University of Messina

Bachelor of Data Analysis

ACADEMIC YEAR - 2023/2024

Software Engineering

E-commerce Clothing Store

(Project report)

Supervisor:
Prof. Salvatore Distefano

Students:
Sahil Nakrani
Henok Taddese
Yoshan Mendis

Acknowledgments

We would like to extend our deepest gratitude to all the individuals and team who contributed to the successful completion of this project.

Firstly, we are immensely grateful to our development team for their unwavering dedication, hard work, and innovative solutions that have been instrumental in bringing this project to life. Our technical expertise, collaboration, and commitment to excellence have been the foundation of our success.

A special thank you goes to our supervisor Professor Salvatore Distefano. Your input has been crucial in shaping the functionalities and ensuring that the platform meets the needs and expectations of our diverse user base.

Our sincere appreciation extends to our partners and third-party service providers, including Stripe, FastBots, and GTranslate. Your services and support have significantly enhanced the capabilities and reach of our platform.

We are grateful to our quality assurance team for their thorough testing and dedication to maintaining the highest standards of quality and reliability in our product. Your attention to detail has ensured a seamless user experience.

Lastly, we would like to thank our families and friends for their understanding and support during the long hours of development and testing. Your encouragement has been a source of strength and motivation for our team.

Thank you all for your contributions, collaboration, and commitment to making this project a success.

Table of Contents

1. **Abstract and Introduction**
    - Project Overview
    - Objectives
    - Scope
    - Innovation and Impact

2. **Choosing a Software Development Methodology**
    - Introduction
    - Factors Influencing the Choice of Methodology
    - Comparative Analysis of Methodologies
        - Water fall
        - Agile
    - Choosing Agile for the E-commerce Clothing Platform Project

3. **Agile Methodology (Scrum) and Sprints Overview**
    - Agile Philosophy Overview
    - Key Principles of Agile
    - Types of Agile Methodologies
    - Key features of Scrum
        - Iterative Process
        - Roles in Scrum
        - Scrum Ceremonies
        - Scrum Artifacts
        - Why was Scrum chosen for this project

4. **Software Requirements Analysis**
    - Objectives of Requirements Analysis
    - Functional Requirements
        - User Authentication
        - Product Management (CRUD operations)
        - Payment Processing
        - Language Translation
        - Chatbot Integration
        - Dark Mode Feature
        - Notification/Mailing Service
    - Non-functional Requirements
        - Performance
        - Security
        - Usability
        - Scalability

5. **Tools and Technologies Used**
    - Backend development tools
    - Frontend development tools
    - API tools
        - Language Translator API
        - Chatbot Services
        - Payment Gateway
        - Dark Mode Libraries
        - Email Services
    - Containerization and Deployment

6. **Actual Sprints Development and Results**
    - Initial Setup and Architecture
    - User Authentication and Profile Management
    - Product Management
    - Payment Processing
    - Notification Service
    - Admin panel
    - Language Translation
    - Chatbot Integration
    - Dark Mode

7. **UML**
    - UseCase Diagram
    - Class Diagram
    - Component Diagram
    - Database Design
    - Deployment Diagram

8. **Features Implementation**
    - Implementation Architecture
    - User Authentication / Authorization
    - CRUD operation
    - ChatBot Integration
    - Payment Integration
    - Language Translator
    - DarkMode Integration
    - Mailing Server Implementation
    - Admin Panel

9. **GUI Mock-up**

10. **Conclusion**

Introduction

In today's digital era, e-commerce has become a vital part of the retail industry. The e-commerce clothing platform is designed to offer a seamless shopping experience for users looking to purchase clothing items online. The platform aims to bridge the gap between physical and online retail by providing features.

Evolution of Online Clothing Stores

E-commerce has evolved significantly over the past two decades. Initially, online stores offered basic functionalities, primarily focusing on listing products and processing orders. However, with advancements in technology and changing consumer expectations, modern e-commerce platforms have become sophisticated systems offering a plethora of features. These include:

- Advanced search and filtering options to help users find products quickly.
- AI-driven recommendations and personalized shopping experiences.
- Multi-language support and global shipping options to cater to international markets.
- Integration with social media for marketing and customer engagement.
- Mobile-optimized interfaces to support the growing number of mobile shoppers.

Key Challenges in E-commerce

While e-commerce offers numerous opportunities, it also presents several challenges that need to be addressed:

- **Security**: Protecting user data and ensuring secure transactions is paramount.
- **Performance**: Ensuring the platform can handle high traffic volumes without compromising on speed and responsiveness.
- **User Retention**: Keeping users engaged and reducing cart abandonment rates through personalized experiences and efficient support systems.
- **Global Reach**: Catering to a global audience requires multi-language support, varied payment options, and efficient logistics.
- **Scalability**: The platform must be able to scale efficiently to accommodate growth and changing user demands.

Technologies in Modern E-commerce Platforms

To meet these challenges, modern e-commerce platforms leverage a variety of technologies:

- **Backend Technologies**: Use of robust frameworks like Django, Ruby on Rails, or Express.js to handle server-side logic and database interactions.
- **Frontend Technologies**: Employing responsive and dynamic front-end frameworks such as React, Angular, or Vue.js to create interactive user interfaces.
- **Database Systems**: Utilizing scalable databases like MySQL, PostgreSQL, or NoSQL databases such as MongoDB to store and manage data.
- **API Integration**: Implementing RESTful APIs to enable seamless communication between different parts of the system and integrate third-party services.
- **Cloud Services**: Leveraging cloud platforms like AWS, Azure, or Google Cloud for hosting, storage, and other infrastructure needs.
- **Security Protocols**: Applying SSL/TLS for secure data transmission, and OAuth or JWT for secure user authentication and authorization.

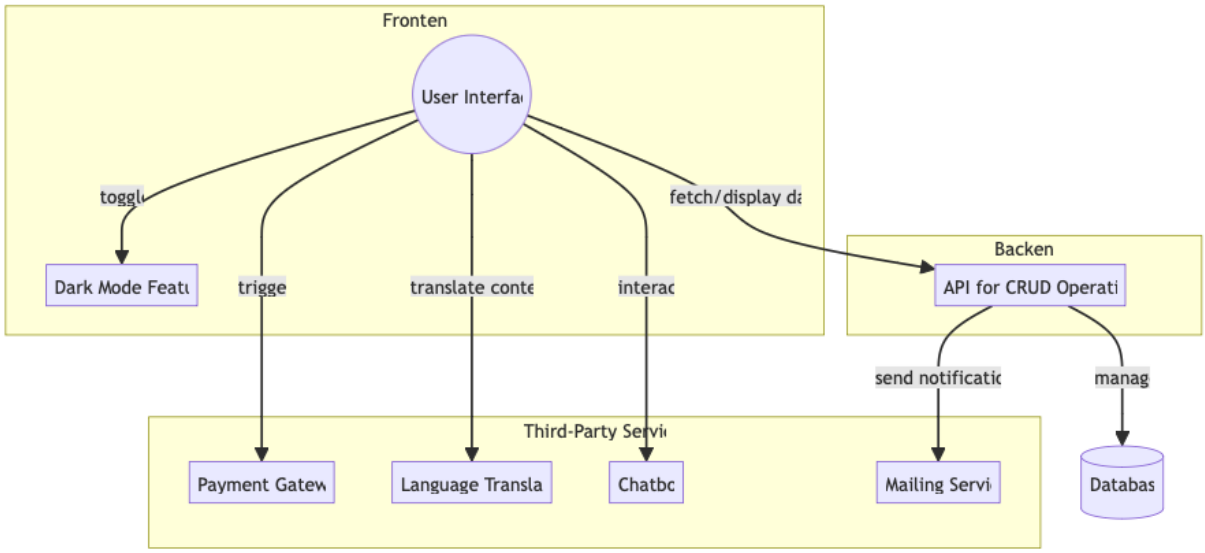
By combining these technologies and addressing the outlined challenges, the e-commerce clothing platform aims to set a benchmark in the online retail space, providing users with a secure, efficient, and enjoyable shopping experience.

Abstract

The e-commerce clothing platform project aims to develop a comprehensive online store that caters to users' needs for purchasing clothing items seamlessly. The platform incorporates a robust backend API for CRUD (Create, Read, Update, Delete) operations, facilitating efficient product management. To enhance user experience, various third-party services are integrated, including a payment gateway, language translator, chatbot, and dark mode feature. Additionally, a mailing service is included to handle user notifications.

The primary objective of this project is to provide a user-friendly and scalable solution with rich features that simplify the online shopping experience while ensuring high performance, security, and usability. By using modern technologies and agile methodologies, the platform is designed to be adaptable to changing market needs and user preferences.

There are key innovations that have been used in the development of this e-commerce platform. These innovations include the integration of real-time language translation to cater to a global audience and of course which supports more than five languages, an AI-powered chatbot for customer support, and a dark mode feature to enhance usability in different lighting conditions. The project's impact is anticipated to be significant in promoting convenient and accessible online shopping, thereby driving higher user engagement and satisfaction.



Objectives

The primary objectives of the e-commerce clothing platform project are as follows:

- **Streamline the Shopping Process:** Simplify the steps involved in browsing, selecting, and purchasing clothing items.
- **Enhance User Convenience:** Integrate various features such as a payment gateway, chatbot, and language translation to make the shopping experience more convenient and accessible.
- **Ensure High Performance and Security:** Develop a secure and high-performing platform that protects user data and provides a fast, responsive interface.
- **Promote Accessibility:** Implement a dark mode feature and multilingual support to cater to a diverse user base with different preferences and needs.

Scope

The project involves the comprehensive development of a full-stack e-commerce application, leveraging modern web technologies to create a robust, scalable, and user-centric platform for online clothing retail. The scope encompasses the entire software development lifecycle, from initial planning and requirements gathering to deployment and maintenance. Here's a detailed breakdown of the scope:

Frontend Development

The frontend will be meticulously designed to deliver an intuitive, engaging, and visually appealing user interface. Key aspects include:

- **Responsive Design**: Ensuring the platform is fully responsive, providing an optimal user experience across a variety of devices including desktops, tablets, and smartphones.
- **User Experience (UX)**: Designing a seamless and enjoyable user journey, from browsing products to completing purchases. This includes user-friendly navigation, advanced search and filtering options, and personalized product recommendations.
- **Interactive Features**: Incorporating interactive elements such as product carousels, hover effects, and dynamic content updates to enhance user engagement.
- **Customization**: Providing users with the ability to customize their experience, including options like dark mode for reduced eye strain and a personalized dashboard for managing their accounts and orders.

Backend Development

The backend will be developed to efficiently handle all core functionalities and ensure smooth operation of the platform. Key components include:

- **CRUD Operations**: Implementing a comprehensive API to manage Create, Read, Update, and Delete operations for products, user accounts, orders, and other essential entities.
- **User Authentication**: Developing a secure authentication system to manage user registration, login, password recovery, and account management. This will include features like two-factor authentication (2FA) for enhanced security.
- **Payment Processing**: Integrating secure and reliable payment gateways to handle transactions. The system will support multiple payment methods including credit/debit cards, digital wallets, and potentially cryptocurrencies.
- **Database Management**: Designing a scalable and efficient database schema to store and manage all necessary data, utilizing modern database technologies such as MySQL, PostgreSQL, or NoSQL solutions like MongoDB.

Third-Party Integrations

To enhance the platform's capabilities and provide additional services, several third-party integrations will be implemented:

- **Language Translation**: Integrating a language translation service to support multiple languages, ensuring the platform is accessible to a global audience.
- **Chatbot Support**: Implementing a chatbot to provide instant customer support, assist with common queries, and guide users through their shopping experience.
- **Dark Mode**: Utilizing third-party libraries to implement a dark mode feature, allowing users to switch between light and dark themes based on their preference.
- **Mailing Service**: Integrating a mailing service to send notifications, order confirmations, promotional emails, and other communications to users.

Choosing a Software Development Methodology

Selecting the appropriate software development methodology is a critical decision that can significantly impact the success of a project. This decision involves evaluating various factors, including project requirements, activities and designing and methodologies. The methodology chosen will guide the entire development process, from planning and execution to delivery and maintenance.

Factors Influencing the Choice of Methodology

Project Requirements

As soon as we start to develop the software we need to consider what services are required and the constraints on the system's operation and development. Requirements engineering process can be:

.

1. Feasibility study - Is it technically and financially feasible to build the system?
2. Requirements elicitation and analysis - What do the system stakeholders require or expect from the system?
3. Requirements specification - Defining the requirements in detail.
4. Requirements validation - Checking the validity of the requirements

Activities Designing

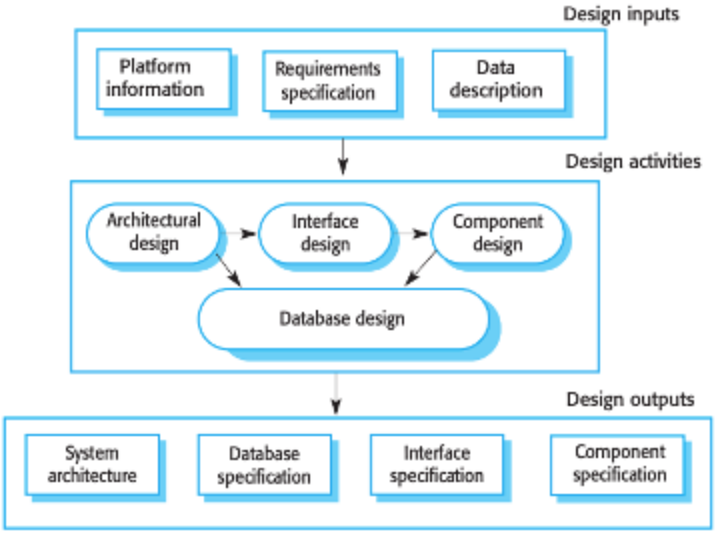
During the process development we also must consider about the description of the activities such as:

1. Products - which are the outcome of an activity.
2. Roles – which are consisting of the responsibility of the people involved into this process
3. Pre- and post-conditions, which are statements that are true before and after a process or a product produced.

After considering these activities we have to designing activities like:

1. Architectural design, where you identify the overall structure of the system, the principal components
2. Interface design, where we define the interfaces between system components.
3. Component design, where you take each system component and design how it will operate.
4. Database design, where you design the system data structures and how these are to be represented in a database.

Then we design these structures that realizes the specification, and then translate this structure into an executable program. A general model of the design process will be like this.



Comparative Analysis of Methodologies

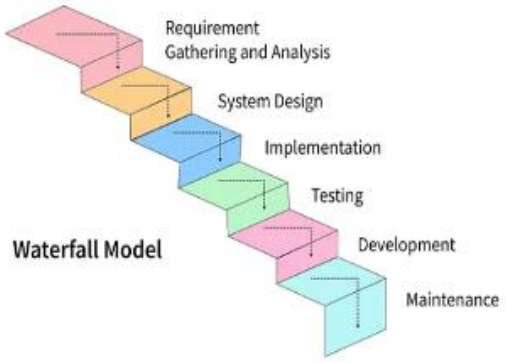
After these evaluation we have to consider how to choose the development model mainly have two model:

Waterfall Model

The Waterfall Model is one of the oldest and most traditional software development methodologies. It is a linear sequential life-cycle model in which each phase must be completed fully before the next phase can begin. There is no overlapping in the phases.

The main phases we have to consider in this case:

- **Requirements Analysis**: All possible requirements of the system to be developed are captured in this phase.
- **System Design**: Helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- **Implementation**: With inputs from the system design, the system is first developed in small programs called units, which are integrated into the next phase.
- **Testing**: All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration, the entire system is tested for any faults and failures.
- **Deployment**: Once functional and non-functional testing is done, the product is deployed in the customer environment or released into the market.
- **Maintenance**: There are always some issues which come up in the client environment. To fix those issues, patches are released. Also, to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.



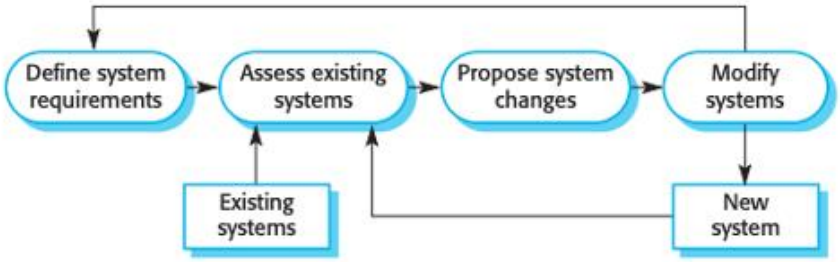
Agile Development Model

Agile Development Model is a type of Incremental model. Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly tested to ensure software quality is maintained. It is used for time-critical applications.

Phases:

- **Planning:** Requirements are gathered for the entire project during the first phase and broken down into individual product development cycles.
- **Design and Development:** Each cycle involves cross-functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.
- **Iteration/Release:** At the end of each iteration, a working product is displayed to the customer and important stakeholders**.**
- **Evaluation:** Customer feedback is gathered with each release and used as input for the next version or iteration.

So, in our case we are using the Agile Model, since requirements change through changing business circumstances, the software that supports the business must also evolve and change. This is technique known as Software evolution



Then another reason to use agile is that we need to do the verification and validation according to the requirement of the user at any level of the development so this Involves checking and reviewing processes and system testing.

So we have to test individual components independently, then we have to do the whole system test. Now the final step is based on accepting testing with customers.

Agile methods

Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:

- Focus on the code rather than the design
- Are based on an iterative approach to software development
- Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.

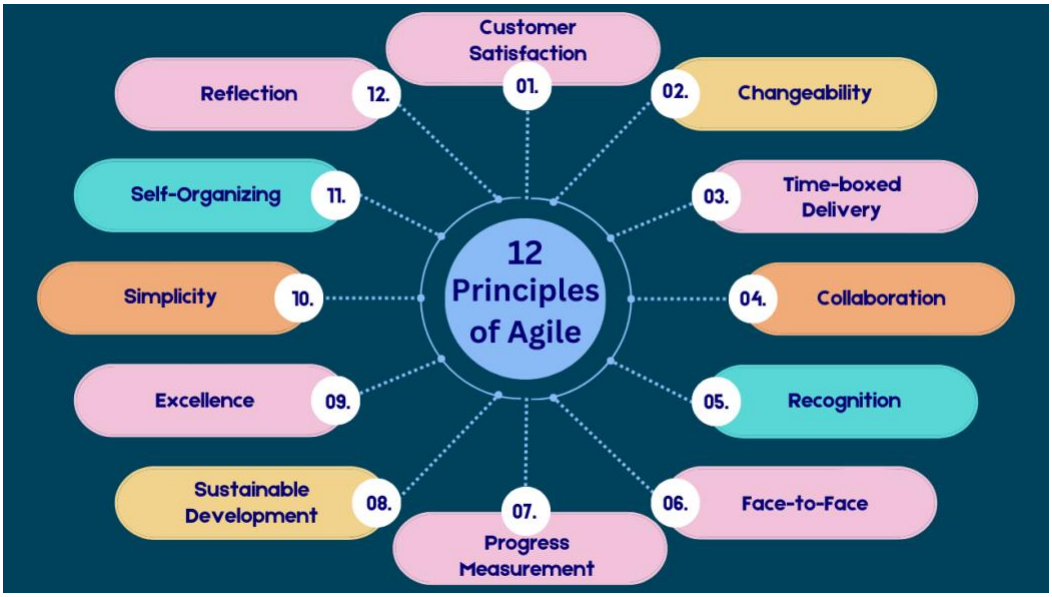
The main aim of the agile methods is to reduce overheads in the software process and to be able to respond quickly to changing requirements without excessive rework.

The Agile Manifesto, formulated in 2001 by a group of software developers, lays down the foundation for modern Agile methodologies. It emphasizes a flexible, iterative approach to software development, promoting collaboration, adaptability, and efficiency; mainly Agile focuses on getting to "Code Complete" faster than with other methods. Here is an expanded explanation of its four key values and how they influence project management.

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

Key Principles of Agile

It also highlights 12 principles which emphasize customer satisfaction, welcoming changing requirements, frequent delivery, collaboration, motivation, face-to-face conversation, working software as the primary measure of progress, sustainable development pace, continuous attention to technical excellence, simplicity, self-organizing teams, and regular reflections to improve effectiveness.



Types of Agile Methodologies

Agile methodologies provide various frameworks that emphasize different aspects of Agile principles and practices like:

1. Extreme Programming (XP) - XP focuses on technical excellence and improving software quality through frequent releases and short development cycles. At the same time Two developers work together at one workstation, improving code quality and fostering knowledge sharing. Regularly integrating code changes into the main codebase to detect issues early.

2. Scrum: - is a framework within the Agile methodology designed to facilitate team collaboration on complex projects. Primarily used in software development, Scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve.

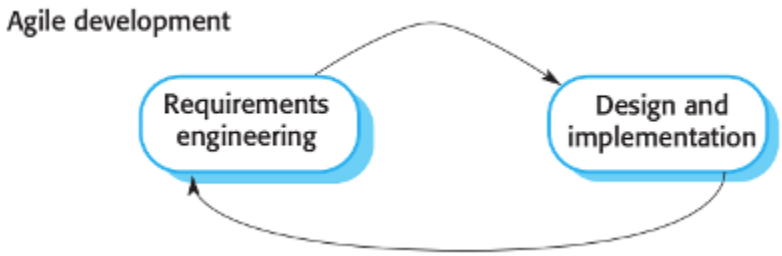
3. Crystal: - is a family of methodologies that adapts to the size and criticality of the project, emphasizing people, interaction, community, skills, talents, and communication. **Crystal Clear:** For small teams (1-6 members) working on non-critical systems.

4. Lean Software Development: - Focuses on minimizing waste and improving efficiency. This methodology is particularly useful in helping businesses with limited resources maximize their development efficiency.

While Product development where a software company is developing a small or medium-sized product for sale. One of the huge benefits of this method is that the customer itself can work with the development where there is a clear commitment from the customer to become involved in the development process.

By other hand most organizations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.

Here we are using an incremental delivery strategy, where we can deliver the software to customers and get rapid feedback from them and add new requirements to our implementation. Agile methods are most effective when the system can be developed with a small co-located team who can communicate informally.



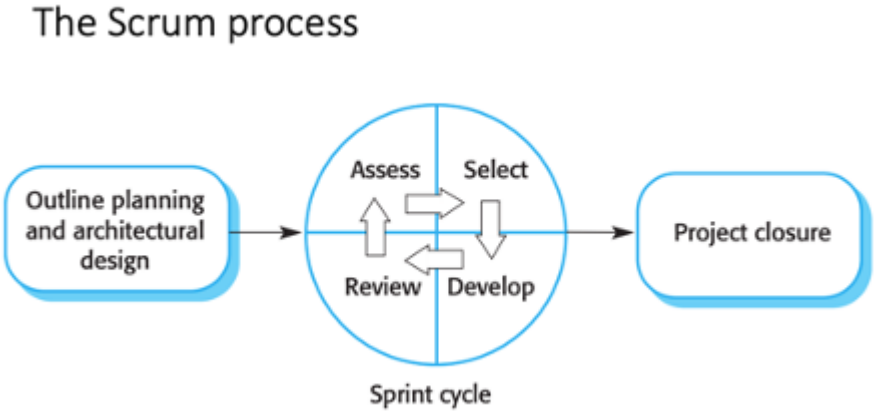
Scrum

Scrum is a framework within the Agile methodology designed to facilitate team collaboration on complex projects. Primarily used in software development, Scrum encourages teams to learn through experiences, self-organize while working on a problem, and reflect on their wins and losses to continuously improve. Many Companies Starting to Use Scrum to Achieve Success, They are finding Scrum an effective tool for addressing the problems. The Rapid growth in the last 3-5 years at leading global companies, like Google, IBM, Nokia, ecc..
The Scrum approach is focusing on managing iterative development rather than specific agile practices. There are three phases in Scrum:

1. The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.

2. The second phase is followed by a series of sprint cycles, where each cycle develops an increment of the system

3. The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.



Key Features of Scrum:

1. **Iterative Process:** Scrum breaks down the development process into short, manageable periods known as **SPRINTS**, which typically last between one to four weeks. Each sprint is aimed at producing a shippable product increment, thereby ensuring regular feedback and the ability to adapt to changing requirements.

2. **Roles:** Scrum defines three primary roles:

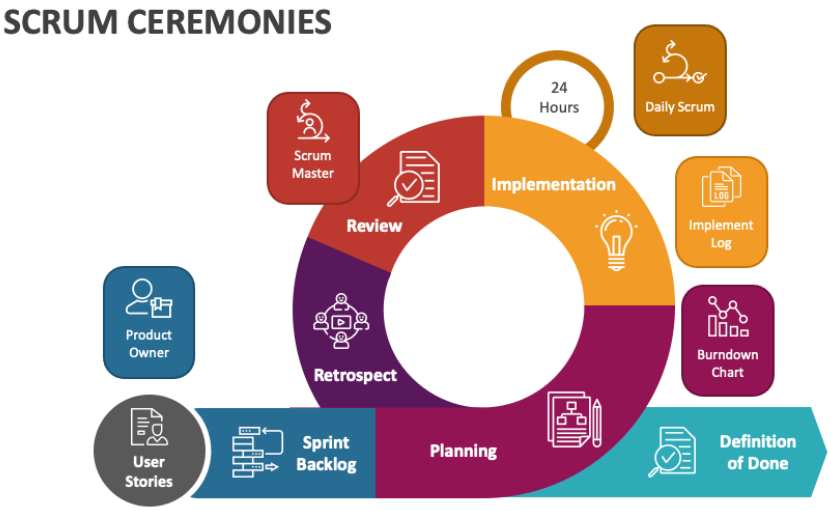
- **Product Owner:** Responsible for defining the project features and prioritizing tasks based on business value. The Product Owner turns this into a single list of what should be produced, prioritized based on business value and risk.

- **Scrum Master:** Acts as a facilitator and coach for the Scrum Team, helping remove obstacles that impede progress and ensuring that the Scrum practices are followed. The Scrum Master supports the team in becoming self-organized and efficient.

- **Development Team:** A cross-functional group of professionals who do the actual work of designing, implementing, testing, and deploying the product. The team works collaboratively to deliver potentially shippable increments of the product at the end of each sprint.

3. **Events Ceremonies:** Scrum incorporates several structured activities to promote regular planning, execution, review, and adaptation, which is know as the **Sprint**:

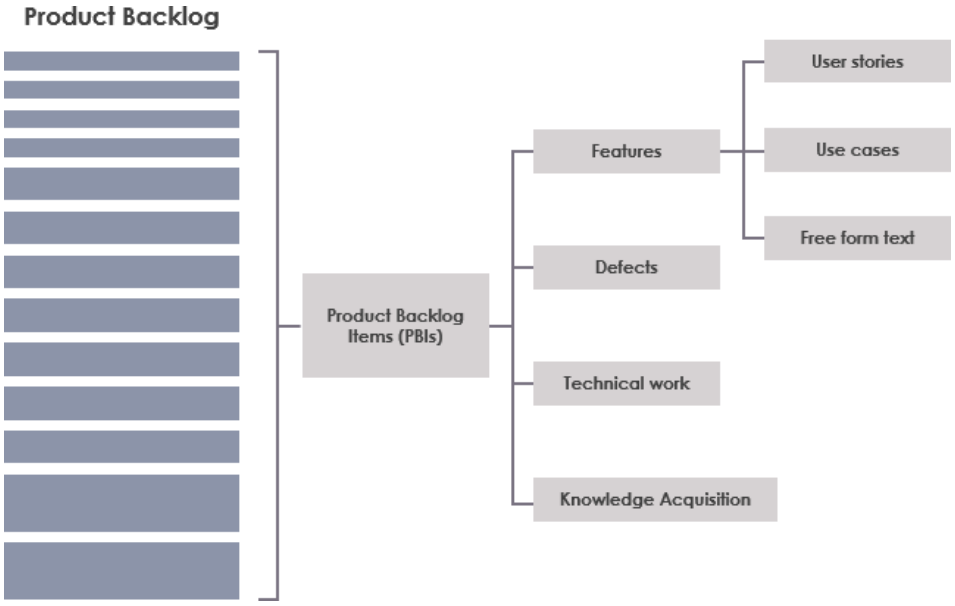
    - **Sprint Planning:** A session where the team selects work from the product backlog to focus on during the upcoming sprint.
    - **Daily Scrum (Daily Stand-up):** A short, daily meeting (typically 15 minutes) where team members synchronize their work and progress, and discuss any impediments they are facing.
    - **Sprint Review:** A meeting at the end of each sprint where the team demonstrates what they have completed during the sprint. This is an opportunity for stakeholders to provide feedback and adjust the backlog as necessary.
    - **Sprint Retrospective:** A meeting after the Sprint Review where the team reflects on the past sprint to identify and agree on continuous process improvements.



4. **Artifacts:** Scrum defines key artifacts that capture and share information about what is to be built, who is building it, and how much work remains:

    - **Product Backlog:** is constantly being revised (items added, removed, modified) by the Product Owner, to maximize the business success of the team's efforts
    - **Sprint Backlog:** lists all the tasks, and the hours remaining for each team for delivering the product increment and achieving the sprint goal.

- **Increment:** The sum of all product backlog items completed during the sprint and all previous sprints, which must be in a usable condition and meet the Scrum team's definition of "done".



The benefits of Scrum are:

- The product is broken down into a set of manageable and understandable chunks.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

Why was Scrum chosen for this project?

We chose to use Scrum for project management due to its structured yet flexible framework, which is ideal for managing projects. Scrum's iterative approach allowed us to break down complex tasks into manageable sprints, making it easier to focus and maintain momentum. This method also facilitated regular evaluation of our progress through sprint reviews and retrospectives, enabling us to adjust the project scope and priorities based on real-time insights and results. Adopting Scrum also helped in maintaining discipline in project execution, ensuring regular progress updates and clear goal-setting. Additionally, using Scrum provided valuable experience in a methodology widely used in the industry.

Software Requirements Analysis

Software Requirements Analysis is a pivotal phase in the software development lifecycle, focusing on gathering, analyzing, and documenting the necessary requirements for a software system. This phase ensures that the development team has a comprehensive understanding of what the stakeholders need and expect from the system, providing a clear blueprint for the subsequent stages of development. The first point to take into consideration is the analysis of our target

Objectives of Requirements Analysis

1. **Understanding User Needs:**
    - The primary objective is to thoroughly understand the needs and expectations of the users and stakeholders. This involves interacting directly with them through various methods such as interviews and surveys.
    - For an e-commerce clothing platform, understanding the need for features like user-friendly navigation, secure payment processing, and efficient order tracking is essential.
2. **Defining System Boundaries:**
    - Clearly defining what the system will and will not do helps set realistic expectations and avoid scope creep. This boundary setting ensures that both stakeholders and the development team are aligned on the project scope.
3. **Identifying Constraints:**
    - Identifying constraints such as budget limits, technological limitations, and regulatory requirements ensures that the project remains feasible and compliant with relevant standards.
4. **Prioritizing Requirements:**
    - Not all requirements hold the same importance. Prioritizing them helps the development team focus on the most critical features first, ensuring that essential functionalities are delivered early.

Functional Requirements

Functional requirements specify what the system should do. For an e-commerce clothing platform, the key functional requirements include user authentication, product management, payment processing, language translation, chatbot integration, dark mode feature, and notification/mailing service.

User Authentication and Authorization

- **Description:** The system must allow users to create accounts, log in, and manage their profiles securely.
- **Details:**
    - Login with email and password.
    - Profile management (update personal information).
    - Give access to the end-point based on role.

Product Management (CRUD operations) (Admin panel)

- **Description:** The system must provide functionalities to create, read, update, and delete product listings.
- **Details:**
    - Add new products with details like name, description, price, and images.
    - Update existing product information.
    - Delete products from the catalog.
    - View product listings and details.
- **Example:** Admins can add new clothing items to the catalog, update prices, or remove discontinued items. Customers can view the products with their details.

Shopping Cart and Checkout Process

- **Shopping Cart Management**:
    - Users have the ability to add, edit, or remove products from their shopping cart.
- **Automated Calculations**:
    - During the checkout process, the system automatically calculates taxes, shipping costs, and applicable discounts.

Payment Processing

- **Description:** The system must enable users to securely complete purchases using various payment methods.
- **Details:**
    - Integration with third-party payment gateways (e.g., PayPal, Stripe).
    - Secure payment processing and storage of transaction records.
    - Support for multiple payment methods (credit/debit cards, on delivery).
- **Example:** Users can choose their preferred payment method and complete the purchase securely.

Language Translation

- **Description:** The system must support multiple languages to cater to a diverse user base.
- **Details:**
    - Integration with a language translation service (e.g., Google Translate API).
    - Automatic translation of product descriptions, user reviews, and other text content.
    - Option for users to select their preferred language.
- **Example:** Users can switch the website language from English to Spanish, Italian, or any other language, and all text content will be translated accordingly.

Chatbot Integration

- **Description:** The system must provide a chatbot for customer support and assistance.
- **Details:**
    - Integration with a chatbot service (e.g., fastbot).
    - Automated responses to common queries (e.g., order status, product information).
- **Example:** Users can interact with a chatbot to check their order status ask about product availability or even to know about the platform.

Dark/Light Modes Feature

- **Description:** The system must offer a dark mode option for better user experience and accessibility.
- **Details:**
    - Toggle button to switch between light and dark modes.
    - Consistent styling across all pages in both modes.
- **Example:** Users can switch to dark mode for a more comfortable viewing experience during night-time browsing.

Notification/Mailing Service

- The system must provide notifications and mailing services to keep users informed about their orders and promotions.
- **Details**
    - Email notifications for order confirmations, and shipping updates.
    - Promotional emails for sales, discounts, and new arrivals.
- **Example:** Users receive an email notification when their order is placed with order details.

Non-functional Requirements

Non-functional requirements describe the system's operational capabilities and constraints that enhance its functionality. These often relate to the quality attributes of the system and how it performs certain functions

1. **Performance**: The application should perform actions such as loading data, processing payments, and responding to user inputs within 2 seconds under normal conditions to ensure a smooth user experience.

2. **Scalability**: The system must be able to scale efficiently to handle increased loads, such as more simultaneous users or data requests, especially during peak usage times. This could be achieved through scalable cloud infrastructure.

3. **Reliability**: The platform should have a high availability rate uptime, and be capable of recovering quickly from any failures without data loss, ensuring continuous service availability.

4. **Security**: Sensitive data, including user personal information and payment details, must be protected against unauthorized access and breaches. Compliance with standards such as PCI DSS for payment data and GDPR for personal data in the EU is required.

5. **Usability**: The application should feature a user-friendly interface that is intuitive to navigate for all user demographics,

6. **Maintainability:** The codebase should be well-documented and structured to facilitate easy updates and maintenance. Adopt a microservices architecture to isolate and address issues without impacting the entire system.

Tools and Technologies Used

In the development of our e-commerce clothing platform, a diverse range of tools and technologies were utilized to ensure a robust, scalable, and user-friendly application. Each technology was chosen based on its suitability for the specific tasks and requirements of the project. Below is a detailed report of the key tools and technologies used in this project:

1. Backend Development

Flask:

- **Description**: Flask is a lightweight and flexible Python web framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications.
- **Usage**: Flask was used for developing the RESTful APIs that handle CRUD operations for product management. Its simplicity and flexibility allowed for rapid development and integration with other backend components.

MySQL:

- **Description**: MySQL is an open-source relational database management system.
- **Usage**: MySQL was chosen for its reliability and performance in handling structured data. It was used to store and manage product information, user data, order details, and other essential records.

2. Frontend Development

Nginx:

- **Description**: Nginx is a high-performance web server that can also be used as a reverse proxy, load balancer, and HTTP cache.
- **Usage**: Nginx was used to serve the frontend of the platform, providing a fast and efficient way to handle HTTP requests and deliver content to users. Its ability to manage high concurrency made it ideal for ensuring smooth user interactions.

Darkmode.js:

- **Description**: Darkmode.js is a lightweight JavaScript library that allows for easy implementation of a dark mode toggle.
- **Usage**: The library was used to implement a dark mode feature, enabling users to switch between light and dark themes seamlessly. It provided a straightforward interface to detect system preferences and apply the appropriate styles.

3. API Tools

- **Chat-Bot Integration**

  FastBots:

  - **Description**: FastBots is a platform that offers easy-to-setup chat-bot solutions for websites and applications.

  - **Usage**: FastBots was used to integrate a chat-bot into the platform, providing users with real-time assistance and support. The chat-bot was configured to handle common queries, guide users through the site, and improve overall user engagement.

- **Multi-Language Translator**

  GTranslate:

  - **Description**: GTranslate is a powerful website translation widget that allows websites to support multiple languages.

  - **Usage**: The GTranslate widget was integrated to provide multi-language support, making the platform accessible to a global audience. It offered seamless language switching and accurate translations, enhancing user experience for non-English speakers.

- **Email Service**

  Python SMTP Library:

  - **Description**: smtplib is a Python library used for sending emails using the Simple Mail Transfer Protocol (SMTP).

  - **Usage**: The smtplib library was used to implement the email notification service within our platform. By creating a custom API, we ensured that users receive timely updates on account activities, promotions, and order statuses. This approach allowed for greater control over email customization and delivery processes.

- **Payment Processing**

  Stripe API:

  Description: Stripe is a technology company that builds economic infrastructure for the internet, providing payment processing software and application programming interfaces for e-commerce websites and mobile applications.

  Usage: Stripe's robust API was integrated into the platform to handle payment processing. It allowed us to securely manage transactions, support multiple payment methods, and ensure compliance with various financial regulations. The API provided a seamless checkout experience for users, managing payment authorizations efficiently.

4. Containerization and Deployment

Docker:

- **Description**: Docker is a platform for developing, shipping, and running applications in containers. Containers allow developers to package applications with all their dependencies, ensuring consistency across different environments.

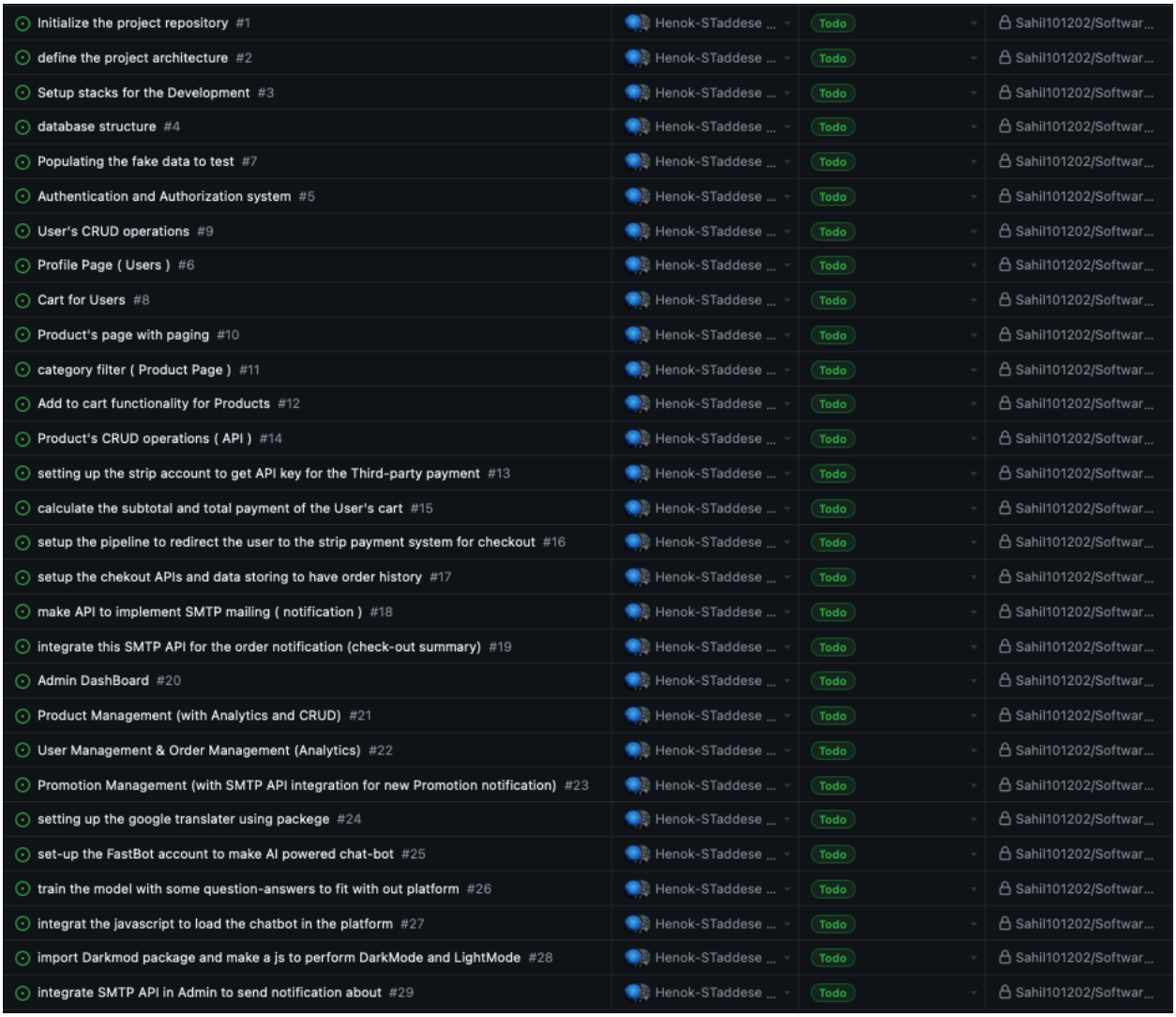
- **Usage**: The entire project was containerized using Docker, which facilitated consistent development and deployment environments. Docker allowed us to package the backend (Flask and MySQL), frontend (Nginx), and other services into isolated containers, simplifying the deployment process and enhancing scalability.

In general, the strategic selection and integration of these tools and technologies were important in the successful development of our e-commerce clothing platform. By leveraging Docker for containerization, Flask and MySQL for backend development, Nginx for frontend serving, Darkmode.js for dark mode functionality, FastBots for chat-bot integration, GTranslate for multi-language support, the smtplib library for email notifications, and Stripe for payment processing, we created a robust, scalable, and user-friendly application. These technologies not only met the immediate needs of the project but also provided a strong foundation for future enhancements and scalability.

Actual Sprints Development and Results

In this section, we will go deep into the actual development process of our e-commerce clothing platform using Agile methodology, specifically focusing on Scrum. We will detail the planning, execution, review, and retrospective of each sprint, highlighting key achievements, challenges, and solutions. The development was divided into six sprints, each targeting specific functionalities to ensure a structured and efficient progression towards our final product.

Product BackLogs



Sprint 1: Project Setup and Core Architecture

This sprint focused on establishing the basic infrastructure and operational protocols necessary for the development of the e-commerce platform. Key activities included setting up the version control repository, defining the system architecture, creating the initial database schema for critical components like users and products, and configuring the development environment for effective team collaboration.

Sprint Development: backlog for the first sprint, inherited from the original backlog table

User story:

- To create a robust development environment that supports efficient coding, testing, and deployment processes.

- To outline and implement a scalable and secure architecture that will support the e-commerce platform's functionality and anticipated growth.

Acceptance Criteria:

1. **Development Environment Setup**

    - Configuration of development tools and servers. Establishment of local and staging environments for development and testing.

2. **Version Control Setup**

    - Setup of Git repositories to manage and track code changes. Definition of branching and merging strategies to ensure smooth collaboration among development team members.Ensured initial commit included the basic project structure and essential documents, such as README and CONTRIBUTING files.

3**. System Architecture Definition and Documentation**

- Collaborated with system architects to define the high-level architecture of the e-commerce platform.Documented the architecture, including diagrams and descriptions of different components and how they interact. This documentation was made available in the repository for team reference.

4. **Database Design**

- Designing the fundamental structure of the platform, including database MySQL schemas, application services, and user interface components. Selection and integration of key technologies and frameworks that will support both frontend and backend development.

Challenges and Solutions

- **Problem**: Variations in developer environments can lead to the "works on machine" syndrome, causing delays and inconsistencies in development and testing. At the same time in order to try what other developers are doing and run those work on our computers may ask to download some new version of the packages and we can get some other errors.

- **Solution**: Created a detailed setup guide that included step-by-step instructions for setting up the development environment.

- Configuration of Docker containers to standardize the development environment regardless of the underlying operating system.

Sprint Review and Retrospective :

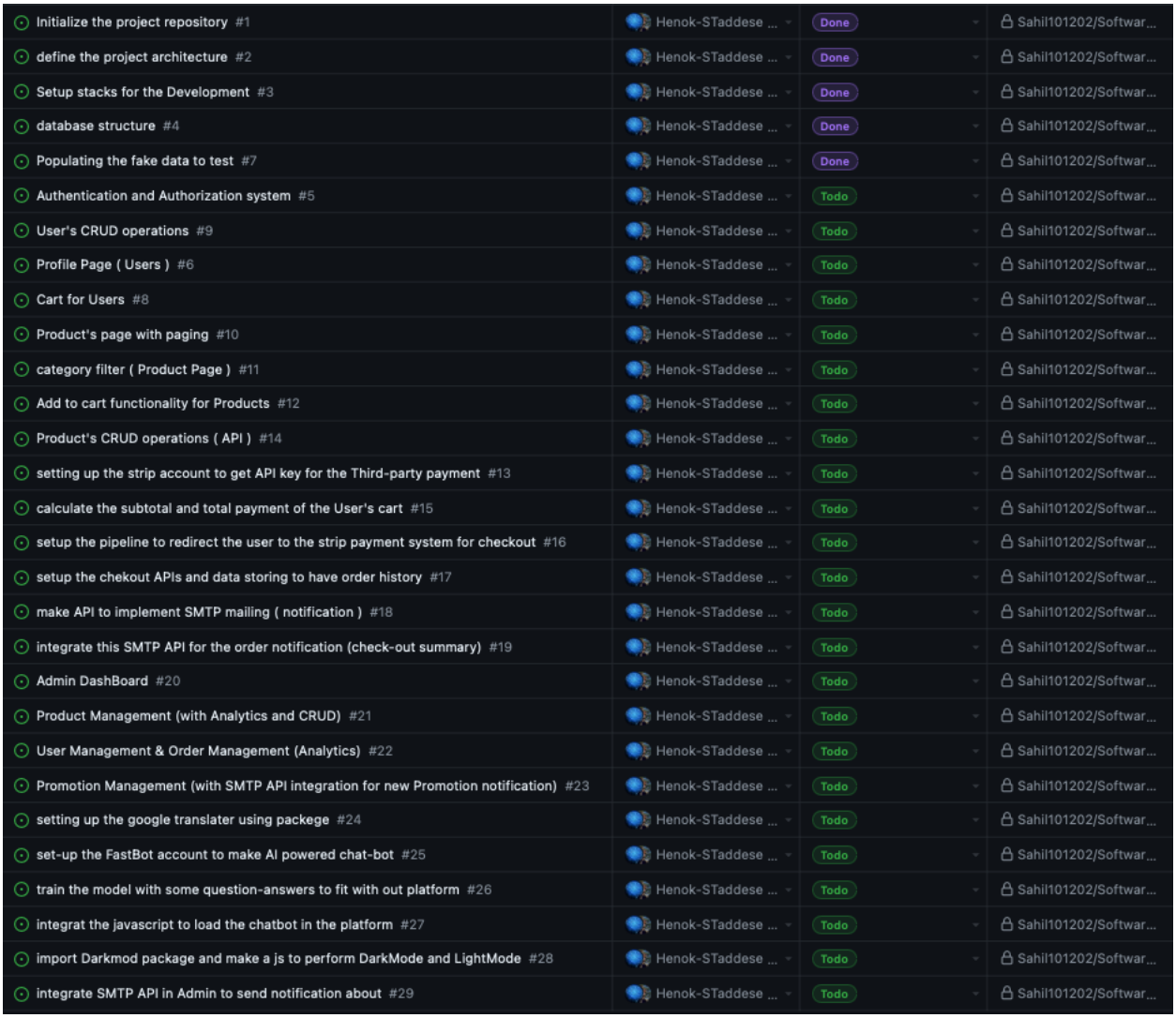
The sprint development went very well and the team successfully configured all necessary development tools and servers. Local and staging environments were set up without significant delays, facilitating immediate start on development tasks.

At the same time we had some Initial delays due to the configuring some development tools, which slightly postponed some team members' start on their tasks.

Sprint 2: User Authentication and Authorization

This Sprint 2, the focus is on establishing a robust user authentication system for the e-commerce platform. Implementing comprehensive user authentication features that include registration, login, and profile management functionalities.

Sprint planning:



Sprint Development: backlog for the second sprint from the original backlog table

User Story 1: Authentication and Authorization system

To implement a robust authentication and authorization system, ensuring secure and differentiated access for users and administrators.

Acceptance Criteria:

- **Authentication System**: Utilize JWT for secure session management and user authentication.
- **Role-Based Access Control**: Implement role-based access to distinguish between regular users and admins.

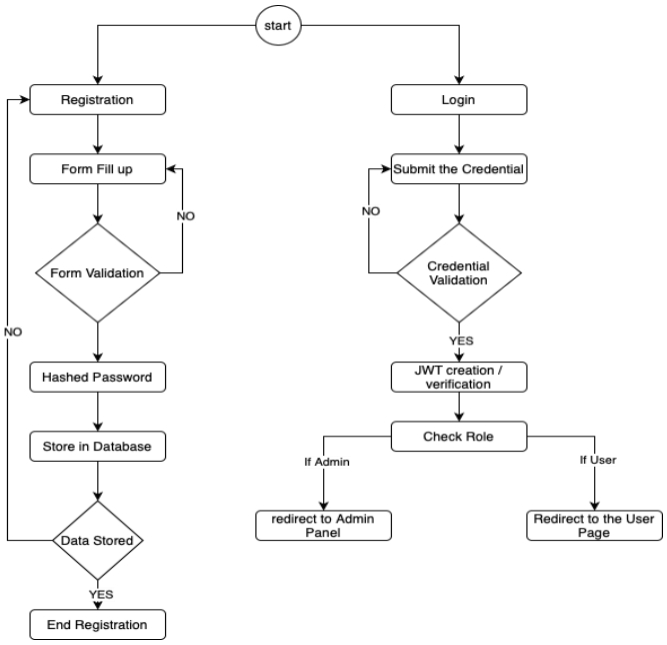
Sprint Planning:

- **Goals**: Establish a secure and scalable authentication system that supports distinct user roles and sessions.
- **Tasks**:
    - Develop APIs to handle JWT-based authentication and session management.
    - Implement role-based access control within the application.

Sprint Execution:

- **Backend Development**: Develop Flask APIs for authentication, utilizing JWT for session management.
- **Frontend Development**: Create login and admin interfaces using Nginx, ensuring secure and separate access.

Activity diagram:



User Story 2: CRUD Operations

To develop a comprehensive system for user CRUD operations, enabling admins to manage user accounts efficiently.

Acceptance Criteria:

- **CRUD Functionality**: Admins can create, read, update, and delete user accounts through a secure interface.

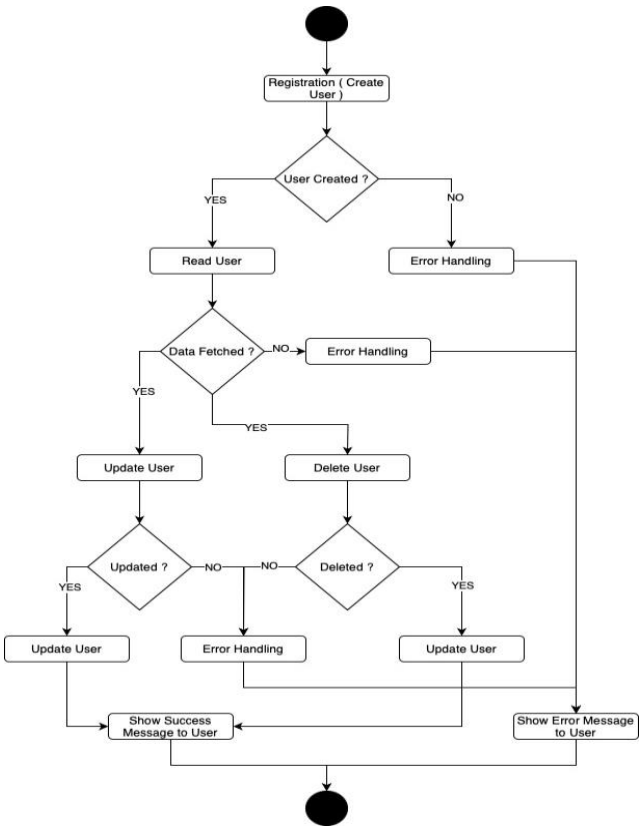
Sprint Planning:

- **Goals**: Implement a complete set of CRUD operations for user management by admins.
- **Tasks**:
    - Develop APIs for handling CRUD operations on user data.
    - Implement a user management interface for admins.

Sprint Execution:

- **Backend Development**: Create APIs using Flask and MySQL for CRUD operations.
- **Frontend Development**: Develop a user management dashboard using Nginx.

Activity diagram:



User Story 3: **Profile Page (with the user related details)**

To implement a user profile page where users can view and edit their personal information securely.

Acceptance Criteria:

- **Profile Management**: Users can securely update their personal information and can also see orders they made.

Sprint Planning:

- **Goals**: Enable users to manage their profiles, enhancing personal data control.
- **Tasks**:
    - Develop APIs for profile viewing and updates.
    - Create a user-friendly profile management interface.

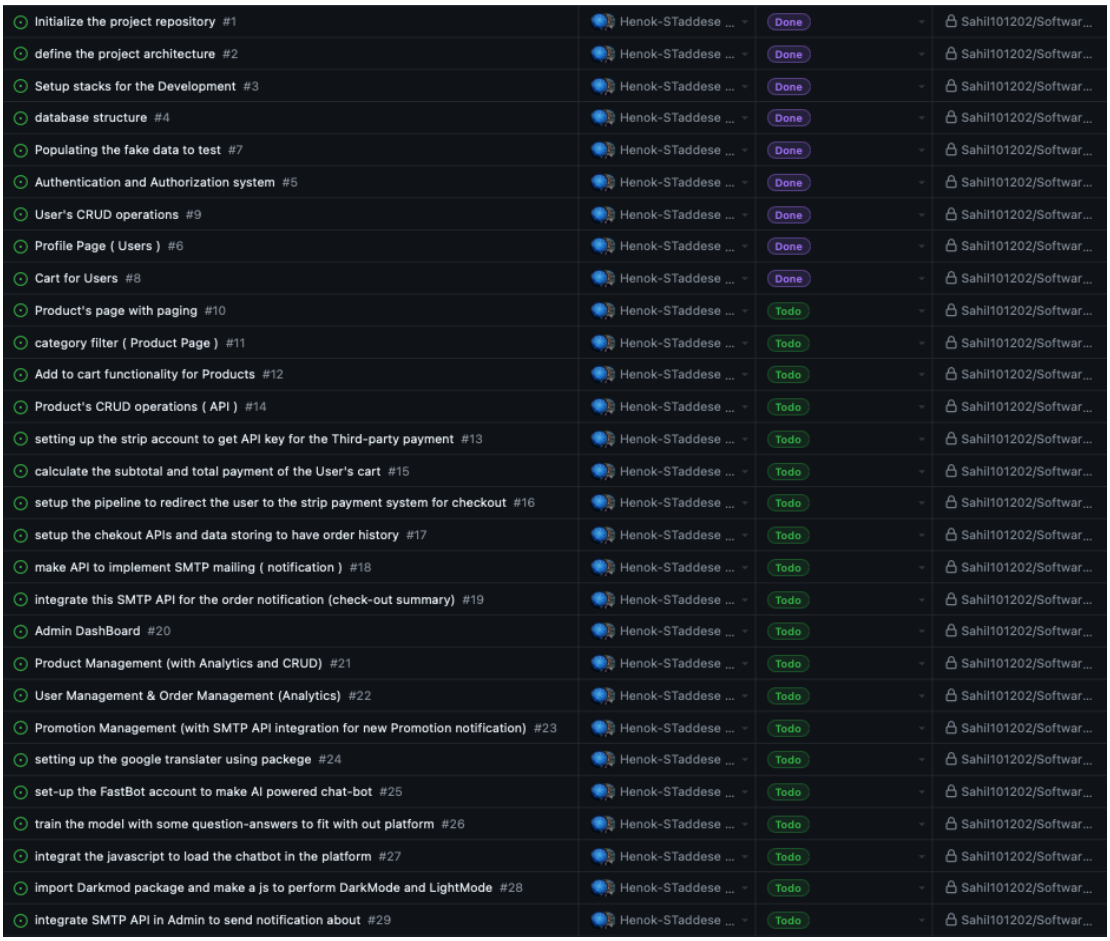
Sprint Execution:

- **Backend Development**: Develop Flask APIs for user profile management.
- **Frontend Development**: Implement the profile page using Nginx with secure data handling with the JWT verification.

Sprint Review and Retrospective:

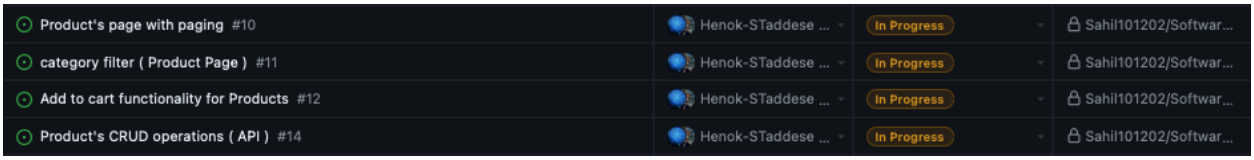
We effectively implemented a user registration form that captures essential information while ensuring data integrity through input validation. The team used secure password storage using hashing algorithms and was successfully integrated. The creation of the login interface was accomplished smoothly, providing a seamless user experience. We had to go through some confusion over the multi-factor authentication setup process, indicating a need for clearer instructions or a more intuitive interface.

Sprint 3: Product Management (CRUD Operations)



Initialize the project repository #1	Henok-STaddese ...	Done	Sahil101202/Softwar...
define the project architecture #2	Henok-STaddese ...	Done	Sahil101202/Softwar...
Setup stacks for the Development #3	Henok-STaddese ...	Done	Sahil101202/Softwar...
database structure #4	Henok-STaddese ...	Done	Sahil101202/Softwar...
Populating the fake data to test #7	Henok-STaddese ...	Done	Sahil101202/Softwar...
Authentication and Authorization system #5	Henok-STaddese ...	Done	Sahil101202/Softwar...
User's CRUD operations #9	Henok-STaddese ...	Done	Sahil101202/Softwar...
Profile Page (Users) #6	Henok-STaddese ...	Done	Sahil101202/Softwar...
Cart for Users #8	Henok-STaddese ...	Done	Sahil101202/Softwar...
Product's page with paging #10	Henok-STaddese ...	Todo	Sahil101202/Softwar...
category filter (Product Page) #11	Henok-STaddese ...	Todo	Sahil101202/Softwar...
Add to cart functionality for Products #12	Henok-STaddese ...	Todo	Sahil101202/Softwar...
Product's CRUD operations (API) #14	Henok-STaddese ...	Todo	Sahil101202/Softwar...
setting up the strip account to get API key for the Third-party payment #13	Henok-STaddese ...	Todo	Sahil101202/Softwar...
calculate the subtotal and total payment of the User's cart #15	Henok-STaddese ...	Todo	Sahil101202/Softwar...
setup the pipeline to redirect the user to the strip payment system for checkout #16	Henok-STaddese ...	Todo	Sahil101202/Softwar...
setup the chekout APIs and data storing to have order history #17	Henok-STaddese ...	Todo	Sahil101202/Softwar...
make API to implement SMTP mailing (notification) #18	Henok-STaddese ...	Todo	Sahil101202/Softwar...
integrate this SMTP API for the order notification (check-out summary) #19	Henok-STaddese ...	Todo	Sahil101202/Softwar...
Admin DashBoard #20	Henok-STaddese ...	Todo	Sahil101202/Softwar...
Product Management (with Analytics and CRUD) #21	Henok-STaddese ...	Todo	Sahil101202/Softwar...
User Management & Order Management (Analytics) #22	Henok-STaddese ...	Todo	Sahil101202/Softwar...
Promotion Management (with SMTP API integration for new Promotion notification) #23	Henok-STaddese ...	Todo	Sahil101202/Softwar...
setting up the google translater using packege #24	Henok-STaddese ...	Todo	Sahil101202/Softwar...
set-up the FastBot account to make AI powered chat-bot #25	Henok-STaddese ...	Todo	Sahil101202/Softwar...
train the model with some question-answers to fit with out platform #26	Henok-STaddese ...	Todo	Sahil101202/Softwar...
integrat the javascript to load the chatbot in the platform #27	Henok-STaddese ...	Todo	Sahil101202/Softwar...
import Darkmod package and make a js to perform DarkMode and LightMode #28	Henok-STaddese ...	Todo	Sahil101202/Softwar...
integrate SMTP API in Admin to send notification about #29	Henok-STaddese ...	Todo	Sahil101202/Softwar...

Sprint Development: backlog for the third sprint from the original backlog table



Product's page with paging #10	Henok-STaddese ...	In Progress	Sahil101202/Softwar...
category filter (Product Page) #11	Henok-STaddese ...	In Progress	Sahil101202/Softwar...
Add to cart functionality for Products #12	Henok-STaddese ...	In Progress	Sahil101202/Softwar...
Product's CRUD operations (API) #14	Henok-STaddese ...	In Progress	Sahil101202/Softwar...

User Story 1: Product Page with Paging

To implement a product page with paging functionality, allowing users to navigate through products efficiently by viewing a subset of products at a time. This feature is essential for enhancing user experience and managing large inventories.

Acceptance Criteria:

1. **Paging Implementation:**

    - Developed APIs to fetch products with paging support.
    - Designed frontend to display products with navigation controls (next, previous, page numbers).

Goals: Implement paging functionality to improve user navigation and manage large inventories on the product page.

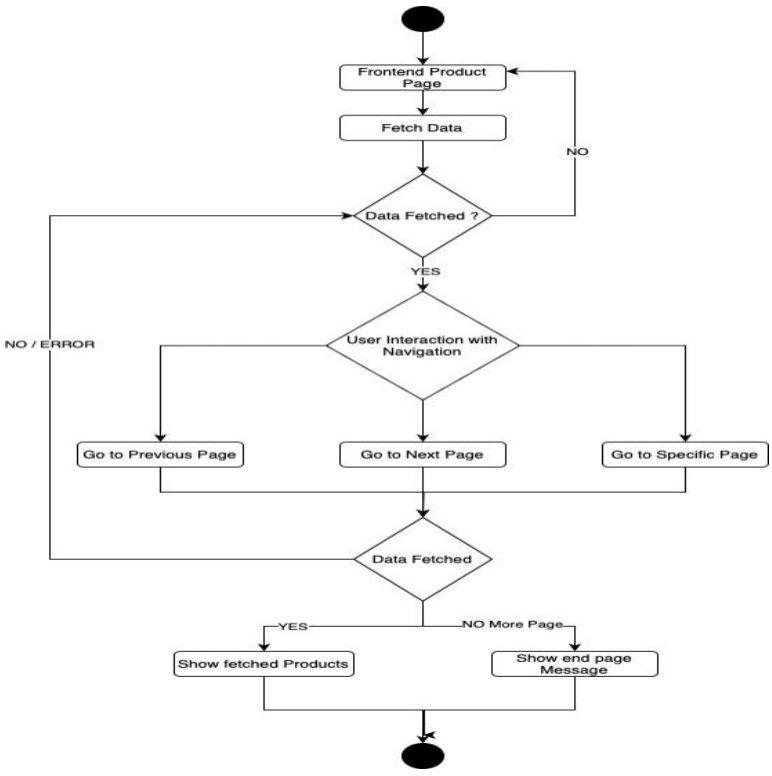
Tasks:

- **Develop APIs:** Create APIs to fetch products with paging parameters (page number, page size).
- **Implement Frontend:** Design and develop the user interface to support product paging, including navigation controls.

Activities:

- **Backend Development:** Developed APIs using Flask and MySQL to support fetching products with paging parameters.
- **Frontend Development:** Created the product page interface using Nginx, incorporating navigation controls for paging.

Activity diagram:



User Story 2: Category Filter (Product Page)

To implement a category filter on the product page, allowing users to filter products based on categories. This feature helps users find products quickly and enhances the overall shopping experience.

Acceptance Criteria:

1. **Category Filter Implementation:**
    - Developed APIs to fetch products based on selected categories.
    - Designed frontend to allow users to select and apply category filters.

Goals: Implement category filter functionality to enhance product search and browsing experience on the product page.

Tasks:

- **Develop APIs:** Create APIs to fetch products filtered by category.
- **Implement Frontend:** Design and develop the user interface to support category filtering.

Activities:

- **Backend Development:** Developed APIs using Flask and MySQL to support fetching products based on selected categories.
- **Frontend Development:** Created the product page interface using Nginx, incorporating category filter options.

Frontend result:

User Story 3: Add to Cart Functionality (Product Page)

To implement the add-to-cart functionality on the product page, allowing users to add products to their cart directly from the product listing. This feature is crucial for streamlining the shopping experience.

Acceptance Criteria:

1. **Add to Cart Implementation:**
    - Developed APIs to handle adding products to the cart.
    - Designed frontend to include add-to-cart buttons on product listings.

Goals: Implement add-to-cart functionality to enhance the shopping experience and streamline the process of adding products to the cart.

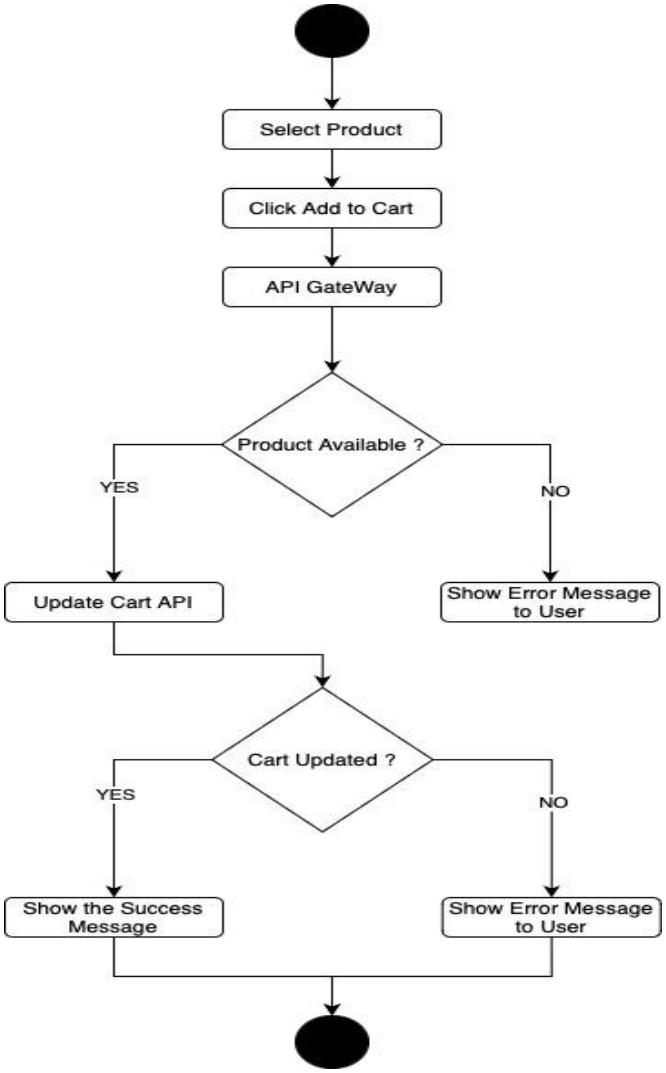
Tasks:

- **Develop APIs:** Create APIs to handle adding products to the cart.
- **Implement Frontend:** Design and develop the user interface to include add-to-cart buttons on product listings.

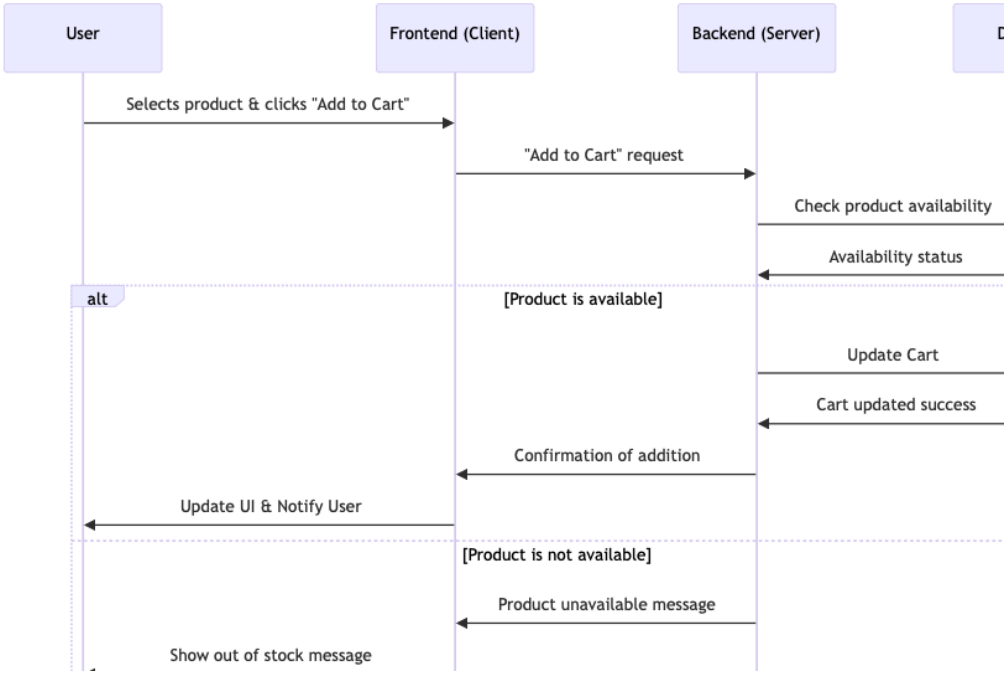
Activities:

- **Backend Development:** Developed APIs using Flask and MySQL to support adding products to the cart.
- **Frontend Development:** Created the product page interface using Nginx, incorporating add-to-cart buttons on product listings.

Activity diagram:



Sequence diagram:



Challenges:

- **Real-time Cart Updates:** Ensuring the cart updates in real-time as products are added.

Solutions:

- **Asynchronous API Calls:** Implemented asynchronous API calls to update the cart in real-time without page reloads.

User Story 4: Product's CRUD Operation (API)

To implement the essential CRUD operations for product management, enabling the creation, reading, updating, and deleting of products. This feature is fundamental for maintaining the product catalog.

Acceptance Criteria:

1. **CRUD Operations Implementation:**
    - Developed APIs to handle creating, reading, updating, and deleting products.
    - Ensured robust validation and error handling for all CRUD operations.

Goals: Implement CRUD operations for effective product management and maintenance of the product catalog.

Tasks:

- **Develop APIs:** Create APIs to handle CRUD operations for products.
- **Implement Validation:** Ensure robust validation and error handling for all CRUD operations.

Activities:

- **Backend Development:** Developed APIs using Flask and MySQL to support creating, reading, updating, and deleting products.
- **Validation and Error Handling:** Implemented robust validation and error handling mechanisms for all CRUD operations.

Activity Diagram:

Challenges:

- **Data Integrity:** Ensuring data integrity and consistency during CRUD operations.

Solutions:

- **Comprehensive Validation:** Implemented comprehensive validation rules and error handling to maintain data integrity.

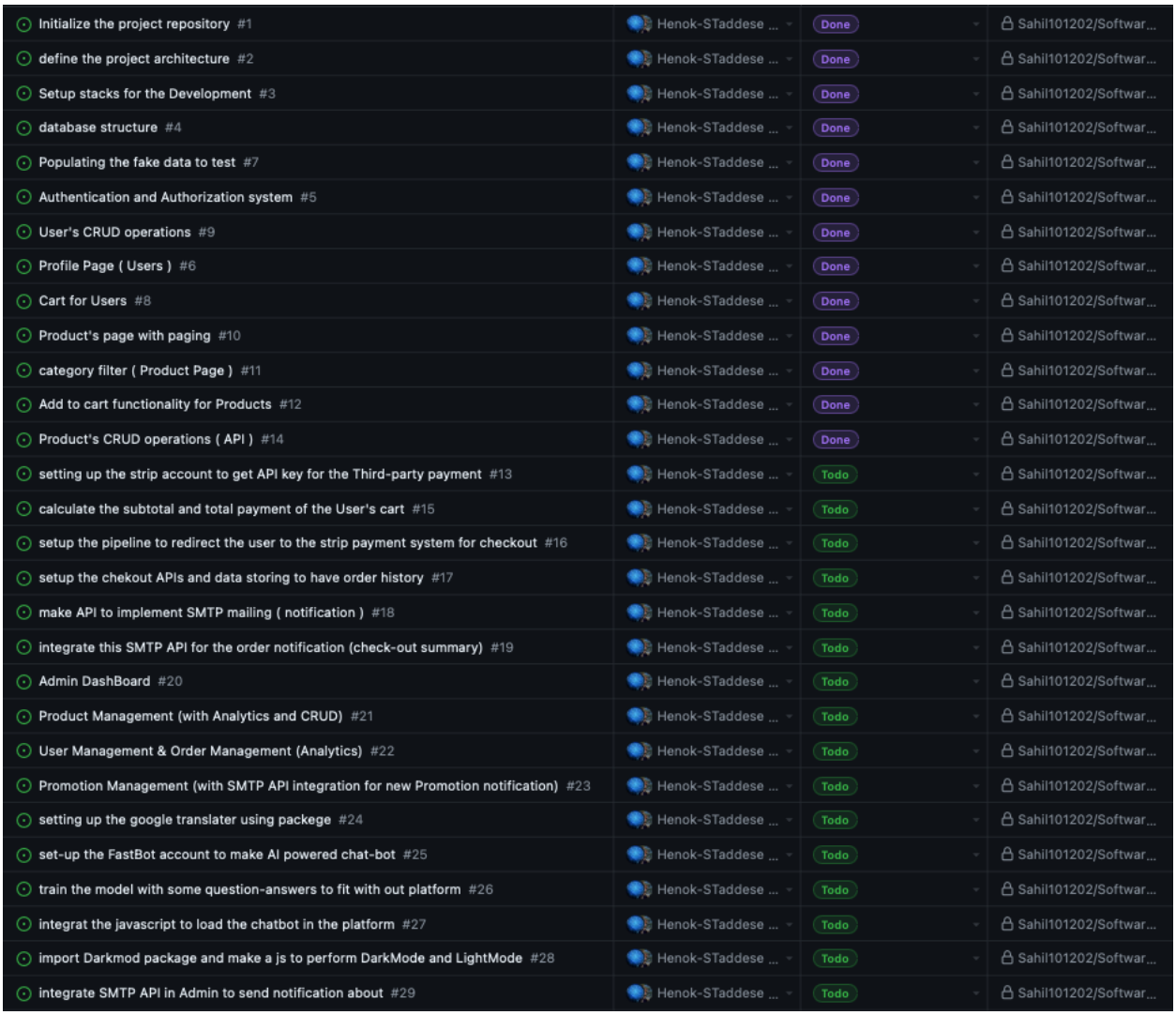
Sprint Review and Retrospective:

The database schema was effectively designed and implemented, supporting complex product attributes such as categories, prices, and inventory status. Integration of database operations with the backend system was accomplished, enabling real-time data processing and immediate updates. User interfaces developed during this sprint were well-received for their usability and design consistency. The responsiveness across different devices was achieved, enhancing user experience and accessibility. The initial complexity of the database schema led to difficulties in adapting and extending it to accommodate unexpected product attributes.

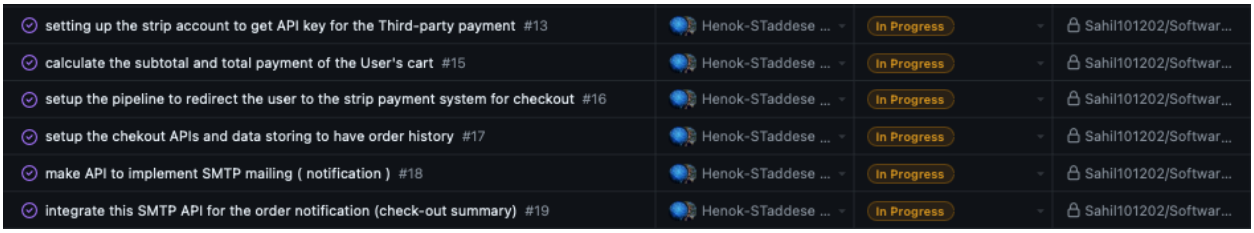
By the end of Sprint 3, we successfully implemented the essential product management features, including product paging, category filtering, add-to-cart functionality, and CRUD operations. These features set a strong precedent for future enhancements and iterations. The feedback and insights gathered during the review and retrospective meetings will guide improvements in subsequent sprints.

Sprint 4: Payment Processing and Mailing Service

Sprint planning:



Sprint development: backlog for the fourth sprint



User Story 1: Setting Up the Stripe Account to Get API Key for Third-Party Payment

To configure and integrate a secure payment processing system using Stripe. This ensures safe and efficient handling of transactions on the e-commerce platform.

Acceptance Criteria:

1. **Stripe Account Configuration:**
    - Set up a Stripe account to obtain the necessary API keys.
    - Ensure secure API integration with the platform.

Goals: Implement a secure payment processing system using Stripe to handle transactions efficiently.

Tasks:

- **Set Up Stripe Account:** Configure the Stripe account to obtain API keys.
- **API Integration:** Integrate the Stripe API with the platform.

Activities:

- **Stripe Account Setup:** Configured the Stripe account and obtained API keys.
- **API Integration:** Integrated the Stripe API securely with the platform, ensuring safe handling of payment transactions.

User Story 2: Calculate the Subtotal and Total Payment of the User's Cart

To develop a system that accurately calculates the subtotal and total payment of the user's cart, ensuring a smooth and transparent checkout process.

Acceptance Criteria:

1. **Payment Calculation:**
    - Implement logic to calculate the subtotal and total payment of the user's cart.
    - Ensure accuracy in the calculations.

Goals: Implement a system to calculate the subtotal and total payment of the user's cart.

Tasks:

- **Develop Calculation Logic:** Implement logic to calculate subtotal and total payment.
- **Ensure Accuracy:** Verify the accuracy of the calculations.

Activities:

- **Implementation:** Developed and integrated the logic to calculate the subtotal and total payment of the user's cart.
- **Verification:** Conducted thorough testing to ensure accuracy in the calculations.

User Story 3: Setup the Pipeline to Redirect the User to the Stripe Payment System for Checkout

To set up a pipeline that redirects users to the Stripe payment system for a seamless checkout experience.

Acceptance Criteria:

1. **Checkout Pipeline:**
    - Develop a pipeline that redirects users to the Stripe payment system.
    - Ensure a seamless and secure checkout experience.

Sprint Planning: Goals: Implement a pipeline to redirect users to the Stripe payment system for checkout.

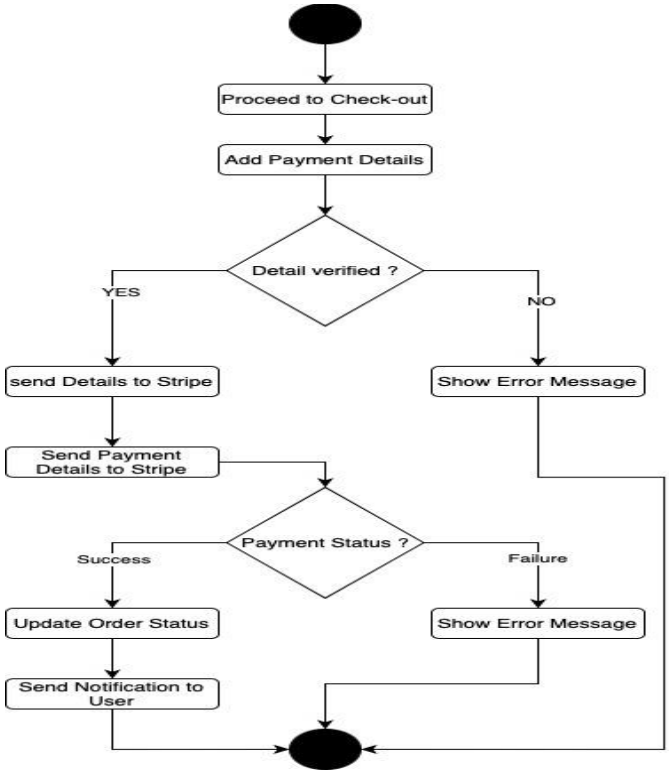
Tasks:

- **Develop Checkout Pipeline:** Create a pipeline to redirect users to the Stripe payment system.
- **Ensure Security:** Ensure the pipeline provides a secure checkout experience.

Activities:

- **Pipeline Development:** Developed a pipeline to redirect users to the Stripe payment system.
- **Security Measures:** Implemented security measures to ensure a safe checkout experience.

Activity diagram:



User Story 4: Setup the Checkout APIs and Data Storing to Have Order History

To develop APIs and data storage solutions to maintain order history, enhancing the platform's ability to manage and track orders.

Acceptance Criteria:

1. **Checkout APIs and Order History:**
    - Develop APIs to handle checkout processes and store order data.
    - Ensure reliable data storage for maintaining order history.

Goals: Implement APIs and data storage solutions to maintain order history.

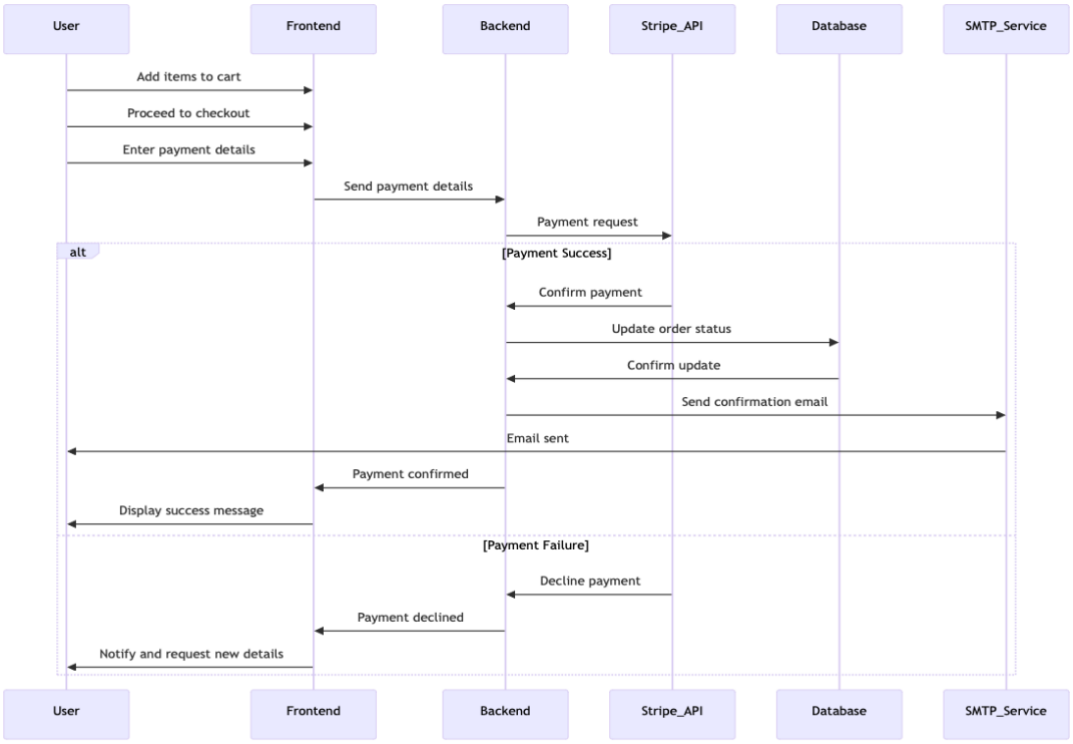
Tasks:

- **Develop Checkout APIs:** Create APIs to handle checkout processes.
- **Implement Data Storage:** Ensure reliable storage of order data.

Activities:

- **API Development:** Developed checkout APIs and integrated them with the platform.
- **Data Storage:** Implemented data storage solutions to maintain order history.

Sequence diagram:



User Story 5: Make API to Implement SMTP Mailing (Notification) and Integrate This SMTP API for Order Notification (Checkout Summary)

To implement an SMTP API for sending notifications and integrate it to send order summaries after checkout, enhancing communication and user experience.

Acceptance Criteria:

1. **SMTP API Implementation:**
    - Develop and integrate the SMTP API to send notifications.
    - Configure the API to send order summaries after checkout.

Goals: Implement the SMTP API for sending notifications and integrate it for order summaries.

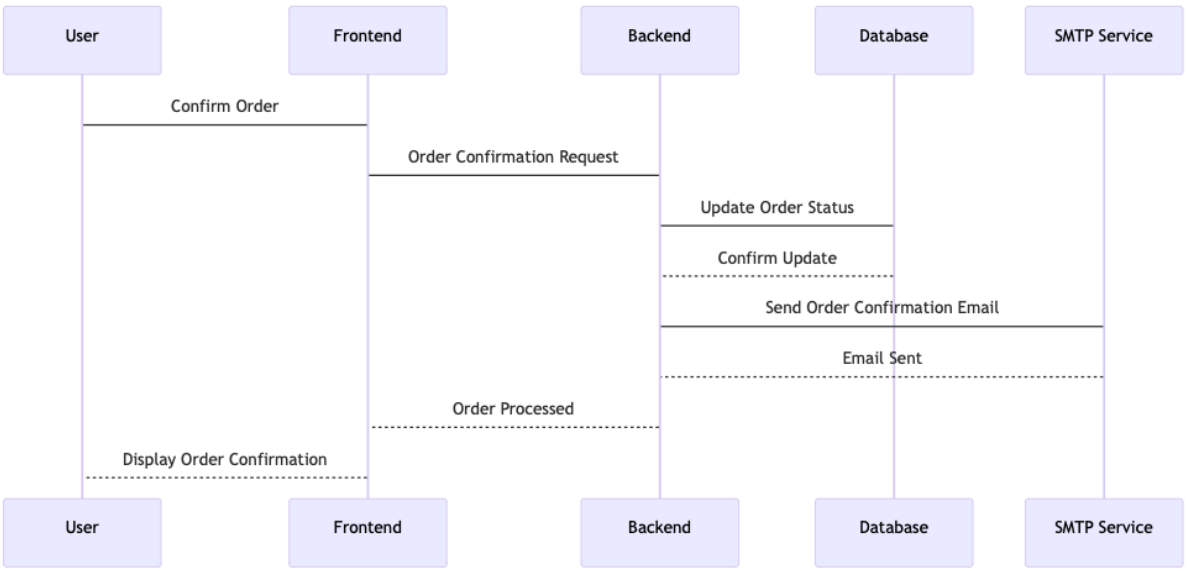
Tasks:

- **Develop SMTP API:** Create and configure the SMTP API for notifications.
- **Integrate with Checkout:** Ensure integration with the checkout process to send order summaries.

Activities:

- **API Development:** Developed the SMTP API and integrated it with the platform.
- **Functionality:** Configured the API to send order summaries after checkout.

Sequence diagram:



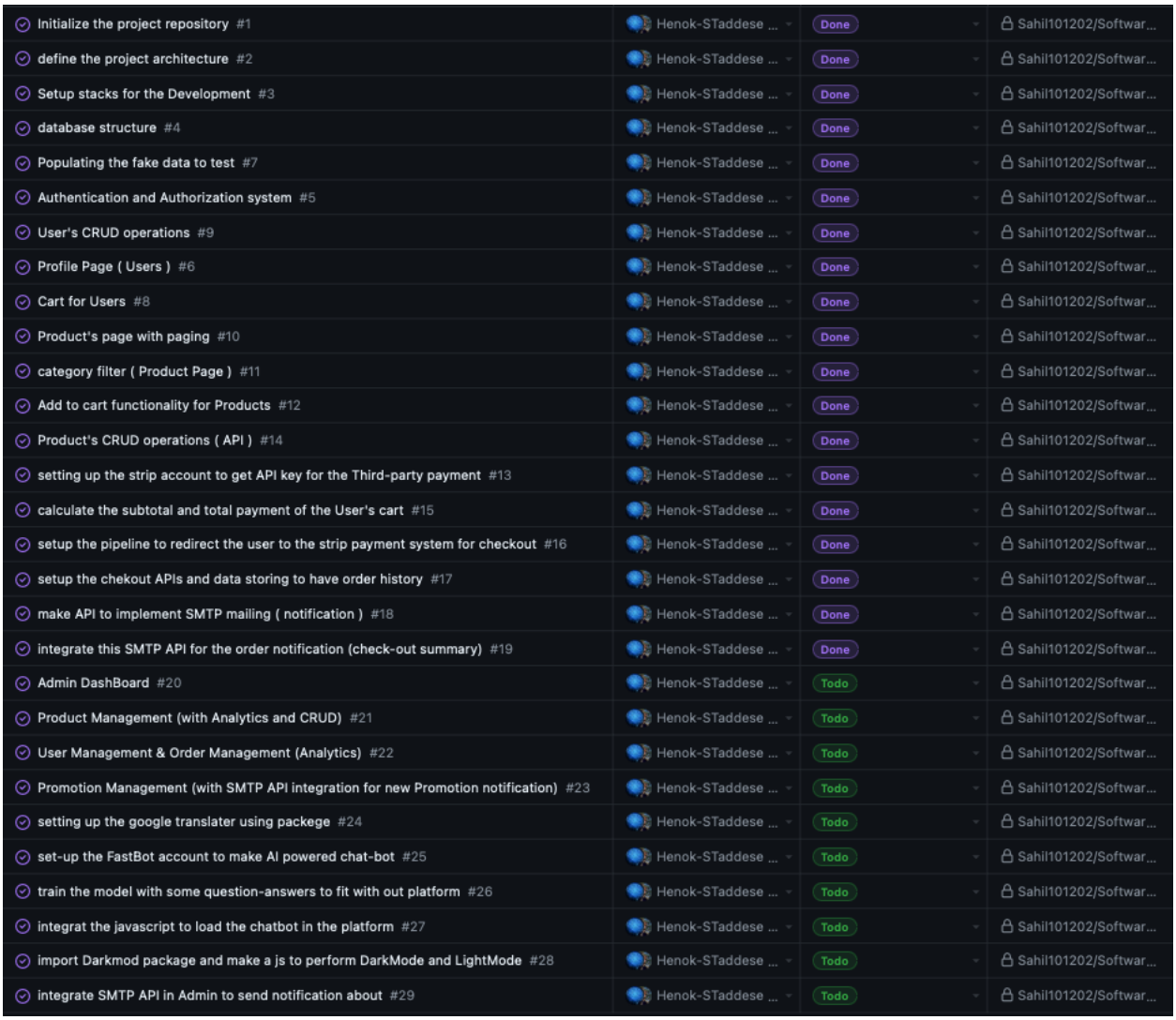
Sprint Review and Retrospective:

The shopping cart functionality was implemented successfully, allowing users to add, modify, and remove items. The cart feature works effectively across Flask API, by providing convenience for the user. We successfully implemented complex backend logic for order processing, including inventory checks and order confirmations.

By the end of Sprint 4, we successfully implemented a secure payment processing system and a reliable mailing service. These features significantly enhance the platform's functionality and user experience.

The feedback and insights gathered during the review and retrospective meetings will guide future improvements and iterations.

Sprint 5: Admin Panel



Sprint 5 is dedicated to enhancing the Admin Panel that was initially established in previous sprints. The goals for this sprint were to expand the panel's capabilities, streamline user interfaces, and implement additional features that facilitate more efficient administration of the e-commerce platform.

User Story 1: Admin Dashboard

To enhance the admin dashboard to provide a more comprehensive and interactive overview of platform activities.

Acceptance Criteria:

- **Dashboard**: Displays key statistics and analytics, including product categories, data, user activity, and inventory levels.

- **Integration**: The dashboard should integrate seamlessly with existing data sources to provide real-time updates.

Sprint Planning:

- **Goals**: Expand the admin dashboard with real-time analytics and improved data visualization.
- **Tasks**:
    - Implement new analytics features.
    - Design an interactive admin interface.
    - Integrate real-time data processing for up-to-date information display.

Sprint Execution:

- **Backend Development**: Enhanced the analytics engine to provide real-time data for the dashboard.

- **Frontend Development**: Designed and implemented a user-friendly dashboard interface using Nginx.

User Story 2: Product Management (with Analytics and CRUD)

Objective: To expand product management capabilities, incorporating advanced analytics and full CRUD operations for products.

Acceptance Criteria:

- **Product Management**: Allows for the addition, removal, and modification of product listings.
- **Analytics**: Provides detailed analytics for product performance.

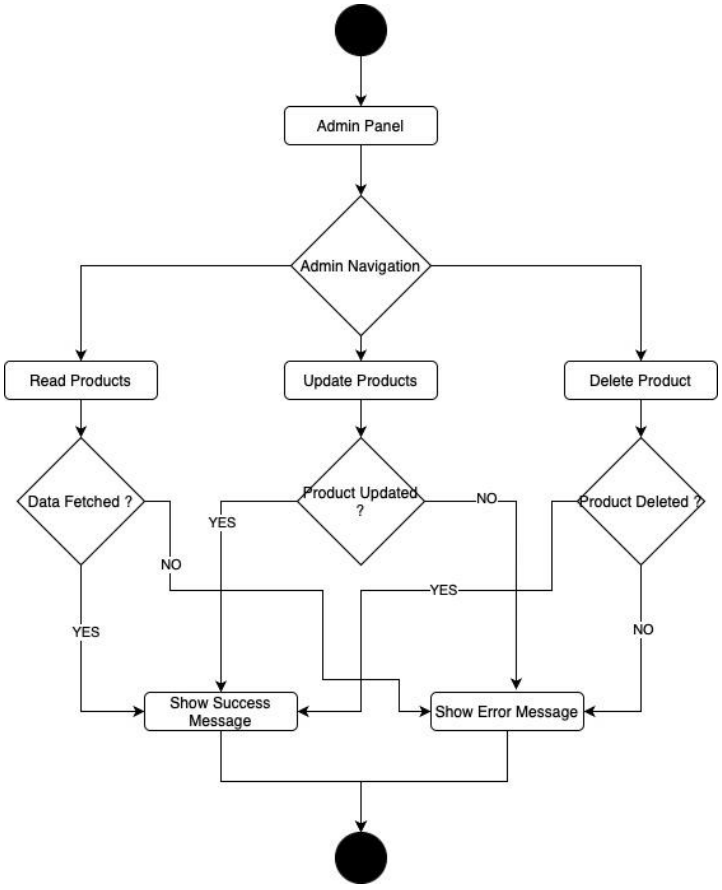
Sprint Planning:

- **Goals**: Implement advanced product management features, including analytics and CRUD operations.
- **Tasks**:
    - Extend existing product management functionalities to include detailed analytics.
    - Implement CRUD operations.

Sprint Execution:

- **Backend Development**: Developed APIs for enhanced product management with analytics and CRUD operations.
- **Frontend Development**: Updated the product management interface to incorporate new features.

Activity diagram:



User Story 3: User Management & Order Management (with Analytics)

Objective: To enhance the user and order management systems with integrated analytics for better decision-making.

Acceptance Criteria:

- **User Management**: Enables viewing user information, including status and activity.
- **Order Management**: Facilitates tracking order statuses and viewing transaction histories.

Sprint Planning:

- **Goals**: Integrate analytics into user and order management to improve administrative decision-making.
- **Tasks**:
    - Integrate analytics into user and order management.
    - Develop a user-friendly interface for system navigation.
    - Ensure data accuracy and timely updates within the management modules.

Sprint Execution:

- **Backend Development**: Enhanced APIs to include analytics for user and order data.
- **Frontend Development**: Updated the management interface to display analytics and facilitate quick actions.

User Story 4: Promotion Management (with SMTP API integration for new Promotion notification)

Objective: To develop a robust promotion management tool that integrates with SMTP for sending notifications about new promotions.

Acceptance Criteria:

- **Promotion Management**: Provides tools to create and manage promotional campaigns and discount codes.
- **SMTP Integration**: Integrate SMTP to send notifications when new promotions are live.

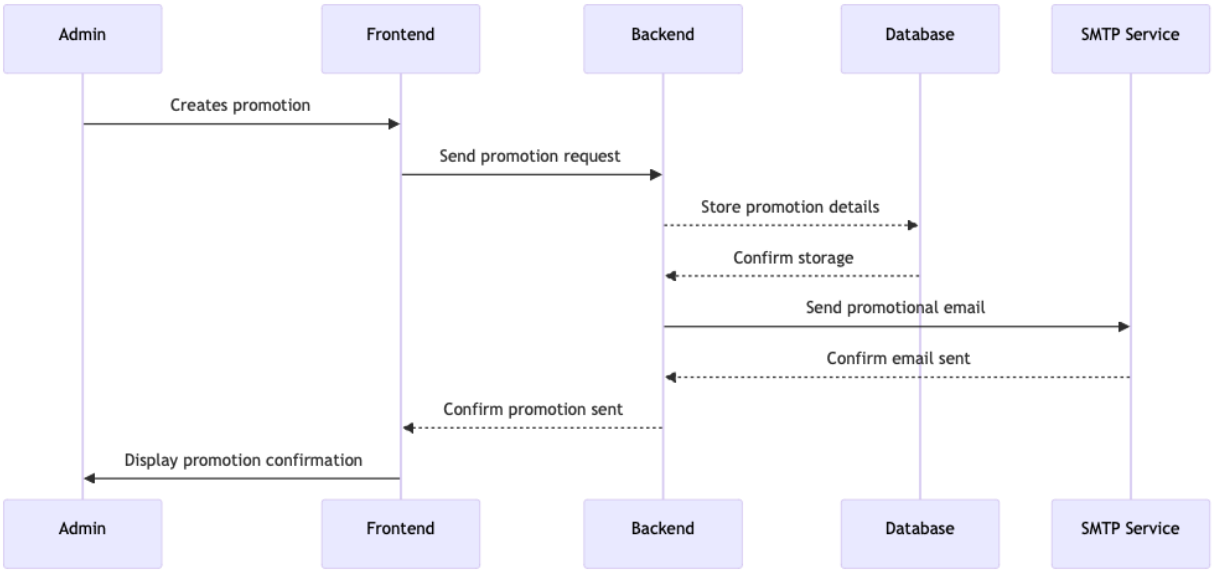
Sprint Planning:

- **Goals**: Enhance promotion management with SMTP integration for effective promotion notifications.
- **Tasks**:
    - Develop the promotion management module.
    - Configure SMTP to send promotional notifications.
    - Test the complete functionality for usability and reliability.

Sprint Execution:

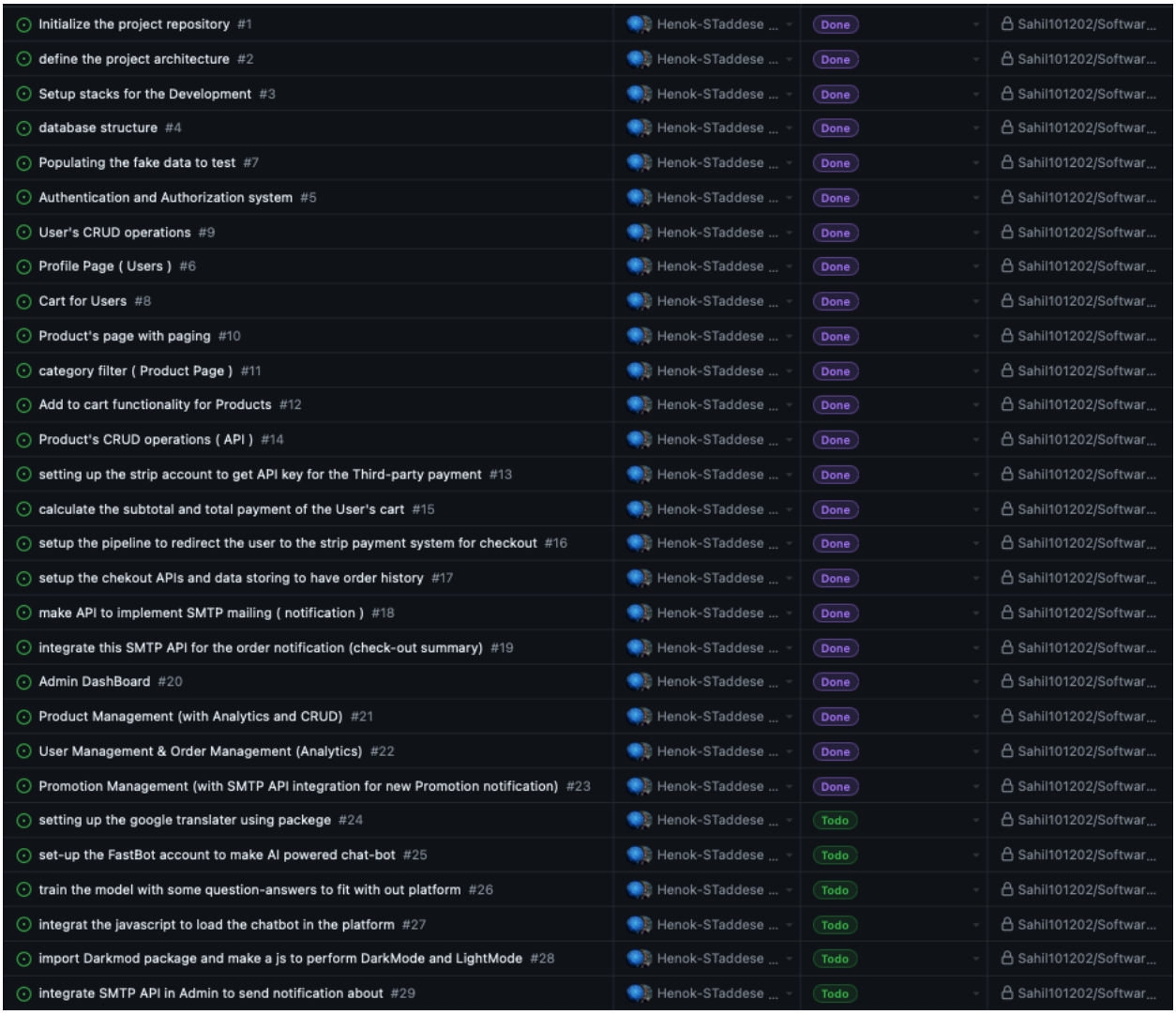
- **Backend Development**: Developed APIs for promotion management and integrated SMTP for notifications.
- **Frontend Development**: Updated the admin interface to include promotion management tools.

Sequence diagram:



Sprint 6: Dark Mode Feature, Chat-Bot, and Multi-Language Translator

Sprint planning:



Sprint Development: backlog for the sixth sprint from the original backlog table

User Story 1: Setting up the Google Translator Using Package

To integrate a multi-language translator to the platform, enabling users to switch between various languages seamlessly. This feature enhances accessibility and usability for a global audience.

Acceptance Criteria:

1. **Multi-Language Translator Implementation:**
    - Added the GTranslate Website Translator Widget to support multiple languages.
    - Configured the widget to allow users to switch languages seamlessly.

Sprint Planning: Goals: Implement the multi-language translator to enhance accessibility and usability for non-English-speaking users.

Tasks:

- **Develop APIs:** Set up and configure the GTranslate package.
- **Implement Frontend:** Integrate the GTranslate widget into the platform.

Sprint Execution: Activities:

- **Implementation:** Added the multi-language translator using the GTranslate Website Translator Widget. Configured the widget to support multiple languages and integrated it into the platform.
- **Functionality:** Ensured the widget allows users to switch to their preferred language seamlessly.

Challenges:

- **Consistent Translation Quality:** Ensuring translations are accurate and contextually appropriate.

Solutions:

- **Widget Configuration:** Thoroughly configured the GTranslate widget and tested various languages to ensure accurate translations.

User Story 2: Set-Up FastBot Account for AI-Powered Chat-Bot

To set up and integrate a chat-bot for user interaction and support, providing assistance with inquiries and navigation. This feature aims to improve customer support and user experience on the platform.

Acceptance Criteria:

1. **Chat-Bot Integration:**
    - Set up a FastBot account and trained the model with relevant question-answer pairs.
    - Integrated the chat-bot into the platform using JavaScript.

Goals: Implement the chat-bot to enhance user interaction and support.

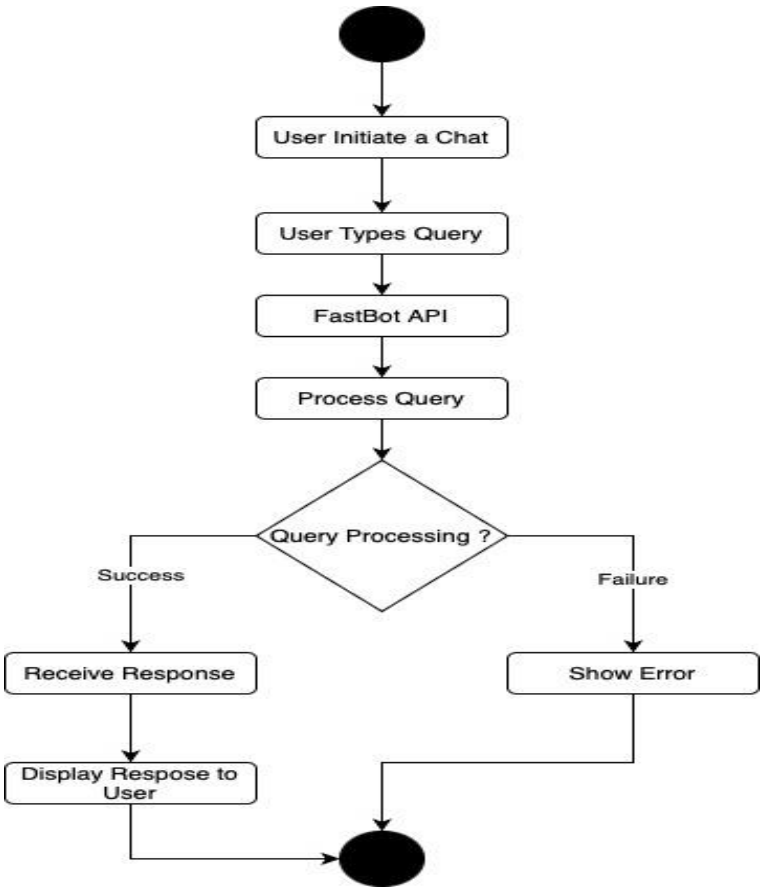
Tasks:

- **Setup FastBot Account:** Create and configure a FastBot account.
- **Train Chat-Bot:** Train the chat-bot with relevant question-answer pairs.
- **Integrate JavaScript:** Load the chat-bot onto the platform using JavaScript.

Activities:

- **Implementation:** Set up the FastBot account and trained the chat-bot model with relevant data. Integrated the chat-bot into the platform using JavaScript.
- **Functionality:** Configured the chat-bot to handle common user queries and provide navigation assistance.

Activity diagram:



Challenges:

- **Chat-Bot Responsiveness:** Ensuring the chat-bot is responsive and provides accurate information.

Solutions:

- **Training and Customization:** Regularly updated and tested the chat-bot to improve its performance and responsiveness.

User Story 3: Import Darkmode Package and Implement Dark Mode

To implement a dark mode feature for the website, allowing users to switch between dark and light themes. This feature aims to enhance user experience by providing a visually comfortable browsing option.

Acceptance Criteria:

1. **Dark Mode Implementation:**
    - Imported the Darkmode.js package.
    - Developed a toggle switch to switch between dark and light modes.
    - Styled the website to ensure consistency across both themes.

Goals: Implement the dark mode feature to enhance user experience and provide a visually comfortable browsing option.

Tasks:

- **Import Darkmode.js:** Import the Darkmode.js package.
- **Develop Toggle Switch:** Create a toggle switch for switching between dark and light modes.
- **Style Website:** Ensure consistent styling across both themes.

Activities:

- **Implementation:** Utilized Darkmode.js to implement the dark mode feature. Created a toggle switch and styled the website to ensure consistency and readability across both themes.
- **Styling:** Used CSS variables and media queries to manage color themes and ensure a seamless transition.

Challenges:

- **Consistent Styling:** Maintaining visual consistency across both dark and light modes.

Solutions:

- **Thorough Testing and Styling Adjustments:** Conducted extensive testing and made styling adjustments to ensure consistent appearance.

Successful implementation of all three key features: multi-language translator, chat-bot, and dark mode and we obtain positive feedback from stakeholders validated. Effective teamwork and collaboration were observed throughout the sprint.

By the end of Sprint 6, we successfully implemented the dark mode feature, chat-bot and multi-language translator. These features significantly enhance the platform's usability and accessibility. The feedback and insights gathered during the review and retrospective meetings will guide future improvements and iterations.

Unified Modeling Language (UML)

UML Overview and Importance

UML (Unified Modeling Language) is a standardized modeling language that provides a set of graphical notation techniques to create visual models of object-oriented software-intensive systems. It is primarily used in software engineering to offer a clear and comprehensive visual blueprint of a system, enabling detailed documentation of its structure and design before the coding begins.

Reasons for Using UML in Software Projects

1. **Standardization**: UML offers a standardized way to visualize the design of a system. This standardization ensures clear understanding among all stakeholders, including developers, project managers, analysts, and new team members. The universal nature of UML makes it particularly beneficial for large teams and geographically distributed projects.

2. **Improves Understanding and Communication**: UML diagrams serve as vital communication tools among project teams and stakeholders. They simplify complex software structures and business processes into easy-to-understand diagrams, facilitating improved communication and ensuring a clear understanding of project requirements and functionalities.

3. **Systematic Planning and Visualization**: UML allows detailed planning and visualization of software projects before coding begins. It helps identify potential issues and bottlenecks in the system architecture, which can be addressed early, thereby reducing project risks and increasing efficiency.

4. **Documentation**: UML provides excellent documentation support, which is crucial for maintenance and future enhancements of the software. This documentation is particularly useful for onboarding new team members, enabling them to quickly understand the project's functional and non-functional details.

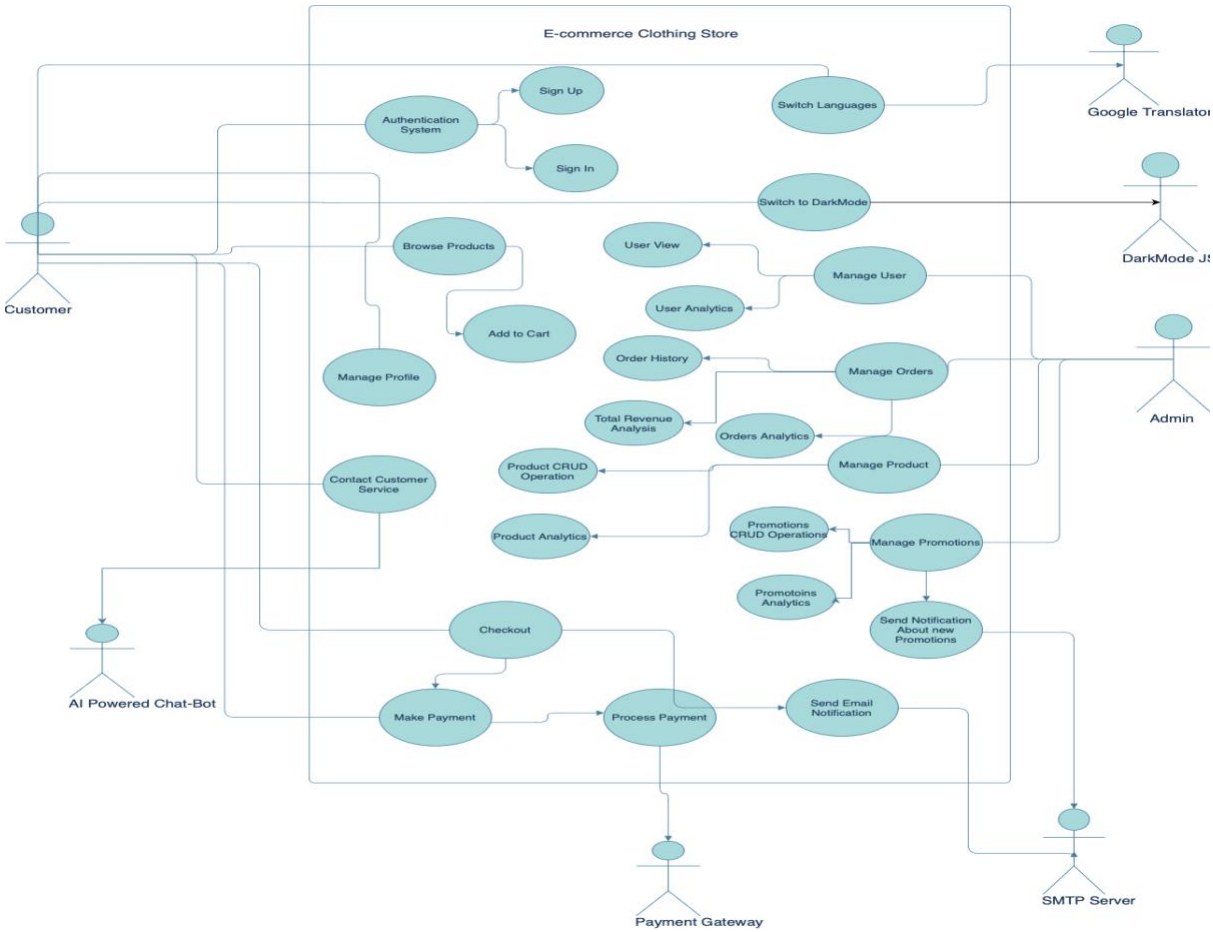
5. **Facilitates Design and Code Reuse**: UML diagrams provide clear schematic representations of existing and planned system features, facilitating the identification of reusable components. This reuse can significantly reduce development time and costs for new systems requiring similar components.

6. **Multiple Perspectives**: UML offers various types of diagrams to cover different aspects of the system. For instance, use case diagrams represent functional requirements, class diagrams illustrate static structures, activity diagrams describe dynamic aspects, and sequence diagrams detail object interactions. This multiplicity allows for a comprehensive examination of the system from various angles.

7. **Integration and Scalability**: UML supports scalable and integrative design of software projects. As projects expand or integrate with other systems, UML diagrams can be updated to accommodate and represent these changes effectively.

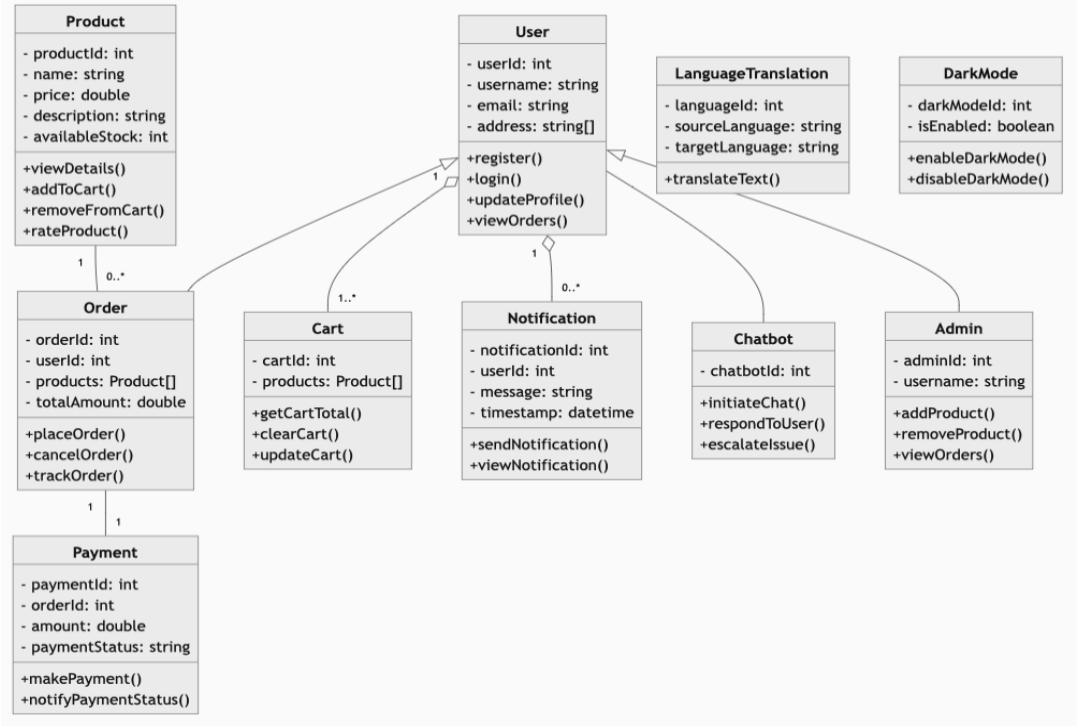
Use Case Diagram

- The following use case diagram shows the main interactions between users and the system, such as user authentication, browsing products, adding to cart, checkout, payment processing, and receiving notifications in our platform.



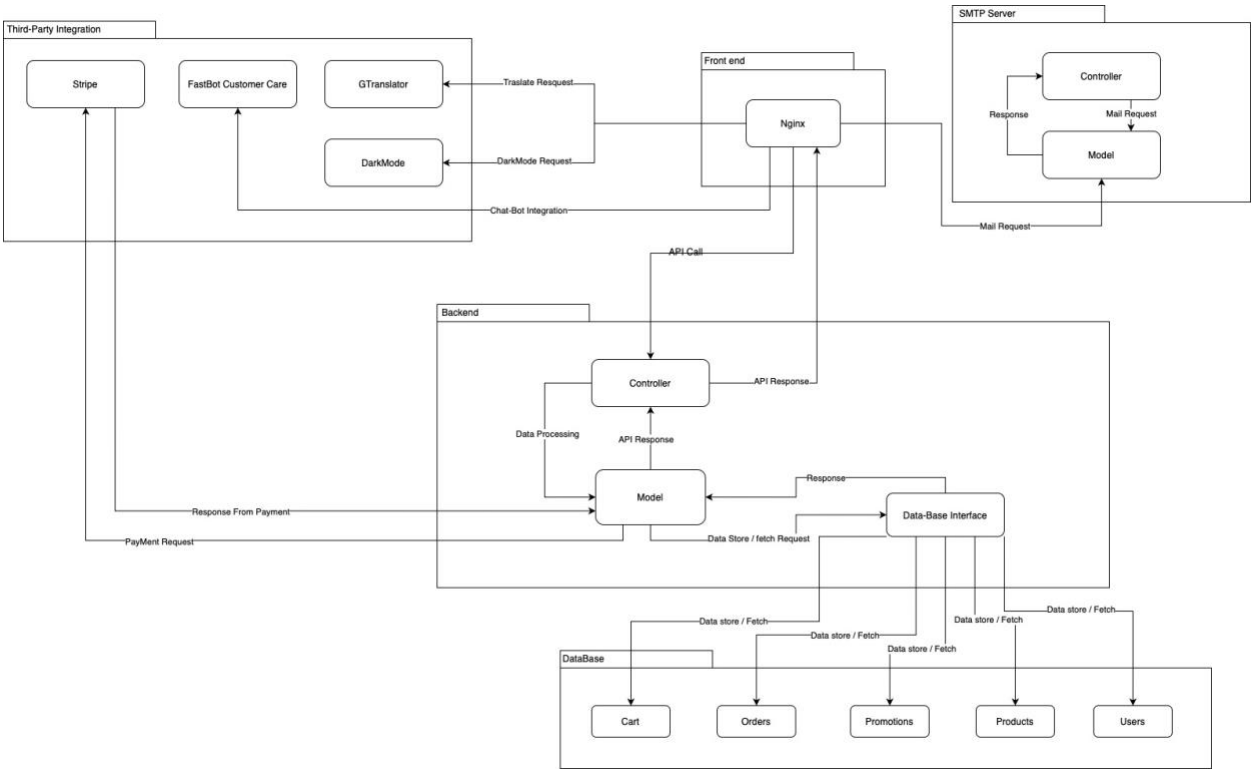
Class Diagram

- Following , we have provided the class diagram detailing the various classes in the system, their attributes, methods, and relationships, such as User, Product, Order, Payment, and Notification classes in our platform.

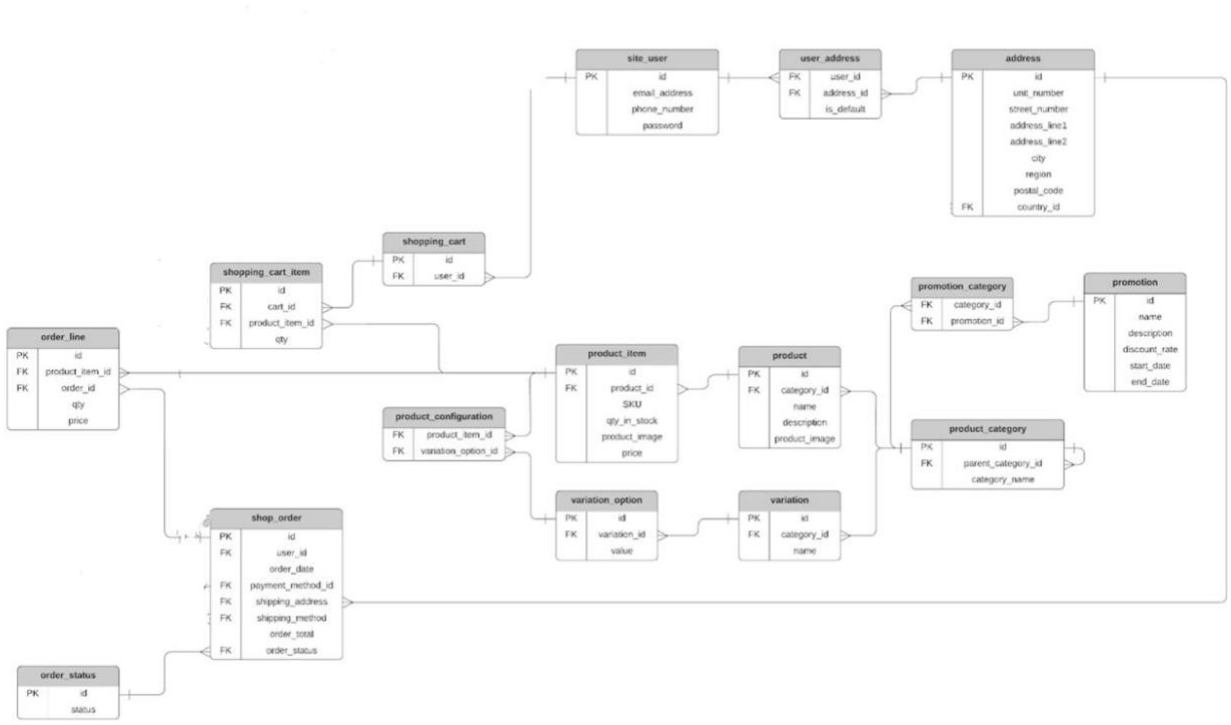


Component Diagram

- Below we present the final component diagram that should depict all subsystems (features) and show the communication between different parts abstractly across the plat form.



DataBase Structure



Software Architecture Description

System Overview

The software architecture of the e-commerce clothing platform is designed to provide a robust and scalable solution for online shopping management. The system employs a microservices architecture to ensure modularity and ease of maintenance. Key components include Nginx for the frontend, Flask for the backend, MySQL database, and integration with external services like payment gateways, chatbot APIs, Google Translator API, dark mode packages, and a separate mailing server.

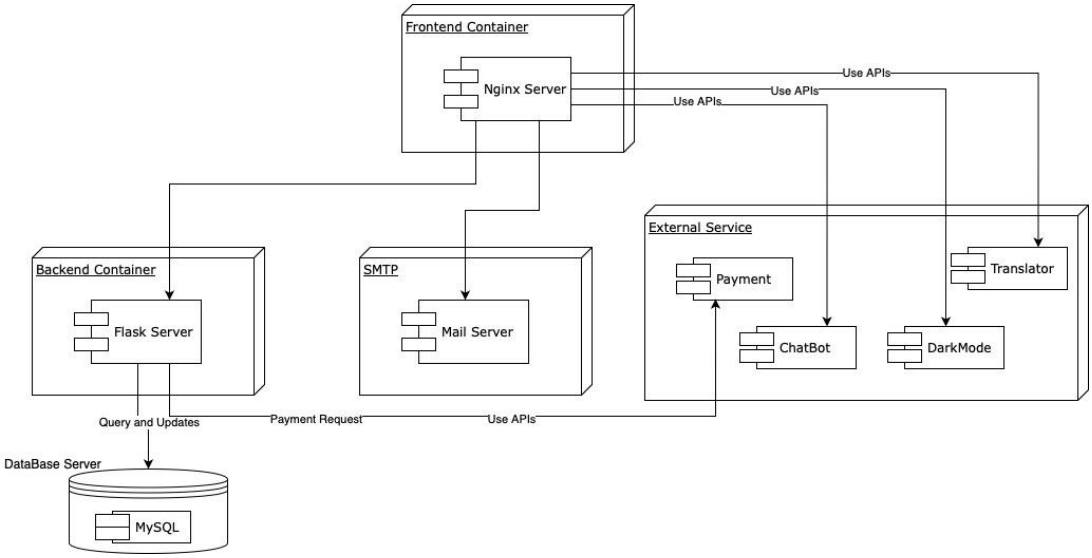
Component Descriptions

1. **Frontend (Nginx)**
    - **User Interface**: Provides the graphical user interface for user interactions. Displays product information and processes user inputs.
    - **Frontend Logic (Controller)**: Manages user input, processes requests, and interacts with the backend for data.
    - **Data Management (Model)**: Manages and prepares data received from the backend for display.
2. **Backend (Flask)**
    - **Business Logic (Controller)**: Processes business logic, making decisions based on user input and data from the MySQL database and external APIs.
    - **Data Handling (Model)**: Manages interactions with the MySQL database through CRUD operations.
    - **Data Presentation (View)**: Formats data responses for the frontend.
3. **MySQL Database**
    - **Data Storage**: Stores all persistent data including User, Product, Order, Payment, and Notification data, structured for efficient access and manipulation.
4. **External APIs**
    - **Payment Gateway**: Processes payment transactions securely.
    - **Chatbot API**: Provides interactive support to users.
    - **Google Translator API**: Translates content to different languages.
    - **Dark Mode Package**: Implements dark mode functionality.
    - **Mailing Server**: Operates as a standalone SMTP server for sending notifications.

Interconnections

- The Nginx frontend communicates with the Flask backend via HTTP/API calls, requesting and sending data based on user interactions.
- The Flask backend fetches and manipulates data from the MySQL database, ensuring data consistency and integrity.
- For payment processing, chatbot interactions, and translation, the backend interacts with the respective external APIs.
- The backend also interfaces with the standalone mailing server to send notifications to users.

Deployment Diagram



Deployment Details

The project employs a microservices architecture, where each component operates as a standalone entity within Docker containers. This approach enhances maintainability and scalability, avoiding the pitfalls of monolithic architecture.

- **Nginx**: Deployed in a Docker container for the frontend.
- **Flask**: Deployed in a separate Docker container for the backend.
- **MySQL Database**: Runs in its own Docker container.
- **External Services**: Payment Gateway, Chatbot API, Google Translator API, and Dark Mode Package are integrated as external services.
- **Mailing Server**: Deployed in its own Docker container as a standalone SMTP server.

GitHub Repositories

- Repository : [GitHub Repository Link]

Features Implementation

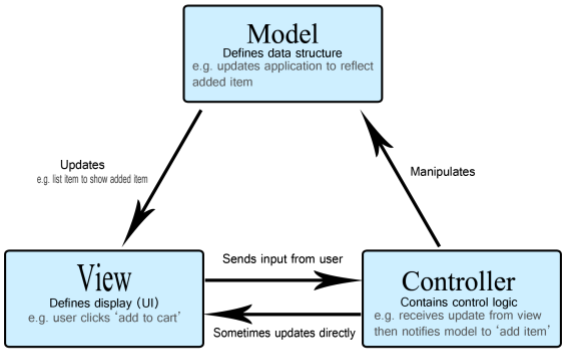
This section details the implementation of key features in our e-commerce clothing platform, highlighting the architecture, steps taken for integration, and including screenshots for each feature.

Implementation Architecture

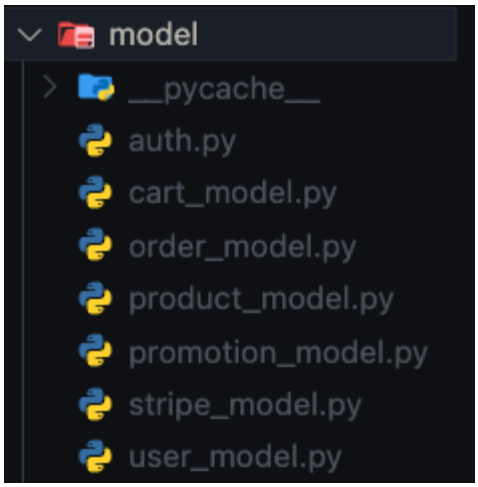
Our platform is designed with a modular architecture to facilitate seamless development and integration of various features. Below is a breakdown of the architecture and implementation details for each feature.

MVC Architecture

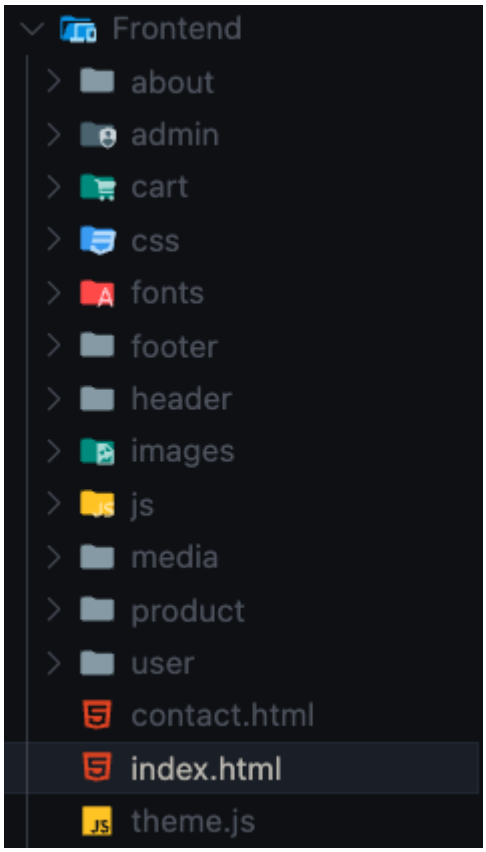
MVC is a software design pattern commonly used for developing web applications. It divides the application into three interconnected components:



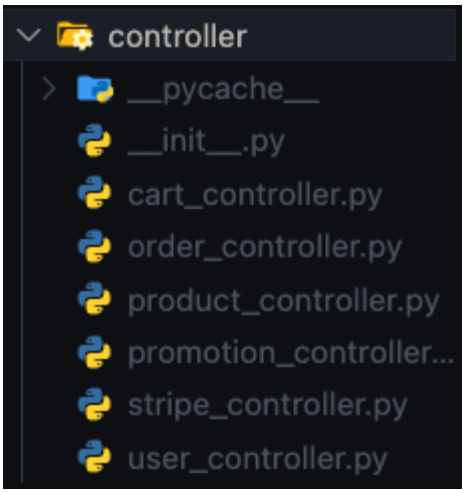
- **Model**: Represents the data and business logic of the application. It interacts with the database, processes data, and responds to requests for information from the View. (e.g. API/model/)



- **View**: Represents the presentation layer of the application. It displays the data to the user and sends user commands to the Controller. Here, we are using NGINX for the front-end structure. (e.g. /Frontend/)



- **Controller**: Acts as an intermediary between Model and View. It processes user input, manipulates data from the Model, and decides which View to render. (e.g. API/controller/)



User Authentication / Authorization

Implementing robust user authentication and authorization mechanisms is essential for securing an e-commerce platform. This involves setting up the database to store user information, creating endpoints for user registration and login, managing user sessions securely, and ensuring the front-end interfaces are intuitive and secure. Using the Model-View-Controller (MVC) design pattern helps in organizing the code effectively.

Implementation Steps:

1. **Registration and Login:**
    - **Endpoints Implementation:** Registration and login endpoints are implemented in Flask to handle user sign-up and authentication. We created an API that is used to manage user authentication and authorization, simplifying the process of securing routes and managing user roles.

```
exptime = datetime.now() + timedelta(hours=24)
exp_epoc_time = exptime.timestamp()
payload = {
    "id": result['id'],
    "role_id": result['role_id'],
    "name": result['name'],
    "email": result['email'],
    "exp": int(exp_epoc_time)
}
jwt_token = jwt.encode(payload, "sahil101202", algorithm="HS256")
jwt_token_str = jwt_token.decode('utf-8')
return make_response({'response': True, 'message': 'User login successful', 'token': jwt_token_str}, 200)
```

User Authorization:

```
def token_auth(self, endpoint=""):
    def inner1(func):
        @wraps(func)
        def inner2(*args, **kwargs):
            endpoint = request.url_rule
            endpoint_str = str(endpoint)
            authorization = request.headers.get("Authorization")
            if re.match("^Bearer *([^ ]+) *$", authorization, flags=0):
                token = authorization.split(" ")[1]
                print(token)
                # return func(*args, *kwargs)
                if token:
                    try:
                        jwtdecoded = jwt.decode(token, 'sahil101202', algorithms="HS256")
                    except jwt.ExpiredSignatureError:
                        return make_response({'responce':False, 'message':'Token Expired!!'}, 401)
                    role_id = jwtdecoded['role_id']
                    self.cur.execute("SELECT roles FROM endpoint_accessibility WHERE endpoint=%s", (endpoint_str,))
                    result = self.cur.fetchall()

                    if len(result)>0:
                        allowed_roles = result[0]['roles']
                        roles_list = json.loads(allowed_roles)
                        if role_id in roles_list:
                            return func(*args, **kwargs)
                        else:
                            return make_response({'responce':False, 'message':'Unauthorized User!!'}, 404)
                    else:
                        return make_response({'responce':False, 'message':'Unknown Endpoint!!'}, 404)

                else:
                    return make_response({'responce':False, 'message':'Invalid Token!!'}, 401)
            else:
                return make_response({'responce':False, 'message':'Invalid Token!!'}, 401)
        return inner2
    return inner1
```

2. **Session Management:**

    - **JWT Implementation:** Implement JWT (JSON Web Tokens) for secure session management. JWT allows for stateless authentication by encoding user information into a token. This token is sent with each request to verify the user's identity without requiring session storage on the server.
    - **Token Generation and Validation:** Generate a JWT upon successful login and include it in the response. Validate the token for protected routes to ensure only authenticated users can access them.

```
const token = json.token;
localStorage.setItem("authToken", token);
window.location.href = "http://localhost:8080/";
alert(json.message);
```

3. **MVC Design Pattern for the Register / Login:**

    - **Model (M):** The model represents the data layer of the application. In this case, the User class interacts with the database to handle user data. The model is responsible for data validation, relationships, and business logic.
    - **View (V):** The view is responsible for rendering the user interface. The login HTML file is a part of the view, providing a form for users to input their credentials.
    - **Controller (C):** The controller handles user input, processes it, and interacts with the model to perform actions. The Flask routes (/register and /login) act as controllers, handling HTTP requests and responses.

By following these steps and using the MVC design pattern, we ensured a secure, efficient, and maintainable user authentication and authorization system. This foundation enhances the security of user data, streamlines the login process, and improves the overall user experience on the platform.

CRUD Operations

Implementation Steps:

1. **API Endpoints**:
    - We have created CRUD operation endpoints for the product, user, and promotions.

Part of User Controller

```python
@user_blueprint.route('/user/profile/<int:id>', methods=['GET'])
# @auth.token_auth()
def user_read_controller(id):
    return user.user_read_one(id)

@user_blueprint.route('/user/register', methods=['POST'])
# @auth.token_auth()
def user_signup_controller():
    return user.user_signup_model(request.json)

@user_blueprint.route('/user/update/<id>', methods=['PUT'])
# @auth.token_auth()
def user_update_controller(id):
    return user.user_update_model(id, request.json)

@user_blueprint.route('/user/delete/<id>', methods=['DELETE'])
# @auth.token_auth()
def user_delete_controller(id):
    return user.user_delete_model(id)

@user_blueprint.route('/user/login', methods=['POST'])
# @auth.token_auth()
def user_login_controller():
    return user.user_login(request.json)
```

Part of Product Controller

```python
# Route to get all products
@product_blueprint.route('/products', methods=['GET'])
def read_all_products_controller():
    return product.read_all_products()

# Route to get details of a specific product by its ID
@product_blueprint.route('/products/<int:product_id>', methods=['GET'])
def product_detail_controller(product_id):
    return product.read_one_product(product_id)

# Route to add a new product
@product_blueprint.route('/products/add', methods=['POST'])
# @auth.token_auth()
def add_prodduct_controller():
    return product.add_product(request.json)

# Route to delete a product by its ID
@product_blueprint.route('/products/<int:product_id>', methods=['DELETE'])
# @auth.token_auth()
def delete_product_controller(product_id):
    return product.delete_product(product_id)

# Route to update a product by its ID
@product_blueprint.route('/products/update', methods=['PUT'])
# @auth.token_auth()
def update_product_controller():
    return product.update_product(request.json)
```

Part of Promotion Controller

```python
@promotion_blueprint.route('/promotions', methods=['POST'])
def add_promotion():
    return promotion_model.add_promotion(request.json)

@promotion_blueprint.route('/allPromotions', methods=['GET'])
def get_all_promotions():
    return promotion_model.get_all_promotions()

@promotion_blueprint.route('/promotions/<int:id>', methods=['GET'])
def get_promotion(id):
    return promotion_model.get_promotion(id)

@promotion_blueprint.route('/promotions', methods=['PUT'])
def update_promotion():
    data = request.get_json()
    return promotion_model.update_promotion(data)

@promotion_blueprint.route('/promotions', methods=['DELETE'])
def delete_promotion():
    return promotion_model.delete_promotion(request.json)

@promotion_blueprint.route('/promotions/counts', methods=['GET'])
def count_promotions():
    return promotion_model.count_promotions()

@promotion_blueprint.route('/promotions/activeCounts', methods=['GET'])
def active_promotions():
    return promotion_model.active_promotions()
```

ChatBot Integration

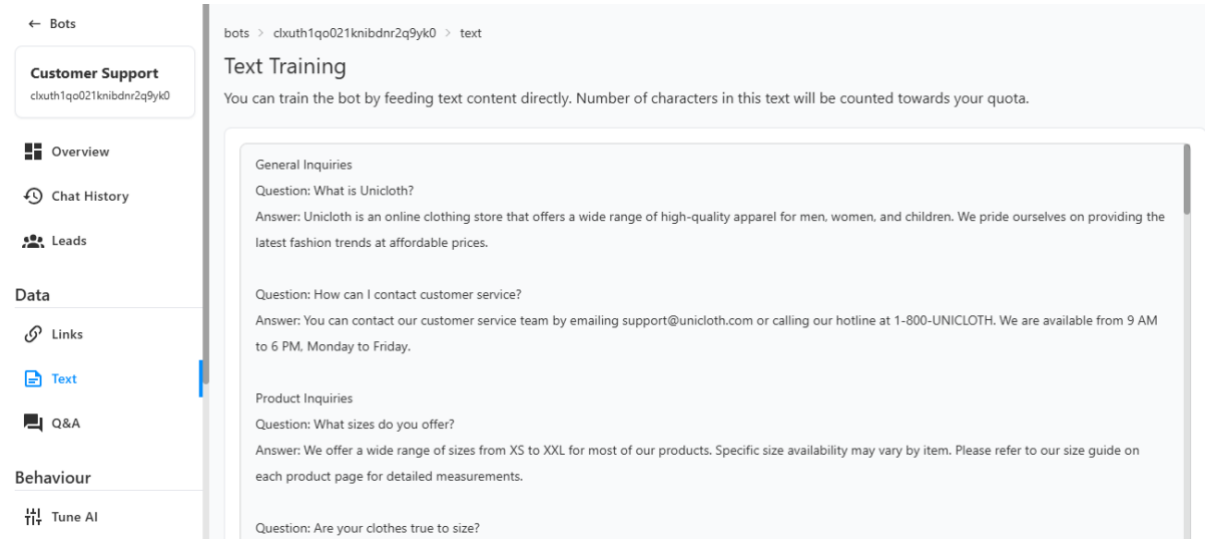
Implementation Steps:

1. Setup FastBot Account:

- **Create FastBot Account:**.
- **Configure Account:**
  - In the FastBot dashboard, we navigated to the settings page.
  - Configure the basic settings, such as the bot's name, language, and initial greeting message.

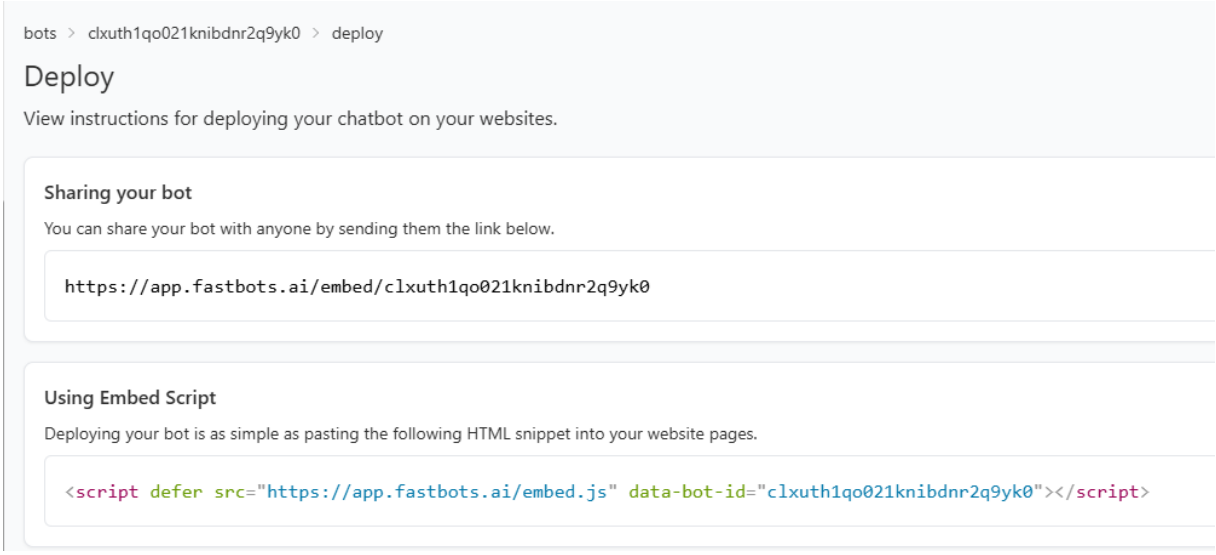
2. Train the Chat-Bot:

- **Data Preparation:**
  - We prepared a list of common user queries and corresponding answers. This list should cover frequently asked questions, navigation assistance, and other relevant interactions.
- **Training the Bot:**
  - In the FastBot dashboard, we went to the training section.
  - Input the question-answer pairs prepared earlier.
  - Then we trained the model by clicking the "Train" button.



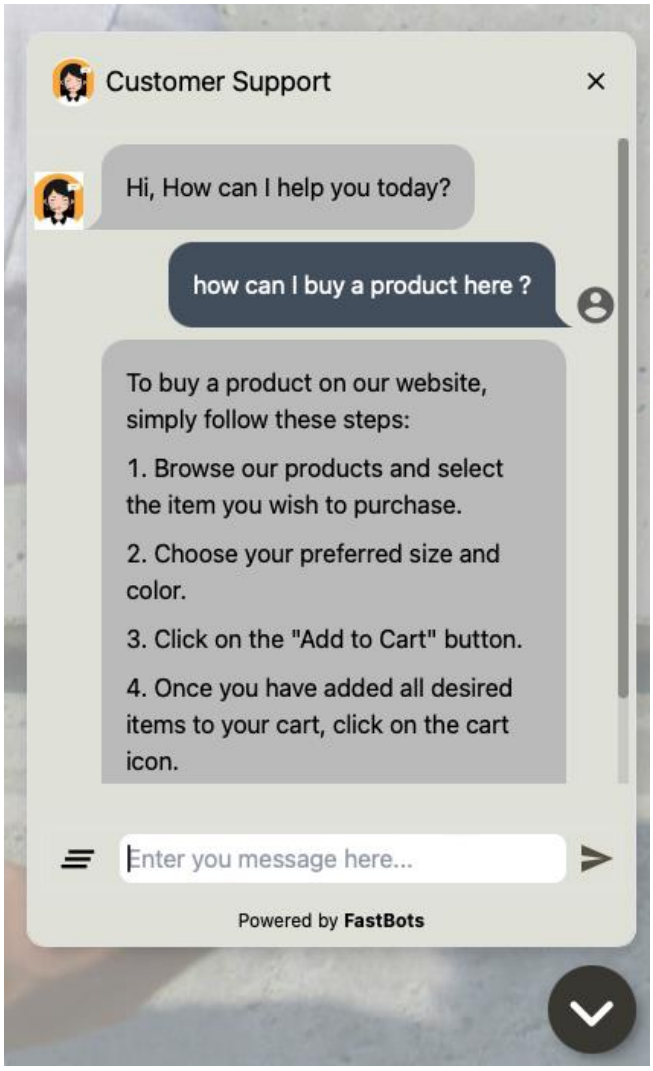
3. Integrate JavaScript to Load the Chat-Bot:

- **Generate Integration Script:**
    - We generated the JavaScript integration script provided by FastBot. This script will be used to embed the chat-bot into your platform.



- **Embed Chat-Bot on the Platform:**
    - Finally, we added the generated JavaScript script to our platform's HTML code. Typically, this is done by including the script in the <head> section of your main HTML file.

The chat-bot is now capable of handling a variety of user queries, providing quick and accurate responses. Below is the image of the result of the chatbot section in the platform after successfully integrating.



Payment Integration

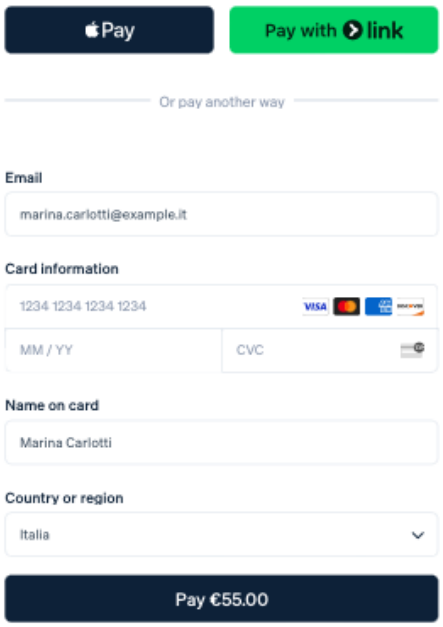
Implementation Steps:

1. **Stripe API Setup:**
    - Firstly, we have configured your account and obtained the necessary API keys for integration.

2. **Backend Integration:**
    - Then, we have implemented Flask endpoints for creating and managing payment sessions. And integrated the API key with the Flask API model to manage sessions.

```python
# Set your secret key. Remember to switch to your live secret key in production!
stripe.api_key = 'sk_test_51PUYzSDxa2xNsUeMK90pm4ounZgnzIcwXrbcW9GAPB4hzwtAcrNzBVlqy3CBI5rbCYeq28FwGYkChL4kzeDZDPRF00sgbU2TGd'

class StripeModel:
    @staticmethod
    def create_checkout_session(data):
        try:
            cart_items = data['data']
            line_items = [{
                'price_data': {
                    'currency': 'usd',
                    'product_data': {
                        'name': item['name'],
                    },
                    'unit_amount': int(item['price'] * 100),
                },
                'quantity': item['qty'],
            } for item in cart_items]
            session = stripe.checkout.Session.create(
                payment_method_types=['card'],
                line_items=line_items,
                mode='payment',
                success_url='http://localhost:8080/',
                cancel_url='http://localhost:8080/',
            )
            return jsonify(id=session.id)

        except stripe.error.StripeError as e:
            return jsonify(error=str(e)), 400

        except Exception as e:
            return jsonify(error="An error occurred"), 500
```

So, here payment integration involves configuring Stripe, implementing backend endpoints in Flask to manage payment sessions, and using Stripe Elements for a secure checkout experience on the frontend. This approach ensures a robust and secure payment processing flow within your application.



Language Translator

Implementation Steps:

1. **Generate the Translator Widget Code:**
    - In the GTranslate dashboard, we generated a widget and customized the widget settings such as default language, supported languages, and the appearance of the language switcher.

WIDGET CODE

```
<div class="gtranslate_wrapper"></div>
<script>window.gtranslateSettings = {"default_language":"en","languages":
["en","fr","de","it","es","am","gu"],"globe_color":"#66aaff","wrapper_selector":".gtranslat
e_wrapper","flag_size":16,"globe_size":40}</script>
<script src="https://cdn.gtranslate.net/widgets/latest/globe.js" defer></script>
```

2. **Integration into the Platform:**
    - Then we copied the generated widget code and pasted the widget code into the platform's main HTML file,in the <head> section.

```
<div class="gtranslate_wrapper"></div>
<script>window.gtranslateSettings = {"default_language":"en","languages":["en","fr","de","it","am","gu"],"globe_color":"#fff",
"wrapper_selector":".gtranslate_wrapper","flag_size":16,"horizontal_position":"left","vertical_position":"bottom","globe_size":40}</
script>
<script src="https://cdn.gtranslate.net/widgets/latest/globe.js" defer></script>
```

The multi-language translator allows users to switch to their preferred language seamlessly, improving accessibility and user experience for non-English-speaking users. Below we have provided an illustration of an image showing the change after successfully integrating it in the platform.

Dark Mode Integration

Implementation Steps:

1. **Import Darkmode.js**

    To implement the dark mode feature, we began by integrating the Darkmode.js library, a lightweight JavaScript library designed to add a dark mode toggle to websites with minimal effort. We imported Darkmode.js into our project to leverage its functionality for detecting and toggling between light and dark themes.

    Adding Darkmode.js:

    We included the Darkmode.js script in our HTML file for seamless integration.

    <script src="https://unpkg.com/darkmode-js@1.5.5/lib/darkmode-js.min.js"></script>

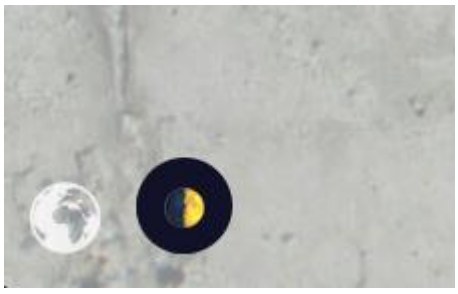
2. **Develop Toggle Switch**

We then developed a user-friendly toggle switch that allows users to switch between dark and light modes. The toggle switch is a crucial component that provides users with control over their browsing experience.

1. **Creating the Toggle Switch:** We Added a button element in the HTML to serve as the dark mode toggle.
2. **Initializing Darkmode.js:** We Initialized Darkmode.js with custom configuration options to define the behavior and appearance of the dark mode toggle.

```
const options = {
    bottom: '18px', // default: '32px'
    right: 'unset', // default: '32px'
    left: '70px', // default: 'unset'
    time: '0.5s', // default: '0.3s'
    mixColor: '#fff', // default: '#fff'
    backgroundColor: '#fff',  // default: '#fff'
    buttonColorDark: '#100f2c',  // default: '#100f2c'
    color: '#fff',
    blockquote: '#fff',
    buttonColorLight: '#fff', // default: '#fff'
    saveInCookies: false, // default: true,
    label: '🌓', // default: ''
    autoMatchOsTheme: false// default: true
  }

  const darkmode = new Darkmode(options);
  darkmode.showWidget();
```

By integrating Darkmode.js and developing a seamless toggle switch, we successfully implemented the dark mode feature on our platform. This feature significantly enhances user experience by providing a visually comfortable browsing option, ensuring consistency and readability across both dark and light themes. Below we have provided an image showing the toggle button after successfully integrating it in our platform.



Admin panel

Implementation Steps

1.Admin Dashboard

The admin panel is a crucial component of the platform, providing administrators with a comprehensive and interactive interface to manage various aspects of the system. The primary objectives of the admin panel are to facilitate efficient management of users, products, orders, and promotions, while also offering advanced analytics and secure access control.

Implementation Details:

Quick Actions:

In order to provide comprehensive management of products by allowing administrators to create, read, update, and delete product information directly from the admin panel and submit it directly to the backend.

Backend Integration:
We developed an API endpoint to handle the creation of new products and handling new products . Use validation to ensure all required fields are filled and data is properly formatted.

```python
product_blueprint = Blueprint('product', __name__)
product = product_model()
auth = auth()

# Route to get all products
@product_blueprint.route('/products', methods=['GET'])
def read_all_products_controller():
    return product.read_all_products()

# Route to get details of a specific product by its ID
@product_blueprint.route('/products/<int:product_id>', methods=['GET'])
def product_detail_controller(product_id):
    return product.read_one_product(product_id)

# Route to add a new product
@product_blueprint.route('/products/add', methods=['POST'])
# @auth.token_auth()
def add_prodduct_controller():
    return product.add_product(request.json)

# Route to delete a product by its ID
@product_blueprint.route('/products/<int:product_id>', methods=['DELETE'])
# @auth.token_auth()
def delete_product_controller(product_id):
    return product.delete_product(product_id)

# Route to update a product by its ID
@product_blueprint.route('/products/update', methods=['PUT'])
# @auth.token_auth()
def update_product_controller():
    return product.update_product(request.json)
```

- Then we have used the same API to handle the users and to track their activity in order to provide a user friendly platform. And also we ensure all required fields are filled and data is properly formatted.

```python
@user_blueprint.route('/user/profile/<int:id>', methods=['GET'])
# @auth.token_auth()
def user_read_controller(id):
    return user.user_read_one(id)

@user_blueprint.route('/user/register', methods=['POST'])
# @auth.token_auth()
def user_signup_controller():
    return user.user_signup_model(request.json)

@user_blueprint.route('/user/update/<id>', methods=['PUT'])
# @auth.token_auth()
def user_update_controller(id):
    return user.user_update_model(id, request.json)

@user_blueprint.route('/user/delete/<id>', methods=['DELETE'])
# @auth.token_auth()
def user_delete_controller(id):
    return user.user_delete_model(id)
```

2.Promotion Management

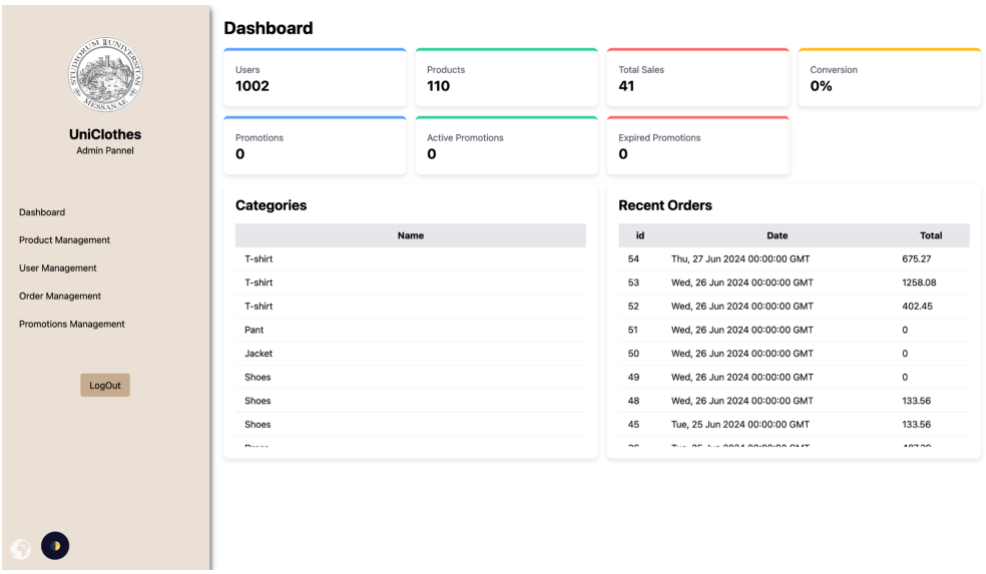
- We develop a robust promotion management that can manage CRUD operation and integrates with SMTP for sending notifications about new promotions. This tool will enable administrators to create, manage, and communicate promotions effectively to users, enhancing marketing efforts and user engagement. We are using the Flak API to handle the creation of new promotions and an external SMTP server to send the notification to all users about new promotions.

```
document.addEventListener("DOMContentLoaded", function() {
    loadPromotions();
    loadPromotionsForSelects();
});

function showForm(formType) {
    hideForm();

    const formContainer = document.getElementById("crud-form-container");
    formContainer.classList.remove("hidden");

    let formTitle = "";
    switch (formType) {
        case "create":
            document.getElementById("create-form").classList.remove("hidden");
            formTitle = "Create Promotion";
            break;
        case "update":
            document.getElementById("update-form").classList.remove("hidden");
            formTitle = "Update Promotion";
            break;
        case "delete":
            document.getElementById("delete-form").classList.remove("hidden");
            formTitle = "Delete Promotion";
            break;
        case "send-email":
            document.getElementById("send-email-form").classList.remove("hidden");
            formTitle = "Send Promotion Email";
            break;
    }

    document.getElementById("crud-form-title").textContent = formTitle;
}
```

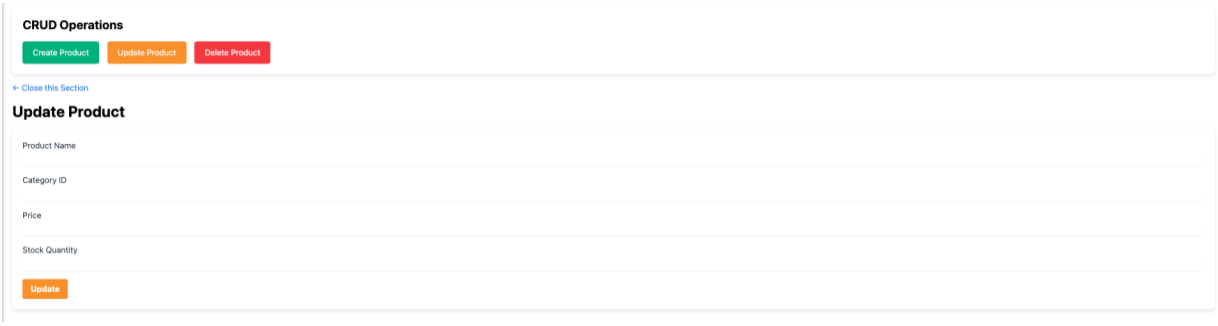
3.Designing Tool:

In order to provide a user-friendly and efficient interface for the administrator we have utilized HTML, CSS, JavaScript, and a frontend framework, Nginx (for the proxy).

Here we are able to see overall design of admin panel



In down below image the admin is able to perform all the CRUD operation about the Products



4.Mailing Service Implementation

We provided a comprehensive implementation for sending order confirmation and promotions emails with invoices using SMTP. This involves setting up the email blueprint, handling email sending logic, and integrating it into the order process.

Implementation Steps

Setting Up the Email Blueprint

- In order to handle email-related routes and logic within the Flask application. First we have defined a blueprint for email-related routes, then we Implemented a route to handle sending order confirmation emails. At last we extract email and product information from the request data.

Email Sending Logic

- We implemented the actual sending of emails using an SMTP server. This involves configuring the SMTP settings and defining functions to send emails and generate invoice content. This invoice is based on the HTML format of the product list.

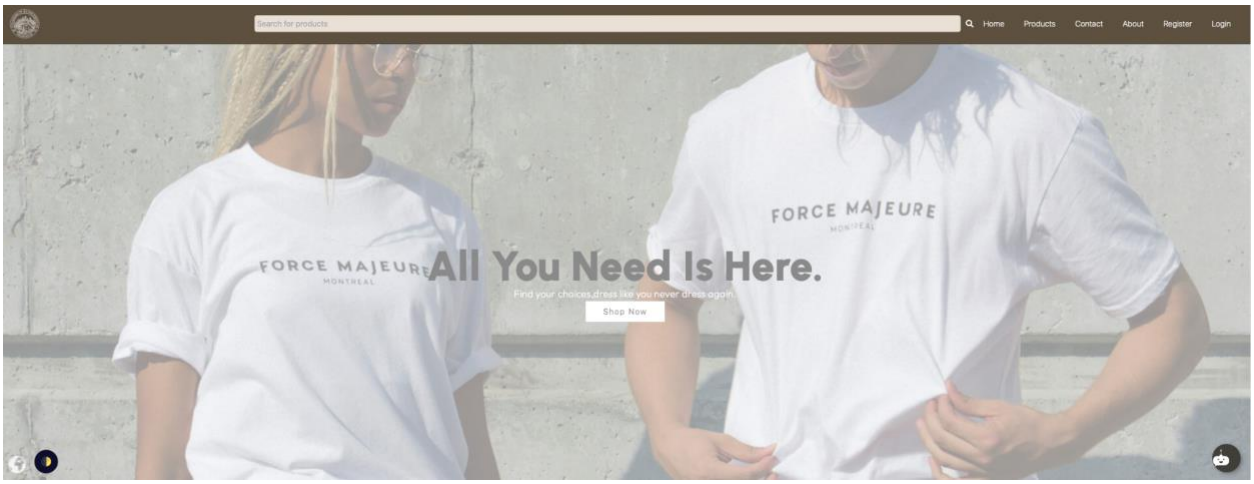
```python
# controller/email_controller.py
from flask import Blueprint, request, make_response
from email_model import send_email, generate_invoice

email_bp = Blueprint('email', __name__)

@email_bp.route('/send-email', methods=['POST'])
def send_order_confirmation_email():
    data = request.json
    recipient_email = data['email']
    products = data['products']  # Ensure products list is passed in the request data

    if not (recipient_email and products):
        return make_response({'status': False, 'error': 'Incomplete data provided'}, 400)

    subject = "Order Confirmation and Invoice"
    invoice_html = generate_invoice(products)
    invoice_plain_text = "Here is your order invoice:\n\n"  # Provide a plain text version if needed

    try:
        send_email(subject, [recipient_email], invoice_html, invoice_plain_text)
        return make_response({'status': True, 'message': 'Email sent successfully'}, 200)
    except Exception as e:
        return make_response({'status': False, 'error': str(e)}, 500)
```

Integrating Email Notification into Order Process

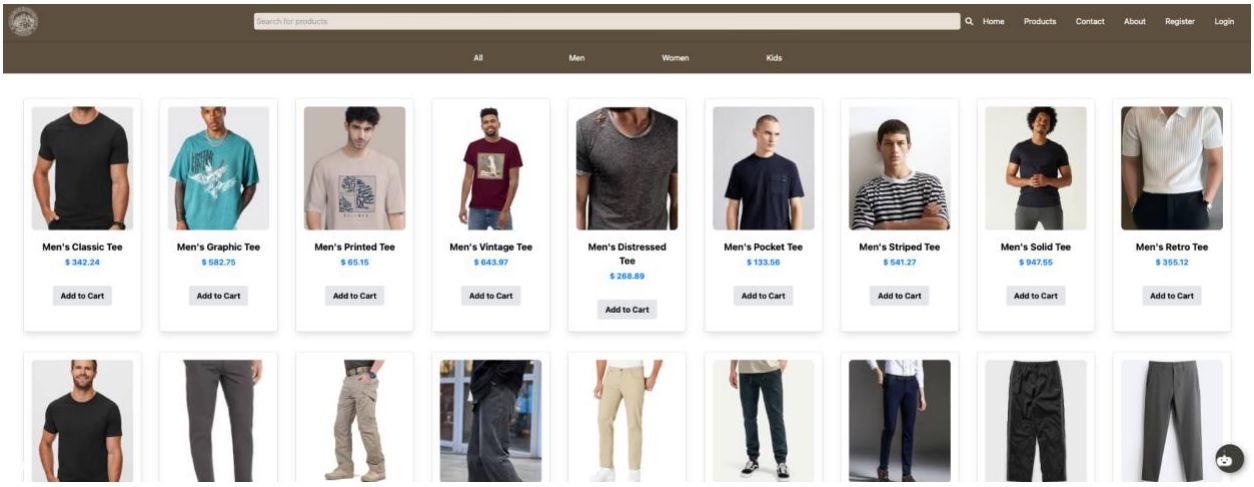
- In order to get the email notification when an order is completed, users receive an order confirmation to the user's personal email.

GUI Mock-up

The Landing Page :

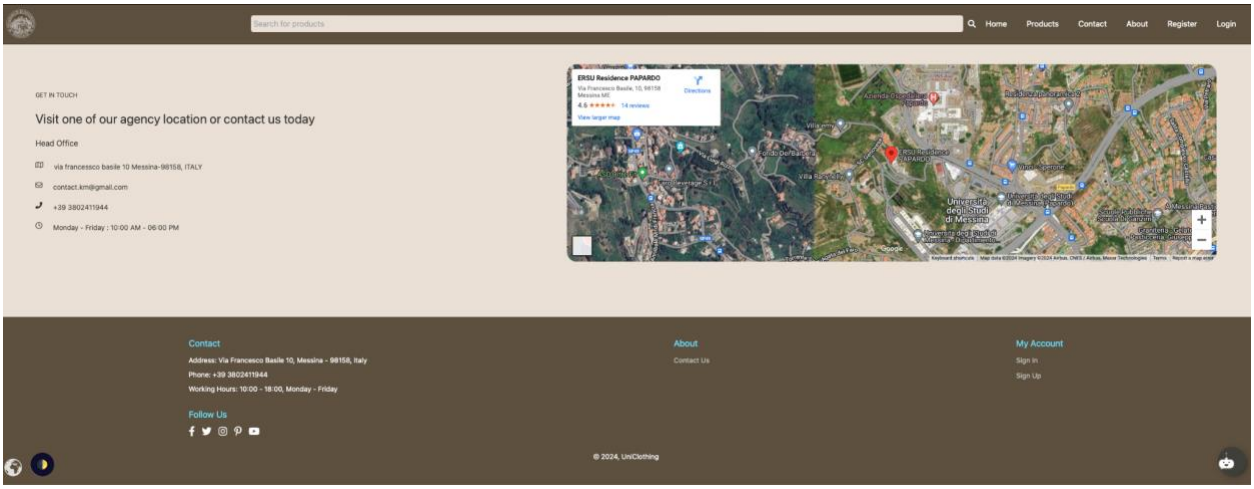


The Product Page :

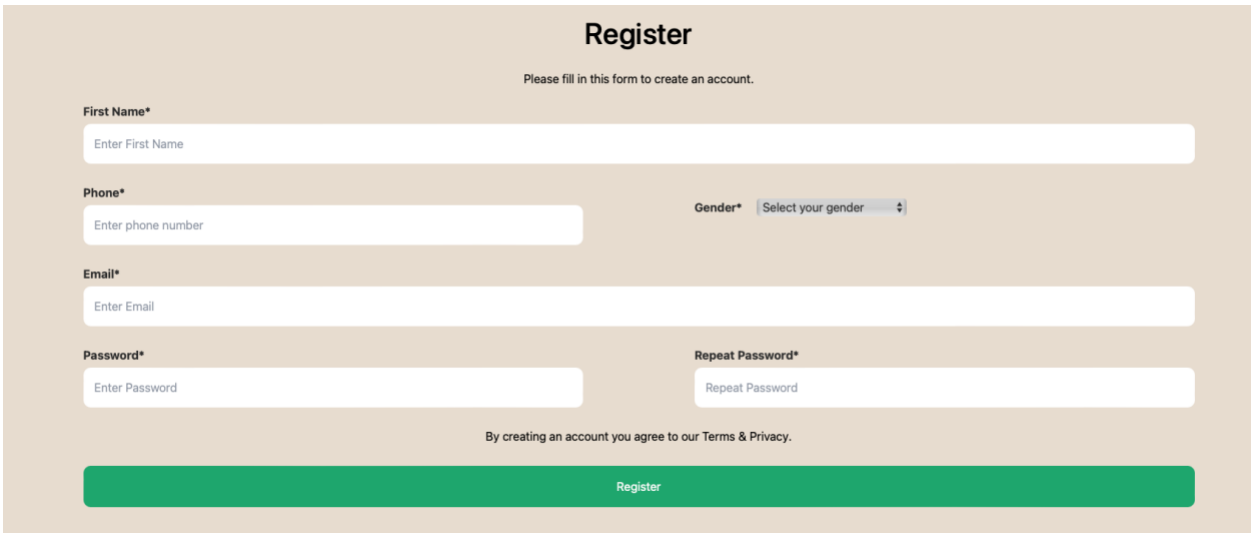


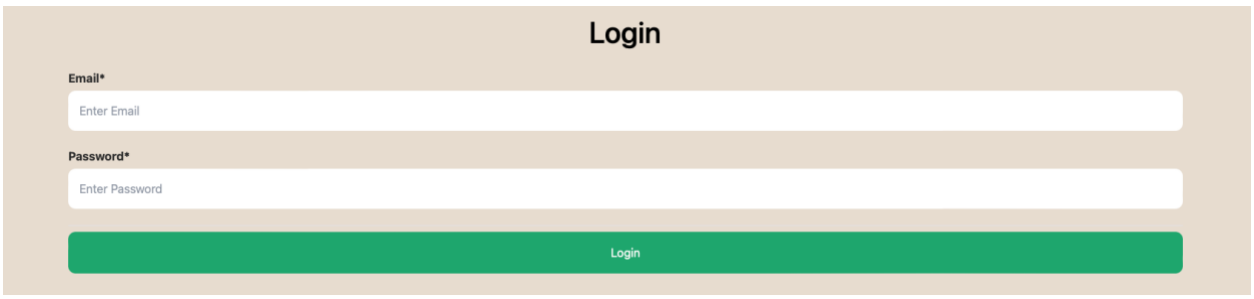
The Chat Bot:

The Contact Page :

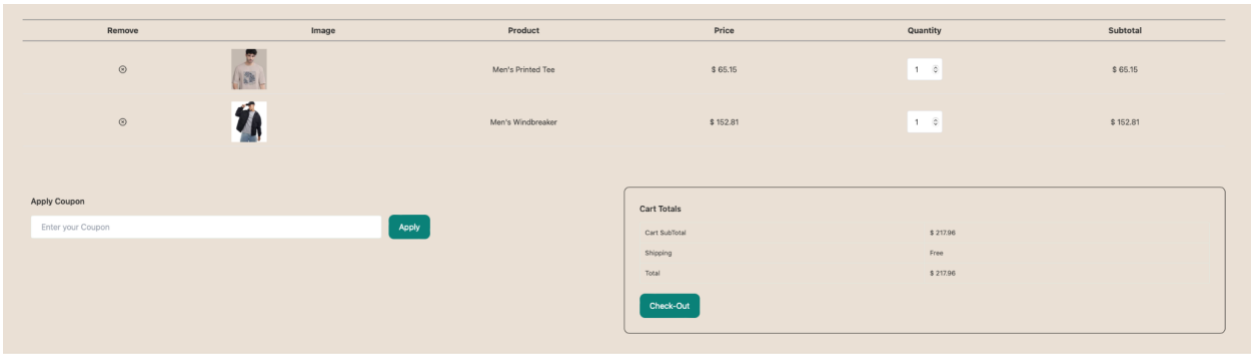


The Registration and Login Page :

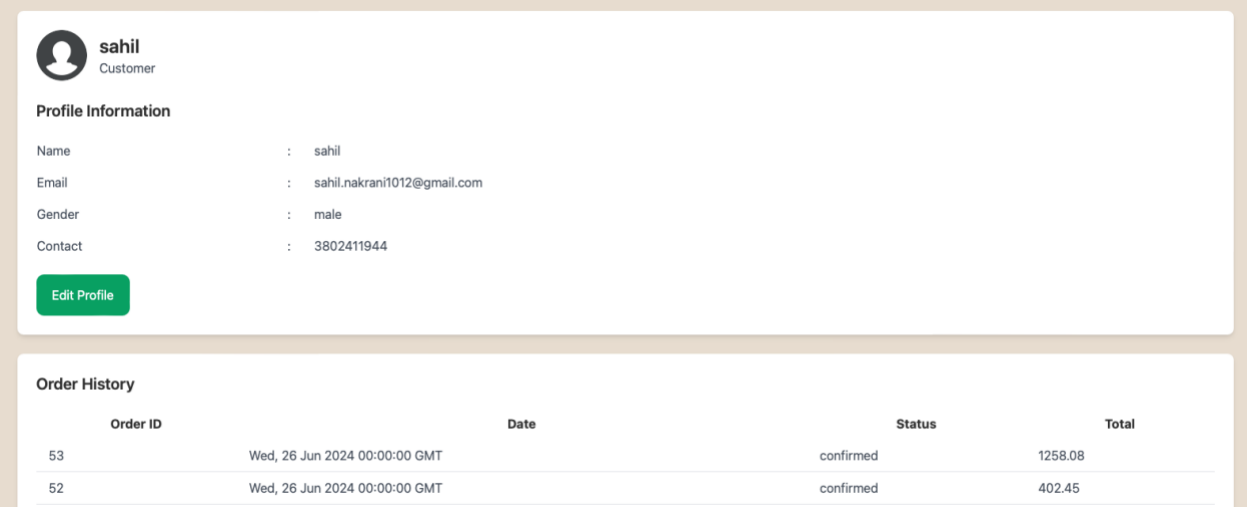




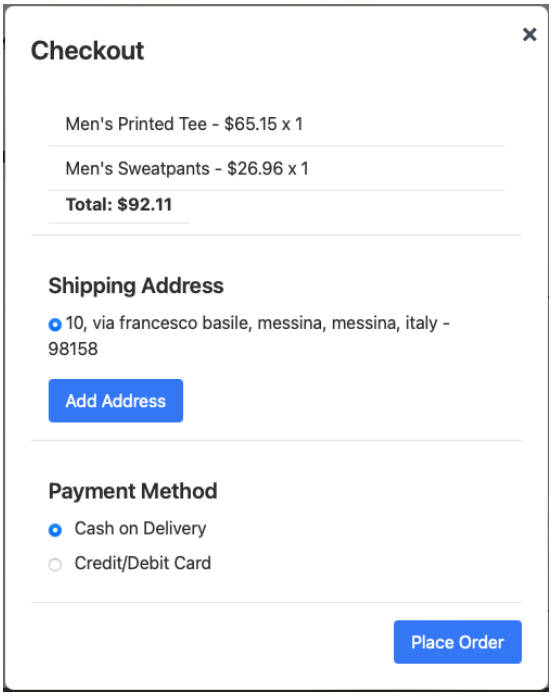
The Cart Page :



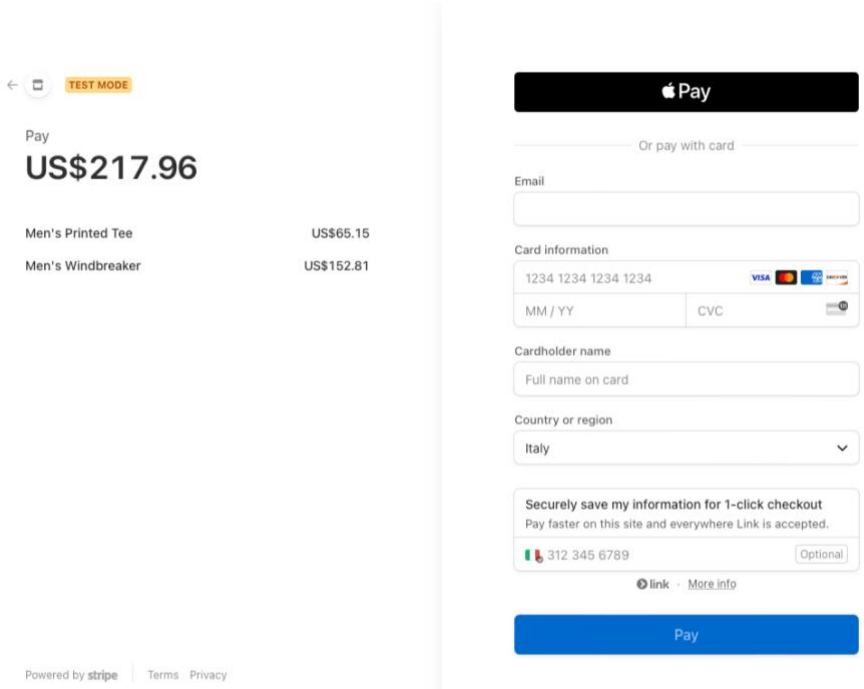
The Profile Page :



The CheckOut Modal:



The Payment Integration Page :



The Email of Order Invoice and Confirmation :

uniclothes.service@gmail.com
to me

Wed, Jun 26, 11:46 PM (8 days ago)

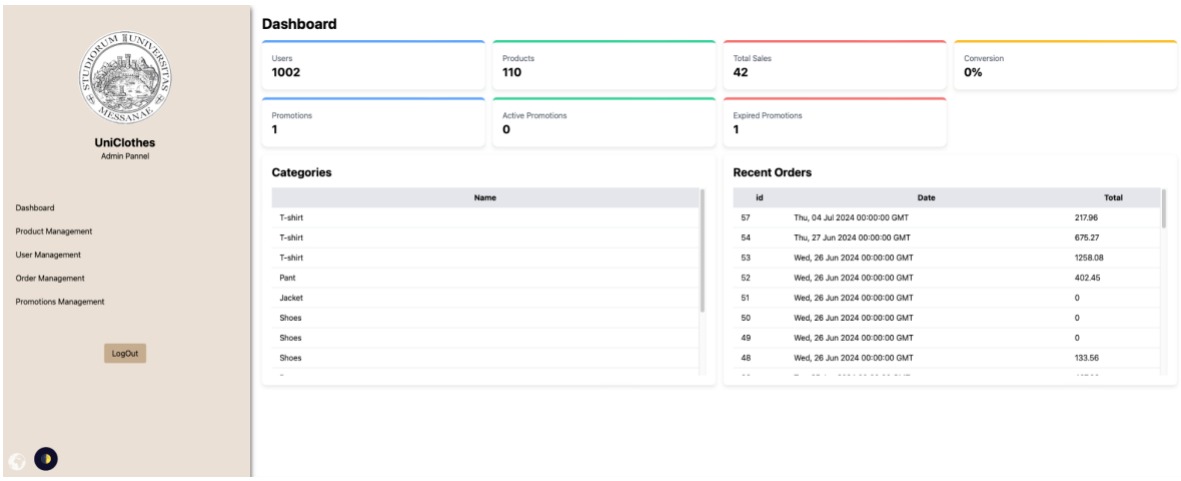
Order Invoice

Dear Customer, Here is your invoice and order summary :

Product Name	Price	Quantity	Total
Men's Printed Tee	$65.15	1	$65.15
Men's Vintage Tee	$643.97	1	$643.97
Men's Denim Jacket	$132.62	1	$132.62
Men's Athletic Shoes	$416.34	1	$416.34
		Total Amount:	$1258.08

Reply Forward

The Admin Panel :



Product Management

Total Products	Categories	Total Sales
110	13	42

Average Price	Maximum Price	Minimum Price
$470.46	$988.69	$17.07

Categories

ID	Name
1	T-shirt
2	T-shirt
3	T-shirt
4	Pant
5	Jacket
6	Shoes
7	Shoes
8	Shoes

All Products

ID	Category	Item	Price	Quantity
1	T-shirt	Men's Classic Tee	342.24	64
2	T-shirt	Men's Graphic Tee	582.75	53
3	T-shirt	Men's Printed Tee	65.15	66
4	T-shirt	Men's Vintage Tee	643.97	84
5	T-shirt	Men's Distressed Tee	268.89	35
6	T-shirt	Men's Pocket Tee	133.56	92
7	T-shirt	Men's Striped Tee	541.27	34
8	T-shirt	Men's Solid Tee	947.55	18

CRUD Operations

Create Product | Update Product | Delete Product

Promotion Management

Total Promotions	Active Promotions	Expired Promotions
1	0	1

Average Discount ($)	Average Discount (%)
0	20%

Recent Promotions

ID	Name	Discount Rate	Start Date	Expire Date
1	MEN20	20	Sat, 29 Jun 2024 00:00:00 GMT	Sun, 30 Jun 2024 00:00:00 GMT

CRUD Operations

Create Promotion | Update Promotion | Delete Promotion | Send Promotion Email

← Close this Section

Send Promotion Email

Email Content

Hello!! there.

Send Email

Order Management

Total Orders
42

Average Order Value
$585.961

Total Revenue
$24,610.38

Recent Orders

ID		Order Date	Order Total	Order Status
57	5	Thu, 04 Jul 2024 00:00:00 GMT	217.96	
54	1004	Thu, 27 Jun 2024 00:00:00 GMT	675.27	
53	5	Wed, 26 Jun 2024 00:00:00 GMT	1258.08	
52	5	Wed, 26 Jun 2024 00:00:00 GMT	402.45	
51	5	Wed, 26 Jun 2024 00:00:00 GMT	0	
50	5	Wed, 26 Jun 2024 00:00:00 GMT	0	
49	5	Wed, 26 Jun 2024 00:00:00 GMT	0	
48	5	Wed, 26 Jun 2024 00:00:00 GMT	133.56	

User Management

Total Users
1002

Male Users
331

Female Users
357

Active Users
42

All Users

ID	Name	Email	Gender	Phone
5	sahil	sahil.nakrani1012@gmail.com	male	3802411944
6	sahil	sahil1@gmail.com	male	0000
7	yoshan	yoshan@gmail.com	female	0000
8	henok	henok@gmail.com	male	0000
9	Kevin Li	paul91@example.net	other	3595546621
10	Alan Garcia	victor88@example.net	female	001-309-875-3120
11	James Fry	jellis@example.net	male	811-917-5632
12	Katherine Carlson	gwhite@example.com	female	(220)582-3959x3618

Conclusion

The development of our e-commerce clothing platform signifies a major success in utilizing modern technology to create a robust, scalable, and user-friendly online shopping experience. Throughout the various sprints of the project, the platform has meticulously evolved to integrate advanced functionalities, ensuring a seamless and efficient shopping experience for end-users and streamlined operations for administrators.

From the initial setup of the repository and architectural design to the detailed implementation of user authentication, product management, payment processing, language translation, chatbot integration, dark mode feature, and a dedicated mailing server, every aspect of the platform has been meticulously planned and executed. The use of Flask for the backend, Nginx for the frontend, and MySQL for data management, coupled with Docker for deployment, has resulted in a highly modular and maintainable system.

The integration of external services, such as payment gateways, chatbot APIs, Google Translator API, and dark mode packages, further enhances the platform’s functionality, offering a comprehensive solution for diverse user needs. The separate mailing server ensures reliable and efficient communication with users, reinforcing the platform's commitment to providing a superior user experience.

The successful launch and operation of the e-commerce platform represent just the beginning of a continuous journey towards innovation and excellence in digital commerce. As the platform evolves, it will remain adaptive to the changing needs of the market and technological advancements, consistently striving to deliver exceptional value to users and stakeholders. This commitment to ongoing improvement and responsiveness to user feedback ensures that the platform will continue to meet and exceed expectations, solidifying its position as a leading solution in the e-commerce domain.