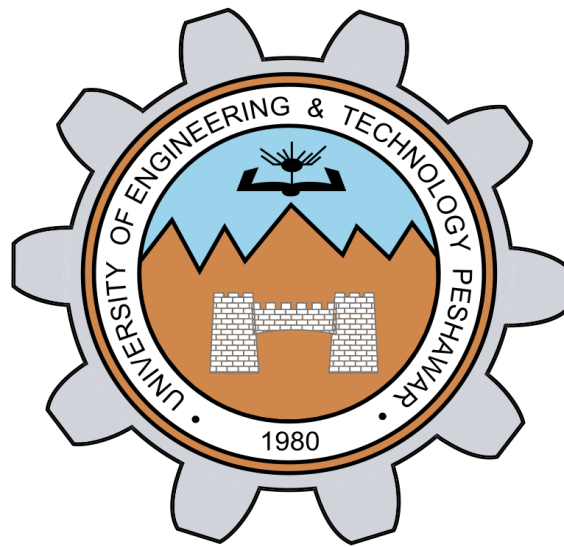


E-Commerce Web Application Micro-Services Based Development



Session 2019-2023

By

Mohammad Danyal

Muhammad Naeem

Muhammad Adeel

Bachelor of Science in Computer Science & Information
Technology

Department of Computer Science
University of Engineering and Technology Peshawar, Jallozai
Campus, Pakistan
July, 2023

E-Commerce Web Application Microservices Based Development

By

Mohammad Danyal

Muhammad Naeem

Muhammad Adeel

CERTIFICATE

A THESIS SUBMITTED IN THE PARTIAL FULFILMENT OF THE
REQUIREMENTS FOR THE DEGREE OF BACHELOR OF SCIENCE IN
COMPUTER SCIENCE & INFORMATION TECHNOLOGY

We accept this dissertation as conforming to the required standards

Supervised by:

Dr. Aamir Saeed

Approved By HOD:

Chairman Dr. Laeeq Ahmed

Department of Computer Science
University of Engineering and Technology Peshawar, Jallozai
Campus, Pakistan
July, 2023

Dedication

We dedicate this project to our beloved parents who are dearer to us and our teachers who are like candlelight on the way to our destiny.

Declaration

We hereby declare that we are the sole authors of this thesis. The work submitted in this thesis is the result of our research except where otherwise stated.

Acknowledgment

First of all, we are thankful to the ALMIGHTY ALLAH who has given us the strength, patience, and knowledge for choosing such an impacting project.

We extend my thanks to the faculty and staff of University of Engineering & Technology Peshawar for providing the necessary resources and facilities that facilitated this study. The access to research materials, computing resources, and the library greatly contributed to the success of this project.

We would like to thank my colleagues and friends for their assistance, discussions, and feedback during the development of the microservice architecture. Their input and collaboration were essential in overcoming challenges and refining the implementation.

Lastly, we would especially like to acknowledge the efforts of our supervisor **Dr. Aamir Saeed**. He kept us motivated throughout this project. His guidance and experience was very helpful in the completion of this project.

This thesis would not have been possible without the collective efforts and contributions of all those mentioned above. Their involvement and support have played a vital role in shaping this research endeavor, and we are sincerely thankful to each and every one of them.

Abstract

This thesis focuses on the development of a microservice architecture for an e-commerce web application. The objective is to design and implement a scalable and modular system that can handle the complex requirements of an e-commerce platform. The microservice approach enables the application to be broken down into smaller, independent services that can be developed, deployed, and scaled independently. This architecture offers benefits such as improved flexibility, scalability, and maintainability.

The thesis begins by exploring the fundamental concepts of microservice architecture and its advantages over monolithic architectures. It then delves into the analysis and design phase, where the system requirements are identified, and the microservices are designed to fulfill those requirements. The implementation phase covers the development of each microservice, including technologies, frameworks, and best practices employed. The deployment and testing strategies are also discussed, ensuring the system's reliability and performance.

Throughout the thesis, attention is given to key aspects such as service communication, data consistency, fault tolerance, and security. Various tools and technologies are utilized, including containerization with Docker, orchestration with Kubernetes, and cloud infrastructure with services like AWS. Additionally, a recommendation system and a payment gateway (Stripe) are integrated to enhance the e-commerce platform's functionality and user experience.

The developed microservice architecture provides a robust foundation for the e-commerce web application, enabling scalability and flexibility as the business grows. The results of the implementation and testing phases demonstrate the feasibility and effectiveness of the proposed architecture. The thesis concludes with a discussion of the achieved goals, limitations, and future directions for further improvement and expansion of the e-commerce web application.

Table of Contents

Dedication.....	4
Declaration	5
Acknowledgment	6
Abstract	7
CHAPTER 01	1
Introduction	1
CHAPTER 02	6
Design and Implementation	6
• Customer.....	8
• Products.....	8
• State Tax	9
• Shopping Cart Items	9
• Order Details	9
• Shipping Type	10
• Credit Card Details.....	10
• Purchase History.....	10
2.3.3 Data Flow Diagram (DFD):.....	12
2.3.4 The Shopping Cart Application.....	16
2.3.5 Registration:	18
2.3.6 User Interface Design	19
2.3.7 Shopping Cart	20
2.3.8 Place an Order.....	21
2.3.9 Check Out.....	23
2.4 Product Design	25
CHAPTER 03	27
Software Requirements Specification	27
CHAPTER 04	29
Microservices Deployment	29
4.1 Microservices Deployment with Docker.....	29
.....	31

.....	32
Docker Container:	32
.....	32
Docker Images:.....	33
.....	33
Docker Volumes:	33
.....	33
4.2 Kubernetes Deployment:.....	34
4.3 Cloud Infrastructure with AWS	36
Docker on AWS EC2:.....	37
• Launch EC2 Instances:	37
• Install Docker:.....	37
• Manage Docker Daemon:	37
• Add Users to the Docker Group:	37
• Test Docker Installation:	37
Kubernetes on AWS EC2:	37
• Install kubectl:	37
• Set Up a Kubernetes Cluster:	38
• Kubeadm:.....	38
• Kops:	38
• EKS (Elastic Kubernetes Service):	38
• Configure kubectl:.....	38
• Test Kubernetes Installation:	38
• Create Docker Image:	38
• Push Docker Image to a Registry:	38
• Kubernetes Deployment:	38
• Kubernetes Service:	39
• Ingress Controller:	39
• Scale and Manage:	39
 CHAPTER 05	 41
Recommendation & Stripe Payment System	41
 CHAPTER 06	 45
Methodology	45
 CHAPTER 07	 47
Evaluation and Results	47
 CHAPTER 08	 49
Software Testing	49

8.11 Microservices Deployment Testing	52
CHAPTER 09	54
Conclusion & Future Work	54
9.1 Conclusion of Chapter 01.....	54
9.2 Conclusion of Chapter 02.....	54
9.3 Conclusion of Chapter 03.....	55
9.4 Conclusion of Chapter 04.....	56
9.5 Conclusion of Chapter 05.....	57
9.6 Conclusion of Chapter 06.....	57
9.7 Conclusion of Chapter 07.....	58
9.8 Conclusion of Chapter 08.....	59
9.9 Future Direction:	59
9.9.1 AI and Personalization:	59
9.9.2 Voice Commerce:	60
9.9.3 Social Commerce:	60
9.9.4 Mobile Commerce (M-Commerce):	60
9.9.5 Fast and Flexible Delivery Options:	60
9.9.6 Data Privacy and Security:	60
9.9.7 Our Team Future Work:.....	61
• Continuous Learning:	61
• Certifications:	61
• Networking and Professional Connections:.....	61
• Experiment with Cloud Technologies:.....	61
• Soft Skills Development:	61
• Seek Internships and Job Opportunities Abroad:	61

CHAPTER 01

Introduction

This thesis aims to design and implement a microservices-based e-commerce web application. The project will utilize front-end development with React, back-end development with Spring Boot, microservices deployment with Docker and Kubernetes, cloud infrastructure with Amazon Web Services (AWS), and recommendation systems with Algolia.

1.1 Background

In this section, the thesis provides an overview of the background information related to our e-commerce websites and their significance. It explains the increasing popularity of online shopping & the need for comprehensive e-commerce platforms that cater to various customer segments.

Our e-commerce websites have become a vital part of modern-day business, and it is essential to develop an efficient and reliable platform to cater to the increasing demands of online commerce. Microservices architecture offers a scalable and flexible solution for building complex applications.

1.2 Problem Statement

Traditional monolithic our e-commerce applications face several challenges such as scalability, fault-tolerance, and maintainability. These challenges increase as the application grows. Microservices architecture addresses these challenges and provides an elegant solution for building scalable and reliable applications.

The problem statement highlights the challenges and gaps in the existing e-commerce websites. It identifies the need for a comprehensive platform that offers a diverse range of products for different customer segments, Along with the requirement for modern technologies and frameworks to enhance the user experience and optimize platform performance.

1.3 Objectives

The objective of this thesis is to design and implement our e-commerce web application using microservices architecture, which will be scalable, fault-tolerant, and efficient. The application will have a responsive and intuitive user interface, and it will incorporate a recommendation system using Algolia.

The main objective is to explore the implementation of a comprehensive of our e-commerce website. It also emphasizes the intention to analyze the effectiveness and efficiency of modern technologies, such as React, Spring Boot, Docker, Kubernetes, AWS, and Algolia, in enhancing the user experience and optimizing the performance of the our e-commerce platform.

1.4 Significance of the Study

The significance of the study highlights the potential contributions of the research. It explains how the findings of the thesis can benefit the e-commerce industry by providing insights into the implementation of a comprehensive e-commerce platform and evaluating the effectiveness of modern technologies.

This study aims to contribute to the development of microservices-based e-commerce applications. The project will provide a reference implementation for building scalable and reliable e-commerce applications using microservices architecture.

1.5 Scope and Limitations

This section defines the scope of the study, including the specific aspects of E-commerce that will be covered and the technologies that will be analyzed. It also acknowledges the limitations of the study, such as time constraints, resource limitations, and potential challenges in implementing and evaluating the platform. The scope of this study includes the design and implementation of a microservices-based e-commerce web application.

The limitations are that the project will focus on using React, Spring Boot, Docker, Kubernetes, and AWS for the implementation.

1.6 Thesis Structure

The thesis structure section provides an overview of the organization of the thesis. It outlines the different chapters and their respective contents, guiding the reader through the flow of the thesis.

The thesis consists of Nine chapters.

1. Introduction
2. Design and Implementation
3. Software Requirements Specification
4. Microservices Deployment
5. Recommendation & Stripe Payment System
6. Methodology
7. Software Testing
8. Evaluation and Results
9. Conclusion & Future Work

1.7 Stakeholders and their Interests

The project Team is responsible for providing the project requirements and approving the project budget. The project manager is responsible for overseeing the project and ensuring that it is completed on time and within budget.

The development team is responsible for developing the website. The marketing team is responsible for promoting the website and generating leads.

The customer support team is responsible for providing support to customers who have problems with the website.

1.8 Project Activities

The project activities include the following:

- Requirements gathering
- System design
- Development
- Testing
- Deployment
- Marketing
- Customer support

1.9 Requirement Gathering and Analysis:

Identify and document the specific functional and non-functional requirements of our e-commerce website. Details explanation below part.

1.10 Design and Wire framing:

We create the visual design, user interface, and wireframes for the website, ensuring an intuitive and user-friendly layout.

1.11 Testing and Quality Assurance:

We also Conduct comprehensive testing, including functional, usability, performance, and security testing, to ensure a robust and reliable website.

1.12 Deployment and Infrastructure Setup:

We configure the necessary infrastructure using Docker, Kubernetes, and AWS to ensure scalability and high availability.

1.13 Payment Gateway Integration:

For Payment we integrate the chosen payment gateway (Stripe) to enable secure and seamless payment transactions.

1.14 Mobile Responsiveness:

Highly users of mobile, so we optimize the website's design and functionality to ensure a seamless user experience across different devices and screen sizes.

Responsive Web Design is method for designing web pages so that they can be loaded properly on various different devices and screen sizes. A responsive web application is often the combination of flexible grids and images. These responsive design patterns support the page elements sizing to work on different screens.

Responsive Web Design can adapt to any device such as a Desktop, a Laptop, a Tablet, or a mobile phone because page sizing is rendered in relative units like percentages, instead of pixels or points.

CHAPTER 02

Design and Implementation

2.1 Architecture Design

This section will describe the overall architecture of the microservices-based e-commerce web application.

The design and implementation chapter begins with the architecture design of the e-commerce website. It presents the overall structure, components, and interactions between different modules or services. It may include diagrams, flowcharts, or other visual representations to illustrate the design.

2.2 Front-end Development with React

This section will describe the implementation of the front-end using React and state management techniques used.

The implementation of the front-end using React. It explains how React components are designed and developed to create an intuitive and responsive user interface. It may also discuss any additional libraries or tools used alongside React.

2.3 Back-end Development with Spring Boot

This section will describe the implementation of the back-end using Spring Boot and MY SQL and the API developed for the microservices.

The implementation of the back-end using Spring Boot and My SQL. It covers the design and development of the server-side logic, database integration, and API endpoints required for the e-commerce platform.

2.3.1 MySQL Database:

In this project, MySQL is used as the backend database. MySQL is an open-source database management system. The features of MySQL are given below:

- MySQL is a relational database management system. A relational database stores Information in different tables, rather than in one giant table. These tables can be referenced to each other, to access and maintain data easily.
- MySQL is open source database system. The database software can be used and Modify by anyone according to their needs.
- It is fast, reliable and easy to use. To improve the performance, MySQL is multi- threaded database engine. A multithreaded application performs many tasks at the same time as if multiple instances of that application were running simultaneously.

In being multithreaded MySQL has many advantages. A separate thread handles each incoming connection with an extra thread that is always running to manage the connections. Multiple clients can perform read operations simultaneously, but while writing, only hold up another client that needs access to the data being updated. Even though the threads share the same process space, they execute individually and because of this separation, multiprocessor machines can spread the thread across many CPUs as long as the host operating system supports multiple CPUs. Multithreading is the key feature to support MySQL's performance design goals. It is the core feature around which MySQL is built.

MySQL database is connected to ASP.NET using an ODBC driver. Open Database Connectivity (ODBC) is a widely accepted application-programming interface (API) for database access. The ODBC driver is a library that implements the functions supported by ODBC API. It processes ODBC function Calls, submits SQL requests to MySQL server, and returns results back to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by MySQL.

2.3.2 Database Design:

In this section, the basic structure of the tables composing the database for the project are shown along with information about primary and foreign keys.

- **Customer**

SNO	NAME	TYPE	DESCRIPTION
1	User_ID	Varchar	Primary key for Customer identification
2	Password	Varchar	Security for Customer
3	First Name	Varchar	
4	Last Name	Varchar	
5	Address	Varchar	
6	City	Varchar	
7	Zip	Integer	
8	State	Varchar	
9	Email Address	Varchar	
10	Phone Number	Varchar	

- **Products**

SNO	NAME	TYPE	DESCRIPTION
1	<u>Inventory ID</u>	Integer	Primary key for Inventory Identification,
2	Products Name	Varchar	
3	Author	Varchar	
5	Nr Products	Integer	
6	Price	Double	

- **State Tax**

SNO	NAME	TYPE	DESCRIPTION
1	<u>State Name</u>	Varchar	Primary key for State Identification
2	Sales Tax Rate	Double	Sales tax for each state

- **Shopping Cart Items**

SNO	NAME	TYPE	DESCRIPTION
1	<u>Shopping Cart ID</u>	Integer	Primary key for Shopping Cart Identification
2	Inventory ID	Varchar	Foreign key to Inventory
3	Price	Double	
4	Date	Date	
5	User ID	Varchar	Foreign key to Customer
6	Quantity	Integer	

- **Order Details**

SNO	NAME	TYPE	DESCRIPTION
1	Order ID	Integer	Primary key for Order identification
1	User ID	Char	Foreign key to Customer
2	Receiver's Name	Char	If order is to be sent to other address rather than to the customer, we need that address
3	Address	Char	
4	City	Char	
5	Zip	Integer	
6	State	Char	Foreign key to State Tax
7	Type of Shipping	Char	Foreign key to Shipping Type
8	Date of Purchase	Date	

- **Shipping Type**

SNO	NAME	TYPE	DESCRIPTION
1	<u>Type of Shipping</u>	Varchar	Primary key to define type of shipping
2	Price	Double	
3	Approximate days for delivery	Integer	

- **Credit Card Details**

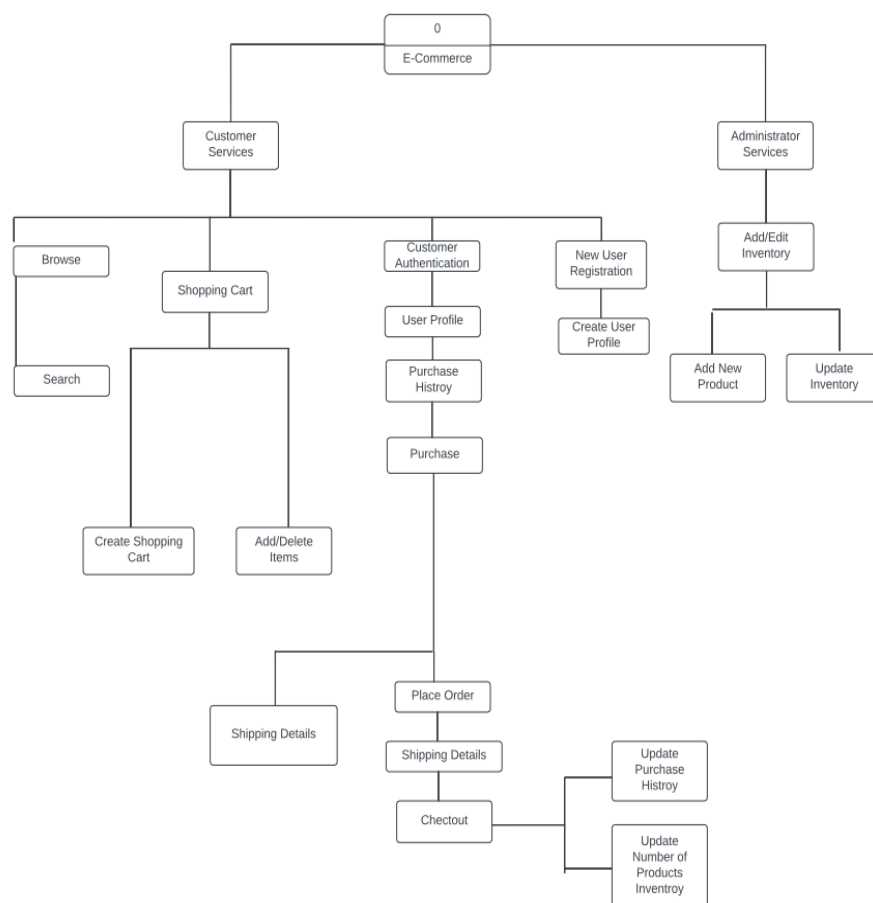
SNO	NAME	TYPE	DESCRIPTION
1	<u>Credit Username</u>	Varchar	Primary key for Customer Identification
2	Credit Card Number	Varchar	
3	Card Type	Varchar	Master Card, Visa, Discover
4	CVV Number	Integer	Number present on the back of the card for extra security
5	Expiry Date	Date	
6	User ID	Varchar	Foreign key to Customer

- **Purchase History**

SNO	NAME	TYPE	DESCRIPTION
1	<u>User ID</u>	Varchar	Primary key for Customer Identification
2	Inventory ID	Varchar	Product purchased by the user
3	Date of Purchase	Date	
4	Order ID	Integer	Foreign key to Order details
5	Quantity	Integer	
6	Price	Double	

2.3.2 Functional Decomposition Diagram

A decomposition diagram shows a top-down functional decomposition of a system and exposes the system's structure. The objective of the Functional Decomposition is to break down a system step by step, beginning with the main function of a system and continuing with the interim levels down to the level of elementary functions. The diagram is the starting point for more detailed process diagrams, such as data flow diagrams (DFD). Figure 2.3.2 shows the Functional Decomposition Diagram for this project.

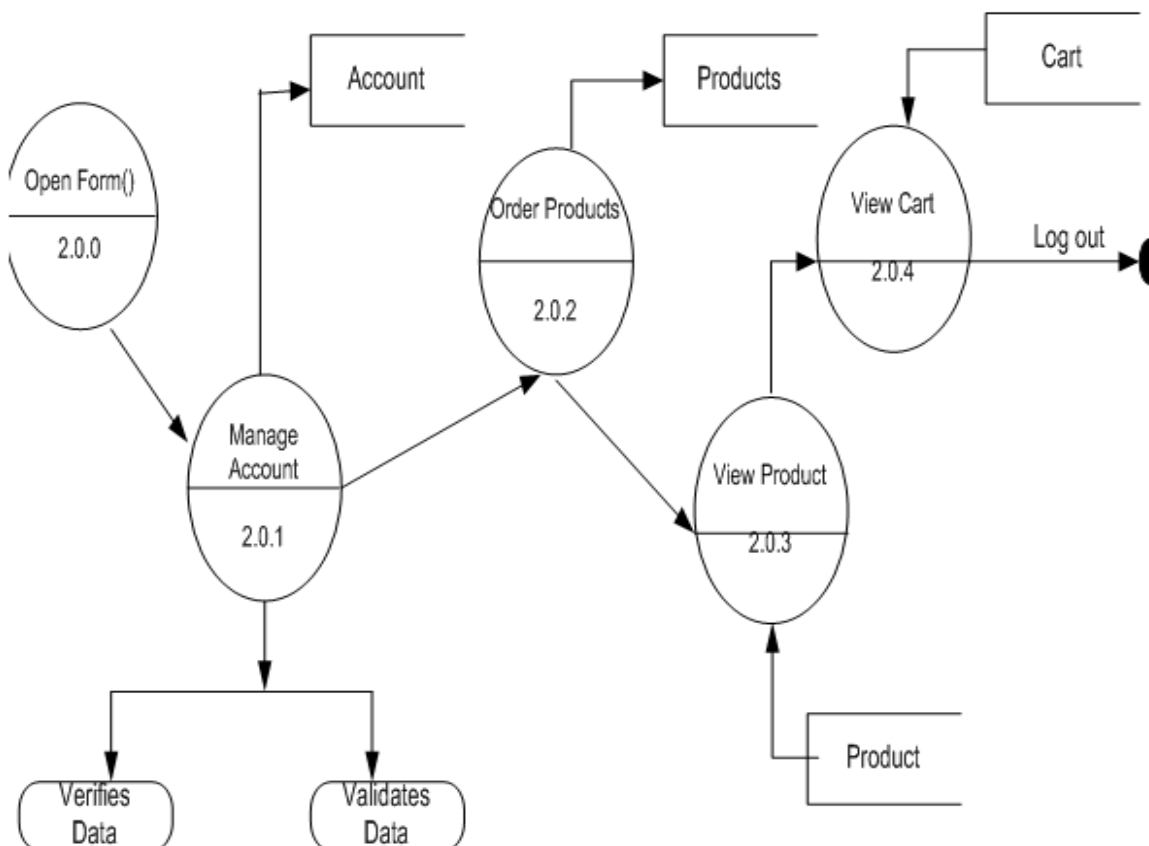
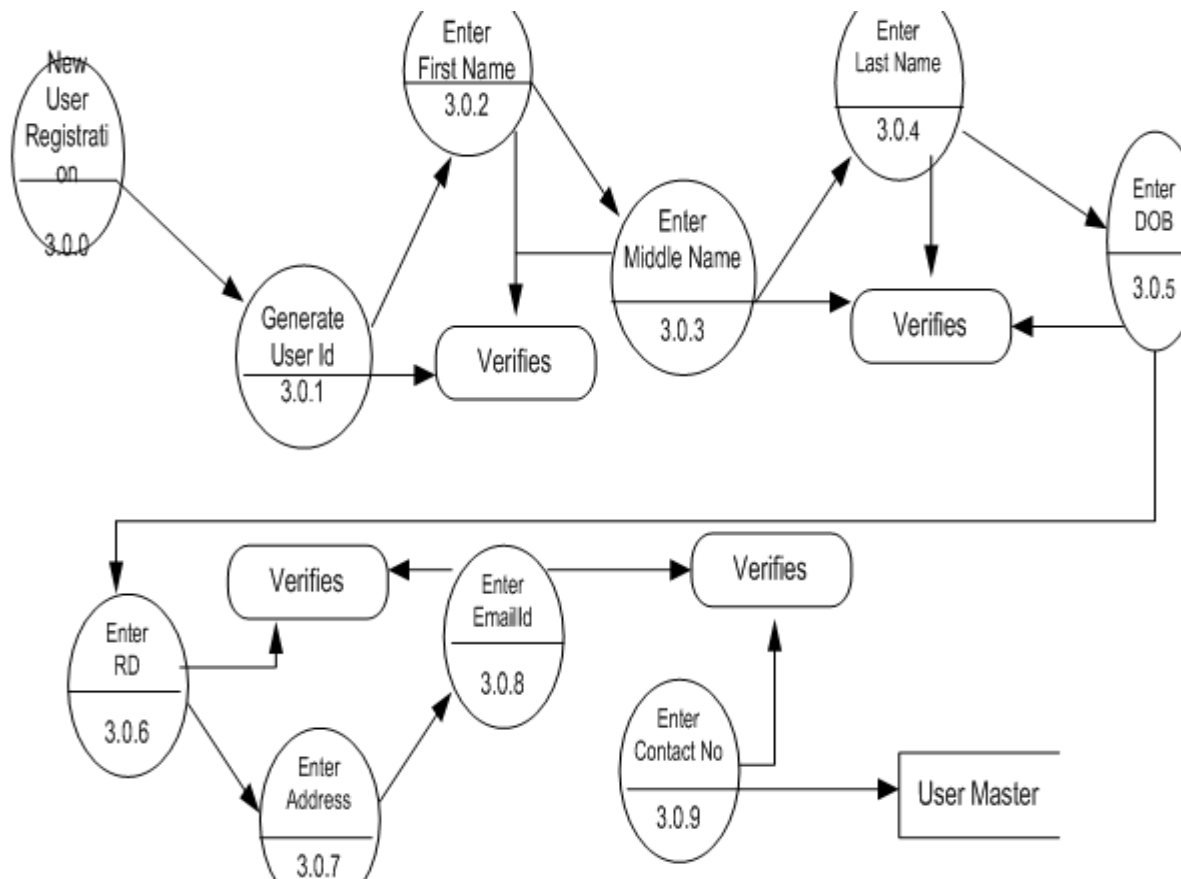


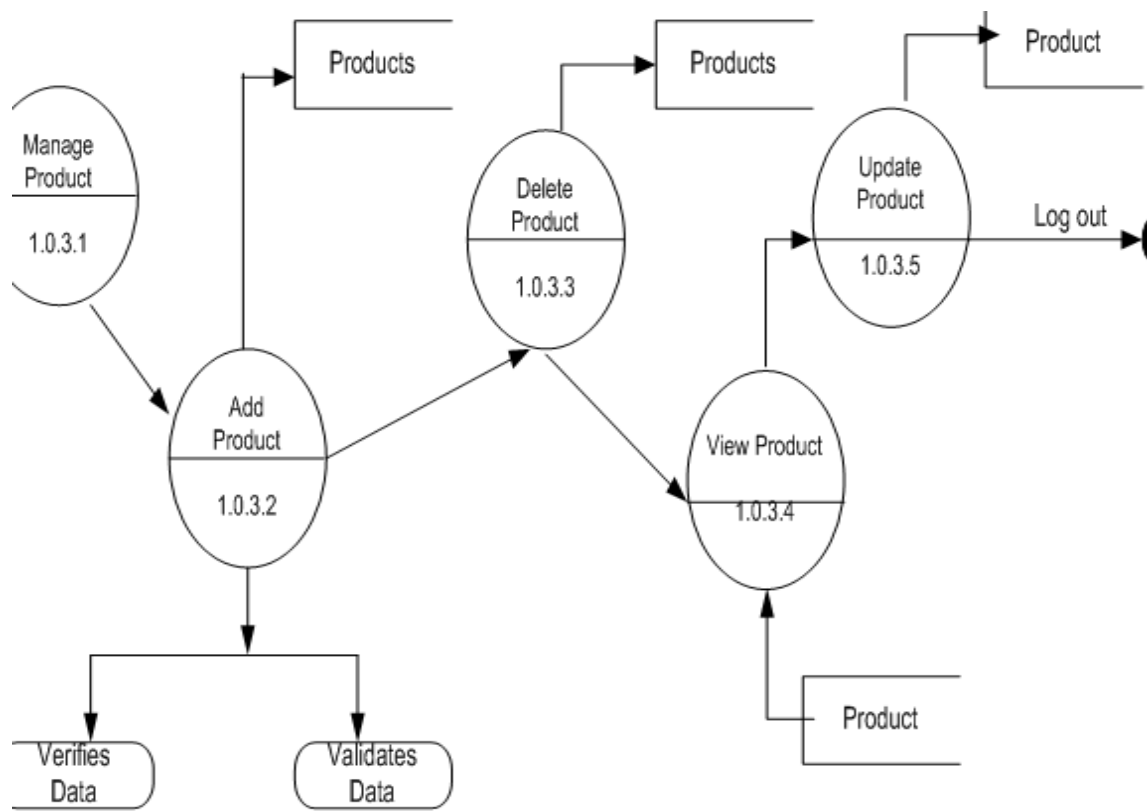
2.3.3 Data Flow Diagram (DFD):

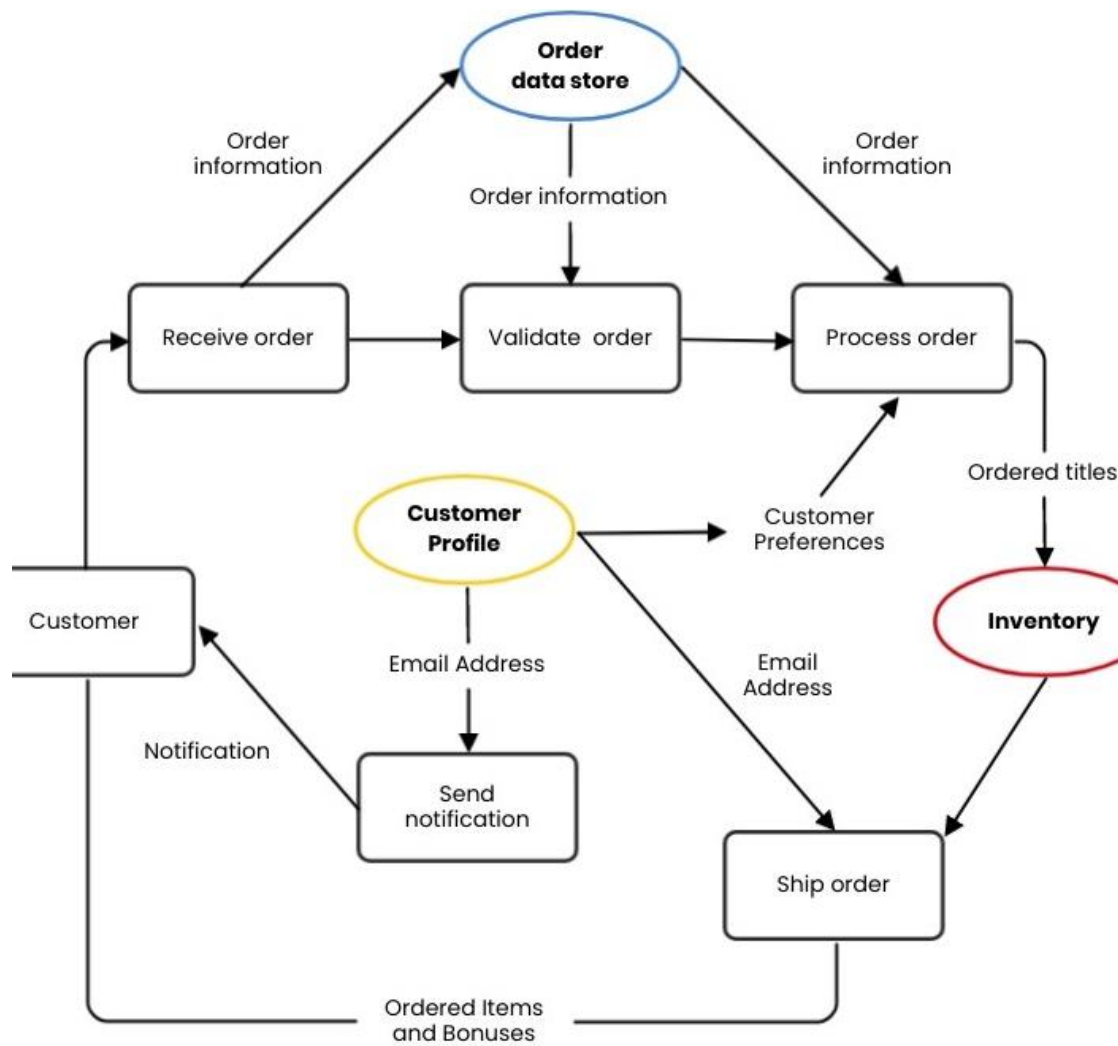
Data Flow Diagrams show the flow of data from external entities into the system, and from one process to another within the system. There are four symbols for drawing a DFD:

- Rectangles representing *external entities*, which are sources or destinations of data.
- Ellipses representing *processes*, which take data as input, validate and process it and output it.
- Arrows representing the *data flows*, which can either, be electronic data or physical items.
- Open-ended rectangles or a Disk symbol representing *data stores*, including electronic stores such as databases or XML files and physical stores such as filing cabinets or stacks of paper.

Figures the Data Flow Diagrams for the current system. Each process within the system is first shown as a Context Level DFD and later as a Detailed DFD. The Context Level DFD provides a conceptual view of the process and its surrounding input, output and data stores. The Detailed DFD provides a more detailed and comprehensive view of the interaction among the sub-processes within the system.







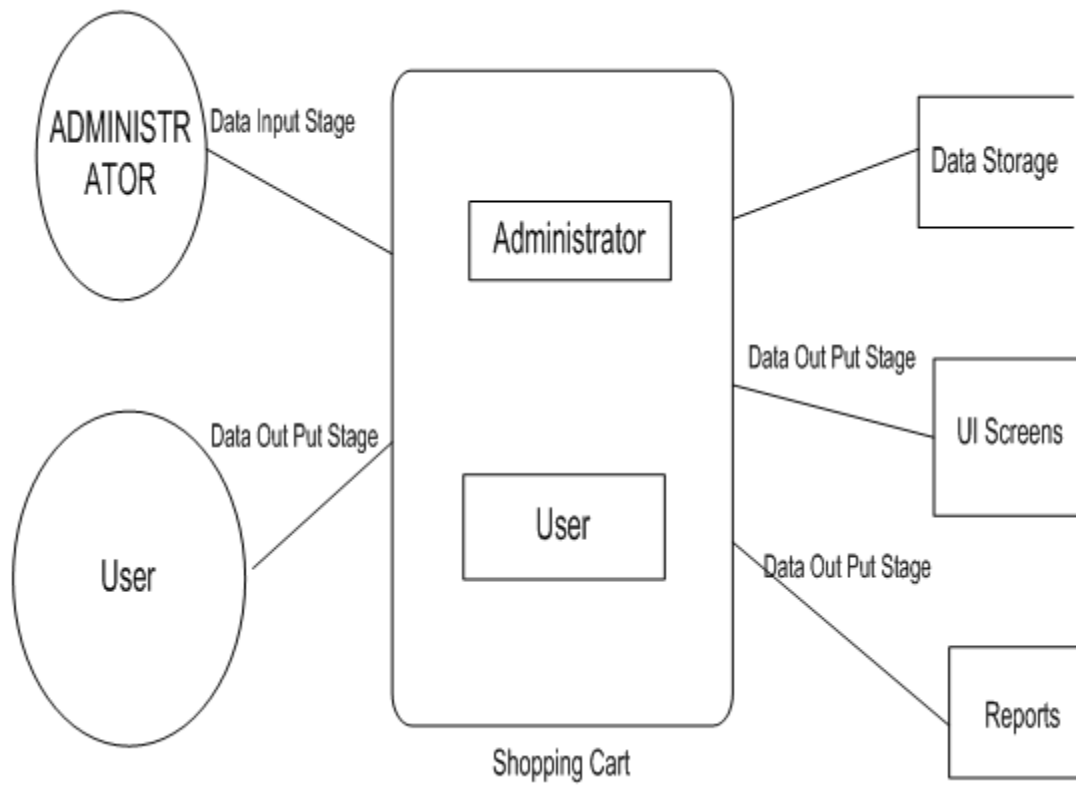
2.3.4 The Shopping Cart Application

The objective of this application is to provide the user an online website where they can buy products from the comfort of their home. A shopping cart is used for the purpose. The user can select the desired items, place them in the shopping cart and purchase them using a Credit Card. The user's order will be shipped according to the type of shipping selected at the time of placing the order.

Website consists of the following web pages:

1. Add Products
2. Product Details
3. Change Password
4. Check Out
5. Final Order
6. Forgot Password
7. Login
8. Log Out/Off
9. Menu
10. Order
11. Purchase History
12. Registration
13. Search
14. Shopping Cart
15. User Details

Below figures show some screenshots taken from running the application. All the functionalities are explained accordingly.

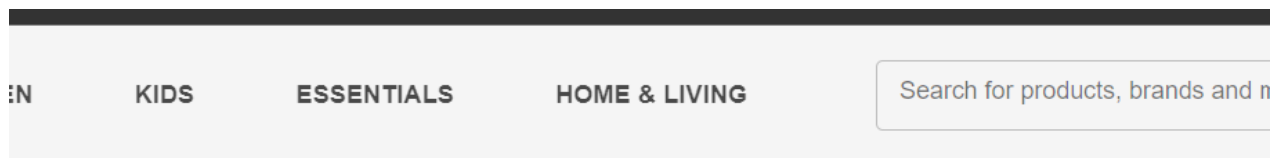


2.3.5 Registration:

A new user can register on the site by clicking on the registration button on the menu at the top of the page, as shown in Below Figure:

The figure shows a registration form titled "Sign Up". At the top, there is a navigation bar with links: MEN, KIDS, ESSENTIALS, and HOME & LIVING. To the right of these links is a search bar with the placeholder text "Search for products, brands and". Below the navigation bar, the "Sign Up" title is centered. The form consists of several input fields, each with a label and a placeholder icon: "First Name" (person icon), "Last Name" (person icon), "Username" (person icon), "Email" (envelope icon), "Password" (lock icon), and "Confirm Password" (lock icon). Below the "Confirm Password" field is a checkbox with the text "I agree to the Terms and Conditions.". At the bottom of the form is a large teal button labeled "SIGN UP".

The required field for successful registration on the site. If the value is not entered, an appropriate message is displayed. If a user with same User Name already exists, the message is displayed. Clicking on Reset will clear all the fields and Submit will submit the information for registration. Upon successful completion, the user is directed to the products page.



Sign In

Username

Password

SIGN IN

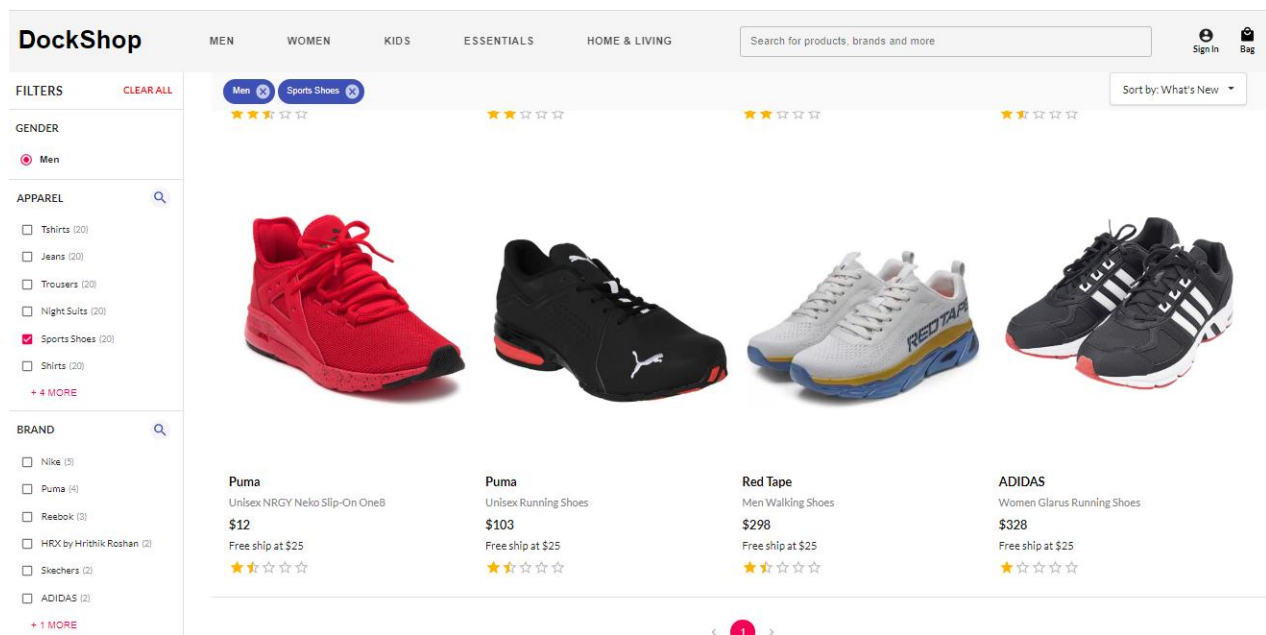
OR

 Google

Not on DockShop yet? [Sign up](#)

2.3.6 User Interface Design

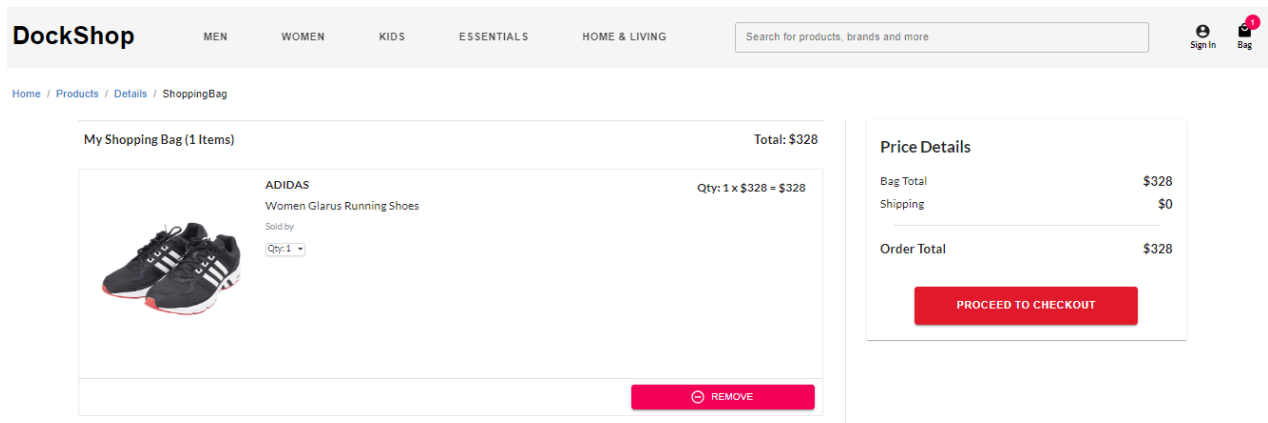
Before implementing the actual design of the project, a user interface designs were constructed to visualize the user interaction with the system as they browse for products, create a shopping cart and purchase items. The user interface design will closely follow our Functional Decomposition Diagram. Figures show the initial designs of the web pages.



2.3.7 Shopping Cart

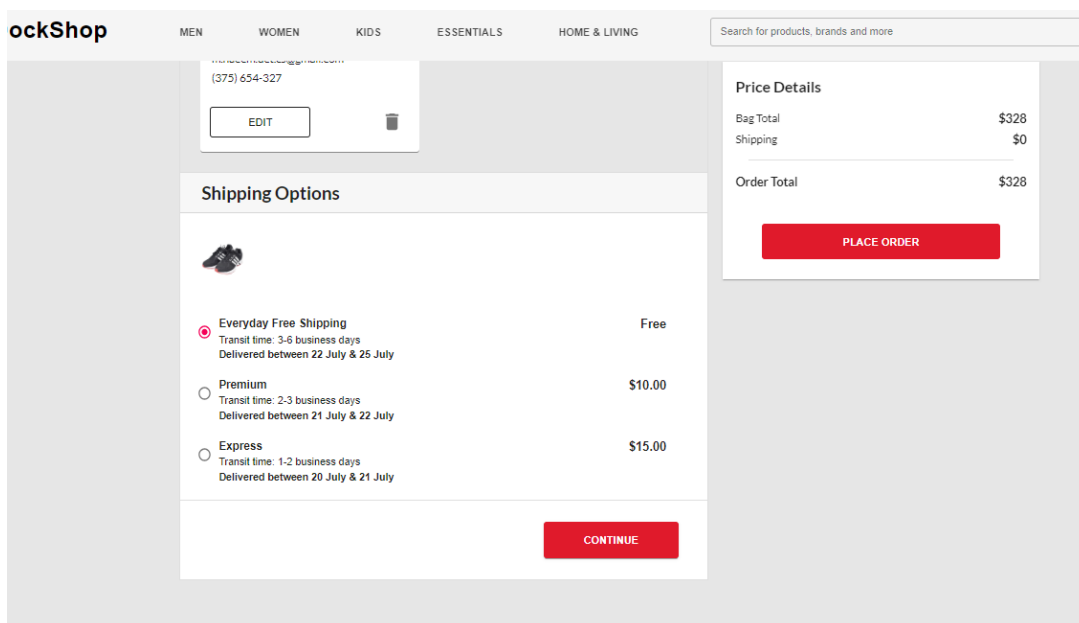
When “Add to Cart” is clicked for any product, it is added to the shopping cart illustrated, If that particular product is already present in the shopping cart, the quantity is increased by 1 and the price is changed accordingly; if not, a new entry is made into the table. All the information in the shopping cart is stored in “shopping cart items” table. Adding item into the shopping cart does not decrease the quantity of item in the items table. It is decreased only after an order is placed for the product.

So, placing the item in the shopping cart does not guarantee the availability of the item at the time of placing the order.



2.3.8 Place an Order

When “Place an Order” button is clicked which is located on the bottom of the shopping cart, the application will ask the user to login if he has not already done so.



If the user is placing an order with the web site for the first time, they will be asked to enter the credit card details as shown in the above figure; if not, only the Card Verification Value (CVV) number of the credit card is asked for verification.

At this point, the user can check the shipping address box if shipping address is same as billing address, otherwise the user has to enter the new shipping address.

DockShop MEN WOMEN KIDS ESSENTIALS HOME & LIVING Search for products, brands and more Sign In Bag

Shipping Address

First Name
Required

Last Name

Email

Address Line 1

Address Line 2 (optional)
Apt, Suite, Bldg, Floor, etc.

Zip Code State

City

Phone Number
123-123-1234

CONTINUE

Price Details

Bag Total	\$328
Shipping	\$0
Order Total	\$328

PLACE ORDER


If the check box provided is checked, the shipping address is obtained from the *Customer* table. The user also has to select the desired type of Shipping for the order. When all the information is entered, the user can **“Proceed to the Checkout”**.

DockShop MEN WOMEN KIDS ESSENTIALS HOME & LIVING Search for products, brands and more € Sign

Home / Products / Details / ShoppingBag

My Shopping Bag (1 Items)

Total: \$328

	ADIDAS Women Glarus Running Shoes Sold by Qty: 1	Qty: 1 x \$328 = \$328
---	--	------------------------

REMOVE

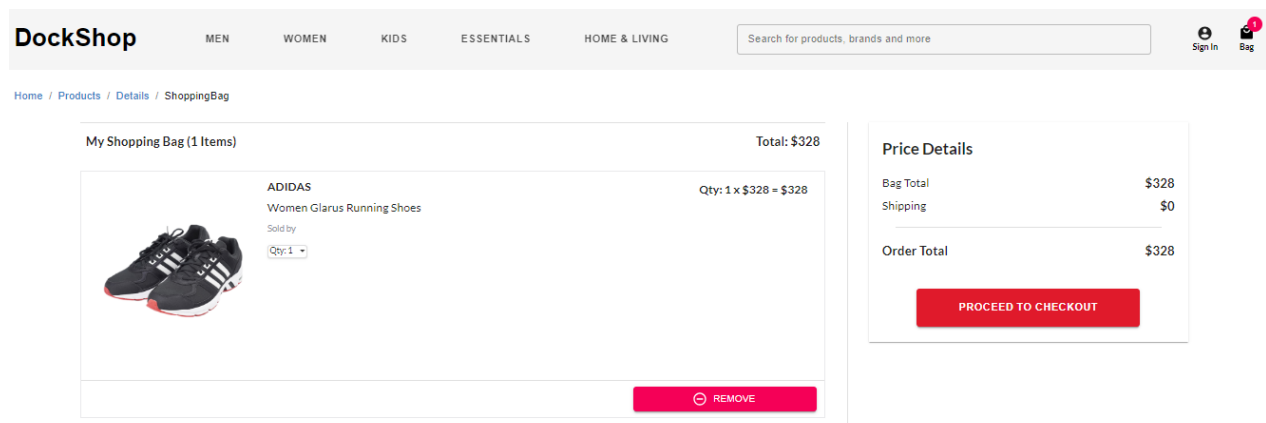
Price Details

Bag Total	\$328
Shipping	\$0
Order Total	\$328

PROCEED TO CHECKOUT

2.3.9 Check Out

Before placing the final order, the user is shown the total price of the order, which includes total price of books selected, shipping rate and state tax as illustrated in Figure. If the user is not satisfied with the order, the order can be cancelled at that point. The information in the shopping cart remains intact, so the user can go back to it and make any changes if necessary. When the “Place Order” button is clicked, the order is placed and the following screen appears which informs the user about the approximate number of days in which the order will be delivered.



Once the order is placed, the quantity of the books is reduced in the *Books* table. The shopping cart for the user is cleared and an appropriate message is displayed, as shown in the figure below.

Payment Successful. Thank You For Shopping at DockShop.

Your order is placed successfully. Your order id is 1689779068041.

Receipt: [Order-Receipt](#)

Delivery Address: Muhammad Naeem
Bang tehsil mastuj district chitral kpk Pakistan
Chitral, undefined - 22323
Mobile - 375654327
Email - m.naeem.uet.cs@gmail.com

Payment Details: VISA ending in 4242
Exp: 12/2034

Paid Amount: \$328

Delivery Details: Everyday Free Shipping
Delivered between 22 July & 25 July



Women Glarus Running Shoes
ADIDAS
Qty: 1 X 328 = 328

2.4 Product Design

- **Design Requirements:**

Identify the key design requirements for each product category. This may include factors such as color schemes, typography, imagery styles, and visual themes. For example, men's products might lean towards bold and minimalistic designs, while kids' products may require vibrant and playful elements.

- **Gather Design Assets:**

We collect relevant design assets that align with our chosen product categories. These assets may include stock images, illustrations, icons, and patterns. Canva provides a vast library of assets to choose from, or we can upload our own.

- **Canva Account and Select a Template:**

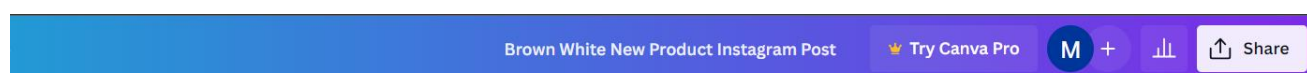
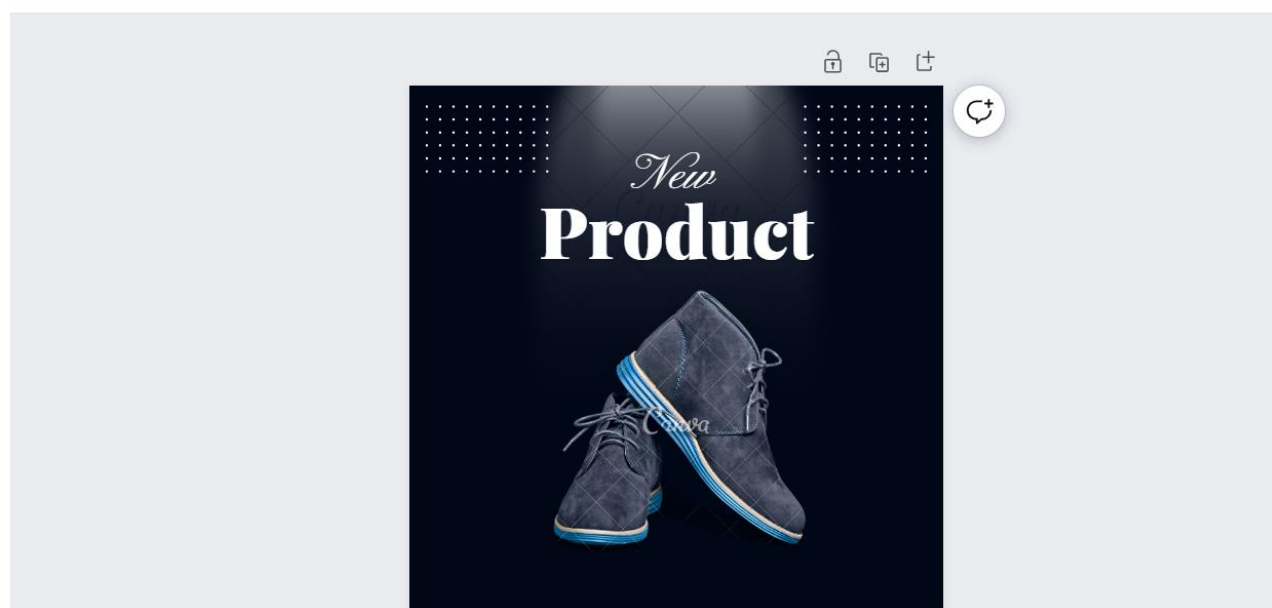
Canva offers numerous pre-designed templates that cater to different purposes, including e-commerce and product promotion.

- **Customize the Template:**

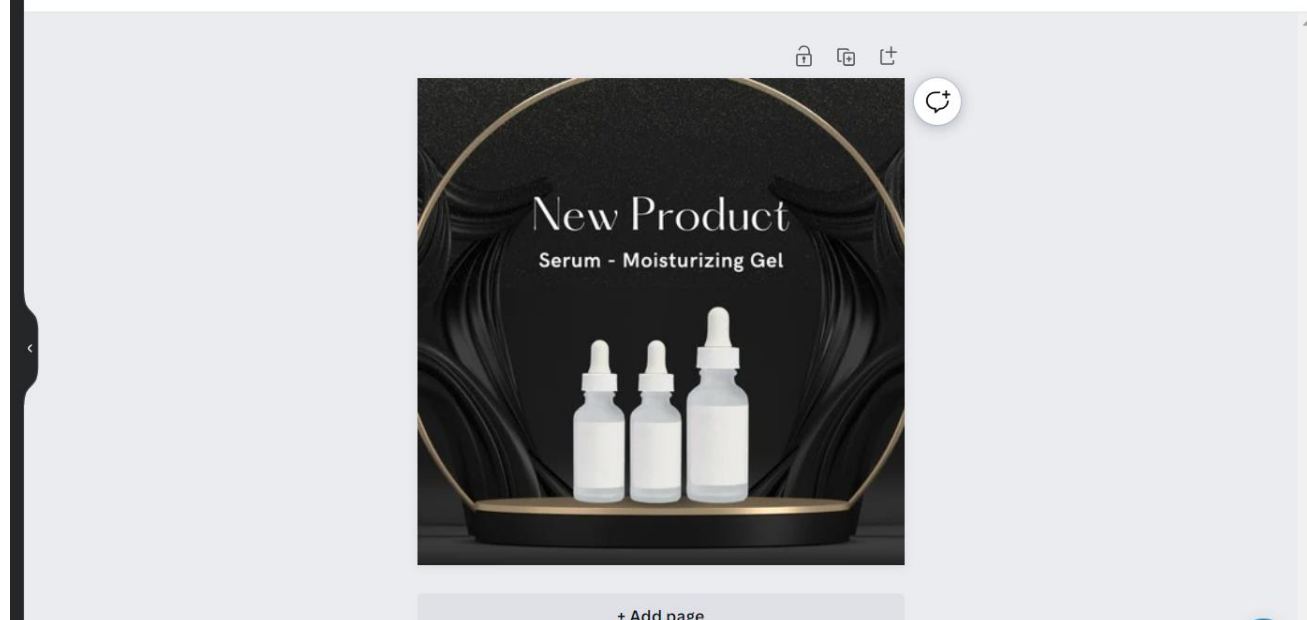
Modify the template to align with the design requirements of each product category. Edit the colors, typography, and layout to match the desired aesthetic. Replace placeholder images and text with relevant visuals and product information. Canva provides an intuitive drag-and-drop interface to make customization easy.



osition



Position



CHAPTER 03

Software Requirements Specification

3.1 Functional Requirements

The e-commerce web application will provide the following functional requirements:

- Users will be able to browse a catalog of products and services.
- Users will be able to add products and services to their shopping cart.
- Users will be able to checkout and complete their purchase.
- Users will be able to track the status of their orders.
- Users will be able to manage their account information.

3.2 Non-Functional Requirements

The e-commerce web application will meet the following non-functional requirements:

- The application must be available 24/7.
- The application must be scalable to handle a large number of users.
- The application must be secure.
- The application must be accessible to users with disabilities.

3.3 Assumptions and Dependencies

The following assumptions and dependencies have been made for the e-commerce web application:

- The application will be developed using the Java programming language.
- The application will be deployed to a cloud-based environment.
- The application will use a microservices architecture

CHAPTER 04

Microservices Deployment

4.1 Microservices Deployment with Docker

This section will discuss Docker and Kubernetes and their usage for microservices deployment and orchestration.

The section on microservices deployment examines the role of Docker and Kubernetes in deploying microservices architecture. It explains how these technologies can enhance scalability, reliability, and ease of deployment in an e-commerce context.

The steps to dockerize an e-commerce web app are explain below:

4.1.1 Docker Installation:

First of all we ensure that we have Docker installed on our development machine. We can download and install Docker from the official website.

4.1.2 Dockerize Web App Code:

We create a Dockerfile in the root directory of our e-commerce web app. The Dockerfile defines the steps to build a Docker image for our application.

4.1.3 Specify Base Image:

We choose a suitable base image for our application. A common choice is the official image, depending on the technology stack of our web app.

4.1.4 Copy Files:

Copy of necessary files and directories from our web app into the Docker image. Use Dockerfile's COPY or ADD command to get it.

4.1.5 Install Dependencies:

We are Install the required dependencies for our web app. For projects, npm install, and for Python projects, pip install, etc.

4.1.6 Configure Environment:

For configuration to set environment variables in the Dockerfile or using Docker Compose to specify any configurations required by our e-commerce app, such as database connection details, API keys, etc.

4.1.7 Expose Ports:

Specify the ports our e-commerce app listens to within the Docker container using the EXPOSE instruction in the Dockerfile.

4.1.7 Build Docker Image:

The docker build command to build the Docker image from the Dockerfile. For example: docker build -t my-ecommerce-app: latest.

4.1.8 Test Docker Image:

For testing we run a container from the built image locally to ensure everything works as expected. The docker run command. For example: docker run -p 80:3000 my-ecommerce-app: latest

4.1.9 Containerize Backend Services:

Our e-commerce app relies on backend services (e.g., database, caching), consider dockerizing those as well. We use separate Dockerfiles or Docker Compose to manage multiple containers.

4.1.10 Compose Multiple Services:

Our app requires multiple containers (e.g., web server, database, caching service), We use Docker Compose to define and manage the services together. This makes managing the whole application stack more manageable.

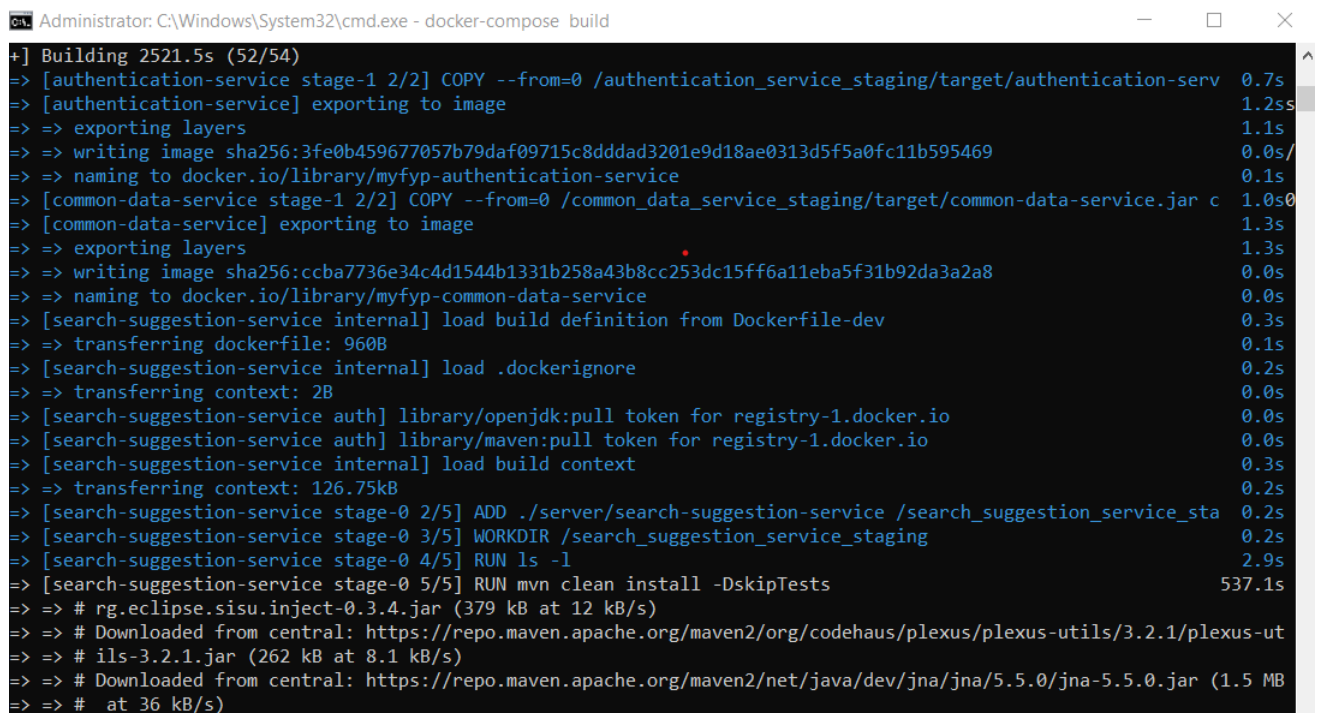
4.1.11 Push Image:

We want to deploy the container to a remote server or a cloud platform, we push Docker image to a container registry like Docker Hub or Amazon ECR.

4.1.12 Deploy Dockerized App:

The last we Deploy the Docker containers to our chosen production environment, either on-premises or in the cloud. Tools like Kubernetes or Docker Swarm can help manage container orchestration and scaling.

Taking some screenshots on deployment time;



```
Administrator: C:\Windows\System32\cmd.exe - docker-compose build
+ Building 2521.5s (52/54)
=> [authentication-service stage-1 2/2] COPY --from=0 /authentication_service_staging/target/authentication-serv 0.7s
=> [authentication-service] exporting to image 1.2s
=> => exporting layers 1.1s
=> => writing image sha256:3fe0b459677057b79daf09715c8dddad3201e9d18ae0313d5f5a0fc11b595469 0.0s/
=> => naming to docker.io/library/myfyp-authentication-service 0.1s
=> [common-data-service stage-1 2/2] COPY --from=0 /common_data_service_staging/target/common-data-service.jar c 1.0s
=> [common-data-service] exporting to image 1.3s
=> => exporting layers 1.3s
=> => writing image sha256:ccba7736e34c4d1544b1331b258a43b8cc253dc15ff6a11eba5f31b92da3a2a8 0.0s
=> => naming to docker.io/library/myfyp-common-data-service 0.0s
=> [search-suggestion-service internal] load build definition from Dockerfile-dev 0.3s
=> => transferring dockerfile: 960B 0.1s
=> [search-suggestion-service internal] load .dockerignore 0.2s
=> => transferring context: 2B 0.0s
=> [search-suggestion-service auth] library/openjdk:pull token for registry-1.docker.io 0.0s
=> [search-suggestion-service auth] library/maven:pull token for registry-1.docker.io 0.0s
=> [search-suggestion-service internal] load build context 0.3s
=> => transferring context: 126.75kB 0.2s
=> [search-suggestion-service stage-0 2/5] ADD ./server/search-suggestion-service /search_suggestion_service_sta 0.2s
=> [search-suggestion-service stage-0 3/5] WORKDIR /search_suggestion_service_staging 0.2s
=> [search-suggestion-service stage-0 4/5] RUN ls -l 2.9s
=> [search-suggestion-service stage-0 5/5] RUN mvn clean install -DskipTests 537.1s
=> => # rg.eclipse.sisu.inject-0.3.4.jar (379 kB at 12 kB/s)
=> => # Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.2.1/plexus-ut
=> => # ils-3.2.1.jar (262 kB at 8.1 kB/s)
=> => # Downloaded from central: https://repo.maven.apache.org/maven2/net/java/dev/jna/jna/5.5.0/jna-5.5.0.jar (1.5 MB
=> => # at 36 kB/s)
```

```

Administrator: C:\Windows\System32\cmd.exe

=> [search-suggestion-service auth] library/maven:pull token for registry-1.docker.io      0.0s
=> [search-suggestion-service internal] load build context                               0.3s
=> => transferring context: 126.75kB                                                    0.2s
=> [search-suggestion-service stage-0 2/5] ADD ./server/search-suggestion-service /search_suggestion_service_sta 0.2s
=> [search-suggestion-service stage-0 3/5] WORKDIR /search_suggestion_service_staging    0.2s
=> [search-suggestion-service stage-0 4/5] RUN ls -l                                    2.9s
=> [search-suggestion-service stage-0 5/5] RUN mvn clean install -DskipTests             549.8s
=> [search-suggestion-service stage-1 2/2] COPY --from=0 /search_suggestion_service_staging/target/search-sugges 0.4s
=> [search-suggestion-service] exporting to image                                       0.5s
=> => exporting layers                                                                  0.5s
=> => writing image sha256:24d727ba8cf756b474bae5aca78a4e9123e972266f43e6244074d6bcd913f6c6 0.0s
=> => naming to docker.io/library/myfyp-search-suggestion-service                     0.0s

C:\Projects\myfyp>docker-compose build up -d
unknown shorthand flag: 'd' in -d

C:\Projects\myfyp>docker-compose build up-d
no such service: up-d

C:\Projects\myfyp>docker-compose up -d
+] Running 7/7
[+] Container redis-cache-container           Started      43.8s
[+] Container mysql-db-container             Started      43.8s
[+] Container react-service-container         Started      45.1s
[+] Container payment-service-container       Started       8.7s
[+] Container authentication-service-container Started       8.8s
[+] Container common-data-service-container   Started       7.7s
[+] Container search-suggestion-service-container Started     11.1s

C:\Projects\myfyp>

```

Docker Container:

The screenshot shows the Docker Desktop application window. The top bar includes the 'Docker Desktop' logo, an 'Upgrade plan' button, a search bar with the text 'Search for images, containers, volumes, extensions and more...', and a 'Ctrl+K' button. The left sidebar contains navigation links: 'Containers', 'Images', 'Volumes', 'Dev Environments BETA', 'Docker Scout EARLY ACCESS', 'Learning Center', 'Extensions', and 'Add Extensions'. The main panel is titled 'Containers' and displays 'Container CPU usage' as 356.14% / 400% (4 cores allocated) and 'Container memory usage' as 1.44GB / 5.95GB. Below this, there is a search bar and a toggle switch for 'Only show running containers'. A table lists the containers:

Name	Image	Status	CPU (%)	Port(s)	Last started
> myfyp	myfyp	Running (8/10)	356.14%		2 minutes ago

Docker Images:

Containers

Images

Volumes

Dev Environments BETA

Docker Scout EARLY ACCESS

Learning Center

Extensions

Add Extensions

Images [Give feedback](#)

Local

Hub

Artifactory

EARLY ACCESS

2.86 GB / 1.9 GB in use 16 images

<input type="checkbox"/>	Name	Tag	Status
<input type="checkbox"/>	myfyp-search-suggestion-service 24d727ba8cf7	latest	In use
<input type="checkbox"/>	myfyp-common-data-service ccba7736e34c	latest	In use
<input type="checkbox"/>	myfyp-authentication-service 3fe0b4596770	latest	In use
<input type="checkbox"/>	myfyp-payment-service a587726817b9	latest	In use
<input type="checkbox"/>	myfyp-react-ui c83c177b35c6	latest	In use
<input type="checkbox"/>	<none> 28be88b0cb33	<none>	Unused (da

Docker Volumes:

Docker Desktop Upgrade plan

Search for images, containers, volumes, extensions and more... **Ctrl+K**

Containers

Images

Volumes

Dev Environments BETA

Docker Scout EARLY ACCESS

Learning Center

Extensions

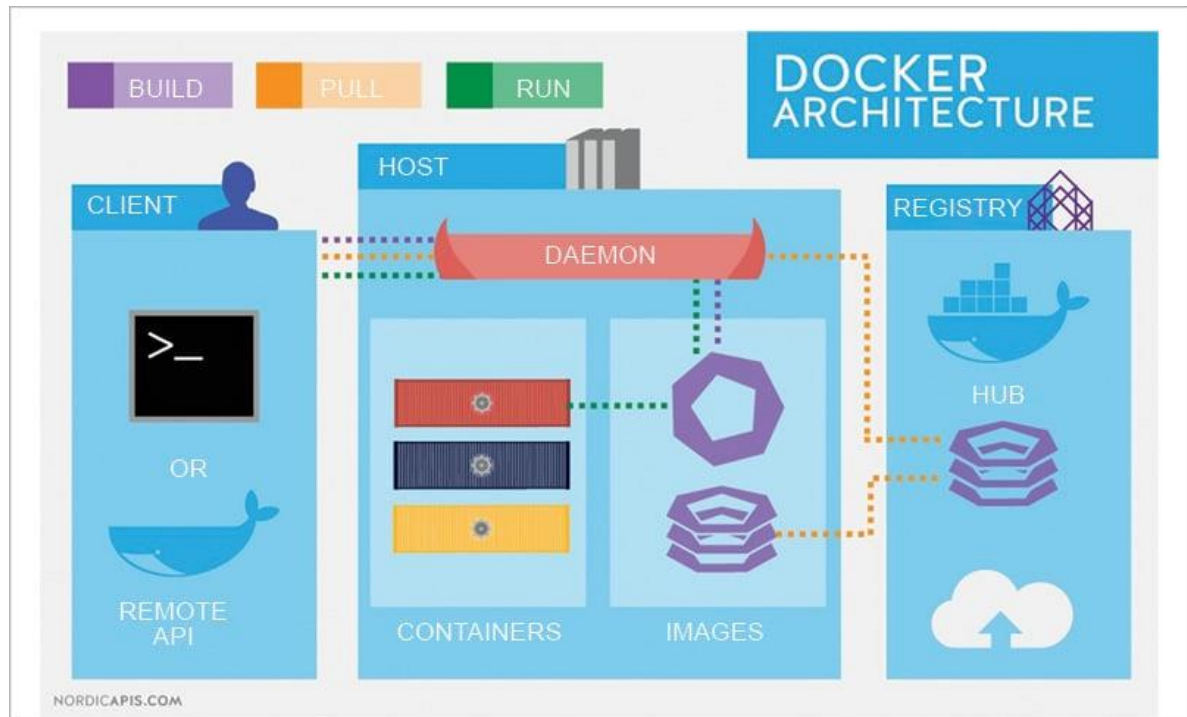
Add Extensions

Volumes [Give feedback](#)

Try the Volumes Docker extension to backup, restore and share volumes.

<input type="checkbox"/>	Name	Status	Created
<input type="checkbox"/>	5429f7ace2d1e7a91875fd95cf278bdda1c03e51d5970057d72e19b55831aad8	in use	1 hour ago
<input type="checkbox"/>	7dd5e37f5dbbcc27ffdc898707a67b06ed2742af1184ecb8401fba649671deca	in use	4 minutes ago
<input type="checkbox"/>	8b0fd4ce1ae96981163ac12de77744188d71c038ae81e05aedccb45bfc44c70	in use	1 hour ago
<input type="checkbox"/>	91a362dfb63cef1872903707b0af46a5e4d8c0c206dc51b9cfe6fc2e80633f74	in use	3 minutes ago
<input type="checkbox"/>	a3c1754f13462ab2d00b9f0b3e1e6a1e2c2d52d637e95215a533457b177b73de	in use	60 minutes ago
<input type="checkbox"/>	b615fb9bce7d3add10cee55f0da72b59e77555c29660e41d11bc2ec72e37fb91	in use	4 minutes ago
<input type="checkbox"/>	b959dd7ae51cd14796bd65e7d2c21652c10c8814c95a9087bb05432d258cb7e4	in use	4 minutes ago
<input type="checkbox"/>	bdc49a4637f7705e7cee22eb4022309a138b4e5b5cae89fd83e3c5482d4ac77	in use	3 minutes ago
<input type="checkbox"/>	cd8041103de50ddd5ad83e425619ff16d0c18e3f36b9630f6c4211c1fccfd678	in use	3 minutes ago

Below the docker architecture:



4.2 Kubernetes Deployment:

The steps to deploy an e-commerce web app on Kubernetes:

4.2.1 Containerize Web App:

Our e-commerce web app is containerized using Docker as described in the previous answer. This means having a Docker image for our web app that encapsulates all dependencies and configurations.

4.2.2 Set Up Kubernetes Cluster:

Then we Install and set up a Kubernetes cluster. And choose from various options like Kubernetes on-premises, managed Kubernetes services from cloud providers (e.g., Amazon EKS, Google Kubernetes Engine, Azure Kubernetes Service), or Kubernetes distributions like Minikube for local development.

4.2.3 Kubernetes Namespace:

We create a dedicated Kubernetes namespace for our e-commerce application to isolate it from other services running in the cluster.

4.2.4 Configure Kubernetes Services:

A Kubernetes Service manifest to expose our e-commerce app to the external world. We can choose between a LoadBalancer, NodePort, or ClusterIP service type, depending on our environment and requirements.

4.2.5 Set up Ingress:

We have multiple services or need more advanced routing and load balancing, consider setting up an Ingress controller to manage external access to our e-commerce app.

4.2.6 Database and Backend Services:

Our e-commerce app relies on backend services, set up Kubernetes resources for those services (e.g., databases, caching services). Depending on the service, you might use StatefulSets, Deployments, or other appropriate resources.

4.2.7 Secrets Management:

K8s also handle sensitive information (e.g., API keys, database credentials) securely using Kubernetes Secrets.

4.2.8 Persistent Storage:

Our e-commerce app requires persistent storage (e.g., for user uploads or data), for that set up Kubernetes Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) to ensure data persists across container restarts.

4.2.9 Monitoring and Logging:

We are configure monitoring and logging solutions to keep track of our application's health and performance. Tools like Prometheus and Grafana can help with monitoring, and Elasticsearch/Fluentd/Kibana (EFK) or Loki/Promtail/Grafana (PLG) stack can handle logging.

4.2.10 Auto-scaling:

We also implement Horizontal Pod Auto scaling (HPA) to automatically scale our e-

commerce app based on resource utilization or custom metrics.

4.2.11 Deployment and Updates:

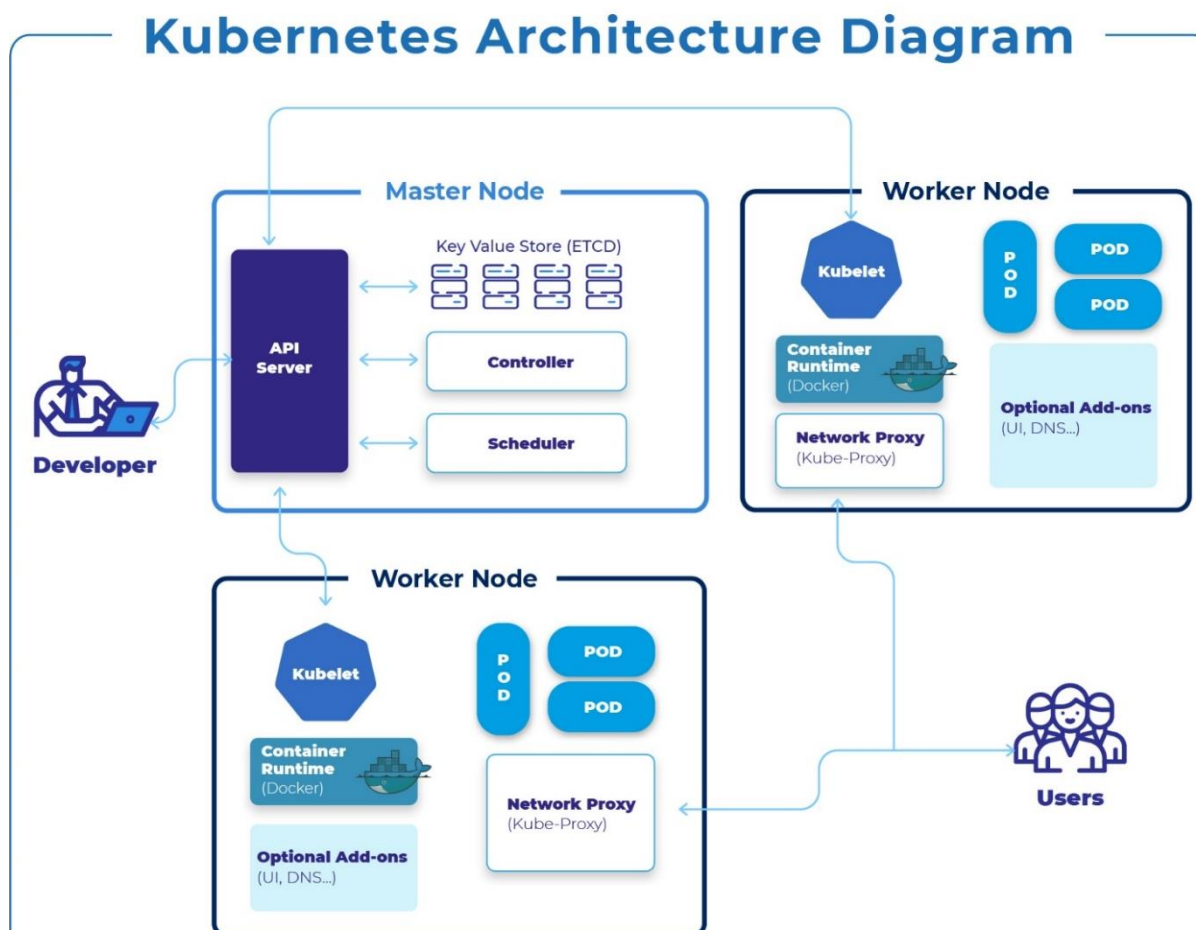
We deploy our e-commerce web app to the Kubernetes cluster using kubectl or a continuous deployment tool. Handle updates and rollbacks gracefully.

4.2.12 Backup and Disaster Recovery:

We consider that implementing a backup and disaster recovery strategy for critical data and configurations.

4.2.13 Kubernetes Architecture:

In below image explain the architecture:



4.3 Cloud Infrastructure with AWS

This section will discuss Amazon Web Services (AWS) and its usage for deploying microservices and cloud infrastructure setup. Here, the literature review explores the

benefits of utilizing Amazon Web Services (AWS) for cloud infrastructure. It discusses the various AWS services that can support the development and deployment of our e-commerce website.

4.3.1 Deploying Microservices-Based E-commerce Web Application on AWS

We use Docker and Kubernetes on AWS EC2 instances involves setting up the required software and configurations to manage containerized applications. Below we explain the steps to use Docker and Kubernetes on AWS EC2:

Docker on AWS EC2:

- **Launch EC2 Instances:**

Firstly we Log in to your AWS Console, navigate to EC2, and launch the desired number of EC2 instances. Ensure that the instances are using an operating system supported by Docker, such as Amazon Linux, Ubuntu, or CentOS.

- **Install Docker:**

Connect to the EC2 instances using SSH and install Docker. We can find installation instructions for different Linux distributions on the Docker website.

- **Manage Docker Daemon:**

So start and enable the Docker daemon to ensure that Docker is running on system boot.

- **Add Users to the Docker Group:**

To avoid using sudo with every Docker command, i can add your user to the docker group:
`sudo usermod -aG docker $USER`

- **Test Docker Installation:**

For checking we run a simple Docker command like `docker --version` or `docker run hello-world` to verify that Docker is installed and working correctly on your EC2 instance.

Kubernetes on AWS EC2:

- **Install kubectl:**

Firstly install the Kubernetes command-line tool, kubectl, on my local machine. This tool allows me

to interact with the Kubernetes cluster.

- **Set Up a Kubernetes Cluster:**

There are several ways to set up a Kubernetes cluster on AWS EC2:

- **Kubeadm:**

We can use kubeadm to bootstrap a cluster on EC2 instances manually. This involves initializing a master node and adding worker nodes to the cluster.

- **Kops:**

Kubernetes Operations (kops) is a popular tool for creating production-grade Kubernetes clusters on AWS. It automates many aspects of cluster provisioning and management.

- **EKS (Elastic Kubernetes Service):**

AWS provides a managed Kubernetes service called Amazon EKS, which abstracts the underlying infrastructure and simplifies cluster management. With EKS, you only need to create the worker nodes (EC2 instances) and connect them to the EKS control plane.

- **Configure kubectl:**

Once the Kubernetes cluster is up and running, you need to configure kubectl to connect to it. Obtain the cluster's access credentials and set them up on your local machine.

- **Test Kubernetes Installation:**

Use kubectl commands like `kubectl get nodes` to verify that the cluster is properly set up, and the nodes are connected.

- **Create Docker Image:**

Dockerize our e-commerce web app by creating a Dockerfile and building a Docker image containing your app and its dependencies.

- **Push Docker Image to a Registry:**

We push the Docker image to a container registry like Amazon ECR, Docker Hub, or any other private registry.

- **Kubernetes Deployment:**

We create Kubernetes Deployment manifests. This includes specifying the Docker image, environment variables, resource requirements, etc.

- **Kubernetes Service:**

A Kubernetes Service to expose your e-commerce app to the external world.

- **Ingress Controller:**

We need more advanced routing and load balancing, for that we set up an Ingress controller and Ingress resources to handle external access to our e-commerce app.

- **Scale and Manage:**

For that we use kubectl commands to scale the number of replicas, update the application, or perform rolling updates as needed.

- **Set up your AWS environment:**

We set up the necessary AWS services based on your requirements, such as Amazon EC2, Amazon RDS, Amazon S3, and Amazon VPC.

We also ensure that we have proper access control and security measures in place.

- **Containerize your microservices:**

Then we use Docker to containerize each microservice, creating separate Docker images for each service.

Then we build Docker images for our microservices, including the necessary dependencies and configurations.

And lastly push the Docker images to a container registry like Amazon ECR.

2.3.2 Create AWS infrastructure for microservices:

We set up an Amazon Elastic Kubernetes Service (EKS) cluster to manage our microservices. Then we configure the EKS cluster to create worker nodes that will host our microservices. After that deploy the required networking components, such as VPC, subnets, and security groups.

- **Deploy microservices to EKS:**

Kubernetes deployment manifests or Helm charts for each microservice, specifying the necessary resources and configurations. Deploy our microservices to the EKS cluster using kubectl or Helm, These are tools. Monitor the deployment status and ensure that all microservices are running successfully.

- **Set up database and storage:**

We managed database service like Amazon RDS or Amazon DynamoDB to store our application data. Configure the necessary permissions and access control for the database. We also set up Amazon S3 or similar storage services for handling static assets like product images or user uploads.

- **Load balancing and scaling:**

AWS load balancer service like Application Load Balancer (ALB) or Network Load Balancer (NLB) to distribute traffic among our microservices. So we configure auto-scaling policies based on metrics like CPU utilization or request rates to automatically scale our microservices.

- **Set up monitoring and logging:**

AWS CloudWatch or third-party monitoring tools to monitor the health, performance, and

Availability of our microservices. Implement centralized logging using services like AWS Cloud Watch Logs or ELK stack for effective troubleshooting and analysis.

- **Configure domain and SSL:**

We register a domain name and configure DNS settings to point for our microservices or load balancer. Set up SSL certificates using AWS Certificate Manager Authority for secure HTTPS communication.

CHAPTER 05

Recommendation & Stripe Payment System

5.1 Recommendation System Overview:

A recommendation system is a technology that analyzes user behavior and preferences to provide personalized suggestions. Its goal is to guide users toward relevant content, products, or services that they might be interested in. This not only improves user satisfaction but also boosts conversion rates and engagement.

- **Integration with the Web Application:**

We've seamlessly integrated the Algolia recommendation system into our e-commerce web application to offer personalized suggestions to our users. Algolia's capabilities enable us to provide real-time and context-aware recommendations, which enhance the overall user experience. The integration involves three main steps:

- **Data Collection:**

We collect and curate user interaction data, such as search queries, viewed products, and purchase history. This data forms the foundation of personalized recommendations.

- **Algorithm Training:**

Algolia's recommendation algorithms use machine learning to analyze user data and identify patterns in user behavior. These algorithms continuously learn and adapt to users' preferences over time.

- **User-Facing Integration:**

The recommendations generated by Algolia are seamlessly integrated into the user interface of our e-commerce website. Users see personalized suggestions while

browsing, searching, or viewing specific products. This integration is designed to enhance user engagement, simplify product discovery, and ultimately drive sales.

5.1.1 Operation in Our E-commerce Website:

In our e-commerce website, the Algolia recommendation system operates as follows:

- **Search Recommendations:**

When user's type in the search bar, Algolia's recommendation system suggests relevant queries based on popular searches and past user behavior. This aids users in refining their searches and finding products faster.

- **Product Recommendations:**

As users explore product pages, Algolia analyzes their viewing and purchase history to offer similar or complementary products. This "Customers Who Bought This Also Bought" section encourages cross-selling and increases the average order value.

- **Personalized Home Page:**

The home page showcases a "Recommended for You" section that displays products tailored to each user's preferences. This dynamic content encourages users to engage with the site and discover new items.

- **Cart Recommendations:**

During the checkout process, Algolia can suggest additional products that complement items already in the user's cart. This encourages upselling and enhances the shopping experience.

5.2 Stripe Payment System Overview:

Stripe is a robust and widely used payment processing platform that offers secure and efficient payment solutions for businesses operating online. Its capabilities range from

handling credit card transactions to managing subscriptions and more. With its developer-friendly API and comprehensive features, Stripe has become a preferred choice for e-commerce businesses to manage their online payments seamlessly.

- **Integration with the Web Application:**

We have seamlessly integrated the Stripe payment system into our e-commerce web application to provide a smooth and secure checkout experience for our users. The integration process involves several key steps:

- **Account Setup:**

We've created a business account with Stripe to access their API and payment processing tools.

- **API Integration:**

Our development team has integrated Stripe's API into our web application's codebase. This allows us to send payment requests and receive payment-related information securely.

- **Checkout Page:**

During the checkout process, users are presented with a secure and user-friendly checkout page. This page is powered by Stripe's technology and allows users to enter their payment information.

- **Payment Processing:**

Once users input their payment details, Stripe securely processes the payment using encryption and anti-fraud measures. User payment data is not stored on our servers, enhancing security.

- **Confirmation:**

After the payment is successfully processed, our application receives a confirmation from Stripe, indicating the successful transaction. Users receive an order confirmation on the website.

5.2.1 Operation in Our E-commerce Website:

In our e-commerce website, the Stripe payment system operates as follows:

- **Secure Checkout:**

When users proceed to checkout, they are directed to a secure payment page hosted by Stripe. This page collects their payment information while ensuring data privacy.

- **Payment Options:**

Stripe supports a variety of payment methods, including credit and debit cards. Users can select their preferred payment method for the transaction.

- **Real-time Processing:**

Once users enter their payment details, Stripe processes the payment in real time, verifying the information and checking for any potential fraud.

- **Subscription Management:**

For subscription-based services, Stripe manages recurring payments and subscription cancellations, streamlining the subscription model for our users.

- **Refunds and Disputes:**

Stripe also provides tools for managing refunds and handling payment disputes, ensuring a high level of customer service.

CHAPTER 06

Methodology

6.1 Research Design

This section will describe the research design used for this study, including the design approach, the development process followed, and the technology stack selection process.

The methodology chapter starts with an overview of the research design. It outlines whether the study is qualitative, quantitative, or a combination of both. It explains the rationale behind the chosen research design and justifies its suitability for the research objectives.

6.2 Data Collection and Analysis

This section will describe the data sources used for this study and the analysis method applied.

The data collection methods and tools that will be used to gather relevant information for the study. It may include surveys, interviews, and data analysis techniques. The thesis also discusses how the collected data will be analyzed to draw meaningful conclusions.

6.3 Development Process

This section will describe the development process followed for the project, including the agile methodology and the development tools used.

The development process that will be followed to implement the e-commerce website. It may include an agile development approach, detailing the stages involved, such as requirements gathering, design, implementation, testing, and deployment.

6.4 Technology Stack Selection

This section will describe the technology stack selection process and the criteria used for selecting a particular framework or tool.

The process of selecting the technology stack for the e-commerce website. It discusses the criteria used to evaluate and choose React, Spring Boot, Docker, Kubernetes, AWS, and Algolia. It also justifies why these technologies were deemed suitable for the project.

CHAPTER 07

Evaluation and Results

7.1 Performance Evaluation Metrics

This section will describe the performance evaluation metrics used and the results obtained.

The evaluation and results chapter begins with the definition of performance evaluation metrics. It identifies the key performance indicators (KPIs) that will be used to assess the effectiveness and efficiency of the implemented e-commerce platform. Examples may include page load time, server response time, and database query performance.

7.2 User Experience Evaluation

This section will describe the user experience evaluation method used and the results obtained.

The methods used to evaluate the user experience of the e-commerce website. It may include user surveys, interviews, or usability testing to gather feedback and insights from users.

7.3 Testing and Validation

This section will describe the testing and validation process used and the results obtained.

The testing and validation process conducted to ensure the functionality and reliability of the e-commerce platform. It covers unit testing, integration testing, and system testing, as well as any tools or frameworks used for automated testing.

7.4 Analysis of Results

This section will analyze and interpret the results obtained from the evaluation.

The findings from the evaluation and testing processes. It discusses the performance metrics, user feedback, and any issues or limitations identified during the evaluation. The results are analyzed to draw conclusions and provide insights into the effectiveness and efficiency of the implemented technologies.

CHAPTER 08

Software Testing

8.1 Functionality Test Case

In this category, the focus is on testing the individual functionalities of the e-commerce web application microservices. Some key test cases include, explain below:

8.2 User Registration Testing

- We verify that users can successfully register with valid information.
- Validate that mandatory fields are properly enforced during registration.
- Test for error handling when invalid or duplicate user registration attempts are made.

8.3 User Login Testing

- We validate that users can log in using valid credentials.
- Test for appropriate error messages when incorrect login credentials are provided.
- Ensure that the login session is maintained correctly.

8.4 User Module Testing

- Test user profile management functionalities, such as updating user information, managing addresses, and resetting passwords.

- Verify that users can view and edit their order history and track their orders.

8.5 User Interface Test Cases

These test cases focus on ensuring a seamless and user-friendly interface for the e-commerce web application. Examples include:

- Verify that all elements of the user interface are displayed correctly and consistently across different devices and browsers.
- Test the responsiveness of the website design to different screen sizes and orientations.
- Validate that navigation menus, buttons, and links are working as intended.

8.6 Performance Test Cases

Performance testing is crucial to ensure that the e-commerce web application can handle expected loads and response times. Test cases may include:

- Load testing to evaluate the system's behavior under expected user loads.
- Stress testing to determine the application's performance and stability under extreme load conditions.
- Measure and optimize response times for critical functionalities like product listing, search, and checkout.

8.7 Integration Test Case

Integration testing is vital for microservices-based development to ensure seamless communication between different services. Key test cases include:

- Test integration between user management, inventory management, and

payment microservices.

- Verify that data is synchronized accurately between services.
- Validate that error handling and exception scenarios are handled appropriately during integration.

8.8 Usability Test Cases

Usability testing focuses on evaluating the user experience and ease of use of the e-commerce web application. Test cases may include:

- Validate that the website's layout and design are intuitive and user-friendly.
- Test the ease of finding products, adding them to the cart, and completing the checkout process.
- Evaluate the effectiveness of search functionalities and product filters.

8.9 Database Test Cases

Database testing ensures the integrity and reliability of data storage and retrieval. Test cases may include:

- Verify that data is accurately stored and retrieved from the database.
- Test the handling of large data volumes, such as product catalogs and user data.
- Validate that database constraints, such as unique keys and foreign key relationships, are enforced correctly.

8.10 Security Test Cases

Security testing is crucial for an e-commerce web application to protect user data and prevent unauthorized access. Test cases may include:

- Validate that user passwords are securely stored using encryption techniques.
- Test for SQL injection and cross-site scripting vulnerabilities.
- Verify that sensitive user data, such as payment information, is transmitted securely using encryption protocols.

8.11 Microservices Deployment Testing

8.11.1 Docker Deployment Testing:

Test the Docker image creation process to ensure it includes all the required dependencies and configurations. We verify that the Docker images can be successfully built and pushed to a Docker registry. Test the deployment of Docker containers on various environments, such as local development machines, staging environments, and production servers. Validate that the deployed containers can communicate with each other as intended.

8.11.2 Kubernetes Deployment Testing:

Test the deployment of microservices on a Kubernetes cluster using Kubernetes manifests or deployment scripts. We verify that the Kubernetes deployment files are correctly configured, including container specifications, resource limits, and environment variables. Conduct tests to ensure that the desired number of replicas are created and distributed across the cluster. Test scaling capabilities by increasing or decreasing the number of replicas and validating that the application remains responsive.

8.11.3 AWS Deployment Testing:

Validate the deployment of microservices on AWS services such as Amazon Elastic Container Service (ECS), Amazon Elastic Kubernetes Service (EKS). Test the integration of the microservices with other AWS resources such as Amazon RDS (Relational Database Service) or Amazon S3 (Simple Storage Service). Verify that the necessary security measures, such as IAM roles and policies, are correctly configured for accessing AWS services. Conduct tests to ensure that the microservices are accessible from the internet through proper networking configurations and load balancer settings.

CHAPTER 09

Conclusion & Future Work

9.1 Conclusion of Chapter 01

In conclusion, this chapter provided a comprehensive introduction to the objectives and scope of this thesis, which focuses on designing and implementing a microservices-based e-commerce web application. By utilizing modern technologies and frameworks such as React, Spring Boot, Docker, Kubernetes, AWS, and Algolia, the aim is to create a scalable, fault-tolerant, and efficient platform that addresses the challenges posed by traditional monolithic e-commerce applications.

The background discussion highlighted the growing significance of e-commerce websites in modern business and the need for robust platforms that cater to diverse customer segments. The problem statement underscored the limitations of existing e-commerce solutions and the potential benefits of adopting microservices architecture.

The objectives outlined the key goals of this thesis, including the design and implementation of an intuitive user interface, the incorporation of a recommendation system, and the evaluation of modern technologies in enhancing user experience and platform performance.

The significance of the study emphasized the potential contributions of the research to the e-commerce industry by offering insights into building comprehensive e-commerce platforms and assessing the effectiveness of cutting-edge technologies.

The scope and limitations section clarified the areas covered in the study, including the use of React, Spring Boot, Docker, Kubernetes, and AWS for implementation, while acknowledging constraints like time and resource limitations.

The chapter structure highlighted the organization of the thesis, with various chapters dedicated to microservices deployment, testing, design and implementation, evaluation, software requirements specification, software testing, and discussion. [CHAPTER 01](#)

9.2 Conclusion of Chapter 02

In the subsequent chapters, the project activities were detailed, ranging from requirements gathering and system design to development, testing, deployment, marketing, and customer support. Each phase plays a crucial role in creating a robust and user-friendly e-commerce platform.

Lastly, the responsive web design approach was discussed, highlighting its importance in ensuring a seamless user experience across diverse devices and screen sizes. By employing responsive design patterns and relative units, the platform aims to provide consistent functionality regardless of the user's device.

In conclusion, this chapter has delved into the intricate details of the design and implementation of the microservices-based e-commerce web application. The architecture design provided an overview of the application's structure and the interactions between its components. Front-end development using React was explored, including the utilization of state management techniques to create a responsive and intuitive user interface.

The back-end development section delved into the implementation of the server-side logic using Spring Boot and MySQL database integration. It highlighted the creation of API endpoints essential for the microservices and showcased the design and structure of the database tables.

The chapter further discussed the MySQL database, its open-source nature, and the advantages it brings to the project. It highlighted the use of ODBC drivers for connecting MySQL to ASP.NET and the advantages of its multithreaded architecture.

The database design section provided a detailed breakdown of the tables comprising the database, elucidating primary and foreign keys, which serve as the foundation for data integrity and relationships.

Functional decomposition diagrams and data flow diagrams were introduced, visualizing the system's structure and the flow of data between processes, external entities, and data stores. These visualizations aid in comprehending the project's complexity and the interplay of its components.

The Shopping Cart Application, replete with registration, user interface design, shopping cart, and order placement, was meticulously explained. Screenshots and descriptions showcased the user journey from selecting products to placing an order, underlining the application's user-centric approach. [CHAPTER 02](#)

9.3 Conclusion of Chapter 03

In conclusion, this chapter has outlined the Software Requirements Specification (SRS) for the microservices-based e-commerce web application. The functional requirements presented a comprehensive set of features that the application will offer to its users, ensuring a seamless shopping experience. These functionalities range from browsing a product catalog to managing user accounts, ultimately aiming to provide a user-centric platform.

The non-functional requirements emphasized the crucial aspects that will govern the application's performance and user experience. The requirement for 24/7 availability underscores the application's commitment to serving users at any time. Scalability is addressed to ensure that the application can handle a large number of users, maintaining responsiveness even during peak usage periods. Security is a paramount concern, promising to protect user data and transactions. Lastly, the commitment to accessibility highlights the application's inclusivity by catering to users with disabilities.

The assumptions and dependencies section delineated the technological choices and foundational principles upon which the application will be built. The use of the Java programming language, deployment in a cloud-based environment, and the adoption of a microservices architecture showcase a forward-looking approach that leverages modern technologies and best practices.

The Software Requirements Specification provided in this chapter serves as a guiding document that outlines the project's goals, features, and performance expectations. The subsequent chapters will delve into the testing and implementation of these requirements, further solidifying the realization of the microservices-based e-commerce web application.

[CHAPTER 03](#)

9.4 Conclusion of Chapter 04

In conclusion, this chapter delved into the intricate process of microservices deployment using Docker and Kubernetes, as well as the integration of Amazon Web Services (AWS) for cloud infrastructure. The deployment of a microservices-based e-commerce web application involves a series of well-defined steps to ensure scalability, reliability, and efficiency.

The Docker section emphasized the importance of containerization for isolating and managing individual services within the application. The outlined steps covered Docker installation, the creation of Docker images, the configuration of environments, and the utilization of Docker Compose for multi-container orchestration. Screenshots provided visual insights into the Docker environment, images, and volumes, making evident the practical implementation of containerization.

The Kubernetes deployment section elucidated the process of deploying an e-commerce web app on a Kubernetes cluster. From containerizing the app to setting up Kubernetes namespaces, services, and backend resources, the process was thoroughly explained. Furthermore, the architecture diagram provided a clear visualization of how the Kubernetes components interact within the deployment ecosystem.

AWS integration was discussed extensively, focusing on the deployment of a microservices-based e-commerce web application on AWS EC2 instances and the configuration of Amazon

EKS for Kubernetes orchestration. The strategic use of services like Amazon RDS, S3, and Elastic Load Balancing was underscored for database management, storage, and load distribution. Moreover, the configuration of monitoring, logging, and SSL for secure communication was emphasized as integral components of deploying on AWS. [CHAPTER 04](#)

9.5 Conclusion of Chapter 05

In conclusion, this chapter explored the significance and integration of recommendation and payment systems within the context of our e-commerce web application. These systems, specifically the Algolia recommendation system and the Stripe payment system, play vital roles in enhancing user experience, boosting engagement, and facilitating seamless transactions.

The recommendation system, powered by Algolia, enriches the user journey by offering personalized suggestions based on user behavior and preferences. The integration seamlessly fits into various aspects of the e-commerce website, including search recommendations, product suggestions, personalized home pages, and cart recommendations. By providing context-aware recommendations, the system drives cross-selling, upselling, and overall user engagement.

Furthermore, the integration of the Stripe payment system ensures a secure, efficient, and user-friendly checkout process for our customers. Through careful account setup and API integration, our e-commerce web application seamlessly connects with Stripe's payment processing tools. Users benefit from the system's secure checkout pages, various payment options, real-time processing, subscription management, and efficient handling of refunds and disputes. [CHAPTER 05](#)

9.6 Conclusion of Chapter 06

In conclusion, this chapter delved into the methodology employed in conducting this study, shedding light on the research design, data collection and analysis, development process, and technology stack selection. Each of these components contributes significantly to the successful execution of the e-commerce web application project and the achievement of its objectives.

The research design of this study was carefully considered, aligning with the nature of the project. By employing a combination of qualitative and quantitative research approaches, we ensure a comprehensive understanding of both user behavior and technical implementation. The chosen research design is tailored to address the research objectives and provide insightful findings.

Data collection and analysis are fundamental aspects of this research. Various data sources, including surveys, interviews, and extensive data analysis, will be used to gather meaningful

insights. These insights will drive decision-making, inform system design, and validate user preferences.

The development process, guided by an agile methodology, ensures flexibility, adaptability, and iterative improvement throughout the project lifecycle. The use of agile principles facilitates efficient collaboration among team members and enhances project transparency. Tools such as requirements gathering, design, implementation, testing, and deployment are integrated seamlessly within this iterative approach.

The selection of the technology stack is a pivotal aspect of the project's success. The methodology outlined the meticulous process undertaken to choose the most suitable frameworks and tools for the e-commerce website. The chosen technologies, including React, Spring Boot, Docker, Kubernetes, AWS, and Algolia, were assessed based on well-defined criteria to ensure they align with the project's requirements and objectives.

[CHAPTER 06](#)

9.7 Conclusion of Chapter 07

In conclusion, this chapter delved into the evaluation and results obtained from the implementation of the e-commerce web application. Through a comprehensive assessment of performance, user experience, testing, and validation, this chapter provides valuable insights into the effectiveness and efficiency of the platform.

The performance evaluation metrics defined in this chapter served as the foundation for gauging the system's efficiency. By measuring key performance indicators such as page load time, server response time, and database query performance, we gained a clear understanding of how well the e-commerce website performs under different conditions.

User experience evaluation played a crucial role in understanding the platform's usability and overall user satisfaction. Through user surveys, interviews, and usability testing, we gathered direct feedback from users, enabling us to identify areas of improvement and validate the user-centric design choices made during the development process.

The rigorous testing and validation process outlined in this chapter were instrumental in ensuring the reliability and functionality of the e-commerce platform. Unit testing, integration testing, and system testing were conducted to identify and rectify any issues or inconsistencies in the system. Automated testing tools and frameworks were employed to streamline the testing process and enhance the accuracy of the results.

The analysis of results is a pivotal aspect of this chapter, providing a comprehensive overview of the findings from the evaluation and testing processes. By analyzing performance metrics, user feedback, and any identified limitations, we draw meaningful conclusions about the effectiveness of the implemented technologies. These insights inform

potential enhancements, refinements, and future iterations of the e-commerce web application. [CHAPTER 07](#)

9.8 Conclusion of Chapter 08

The culmination of meticulous software testing is an essential chapter that validates the robustness, reliability, and security of the e-commerce web application. Through a comprehensive range of test cases spanning functionality, user interface, performance, integration, usability, database, security, and microservices deployment, this chapter provides a comprehensive evaluation of the application's readiness for deployment.

The functionality test cases outlined in this chapter underscore the importance of individual features and functionalities. From user registration to login, user profile management to tracking orders, each aspect is scrutinized to ensure they operate as intended, handle errors gracefully, and provide a seamless user experience.

User interface test cases validate the responsive and user-friendly design across various devices and browsers. Performance test cases gauge the application's ability to handle anticipated user loads, assess its resilience under stress, and optimize response times for critical functions.

Integration test cases are pivotal for ensuring the harmonious communication between microservices. The usability test cases put user experience to the forefront, ensuring intuitive navigation, efficient product discovery, and a streamlined checkout process.

Database test cases ensure data integrity and reliability, while security test cases establish a fortified barrier against unauthorized access and potential vulnerabilities.

Lastly, the microservices deployment testing section examines the successful creation, deployment, and scalability of Docker containers within Kubernetes clusters or AWS environments. These tests underscore the seamless orchestration and scaling of microservices, highlighting their reliability in real-world scenarios. [CHAPTER 08](#)

9.9 Future Direction:

This section will discuss future directions based on the findings of the study. Below we explain step-by-step:

9.9.1 AI and Personalization:

Artificial Intelligence (AI) will play a crucial role in enhancing customer experiences by offering personalized product recommendations, personalized marketing, and

chatbot-powered customer support. AI-driven data analysis will help e-commerce businesses understand customer behavior and preferences better.

9.9.2 Voice Commerce:

With the growing popularity of smart speakers and voice assistants, voice commerce is expected to rise. Customers will use voice commands to search for products, place orders, and interact with e-commerce platforms through voice interfaces.

9.9.3 Social Commerce:

Social media platforms are becoming increasingly important in driving our e-commerce sales. Social commerce integration will enable users to discover and purchase products directly from social media platforms.

9.9.4 Mobile Commerce (M-Commerce):

Mobile shopping will remain a significant focus as more consumers prefer to shop on their smartphones and tablets. E-commerce businesses will optimize their mobile apps and websites for a seamless mobile shopping experience.

9.9.5 Fast and Flexible Delivery Options:

E-commerce invest in faster and more flexible delivery options, such as same-day or one-hour delivery, to meet the growing demand for quick and convenient shipping.

9.9.6 Data Privacy and Security:

As data breaches become more prevalent, our e-commerce companies will prioritize data privacy and security measures to protect customer information and build trust

with their audience.

9.9.7 Our Team Future Work:

Future of our team in the IT field! Pursuing higher education and gaining expertise in DevOps and MERN stack development are excellent choices, as both areas are in high demand and critical for modern software development. Here below are some suggestions for our team's future work and growth:

- **Continuous Learning:**

Learning never stop...! In the fast-paced IT industry, staying up-to-date with the latest technologies, tools, and best practices is crucial. We also attend webinars, workshops, and conferences, and keep exploring new areas related to DevOps and MERN stack development.

- **Certifications:**

Certifications can add credibility to our team's skills and expertise and provide a competitive edge in the job market.

- **Networking and Professional Connections:**

Our team to attend industry events, meetups, and networking sessions. Building professional connections can lead to new opportunities and collaborations.

- **Experiment with Cloud Technologies:**

DevOps and MERN stack development often involve deploying applications to the cloud. Familiarize ourselves with cloud platforms like AWS, Azure, or Google Cloud, as these platforms are widely used in the industry.

- **Soft Skills Development:**

Technical skills, soft skills are equally important for career growth. Our team is work on communication, leadership, and problem-solving skills.

- **Seek Internships and Job Opportunities Abroad:**

Our team members are also interested in pursuing a master's degree abroad, encourage them to explore internship or job opportunities in their respective fields in the target country. Gaining international experience can be valuable for our careers.

References

Microservices Architecture:

<https://www.scnsoft.com/software-development/microservices-consulting> CHAPTER 04

<https://www.opensourceforu.com/wp-content/uploads/2018/04/Figure-1-Docker-container-architecture.jpg> CHAPTER 04

<https://www.clickittech.com/devops/kubernetes-architecture-diagram/>

<https://microservices.io/>

Docker

https://docs.docker.com/get-started/08_using_compose/ Microservices Deployment

Kubernetes

<https://kubernetes.io/docs/concepts/containers/> 4.2 Kubernetes Deployment:

Cloud Computing:

https://docs.aws.amazon.com/?nc2=h_ql_doc_do 4.3 Cloud Infrastructure with AWS

Products Design:

<https://www.canva.com/> 2.4 Product Design

Data Flow Diagrams

https://lucid.app/lucidchart/270d1397-3870-4c82-9ce7-c400267e2c81/edit?beaconFlowId=91B5E41DD2B47D0D&invitationId=inv_8ea94d34-680c-40d8-8108-45871a999a9e&page=0_0# Design and Implementation

Books

Software Engineering_

<https://classroom.google.com/u/3/c/NDI4ODI4NDY1Mzg4/m/NDI4ODI4NDY1NDA4/details>
Software Requirements Specification