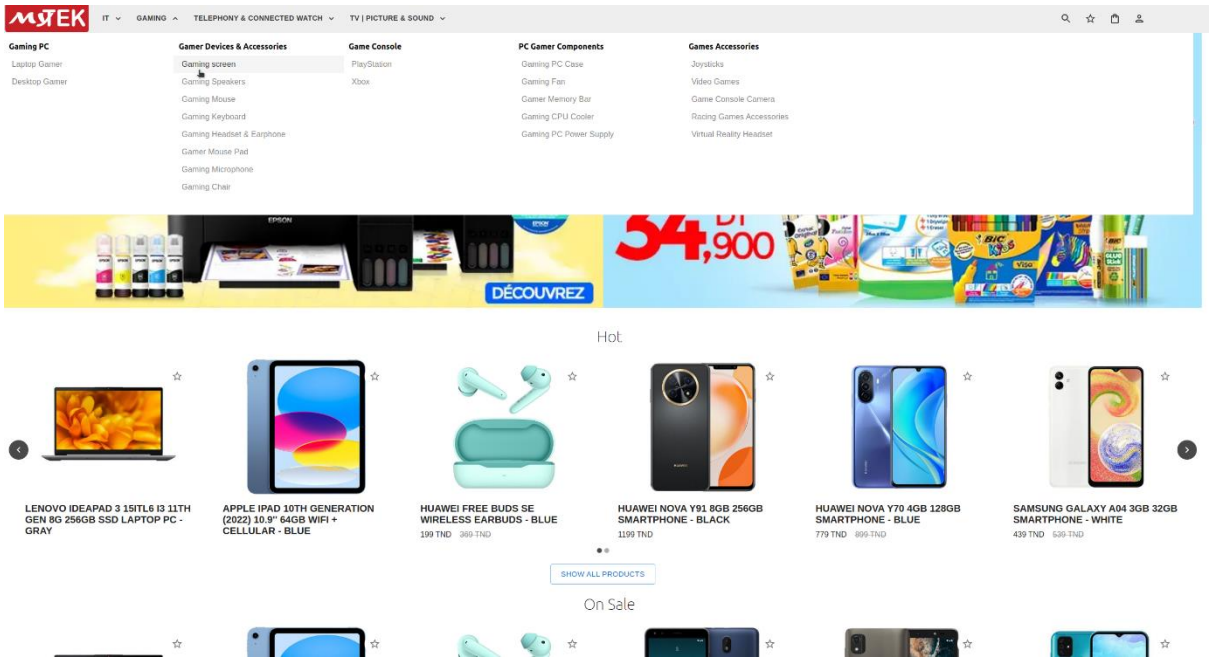

Figure 44: Home Page Interface

❖ **Category Products Showcase**

When a shopper clicks on a category or product type from the menu, they are easily directed to a dedicated page that showcases the products falling within the chosen category. An illustrative example of this can be observed in figure 43, where the user has selected the "Smartphone" category. The interface is designed to enhance the shopper's browsing experience and streamline their product discovery journey.

On the left side of the page, a collection of filtering options is conveniently positioned. These filters encompass both common criteria that apply across all product types as well as criteria specific to the chosen product type. The general filters include attributes such as brand, availability, tags, and price. On the other hand, the specific attributes reflect the technical specifications determined by the merchant via the customization module during the creation of that specific product type. These attributes include details such as smartphone model, operating system, processor, and more.

This level of filtering ensures that shoppers can quickly narrow down their search results and spot products that align with their desired specifications and preferences.

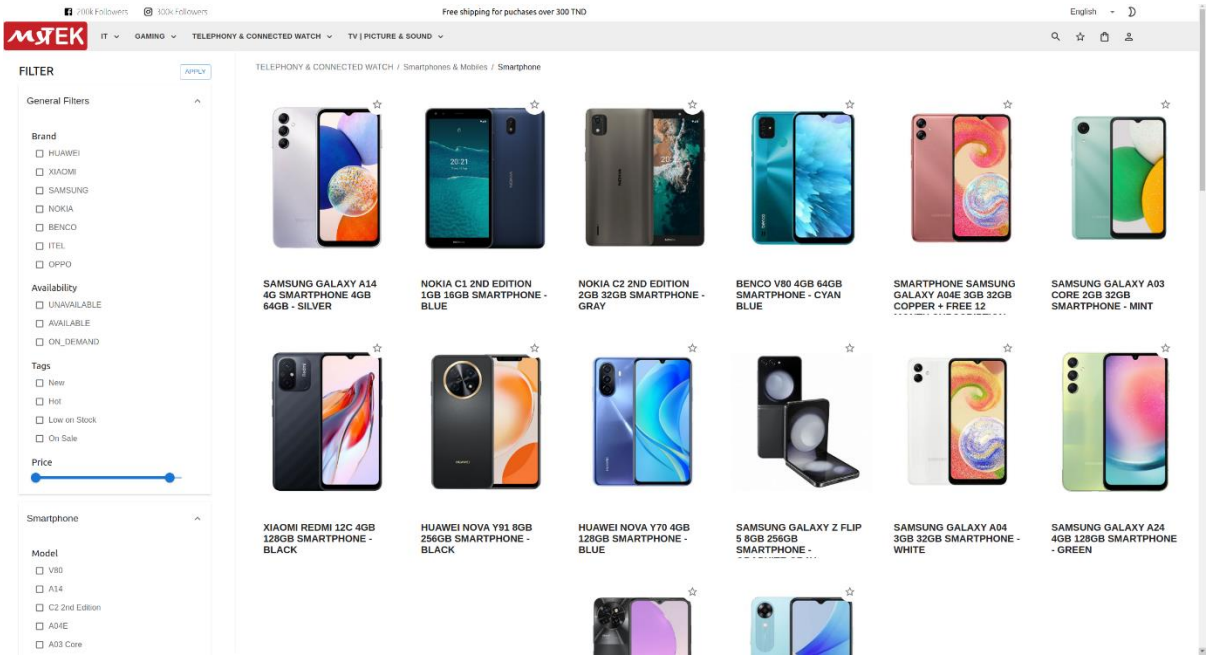

Figure 45: Category Products Interface

❖ Product Search

The product search functionality provides shoppers with an efficient means of discovering products that match their preferences. To illustrate this feature, let's consider the scenario where the keyword "smart" is entered into the search field (figure 44). Upon clicking the search button, the system responds by presenting a collection of products whose titles or descriptions contain the specified keyword. This user-friendly process enhances the shopper's experience by connecting them with products that align with their interests.

Navigating to the left side of the search results page, the interface positions a variety of filtering options. In addition to the general filters, specific filters linked to the product types are strategically integrated. For instance, if the search results include products of the "Smartphones", "Clock Radio & Weather Station", and "Connected Watch" types, these types are presented as specific filtering options. Selecting any of these product types reveals a set of technical attributes associated with them. These attributes enable shoppers to further refine their search results, ensuring that they can effectively and efficiently locate the products that align with their specific requirements and preferences.

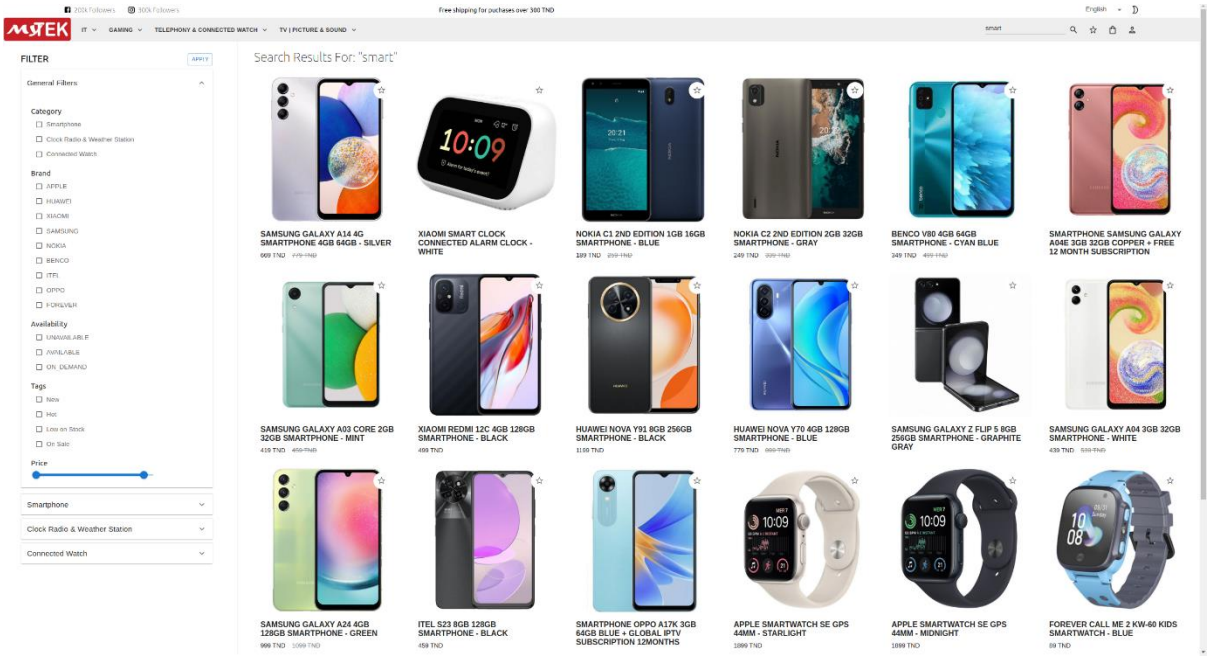

Figure 46: Product Search Results Interface

❖ Shopping Cart

The Shopping Cart interface offers shoppers an easy and intuitive means to check their desired products before finalizing their order. The process begins as shoppers navigate through the variety of products, adding the items they wish to purchase to their shopping cart. Once their shopping is complete, they simply click on the shopping cart icon, directing them to the Shopping Cart page for a comprehensive overview of their selections.

On the Shopping Cart page, the detailed breakdown of the order is revealed. Each product chosen by the shopper is listed, complete with its title, price, selected quantity, and the corresponding total price based on the quantity chosen. Shoppers have the flexibility of adjusting the quantity of any ordered product. This adjustment is automatically reflected in the total price, ensuring that shoppers have a real-time understanding of the cost implications associated with their choices.

Furthermore, the Shopping Cart page provides shoppers with the autonomy to refine their selections. Shoppers can opt to remove products from their shopping cart, aligning their choices with their immediate needs and preferences. Beneath the list of products, the Shopping Cart page furnishes shoppers with essential order insights. The sub-total is presented, encompassing the cumulative total of all the items within the shopping cart, alongside any applicable shipping fees. The complete cost of the order is then clearly displayed as the final total, providing shoppers with a comprehensive overview of the cost.

In addition to managing the specifics of their order, shoppers are provided with the option to include a personalized note before proceeding to the checkout process. Moreover, for those in possession of a coupon, a convenient space is allocated for the application of the coupon code, ensuring that shoppers can optimize the value of their purchase.

Upon finalizing their selections and reviewing the details of their order, shoppers can confidently proceed to the checkout by clicking the designated button. This thoughtful and user-centric Shopping Cart interface streamlines the purchase process, equipping shoppers with all the tools needed to make informed choices and proceed with ease.

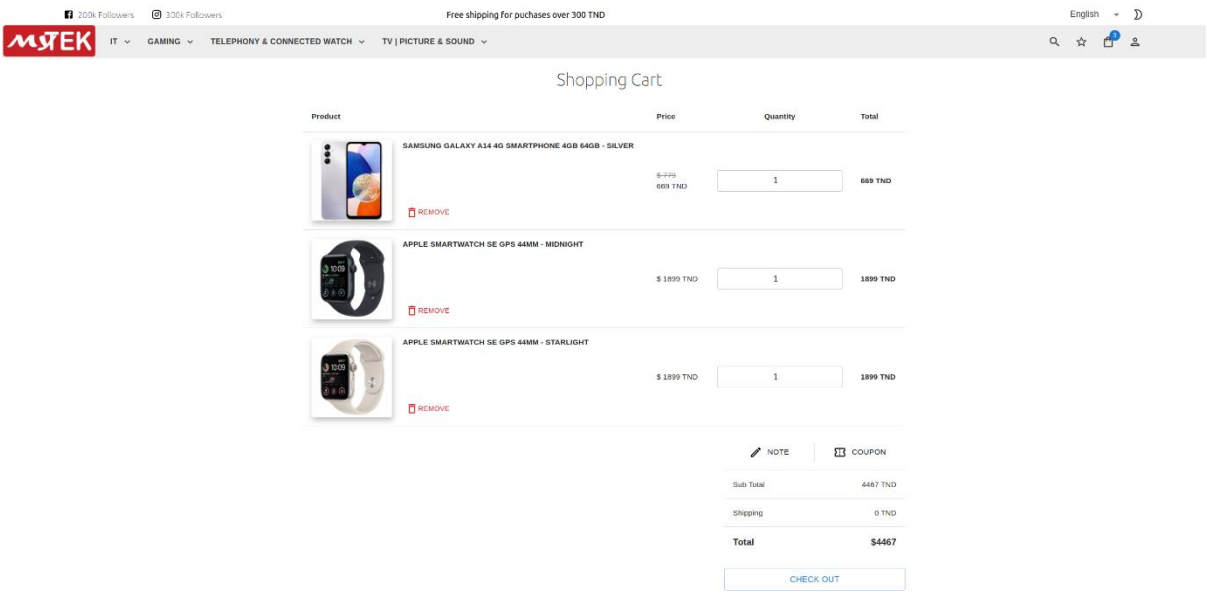

Figure 47: Shopping Cart Interface

❖ Checkout Page

The Checkout Page serves as the final destination of the shopper's journey, encapsulating the critical steps required to complete the order. The checkout process comprises several stages, each carefully designed to ensure a seamless and efficient user experience. Below, we outline the key steps that constitute the Checkout Page:

1. Delivery Details:

Upon initiating the checkout process, the shopper is guided to the first step, exemplified in figure 46. At this stage, the focus lies on capturing the essential delivery information. The system draws upon the stored data from the database, including the shopper's first and last names, email, and default delivery address. Users are provided with the flexibility to edit or input any missing information. Additionally, the option to specify a distinct billing address can be selected, prompting the shopper to provide the relevant details. If the shopper decides to

cancel the checkout process at this stage or any upcoming stages, they can do so with the "Cancel" option. Once these fields are completed, the shopper advances to the subsequent step by selecting "Next".

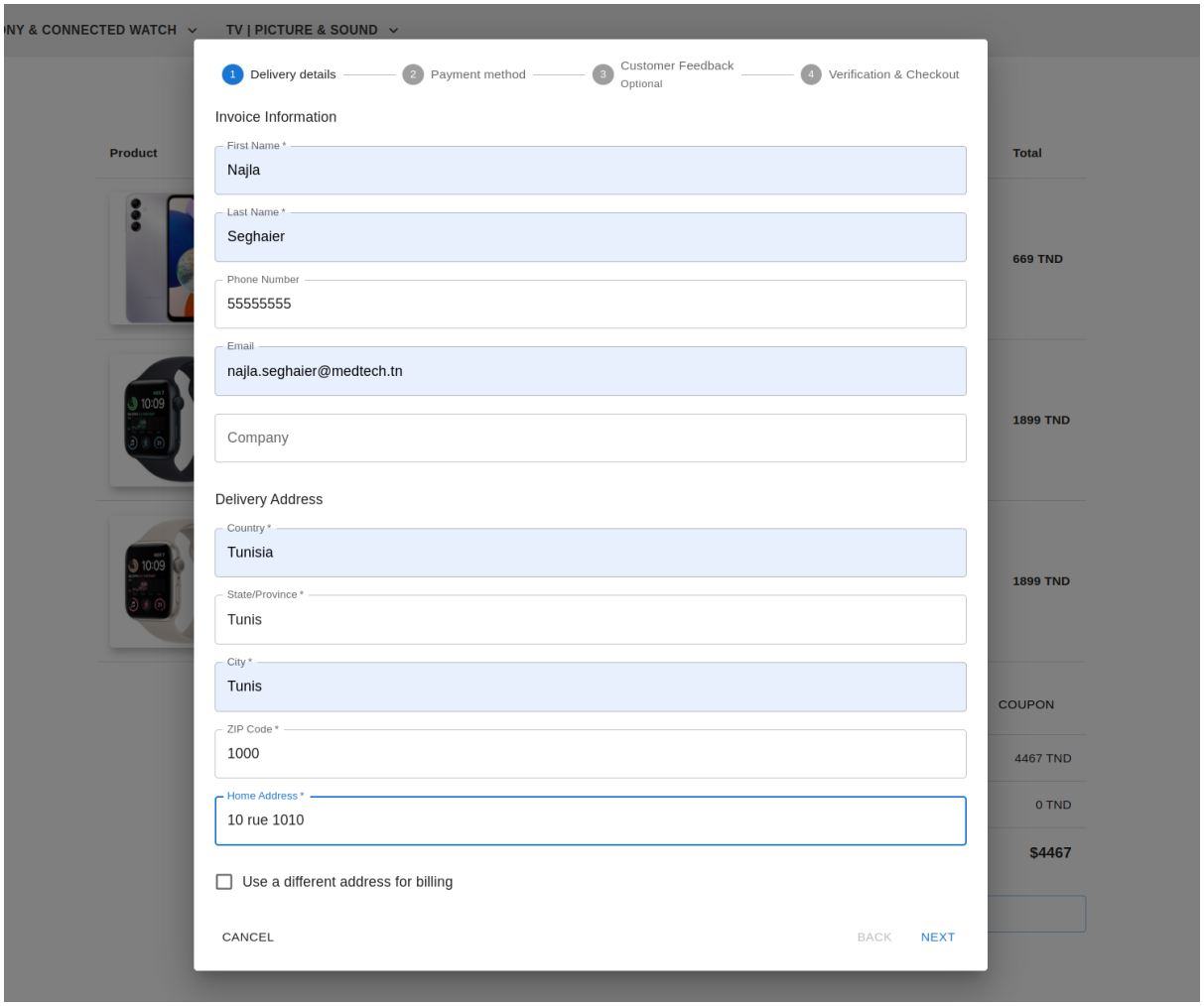

Figure 48: Checkout Page Step 1 - Delivery Details

2. Payment Method:

The second step, illustrated in figure 47, is dedicated to the selection of the preferred payment method. The current options encompass "On Delivery", "Credit Card", and "Bank Transfer". While only the "On Delivery" option is currently enabled, future integrations will extend the range of available payment choices. Concurrently, the total order cost is prominently displayed, ensuring that the shopper remains well-informed about their financial commitment before progressing further. If shoppers wish to revisit the previous step, they can simply click the "Back" button. Conversely, to proceed to the subsequent stage, shoppers can click "Next".

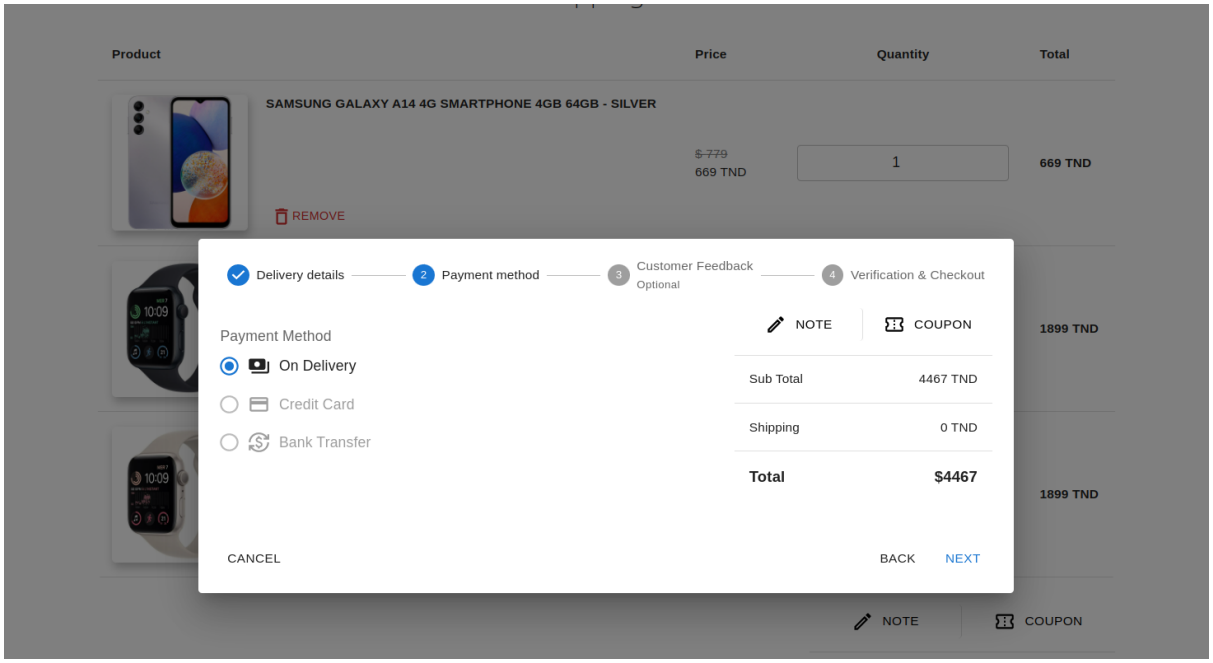

Figure 49: Checkout Page Step 2 - Payment Method

3. Customer Feedback (Optional):

Progressing to the next stage, the "Customer Feedback" phase (figure 48) offers an optional avenue for shoppers to share their insights and experiences. Here, users are provided with the opportunity to rate their overall experience on the platform and to provide additional feedback. For those who choose to opt-out of this step, a "Skip" option is available, enabling a seamless transition to the subsequent stage.

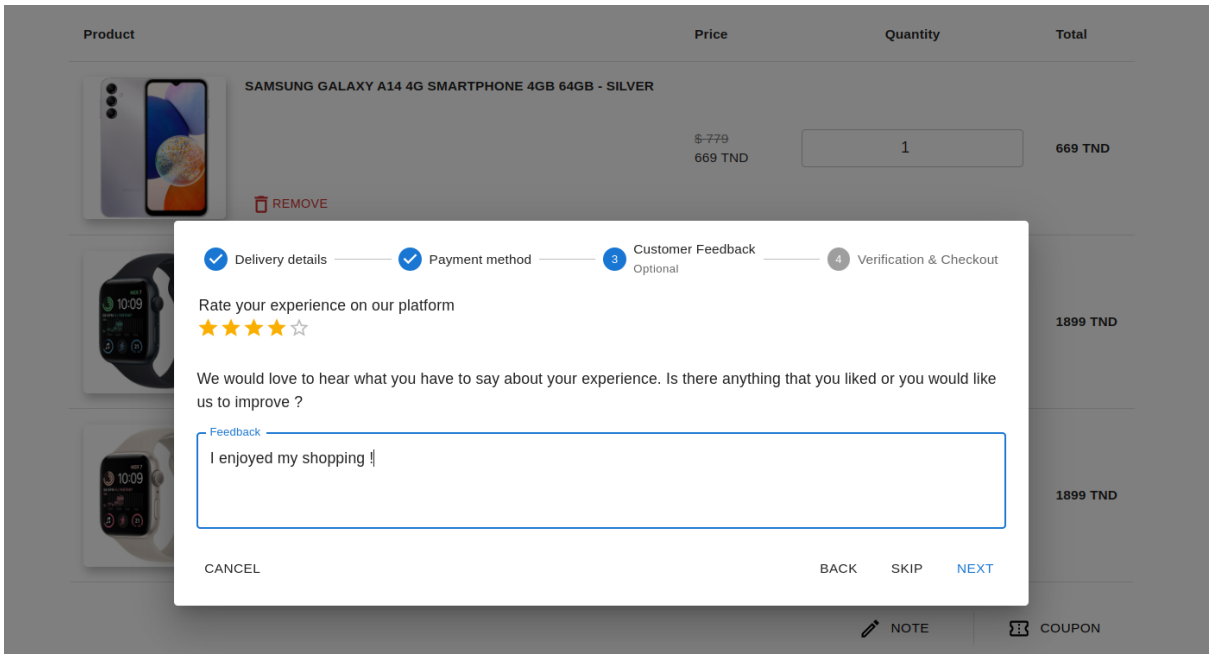

Figure 50: Checkout Page Step 3 - Customer Feedback

4. Final Verification:

The final stage, illustrated in figure 49, culminates in the verification and confirmation of the order. The Checkout Page aggregates all pertinent information for review. This includes the detailed invoice, comprising the shopper's full name and phone number. Additionally, the delivery and billing address details are presented. In cases where the billing address option was not selected during the initial step, the billing address would be the same as the delivery address. However, if the shopper opted for a distinct billing address, that choice is reflected. The payment method, which in our case remains "On Delivery", is restated alongside the comprehensive payment amount breakdown. After carefully examining these details, shoppers can confidently proceed to complete the order by clicking the "Finish" button. This action translates to successful creation of the order.

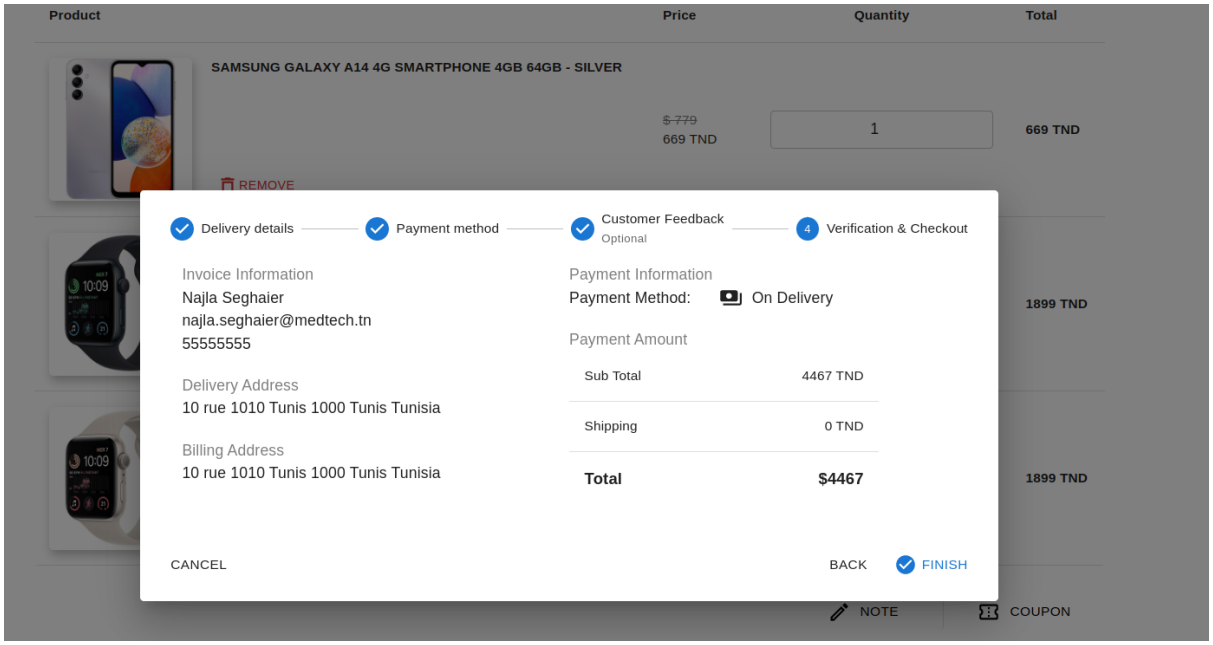

Figure 51: Checkout Page Step 4 - Verification & Checkout

Containerization

Containerization has played a pivotal aspect of our system development. We have leveraged Docker by creating dedicated Dockerfiles for our core API and customization API, resulting in separate Docker images for each of them. This approach provides us with the valuable capability of deploying and scaling these components independently, enhancing the flexibility of our architecture.

For our development environment dependencies, such as MongoDB, we have also opted for using containerized solution rather than installing them directly on the developer's machine. This has the benefit of streamlining experimentation with potential future upgrades. Moreover, this choice facilitates the replication of the exact setup across both production and development environments, effectively mitigating any unexpected issues caused by variations in versions or configurations.

Finally, to orchestrate the various containers essential to our system, we have introduced a Docker Compose file. This file simplifies the task of initiating or shutting down system components while retaining crucial configurations, including port mappings, specifying which Dockerfile to employ for building each service, and preserving data on volumes, among others. This orchestration greatly enhances the manageability and consistency of our containerized system.

Security Implementation

In the following section, we present a series of robust security measures implemented within our platform to ensure protection against a range of threats, encompassing aspects like service denial attacks prevention, secure communication, data encryption, user authentication, and password hashing. These measures collectively ensure the integrity and reliability of our multi-tenant e-commerce platform, reinforcing user trust and safeguarding sensitive information.

1) DoS Attacks Prevention

To enhance our platform's security against Denial-of-Service (DoS) attacks, we have implemented robust measures utilizing NGINX. These measures encompass three approaches designed to block attackers and ensure uninterrupted service:

- **Rate Limiting Requests:** Employing this technique, we strategically restrict the rate at which NGINX accepts incoming requests to a threshold consistent with typical user behavior. For instance, we established that authentic users accessing a login page should be able to

make a request no more frequently than every two seconds. By imposing this limitation, we effectively mitigate the risk posed by malicious scripts attempting simultaneous requests, thus preventing the system from becoming overwhelmed.

- **Connection Count Control:** Another layer of defense involves setting a limit on the number of connections a single client IP address can establish, aligning it with authentic user patterns. This constraint is calibrated to match reasonable user behavior, allowing, in our case, up to ten connections per client IP address. This proactive measure resists against a specific type of DoS attack that seeks to overwhelm the server by flooding it with an excessive number of connections, thereby preserving the system's responsiveness for genuine users.
- **IP Address Denylisting:** In situations where recurrent attack attempts emerge from specific IP addresses, a countermeasure involves adding these malicious sources to an IP address denylist. By doing so, we effectively leave these addresses unable to send any requests to the system. This countermeasure is particularly valuable in combatting persistent attackers and safeguarding the system's stability and performance.

Incorporating these NGINX-based measures for detecting and mitigating service DoS attacks reinforces our platform's resilience against potential threats, ensuring continuous service availability and maintaining a smooth user experience.

2) Audit Trail

We have configured NGINX to maintain access logs alongside error logs. This setup enables us to keep track of the requests directed to our system and identify the source of each request. Currently, we have focused on implementing the logging configuration for these events. However, our future plans involve leveraging these logged entries in combination with threat assessment tools to analyze user behavior and detect any potentially suspicious activities within the system.

3) Authentication and Authorization with JWT

To ensure that every action is carried out only by authenticated users with appropriate authorization, we have implemented Json Web Tokens (JWT). These tokens are included in the "Authorization" header of every request sent to the backend. Within the backend, we have established an authentication middleware that validates user authentication and role-based permissions (Shopper or Merchant) before granting access to the requested endpoint.

4) Secure Client-Server Communication via HTTPS

The fourth security measure involves ensuring secure client-server communication through the implementation of the Hypertext Transfer Protocol Secure (HTTPS) protocol. To achieve this, we configured NGINX to support HTTPS and to automatically redirect any insecure HTTP requests to the secure HTTPS connection. For this purpose, we generated an SSL certificate. While the current certificate is self-signed, it's essential to note that our plan is to replace it with an official certificate as we approach the product's release phase.

5) Encryption of Sensitive Data, including Buyer Addresses (using AES)

To ensure the security of end users' personal information, our system employs the Advanced Encryption Standard (AES) encryption algorithm to store encrypted ciphers instead of plaintext data. This approach guarantees that even in the event of a compromised database, attackers are unable to access users' data due to its encrypted format. As we progress towards the deployment of the platform, our strategy involves storing the encryption keys used for data encryption and decryption on a separate server with extremely restricted access permissions. This configuration not only adds additional security layers to the encryption keys server but also imposes a substantial challenge for attackers, as they must breach two distinct machines almost simultaneously. In the event of a detected attack on the database, access to the encryption keys server will be automatically locked down, further fortifying the overall security framework.

6) Password Hashing

In contrast to encryption, hashing operates in a manner that prevents the original text from being derived from the hashed result. Hashing also offers valuable features, including the generation of unique and entirely distinct hashes for all input texts, irrespective of their similarities. These attributes make hashing a secure approach for ensuring the protection of users' passwords from unauthorized access. In our implementation, we utilized Node.js's bcrypt library to hash the passwords. Our approach involves storing password hashes instead of the actual passwords. In the unfortunate event that an attacker gains access to the database, they will only acquire a seemingly random sequence of characters that cannot be reversed into the original password. Furthermore, due to the uniqueness of hashes for each input text, we can compare the stored hash to the hash of the password provided by the user during login, allowing us to accurately verify its authenticity against the original password.

4.3.5. Solution Evaluation

In this section, we assess the effectiveness of our solution through rigorous testing and evaluation processes.

We implemented unit tests for both controllers and services, with each test focusing on specific aspects of their respective responsibilities. In the case of controllers, our tests primarily concentrated on request validations and the responses generated by the endpoints. For instance, as depicted in Figure 52, the categories controller would return a "bad request" response if the request body did not contain all the necessary data for creating a category. Conversely, after a successful operation, the controller was expected to return the newly created category's data in the response.

In contrast, service tests delved into the business logic. When creating an order, for instance, the service checked that all selected products existed in the database and were available for purchase. Additionally, the service executed complex logic to calculate the order's total amount, taking into consideration discounts on products and applied coupons.

To ensure the quality and reliability of our unit tests, we adhered to industry-standard testing principles:

- **Repeatability:** Unit tests were designed to produce consistent results, failing or succeeding only when parameters changed.
- **Isolation:** Prior to running each test case, shared resources between multiple unit tests were reset, ensuring test independence and reliability.
- **Limited Scope:** Unit tests for a specific method operated without relying on the correctness of other methods or classes. When necessary, external methods were mocked, returning expected valid results.
- **Lightweight:** Our unit tests refrained from interacting with the database or making network requests to external APIs. In such cases, we created mocked (fake) classes or methods to simulate expected responses.

These testing practices upheld the integrity of our unit tests and contributed to the overall robustness and quality of our solution.

❖ Categories Unit testing

a) Categories controller

```
> backend@1.0.0 test
> mocha ./tests/**/*.test.js "./tests/controllers/categories" "./tests/controllers/orders"


  Api
    controllers
      categories
        GET /categories
          ✔ should return a bad request response if globalCategoryId query parameter has an invalid format
          ✔ should return all categories in category group if globalCategoryId query parameter is provided
          ✔ should return all categories returned by the service
        PUT /categories/:id
          ✔ should return a bad request response if the id parameter has an invalid format
          ✔ should return a bad request response if the category label is missing
          ✔ should return a not found response if the service throws a not found response
          ✔ should return the updated category
        POST /categories
          ✔ should return an unauthorized response if the user is a SHOPPER
          ✔ should return a bad request response if the category label is missing
          ✔ should return a bad request response if the parent global category id is missing
          ✔ should return the newly created category
        DELETE /categories/:id
          ✔ should return an unauthorized response if the user is a SHOPPER
          ✔ should return a bad request response if the id parameter has an invalid format
          ✔ should return a not found response if the service throws a not found error
          ✔ should return an empty response


  15 passing (6ms)
```

Figure 52: Categories Controller Unit Tests

b) Categories service

```
> backend@1.0.0 test
> mocha ./tests/**/*.test.js


  Api
    services
      categories
        getAll()
          ✔ should return all categories in the store based on storeId parameter
          ✔ should fetch categories in a category group if globalCategoryId parameter is provided
        create()
          ✔ should insert new category in the database
          ✔ should attach the newly created category to the category group
        deleteById()
          ✔ should throw a not found error if the category does not exist
          ✔ should throw a not found error if the category belongs to another store
          ✔ should delete all related product types
          ✔ should update the category group that the category used to belong to


  8 passing (5ms)
```

Figure 53: Categories Service Unit Tests

❖ Orders Unit testing

a) Orders controller

```
> backend@1.0.0 test
> mocha ./tests/**/*.test.js


  Api
    controllers
      orders
        GET /orders
          ✔ should return all orders returned by the service
          ✔ should return populated shopper field if returned by the service
        GET /orders/:id
          ✔ should return a bad request response if the id parameter has an invalid format
          ✔ should return not found if the service throws a not found error
          ✔ should return the list of orders returned by the service
        POST /orders
          ✔ should return a bad request error if the order products are undefined
          ✔ should return a bad request error if the order products is an empty array
          ✔ should return a bad request error if the order products have an invalid format
          ✔ return the created order
        PUT /orders/:id
          ✔ should return a bad request response if the id parameter has an invalid format
          ✔ should return a not found response if the service throws a not found error
          ✔ should reject the request if it does not update the status, list of products and delivery date
          ✔ should return a forbidden response if the service throws a forbidden error
          ✔ return the updated order


  14 passing (5ms)
```

Figure 54: Orders Controller Unit Tests

b) Orders service

```
> backend@1.0.0 test
> mocha ./tests/**/*.test.js


  Api
    services
      orders
        getAll()
          ✔ should return all orders in the store for merchant
          ✔ should populate the shopper field for merchant
          ✔ should return only user's orders for shopper
        getById()
          ✔ should throw a not found error if the order does not exist
          ✔ should throw a not found error if the order belongs to another store
          ✔ should throw a not found error if the order belongs to another user for SHOPPER
          ✔ should return any order in the store for merchant
          ✔ should populate the shopper field for merchant
        create()
          ✔ should verify that all products exist and are available or on demand
          ✔ should calculate total amount and set status to pending for new orders
        updateById()
          ✔ should throw not found error if the order does not exist
          ✔ should throw not found error if the order belongs to another store
          ✔ should throw not found error for SHOPPER if the order belongs to another user
          ✔ should throw a bad request error if the data does not update the status, list of products and delivery date
          ✔ should throw forbidden error for SHOPPER if the order is NOT in pending state
          ✔ should throw forbidden error for SHOPPER if the order is NOT in PENDING state
          ✔ should throw forbidden error for MERCHANT if the order is NOT in PENDING or CONFIRMED state
          ✔ should verify that all products exist and are available or on demand and recalculate the total amount if the list of products is updated
          ✔ should NOT update delivery date if it was NOT provided in the data
          ✔ should NOT update products list if it was NOT provided in the data
          ✔ should NOT update status if it was NOT provided in the data


  21 passing (7ms)
```

Figure 55: Orders Service Unit Tests

5. RESULTS AND FINDINGS

The primary objective of this section is to present and explain the key results and findings in relation to each of the stated research questions.

5.1. Research Question 1

What are the key design considerations and trade-offs in implementing a multi-tenant system?

In the pursuit of investigating the essential design considerations and trade-offs inherent in the implementation of a multi-tenant system, we navigate the following domains:

a) Data Architecture Consideration: Balancing Scalability and Cost Efficiency

In the multi-tenant system domain, the selection of a data architecture significantly shapes both its quality attributes and the associated financial implications. Our focus on scalability led us to opt for the Shared Database, Shared Collections approach. This strategic choice aligns scalability and cost-effectiveness, striking an optimal balance.

While Dedicated Databases offer peak scalability but come with high costs, Shared Database with dedicated collections can limit scalability due to collection constraints. Our selection of the Shared Database, Shared Collections approach offers the best of both worlds, fostering horizontal scalability for tenant growth. New nodes and resources can be added to distribute database loads, enhancing scalability without inflating costs.

This architectural model not only addresses scalability limitations tied to collection constraints but also promotes cost efficiency. Shared infrastructure minimizes hardware requirements, streamlining implementation and maintenance, and benefiting from the economy of scale principle.

b) System Architecture Consideration: Balancing Scalability and Complexity

The foundation of our system architecture centers around the Service-Oriented Architecture, positioned at the intersection of complexity and scalability. The range of available architectural options introduces a spectrum of trade-offs that need to be considered.

At one end of the spectrum, we have the Monolithic architecture, characterized by its simplicity and ease of management. However, it comes with the trade-off of limited scalability.

While this approach serves well for straightforward applications, it may pose challenges when aiming to accommodate substantial growth.

On the opposite end of the spectrum, the Microservices architecture stands out for its scalability potential. This approach involves breaking down the system into independent services that can be scaled individually. Yet, it introduces an increased level of complexity and demands careful orchestration.

In the context of these architectural choices, our strategic choice of the SOA represents a balanced compromise. By aligning complexity and scalability, we position the platform to achieve optimal outcomes. This approach allows us to harness the benefits of scalability without overwhelming the system with unnecessary intricacies.

c) Quality Attributes Consideration: Balancing Maintainability and Flexibility

In the context of quality attributes, we confront a pivotal trade-off between maintainability and flexibility, where we must carefully weigh the long-term health of the system against its capacity to adapt to evolving and diverse demands.

On one side, an elevated degree of flexibility holds the promise of attracting a wider range of customers and catering to diverse needs. However, it also introduces an increased level of system complexity, leading to potential challenges in the system's maintenance over time.

In response to this intricate trade-off, our solution employs a strategic approach. It exhibits the necessary flexibility to accommodate most common use cases across a variety of business domains. Meanwhile, it reserves the handling of more specific scenarios for on-demand feature requests.

By striking this balance between maintainability and flexibility, we craft a solution that not only caters to immediate customer needs but also safeguards against the potential complexities that can arise in the maintenance phase.

5.2. Research Question 2

How can a shared platform meet the specific needs of different businesses?

The second research question delves into the strategies for developing a shared platform that can effectively cater to the unique demands of various businesses. This section highlights the incorporation of customization features within the platform, facilitating merchants to create distinctive user experiences for their customers. Furthermore, we will explore the inherent adaptability of the platform to diverse business domains.

a) Customization Features for Tailored Experiences

To accommodate the specific needs of different businesses, our approach involved the incorporation of different customization features within the platform. By empowering merchants to form their digital storefronts, we provide them with the potential of personalized interactions with their customers. Key elements of customization encompass:

- **Branding Elements:** Customizable elements such as store information and logos allow merchants to infuse their brand identity throughout the platform.
- **Template Selection**: Merchants can choose from an array of home page templates, aligning their store's visual identity with their brand.
- **Product Display:** Flexibility in product arrangement and showcasing enables merchants to highlight their unique offerings effectively.
- **Home Page Content:** Tailoring the content displayed on the home page offers a prime opportunity to communicate the brand's story and value proposition.
- **Store Configuration:** Merchants can configure store policies, payment methods, and shipping options, providing a tailored experience to their customers.

By empowering merchants with these tools, we foster an environment where businesses can deliver an authentic and captivating experience to their customers, enhancing brand loyalty and customer engagement.

b) Platform Adaptability: A Universal Foundation

In our study to design a shared platform capable of meeting diverse business needs, we focused on establishing a universal foundation. This adaptable architecture serves as the basis for accommodating a range of industries and use cases. Key attributes that contribute to the platform's adaptability include:

- **Modular Components:** The platform's architecture is built on modular components, enabling seamless integration of tailored features and functionalities as required by different business domains.

- **Data Structure Flexibility:** The data model is designed to be flexible, accommodating varying business structures, attributes, and data requirements. For instance, the introduction of a flexible schema empowers merchants to set additional fields specific to each product type. These fields are made available during product creation, facilitating the incorporation of supplementary information about the product. Furthermore, merchants can determine whether each field should serve as a filtering option for shoppers during product search.

- **Merchant-Defined Lists:** Merchants have the authority to define their own lists of category groups, categories, and product types, promoting tailored categorization to suit their business needs.

This approach to adaptability and customization not only caters to the specific needs of different businesses but also positions the platform as a dynamic solution that can evolve alongside the ever-changing landscape of e-commerce.

5.3. Research Question 3

How can an e-commerce platform be designed and developed to meet the quality requirements of scalability, security, and maintainability?

Addressing the question of how to design and develop an e-commerce platform that adequately aligns with the crucial attributes of scalability, security, and maintainability constitutes a central interest in our investigation. The multifaceted nature of this inquiry necessitates thoughtful considerations across these critical domains.

a) Scalability Strategies:

Our focus on scalability is underscored by a series of strategic measures designed to accommodate the e-commerce platform's growth and demands:

- **Leveraging SOA for Containerization:** At the heart of our scalability approach lies the adoption of a Service-Oriented Architecture. This architectural paradigm empowers us to encapsulate functionalities into containerized services. These self-contained units are characterized by their deployability and replicability, ensuring a level of

operational isolation that is vital for scalability. By adhering to SOA principles, we pave the way for agile and dynamic scalability. When demand surges, new instances of containerized services can be easily introduced, allowing the system to effortlessly expand its capacity. Conversely, in periods of reduced demand, services can be scaled down or deactivated, optimizing resource utilization. This dynamic orchestration of services exemplifies our commitment to providing a responsive and resilient platform.

- **Embracing MongoDB for Distributed Data:** Our choice to embrace MongoDB as the cornerstone of our distributed data strategy is deeply aligned with our quest for scalability. MongoDB's schema flexibility and distributed architecture enable us to effortlessly scale horizontally, an essential requirement for accommodating the ever-increasing data volumes in an e-commerce ecosystem. The sharding capabilities of MongoDB enable data to be partitioned and distributed across multiple servers, ensuring efficient data retrieval and processing. This strategic utilization of MongoDB equips our platform with the inherent ability to gracefully expand as data loads surge, all the while maintaining optimal system performance. The seamless addition of new servers ensures that the system can gracefully accommodate the growing demand while avoiding bottlenecks.

b) Security Strategies:

Our platform's security enhancements encompass an array of safeguards:

- **Resisting DoS Attacks:** Implementing specialized tools for the identification and prevention of service denial attacks serves as a preemptive measure against disruptions to service availability.

- **Facilitating Audit Trails:** The incorporation of audit trail tools supports accountability and traceability, enhancing security through comprehensive record-keeping.

- **Authentication and Authorization via JWT:** Employing JWT for authentication and authorization ensures secure user access to the platform's functionalities.

- **Secure Client-Service Communication:** The use of HTTPS protocols establishes a secure communication channel, safeguarding data integrity and confidentiality during transit.

- **Data Encryption:** The encryption of sensitive information, such as shopper addresses through the AES algorithm, augments the security of critical data against unauthorized access.

- **Password Hashing:** Employing password hashing mechanisms ensures that user credentials are stored securely, minimizing the risk of unauthorized exposure.

c) Maintainability Strategies:

To ensure the platform's long-term maintainability, a series of strategic considerations were implemented:

- **Guiding Code Practices:** Throughout the project, shared code best practices were defined, supported by ESLint rules, to maintain uniformity and comprehensibility across codebases.
- **Structured Project Organization:** A carefully thought project structure was devised to foster consistency between the FE the BE codebases. This facilitates efficient identification of code components controlling specific features.
- **Modularity Emphasis:** The creation of independent modules in both FE and BE was based on actor roles (Merchant or Shopper) and feature types (Core or Customization) respectively, promoting code modularity and separation of concerns.
- **Centralized Shared Logic:** The encapsulation of shared logic within reusable components in the FE (the "Common" packages) and utility services in the BE (the "Shared Lib" package) not only promotes consistency but also enhances maintainability by reducing redundancy.
- **Version Control:** Version control, facilitated by Git, is a fundamental pillar of our maintenance strategy. Git enables us to track changes to the codebase, providing a comprehensive history of modifications, bug fixes, and enhancements. By maintaining a version-controlled repository, we ensure that code changes are well-documented and can be easily reverted if necessary. Additionally, Git's branching and merging capabilities allow us to work on new features and fixes in isolated branches, minimizing disruption to the main codebase. This practice promotes collaboration among development teams, reduces the risk of code conflicts, and enhances the overall maintainability of the platform.

This synthesis of strategic scalability, security, and maintainability measures underscores our commitment to developing an e-commerce platform that thrives in a dynamic digital landscape while prioritizing user security and fostering a sustainable development ecosystem.

6. DISCUSSION

Our study has encompassed a thorough analysis of the existing state-of-the-art, culminating in the development of a comprehensive and cost-effective solution. This solution is tailored to empower businesses to establish a foothold in the dynamic e-commerce landscape, enabling them to transition into the digital era and streamline their operational processes. In this section, we delve into a detailed examination of the impact our study holds for both the research domain and the practical industry landscape. Additionally, we address potential concerns pertaining to the validity of our findings, and highlight the approach undertaken to ensure the robustness and reliability of our results.

6.1. Impact on Research

This study represents an attempt to advance the field of e-commerce and multi-tenant platforms, and contributes to the existing body of knowledge in several ways:

1. **Advancement of Multi-Tenancy Understanding:** By delving into the concept of multi-tenancy within e-commerce platforms, our research has expanded the understanding of how businesses can efficiently share a single platform while maintaining their distinct identities and data. This exploration opens avenues for further investigation into the architectural, security, and customization aspects of multi-tenant systems.
2. **Application of Design Science Methodology:** Our utilization of the design science research methodology offers a structured framework that can be adopted by future researchers. This approach not only ensures the development of a robust e-commerce platform but also provides a systematic model for addressing complex problems in the domain of software engineering and technology.
3. **Innovation in Customization and Maintainability:** The innovative incorporation of customization options within a multi-tenant framework challenges the conventional understanding of multi-tenant platform design. Meanwhile, we paid close attention to how customization can limit the project's maintainability and made sure to find a healthy balance between the two. This prompts future researchers to explore the delicate balance between standardization and personalization, offering potential avenues for investigating enhanced user experiences and customer engagement.
4. **Security and Architecture Considerations:** The integration of robust security measures and architectural considerations highlights the importance of sustainable and secure e-commerce platforms. This presents opportunities for in-depth studies on fortifying the

multi-tenant environment against evolving cyber threats and further optimizing platform architectures.

5. **Scalability Insights:** Our study contributes to the understanding of scalability within multi-tenant e-commerce platforms. We explore how businesses can efficiently expand their operations to meet growing demands, offering insights into optimizing resource allocation, load balancing, and system responsiveness. This adds to the knowledge base for creating resilient and adaptable e-commerce ecosystems, preparing businesses for the dynamic digital marketplace.
6. **Addressing Practical Business Challenges:** Our study sheds light on the practical challenges faced by SMBs in entering the e-commerce landscape. By offering a scalable and cost-effective solution, our findings underscore the significance of catering to SMBs and inspire future research into devising tailored strategies for this critical segment.

6.2. Impact on Industry

Our study's implications extend beyond academia, providing substantial value to the e-commerce industry and businesses:

1. **Enhanced Business Viability:** By introducing a comprehensive and affordable solution for businesses seeking to establish a digital presence, our study directly addresses the hurdles faced by SMBs in entering the e-commerce field. Our platform's customizable and scalable architecture empowers SMBs to navigate the digital domain with greater ease and efficacy, enabling them to compete and thrive in a dynamic marketplace.
2. **Streamlined Operational Efficiency:** The multi-tenant e-commerce platform crafted in this study offers businesses a centralized hub for managing their online stores, products, and orders. This translates into streamlined operational processes, reduced complexities, and enhanced productivity.
3. **Tailored Brand Signature:** The customization options embedded within our platform empower merchants to tailor their digital storefronts, reinforcing their brand identity and fostering customer loyalty. This customization potential not only supports businesses' marketing strategies but also encourages entrepreneurship by providing a level playing field for SMBs to present their unique offerings to a wider audience.
4. **Facilitated Digital Transformation:** In an era where digital commerce is central to success, our study's findings offer a roadmap for businesses aiming to shift from traditional modes of operation to thriving within the digital landscape. The strategic understanding of multi-tenancy, architectural considerations, and customization

empowers industry leaders to strategically leverage technology, propelling sustainable growth and enhanced market competitiveness.

The contributions made in this study serve as a foundation for future research endeavors, providing a platform for exploring the intricacies of multi-tenant e-commerce systems and their implications for both academia and industry. As the digital landscape continues to evolve, our work invites scholars to build upon these findings, refine methodologies, and uncover new dimensions in the domain of e-commerce platform development.

6.3. Threats to Validity

While every effort was made to ensure the rigor and reliability of this study, it is essential to acknowledge potential threats to the validity of our findings. By acknowledging these limitations, we aim to provide a comprehensive assessment of the study's scope and its potential impact.

6.3.1. Construct Validity

The development of a multi-tenant e-commerce platform necessitated making certain design decisions based on our understanding of business needs and technical concepts. This may introduce construct validity concerns as alternative design choices or misunderstood notions could yield different outcomes. To mitigate this threat, we engaged in an extensive analysis of the existing related work to align our understanding of the key concepts with the state-of--art.

6.3.2. Internal Validity

The study employed a design science research methodology, which involves iterative development and testing. However, changes in external factors, technological advancements, or unforeseen interactions could impact the internal validity of our findings. We addressed this threat by carefully documenting our methodology and design decisions, enabling future researchers to replicate and extend our work.

6.3.3. External Validity

As the study's scope focused on addressing the challenges of businesses in a specific context, namely SMBs in the e-commerce landscape, the generalizability of our findings to other industries or larger corporations might be limited. We attempted to mitigate this threat by providing a detailed description of the study's context and methodology, allowing readers to assess the applicability of our insights to their specific scenarios.

Additionally, the qualitative insights gathered from business owners might be subject to sampling bias, as the selected individuals' perspectives might not fully represent the entire industry. To address this, we aimed to engage with a diverse range of merchants, ensuring a broader representation of viewpoints.

6.3.4. Conclusion Validity

The reliability of our findings could be influenced by variations in the design and implementation of the developed platform. To enhance the reliability of our research, we documented the platform's architecture, codebase, and technological choices comprehensively, enabling future researchers to reproduce our achievements.

While these threats to validity are acknowledged, they do not diminish the significance of our research and its contributions. By transparently discussing these limitations, we provide a framework for future researchers to build upon our work, further refine methodologies, and explore new dimensions in the domain of e-commerce platform development.

7. CONCLUSIONS

In the dynamic landscape of modern commerce, where businesses seek to thrive in the digital era, our study has culminated in a robust and comprehensive multi-tenant e-commerce platform. Our journey through the intricacies of architecture, customization, and scalability has yielded valuable insights that resonate both in the field of research and the practical world of industry.

Our endeavor was rooted in the pursuit of providing a tailored solution to the challenges faced by businesses, particularly SMBs, in their quest to establish a digital presence. Through the application of design science research methodology, the leverage of cutting-edge technological approaches, and a firm commitment to user-centered design, we present a platform that empowers merchants to seamlessly transition into the digital commerce landscape. The convergence of multi-tenancy, architectural considerations, and customization options within our platform highlights the balance between maintainability and personalization, bridging the gap between standardized offerings and individual brand identities.

In navigating the intricate facets of e-commerce, our findings have direct implications for both academia and industry. Through the lens of research, our study advances the understanding of multi-tenant e-commerce platforms, offering fresh perspectives on architecture, security, customization, and scalability. The systematic application of design science research methodology serves as a guidance for future scholars, providing a structured framework for tackling complex problems and fostering innovation in software engineering and technology.

In the practical field of industry, our study unlocks doors to enhanced business viability, and streamlined operational efficiency. SMBs, often constrained by resources, can now harness our platform to organize distinct online storefronts, nurture customer loyalty, and carve their unique niche in the digital world. Thus, our work has the potential to shape the trajectory of businesses, helping them prosper in a world where digital commerce is paramount.

As we reflect on our journey, we acknowledge the inherent limitations and potential threats to validity that accompany any endeavor of this nature. However, these limitations do not overshadow the significance of our contributions. Instead, they serve as signposts for future explorations, encouraging scholars to delve deeper, refine methodologies, and expand the horizons of e-commerce platform development.

Looking forward, we envision a road paved with opportunities for future research. The realm of multi-tenant e-commerce platforms continues to evolve, offering avenues for in-depth

studies in areas such as security measures, architectural innovations, and user-centric customization strategies. Moreover, the integration of emerging technologies, such as artificial intelligence and blockchain, holds the promise of reshaping the landscape once again, demanding novel insights and inventive solutions.

Furthermore, in alignment with the initial objectives outlined in this report, our ongoing dedication persists as we plan to move ahead with the implementation of a centralized social media management system within our platform. This enhancement will empower businesses to seamlessly post and analyze their social media activities across diverse platforms, all from a singular and streamlined hub. This progression underscores our commitment to continuous improvement and adaptation to the evolving needs of modern commerce.

Additionally, we intend to study the feasibility of deploying the platform on the cloud, driven by its numerous advantages, particularly in scalability and efficient resource utilization. These attributes hold exceptional significance for our project, as our foundation rests on a multi-tenant platform that demands seamless expansion and resource optimization.

As our platform scales and data volume expands, the significance of a robust search mechanism becomes evident in ensuring a smooth shopping experience. To this end, we also plan to integrate the Elasticsearch engine to optimize the search functionality and enable rapid information retrieval.

Concurrently, our commitment to broadening our suite of customization features remains solid, empowering merchants to curate highly distinctive storefronts that deeply resonate with their target audience.

In conclusion, our study highlights the significance of interdisciplinary collaboration, rigorous methodology, and a holistic approach to addressing contemporary challenges. The essence of our work, tied with theory, innovation, and practicality, holds the potential to shape the future of e-commerce in Tunisia, inspire innovation, and empower businesses to navigate the digital world with confidence.

REFERENCES

1. F. Chong, G. Carraro, and R. Wolter. (2006, June). "Multi-Tenant Data Architecture". Microsoft [Online]. Available: https://renatoargh.files.wordpress.com/2018/01/article-multi-tenant-data-architecture-2006.pdf
2. Z. H. Wang, C. J. Guo, B. Gao, W. Sun, Z. Zhang and W. H. An, "A Study and Performance Evaluation of the Multi-Tenant Data Tier Design Patterns for Service Oriented Computing", 2008 IEEE International Conference on e-Business Engineering, Xi'an, China, 2008, pp. 94-101, doi: 10.1109/ICEBE.2008.60.
3. K. Wood and M. Anderson, "Understanding the Complexity Surrounding Multitenancy in Cloud Computing", 2011 IEEE 8th International Conference on e-Business Engineering, Beijing, China, 2011, pp. 119-124, doi: 10.1109/ICEBE.2011.68.
4. W. J. Brown, V. Anderson and Q. Tan, "Multitenancy - Security Risks and Countermeasures", 2012 15th International Conference on Network-Based Information Systems, Melbourne, VIC, Australia, 2012, pp. 7-13, doi: 10.1109/NBiS.2012.142.
5. S. Architects, "Platform Multitenant architecture", *Salesforce Architects*, Aug. 2022, [Online]. Available: https://architect.salesforce.com/fundamentals/platform-multitenant-architecture
6. C. Momm and R. Krebs, "A Qualitative Discussion of Different Approaches for Implementing Multi-Tenant SaaS Offerings", in Proceedings of Software Engineering 2011 (SE2011), Workshop(ESoSyM- 2011).
7. R. Krebs, C. Momm, and S. Kounev, "Architectural Concerns in Multi-Tenant SAAS Applications", ResearchGate, Apr. 2012, [Online]. Available: https://www.researchgate.net/publication/264942141_Architectural_Concerns_in_Multi-Tenant_SaaS_Applications
8. bitcurrent, "Bitcurrent cloud computing survey 2011", bitcurrent, Tech. Rep., 2011
9. H. Lin, K. Sun, S. Zhao and Y. Han, "Feedback-Control-Based Performance Regulation for Multi-Tenant Applications", 2009 15th International Conference on Parallel and Distributed Systems, Shenzhen, China, 2009, pp. 134-141, doi: 10.1109/ICPADS.2009.22.
10. K. -f. Li, H. -y. Wen and Z. -j. Yang, "Business logic structure of SaaS-based E-commerce system", 2010 IEEE 17Th International Conference on Industrial Engineering and Engineering Management, Xiamen, China, 2010, pp. 316-319, doi: 10.1109/ICIEEM.2010.5646604.
11. M. Turner, D. Budgen and P. Brereton, "Turning software into a service", in Computer, vol. 36, no. 10, pp. 38-44, Oct. 2003, doi: 10.1109/MC.2003.1236470.
12. D. Ma, "The Business Model of "Software-As-A-Service"", in 2007 IEEE International Conference on Services Computing, Salt Lake City, UT, 2007 pp. 701-702. doi: 10.1109/SCC.2007.118

13. W.-T. Tsai, Q. Shao, Y. Huang, and X. Bai, "Towards a Scalable and Robust Multi-tenancy SaaS", Proceedings of the Second Asia-Pacific Symposium on Internetware - Internetware '10, 2010, doi: https://doi.org/10.1145/2020723.2020731.
14. Google Cloud. App Engine [Online]. Available: https://cloud.google.com/appengine
15. AWS. (2019). Amazon EC2 [Online]. Available: https://aws.amazon.com/ec2/
16. Salesforce. The number 1 CRM software [Online]. Available: https://salesforce.com/
17. W. -T. Tsai and X. Sun, "SaaS Multi-tenant Application Customization", 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, San Francisco, CA, USA, 2013, pp. 1-12, doi: 10.1109/SOSE.2013.44.
18. F. Chong, "Multi-tenancy and Virtualization", blog, 23 Oct. 2006; http://blogs.msdn.com/b/fred_chong/ archive/2006/10/23/multi-tenancy-and-virtualization. aspx.
19. H. Yaish, M. Goyal, and G. Feuerlicht "A Novel Multi-Tenant Architecture Design for Software as a Service Applications", Proceedings of the 2nd International Conference on Cloud Computing and Services Science, 2012, 82-88.
20. W. -T. Tsai and X. Sun, "SaaS Multi-tenant Application Customization", 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering, San Francisco, CA, USA, 2013, pp. 1-12, doi: 10.1109/SOSE.2013.44.
21. R. Mietzner and F. Leymann, "Generation of BPEL Customization Processes for SaaS Applications from Variability Descriptors", in *Services Computing*, 2008.
22. R. Mietzner, A. Metzger, F. Leymann and K. Pohl, "Variability Modeling to Support Customization and Deployment on Multi-Tenant-Aware Software as a Service Applications", in ICSE PESOS Workshop, 2009.
23. R. Mietzner, F. Leymann and M. P. Papazoglou, "Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-Tenancy Patterns", in 3rd Intl. Conference on Internet and Web Applications and Services, 2008, pp 156-161.
24. B. Sengupta and A. Roychoudhury, "Engineering Multi-Tenant Software-as-a-Service Systems", in Proceedings of the 3rd International Workshop on Principles of Engineering Service-Oriented Systems (PESOS '11) Association for Computing Machinery, New York, NY, USA, 2011, pp 15–21, doi: https://doi.org/10.1145/1985394.1985397
25. K. f. Li, H. y. Wen and Z. j. Yang, "Business logic structure of SaaS-based E-commerce system", 2010 IEEE 17Th International Conference on Industrial Engineering and Engineering Management, Xiamen, China, 2010, pp. 316-319, doi: 10.1109/ICIEEM.2010.5646604.
26. E. Newcomer and G. Lomow, Understanding SOA with web services, Addison Wesley Professional, 2004

27. Y. Z. jun and X. H. xia, Research of online software system development solution based on SaaS, Computer engineering and design, 2009.
28. R. Krebs, C. Momm and S. Kounev, "Architectural Concerns in Multi-Tenant SaaS Applications", in Proceedings of the 2nd International Conference on Cloud Computing and Services Science, 2012.
29. H. Koziolek, "The SPOSAD Architectural Style for Multi-tenant Software Applications", 2011 Ninth Working IEEE/IFIP Conference on Software Architecture, Boulder, CO, USA, 2011, pp. 320-327, doi: 10.1109/WICSA.2011.50.
30. R. Mietzner, F. Leymann and M. P. Papazoglou, "Defining Composite Configurable SaaS Application Packages Using SCA, Variability Descriptors and Multi-tenancy Patterns", 2008 Third International Conference on Internet and Web Applications and Services, Athens, Greece, 2008, pp. 156-161, doi: 10.1109/ICIW.2008.68.
31. R. Mietzner, T. Unger, R. Titze and F. Leymann, "Combining Different Multi-tenancy Patterns in Service-Oriented Applications", 2009 IEEE International Enterprise Distributed Object Computing Conference, Auckland, New Zealand, 2009, pp. 131-140, doi: 10.1109/EDOC.2009.13.
32. W. J. Brown, V. Anderson and Q. Tan, "Multitenancy - Security Risks and Countermeasures", 2012 15th International Conference on Network-Based Information Systems, Melbourne, VIC, Australia, 2012, pp. 7-13, doi: 10.1109/NBiS.2012.142.
33. Z. Feng, B. Bai, B. Zhao and J. Su, "Shrew Attack in Cloud Data Center Networks", 2011 Seventh International Conference on Mobile Ad-hoc and Sensor Networks, Beijing, China, 2011, pp. 441-445, doi: 10.1109/MSN.2011.71.
34. W. T. Tsai and Q. Shao, "Role-Based Access-Control Using Reference Ontology in Clouds", 2011 Tenth International Symposium on Autonomous Decentralized Systems, Tokyo, Japan, 2011, pp. 121-128, doi: 10.1109/ISADS.2011.21.
35. C. Momm and W. Theilmann, "A Combined Workload Planning Approach for Multi-tenant Business Applications", 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops, Munich, Germany, 2011, pp. 255-260, doi: 10.1109/COMPSACW.2011.96.
36. C. P. Bezemer and A. Zaidman(2010) "Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare?", in Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE) (IWPSE-EVOL '10). Association for Computing Machinery, New York, NY, USA, 2010, pp. 88–92, doi: https://doi.org/10.1145/1862372.1862393
37. S. Jansen, S. Brinkkemper, G. Ballintijn and A. van Nieuwland, "Integrated development and maintenance of software products to support efficient updating of customer configurations: a case study in mass market ERP software", 21st IEEE International

Conference on Software Maintenance (ICSM'05), Budapest, Hungary, 2005, pp. 253-262, doi: 10.1109/ICSM.2005.56.

38. R. J. Wieringa, "Design science methodology for information systems and software engineering", 2014.
39. A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in Information Systems Research", *Management Information Systems Quarterly*, vol. 28, no. 1, p. 75, Jan. 2004, doi: 10.2307/25148625.
40. J. Martin, An Information Systems Manifesto. Prentice-Hall, 1984.
41. P. Sapra. (2021, Dec. 26). Kruchten's 4 + 1 views of Software Design [Online]. Available: https://medium.com/the-mighty-programmer/kruchtens-views-of-software-design-e9088398c592