Text

Description automatically generated with medium confidence

TESSA CONSULTING – E-COMMERCE SITE

(WEB APPLICATION)

SOFTWARE ENGINEERING

(Semester 5, 2023)

**Students:** **Mentor:**

Dardan Osmani (129239) Prof. Besnik Selimi

Nazif Selimi (129248)

Nimet Sherifi (129260)

Vullnet Azizi (129402)

**FACULTY OF CONTEMPORARY SCIENCES AND TECHNOLOGIES – TETOVO**

**February 2023, Tetovo**

Contents

[**1.** **ABSTRACT** 3](#_Toc127299757)

[**2.** **PROCESS MODEL** 4](#_Toc127299758)

[**2.1. Agile Development** 4](#_Toc127299759)

[**2.2.** **Extreme Programming (XP) techniques** 5](#_Toc127299760)

[**2.3.** **Scrum** 5](#_Toc127299761)

[**2.4.** **Dealing with changes** 6](#_Toc127299762)

[**3.** **SYSTEM STAKEHOLDERS** 7](#_Toc127299763)

[**4.** **REQUIREMENTS ENGINEERING** 8](#_Toc127299764)

[**4.1. User Requirements** 8](#_Toc127299765)

[**4.2.** **System requirements** 8](#_Toc127299766)

[**4.3.** **Functional Requirements** 9](#_Toc127299767)

[**4.4.** **Non-functional requirements** 10](#_Toc127299768)

[**5.** **SYSTEM MODELING** 11](#_Toc127299769)

[**5.1. Use-Case Diagram** 12](#_Toc127299770)

[**5.2.** **Sequence Diagram** 13](#_Toc127299771)

[**5.3.** **Class Diagram** 14](#_Toc127299772)

[**6.** **SOFTWARE DESIGN** 15](#_Toc127299773)

[**6.1.**  **Architectural Design Decisions** 16](#_Toc127299774)

[**6.2.** **Architectural Pattern** 16](#_Toc127299775)

[**6.3.** **Front-End Design** 18](#_Toc127299776)

[**7.** **SOFTWARE TESTING** 19](#_Toc127299777)

[**7.1.** **PHPunit** 19](#_Toc127299778)

[**7.2.** **Feature Testing** 19](#_Toc127299779)

[**7.3.** **Unit Testing and Test-Driven Development** 21](#_Toc127299780)

[**7.4.** **User Testing** 21](#_Toc127299781)

[**8.** **DEPLOYMENT & MAINTENANCE** 22](#_Toc127299782)

[**9.** **GITHUB** 24](#_Toc127299783)

# **ABSTRACT**

During the course of Software Engineering, we were introduced to many concepts relating to the development of a software system, from its initial conception to its operation and the maintenance that follows. All of the activities that were highlighted give us a roadmap to the inner workings of how teams and groups of individuals come together and organize their work in order to produce an end product that meets the needs of clients and consumers alike.

For our project in this subject, we decided to take on a customized software for a real-life client organization. The organization in question is **Tessa Consulting**, a retail market store based in Gostivar, North Macedonia, which sells hair products and commodities to regular customers and hair stylists nationwide. The company in question requested a website that would allow its customers and clients (hairstylists that it had agreements with) to buy Tessa’s hair products online from the site, and for the products to be delivered at their desired location. The project that we undertook consists of a server-side web application with all the functionality and features of modern E-Commerce sites. It will be used in the traditional Model-View-Controller (MVC) sense where data about the products will be retrieved from the main company database and that same data will be displayed to the users of the site who can purchase the products. This would essentially make Tessa Consulting an e-commerce business aside from it already being an on-location store.

We as a group saw this a great opportunity and realized that the best option would be to document our journey of development in this documentation. By the end of this endeavor, the goal is to create a seamless website with all the necessary features required for a customer to log-in, browse items, place orders, view their purchase history and more. Therefore, what follows is the utilization of all the points covered during our course in this subject.

# **PROCESS MODEL**

## **2.1. Agile Development**

One of the most crucial steps in Software Engineering is identifying the development process that will lead to the production of the software system. For this particular project, we decided from the beginning that we will use the concepts of **Incremental (Agile) Development**, and there are a couple of reasons why we decided this:

1. First, we believed that it would give us more **freedom and fluidity** when it came to going back and forth across the different phases of development. We didn’t want to restrict ourselves because we knew beforehand that alterations to the web application would be needed if necessary. It also had to do with the inexperience of the group, none of us could guarantee first of all, that a phase of development was completely done so that we could safely move on to the next one.
2. Second, it allowed us to develop the project in **stages**, therefore enabling us to showcase each stage to the project stakeholder to get their opinion and **feedback**. We would then use that feedback to make the necessary changes before moving on to the next phase. The idea was to avoid making drastic changes to the whole website and its functionality if the client wasn’t happy with the final version.
3. Third, the stakeholder made it very clear to us from the very beginning that the requirements and requests that were proposed were not set in stone and that we should be prepared for **minor changes** here and there in the way that the website would ultimately function. At this point, it was very clear to us that the way to go would be using agile methods, however further reasons contributed to this decision as well.
4. At the same time, the client stressed the importance of getting the website up and running as soon as possible, and when asked by us if he would accept the system to be deployed with the most crucial functionality first, with additional features to be added on at a later stage, he accepted. **Early delivery and deployment** of useful software is most feasible using agile methods, because of the ease of just going back on the development process and adding components on top of shipped components.

We also incorporated some aspects of integration and configuration, where we often relied on reusable components and templates to fulfill some of the requirements. All these components will be stated later in this documentation.

## **Extreme Programming (XP) techniques**

During development, we examined both Extreme Programming and Scrum methodologies to see if we could implement techniques that would guide our way of developing. Some of the principles of XP that we utilized were:

* Even though in the very first few weeks of development we split ourselves into two groups of 2, where one group focused on the design and look of the web application (**frontend**), while the other group defined the database and the relationships between entities (**backend**), we then started to collectively work on each increment of the software, and formed a **collective ownership**, working on all areas of the system.
* One of the members of the group was directly connected to the clients of the system that we were developing, filling the role of an **on-site customer**. Most of the time the changes to the requirements made by the client were requested through him, who provided insight and directives to the entire team.
* When new code and functionality was introduced to the system, the person who added that code had the duty of making sure that it did not break current existing features, and if it did, the person was automatically assigned the task to fix it, utilizing ideas from **continuous integration** and **refactoring**.

## **Scrum**

The above-mentioned XP practices assured a fast-paced environment where each member of the team knew their roles and did their best to fulfill their duties. Assigning said duties to each member of the group was done during daily **Scrum** like meetings where we reviewed the progress done and decided on new tasks or assigned different people to tasks that could not be completed previously. We had these meetings every day or every two days, which was usually how long it took us to complete our assigned tasks. The time that it took to develop a shippable product i.e., a **Sprint**, was about two to three weeks. After that the subsequent sprints lasted considerably shorter because the main product with the primary functionality was developed in the first increment.

Agile methods of software development proved to be very helpful when developing an e-commerce site due to their focus on delivering value to the end user through iterative and incremental development. Using these methods, we were able to prioritize customer collaboration and adaptability, allowing teams to respond quickly to changing requirements and customer feedback. This approach helped us ensure that the end product meets the specific needs of the e-commerce business and its customers. In addition, the ability to frequently demonstrate working software is much needed for continual refinement and improvement. Furthermore, the use of agile methods lead to increased team morale and motivation, as well as better project visibility and predictability. Overall, the agile approach to software development was highly effective and efficient way to build the Tessa E-Commerce site that we believe meets the needs of the client and consumers.

## **Dealing with changes**

As stated previously, one of the reasons why we chose to develop this project using agile methods is because our client told us beforehand that the requirements, were not final and that we should expect alterations to the demands. Therefore, we had to develop strategies to anticipate and cope with the expected changes.

* By developing the website in increments and showcasing each increment to the stakeholder to get their input and feedback, we avoided the massive costs of **rework** that it would have taken if we fully finished the project and then presented it to the client.

This would be considered a form of **prototype showcasing**, since very early versions of the system were developed with intention of checking with the clients’ requirements and whether they liked the design decisions.

# **SYSTEM STAKEHOLDERS**

When developing an E-Commerce website where people from around the country will be able to purchase products offered from your service, it is important to consider the different stakeholders who are affected or have a legitimate interest in the site.

There are several stakeholders involved in this e-commerce site, and they include:

1. **Customers:** The end users who will purchase products or services through the site.
2. **Business Owners:** The individuals and organization who own and who will operate the Ecommerce site.
3. **Developers:** The team responsible for building and maintaining the technical aspects of the site.
4. **Product Managers:** The individuals responsible for managing the products and services offered on the site.
5. **Marketing and Sales Teams:** The teams responsible for promoting the site and its offerings to customers.
6. **Payment Providers:** The companies that provide payment processing services for the e-commerce site.
7. **Logistics and Shipping Providers:** The companies responsible for delivering products to customers.

These stakeholders will each play a critical role in the success of the Tessa E-commerce site and must work together to ensure a positive customer experience. It’s also worth noting the relationships between the different stakeholders. The business owners hire the development team to design a site that will be used by customers. Once a customer buys a product, it’s the job of the payment providers to transfer the funds and the shipping providers to ship the product to its desired location.

# **REQUIREMENTS ENGINEERING**

Understanding the sort of requirements the client of the system requests, validating them and putting forward the requirements that the development team can implement on the software are important steppingstones as to what the system will inevitably look like. As all development processes, the first step to getting started on a new project is to fully understand the services that the system should offer and the constraints on its operation.

## **4.1. User Requirements**

The client layed out some demands that were standard for E-Commerce sites, and other ones that were very specific and had to do with how the store ran its sales and distribution network. Through techniques such as interviewing the client and different workers, we were able to pull the very essential user requirements:

**User requirements definition**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

The site should serve as a place where both regular customers and registered hairstylists can buy different hair products and items online. The regular customers will have accessibility to products based on hair type, brand, and category, they can place those products on their shopping cart, buy them and have them delivered to their homes via a postal service. In order to have access to the shopping cart, buy items, and see the orders that they made previously, users must have an account on the website, where they will be identified by name, surname, email and so on. Hairstylists on the other hand will have to make requests for their accounts, which must be approved by the administrator of the site. Once they have an account, they are granted special prices and offers on the same products (Hairstylists get cheaper prices than regular customers). Users of the site are given the possibility to pay with credit cards. Users of the site can view the different hairstylist courses that are held at our premises.

## **System requirements**

By examining the user requirements above we were able to come up with a detailed description of the website’s requirements in order to have a better view of exactly what is to be implemented. This detailed description is also referred to as a **functional specification**. This specification was then used by the team to brainstorm ideas on how to implement the points and to decide which of the points were more important to tackle first.

**System requirements specification**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

* 1. There will be three types of users: admin, customer and hairstylist. The view of the admin will be different to the customer and hairstylist. The admin will have the options to add, delete, edit products, view orders, handle users, view performance metrics and so on.
  2. The view that will be generated for the customer and hairstylist will be the same, with the only difference being that before displaying the price of products, a variable will determine whether that the user that is logged in is a regular user or a hairstylist.
  3. When a user requests an hairstylist account, an email will be sent to the admin with the information that was passed in from the user. The admin will then have the ability to accept the hairstylist request or not.
  4. Products need to be categorized by category, brand, hair type and the option must exist for the user to view the products based on what category, brand or hair type they select.
  5. An API needs to be implemented to handle card payment purchases.

## **Functional Requirements**

**Functional requirements** describe what the system should do and the features it should include. A good way to define functional requirements is by looking at the site from the point of view of the user, what is available to the user and what can he do with certain features. Some examples of the functional requirements for this website were:

* Product Catalog

A system for managing and displaying products and services offered on the site.

* Shopping Cart

A system for allowing customers to add items to a virtual cart and proceed to checkout.

* Payment Processing

The ability to accept and process payments from customers securely and efficiently.

* Order Management

A system for tracking customer orders, processing payments, and shipping products.

* Customer Accounts

The ability for customers to create and manage their own accounts, including saved billing information.

* Search and Navigation

Tools for helping customers find the products they are looking for, including search functionality.

* Mobile Optimization

A responsive design that provides a seamless shopping experience on mobile devices.

* Analytics and Reporting

The ability to track key metrics, such as sales and customer behavior, and generate reports for analysis.

* Customer Support

A system for providing customer support, such as email, and phone support.

## **Non-functional requirements**

In addition to functional requirements, **non-functional requirements** are also important considerations for an e-commerce site. Non-functional requirements describe the quality attributes of a system, such as performance, scalability, and security. Some non-functional requirements that were made clear to us by the business owner of the site include:

* Performance

The response time and speed of the site, including page load times, search results, and checkout processes needs to be quick and non-noticeable.

* Scalability

The ability of the site to handle increased traffic and demand. Even though the number at the beginning will be low, they’re bound to increase.

* Security

Measures to ensure the security of customer data, transactions, and the site as a whole. This is extremely important since customer will also input their credit card information.

* Usability

The ease of use and user-friendliness of the site, including navigation, search functionality, and checkout processes.

* Localization

The ability of the site to support multiple languages.

* Maintainability

The ability to easily maintain and update the site over time, including the use of modular, reusable code.

Meeting these non-functional requirements is crucial to the success of the ecommerce site, as they impact the overall customer experience and the ability of the site to meet business goals. Things like security and performance are standard features for sites like these, and they are just as important as the design and look of the site along with its functionality.

During the **requirements elicitation** phase, we also asked the business owner to come up with a story or scenario concerning how the hairstylists, who would have their own special accounts, would use the site. The way the managers were describing the role of hairstylists on the site wasn’t very clear to us, so the following user scenario really helped us understand what they meant.

**User Scenario**

The owner of a barbershop in Skopje, gets his supplies every month from Tessa Consulting in Gostivar. In the past he had to contact us via telephone or email in order to let us know the supplies he needs. In some cases, it was necessary for him to visit us on-location which may have been considered tiring by him, since he probably deals with a lot of clients at his barbershop daily. Now with the website, after having his account request accepted by us, he can simply log in to his account, select the products that he needs, add them to his cart, pay for them with his credit card, and finally have all the products delivered to his store via the postal service. Not only is the barber happy because of the simplified process, but also it will be an easier process for us since we don’t need to organize deliveries anymore.

After the requirements were identified by the development team, we checked back with the client in the **requirements validation** phase, just to make sure that none of the main requirements were changed and to relay to the client the features that were doable and the features that weren’t. After a consensus was reached, we knew that it was time to lay out a plan for the next stages of development.

# **SYSTEM MODELING**

Next, we needed to get different views and perspectives of the system that we were developing. **The Unified Modeling Language (UML)** give us diagram types that are used by software developers for a variety of different reasons, for example they can be used to derive system requirements. The following diagrams that we are about to show helped the engineering team understand the implementation of the system by generating the source code and to design the database relationships. The diagrams were also used to showcase the system functionality to the client and to compare the features with similar E-Commerce sites that are on the market.

## **5.1. Use-Case Diagram**

Use-Case Diagrams are used to model interactions between a system and the human users on the other side, basically it showcases what the customer expects from the system.

1. This first Use-Case Diagram demonstrates the steps of how a customer or hairstylist buys a product online from the website.

Diagram

Description automatically generated

This particular Use-Case Diagram was developed early-on in the development process to assess just how exactly the customers and hairstylist would interact with the system and how the system would interact with them. As you can see the users first must be logged in, and then when making a purchase they have the option of viewing a shopping cart, but it is a necessity to open the checkout page in order to view the products that are in your shopping cart and to give the information necessary to make a purchase. From there the checkout process also involves the payment methods and calculating taxing and shipping.

## **Sequence Diagram**

Sequence Diagrams are interaction-based diagrams (same as use-case diagrams) in the UML to model the sequence of interactions between the users and the objects in a system and the interactions between the objects themselves.

1. The following is a Sequence Diagram modelling the process of how a hairdresser logs into the system and purchases necessary products.

Diagram, schematic

Description automatically generated

As we can see from the diagram the system is displayed in three different layers:

* **The website** which represents the interface that the users interact with.
* **The system** **authentication** which is used to verify and authenticate different log-in request to the system.
* **The database** which serves as the hub from where all data is presented to the site.

## **Class Diagram**

Since the project we are developing is an object-oriented system, it is necessary to show the classes in the system and the associations between these classes.

1. This **sematic data model** was also used in the database design of the system.

Diagram

Description automatically generated

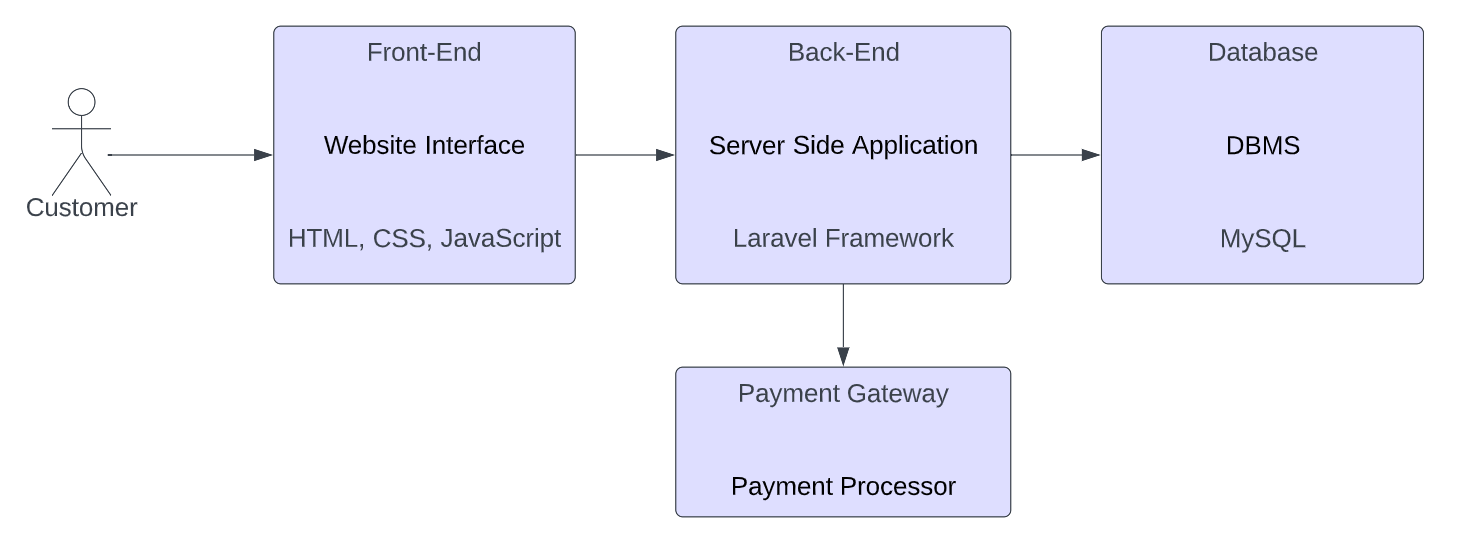
The class diagram for our system consists of 5 classes. Two classes of which, Regular Customer and Hairdresser are subclasses of the superclass Customer, meaning they inherit all the attributes of Customer, while adding to their own attributes.

As mentioned, the above class diagram helped with the design of the database entity-relationship model, where we were able to map the relationships and restrictions of the different classes.

# **SOFTWARE DESIGN**

The first stage in the software design process of the system is the **Architectural design.** The task was to identify the overall system architecture, a description of how the system will be organized as a set of communicating components.

The architecture of the Tessa E-commerce site will consist of the following:



1. **Front-end:** The front-end is the user interface of the e-commerce system. It consists of the website application that customers use to browse products, place orders, and interact with the e-commerce system. The front-end will be built using HTML, CSS, JavaScript.
2. **Back-end:** The back end is responsible for the processing and management of data. It consists of server-side applications and the database that stores customer information, product data, and order history. The back end is built using the Laravel Framework.
3. **Database:** The database is where all the data of the e-commerce system is stored. This includes customer information, product data, and order history. The database management system that is in use in our case is MySQL.
4. **Payment gateway:** The payment gateway is a secure system that handles the process of collecting and processing payment from customers. It integrates with the back end of the e-commerce system and communicates with the payment processor to confirm the payment and complete the transaction.

## **6.1. Architectural Design Decisions**

There are important **architectural design decisions** to evaluate when developing an Ecommerce site such as this one. It’s important to keep in mind that there will be no limitation to who can access the site and create their personal account there. Those same users will anticipate their personal data to be secure, the interface to be easy to understand and dependable, and the performance to be top-notch. Therefore, taking these points into consideration it’s important that the architectural design has features for:

* Scalability where the site is able to handle an increasing number of customers, products, and orders. The architecture is designed to support growth, with the ability to add additional servers, storage, and other resources as needed.
* Security, meaning the e-commerce site must be secure, with robust protection against hacking, fraud, and other security threats.
* User experience since the main audience who will view it are the customers of Tessa Consulting. The application provides a seamless and user-friendly experience, with a clear and easy-to-use interface, and minimal friction during the checkout process.
* Product management includes having an efficient and scalable system for managing products, categories, and product data. The architecture is designed to support product management, with the ability to import and export product data, and to manage product information, images, and other data at scale.

## **Architectural Pattern**

The Laravel Framework is used as a **Model-View-Controller (MVC) framework** to build server-side web applications. A model represents data that can live in a wide variety of databases. A view represents what the end user sees through the interface. In Laravel views are powered by blade templates that combine PHP and HTML code into a single file. Finally, we have a router and controllers in the middle which allows the user to safely interact with the data model. By using the Laravel Framework we automatically implement the MVC architectural pattern.

During the middle and later stages of project development and once the front-end design of the website was considered done, all four developers turned their attention towards the backend part of the application.

We encountered many benefits using the MVC pattern during development. The MVC pattern helped us to separate the different components of the site, making it easier to manage the code and maintain the site as the business requirements changed. We had a clear structure for organizing the code, making it easier to understand and maintain the site. The MVC pattern allowed us to test different components of the site separately, improving the reliability and maintainability of the code. We also reused code and components, reducing the time and effort required to build and maintain the site.

In our project, the model represents the data and business logic related to products, customers, orders, and other aspects of the business. The view is responsible for displaying the data to the customer, administrator, and hairstylist, in the form of HTML templates. The controller in Laravel is the router which handles user interactions, such as processing a request to add an item to the shopping cart and updating the model and view as necessary.

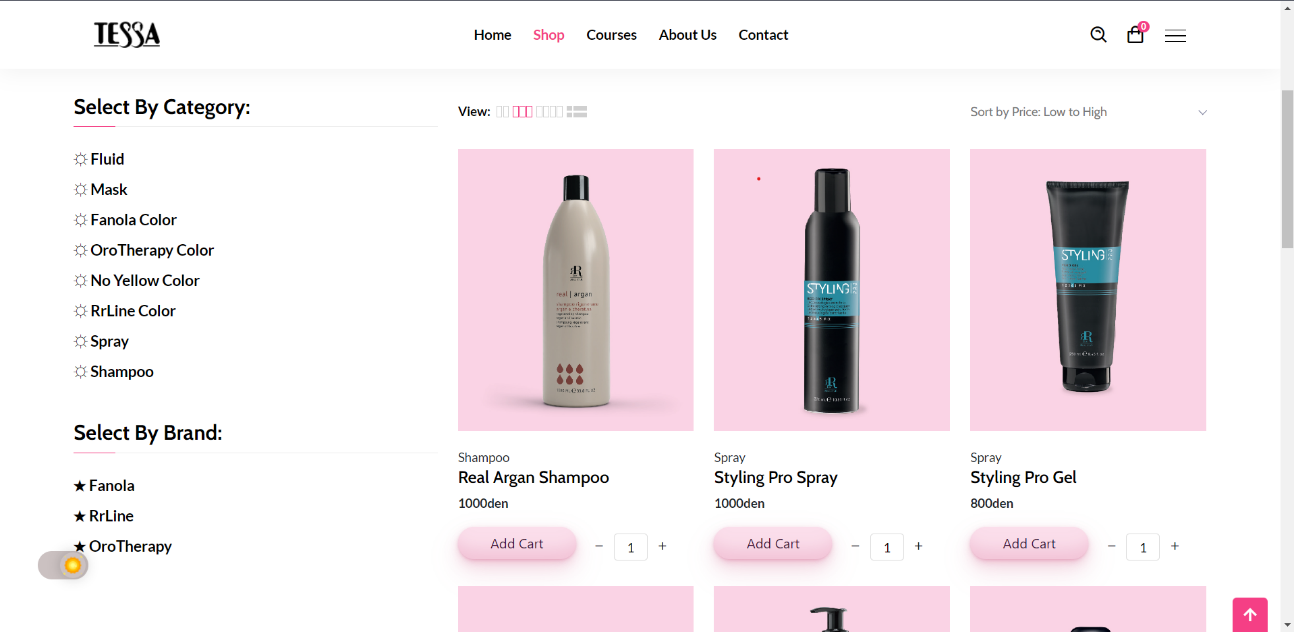
By using the MVC pattern, we were able to improve the structure and organization of our code, improve the reliability and maintainability of the site, and provide a better user experience.

Diagram

Description automatically generated

## **Front-End Design**

One of the major decisions that we made at the start of our development process was to purchase a ready-made template and adjust it according to the demands of the customer. The template in question matched the description of what the client wanted the website to look like, which was a simple and non-convoluted site with a minimalistic feel. Aside from that, the client didn’t get too involved on the design of the website, with the exception being the time he told us to try to incorporate the color pink, which was the main company color.



The website navbar consists of a home page, a shop page, a courses page, an about us page, and a contact page. The home page has hyperlinks to other pages on the site and also contains two sets of products, the most recent products and the most popular products. The shop page has the list of all products where you can choose to see products by category, brand, hair type etc. The courses page contains information on all the courses Tessa Consulting hosts for hairstylists and barbers. The about us page has details about the organization and the contact page contains a form where users can submit questions on any matter they choose.

While creating the shopping cart functionality we utilized the **Observer Pattern** to track changes in the shopping cart. The observer pattern is used to allow objects to observe and respond to changes in other objects, for example, in our case we used it to update the user interface in real-time as items are added or removed from the shopping cart.

# **SOFTWARE TESTING**

**Software Testing** is an important step in ensuring that the site is functioning correctly and providing a good user experience. Small scale **software inspections** were done by all members of the team on their respective code blocks. During the inspections many errors were found that could not be discovered through testing alone, and we took initiative to adopt the **D.R.Y.** (Don’t repeat yourself) principle, essentially getting rid of redundant and repeating code, so that the site will be better optimized for users.

Before we started to write test, we created a testing plan. This plan included the types of tests that we would perform, the tools and frameworks we would use, and the criteria for success.

## **PHPunit**

There are a great number of testing tools that are designed to test code and functionality on the Laravel Framework. PHPunit, Mockery, PHPSpec & Storyplayer are some of the most used testing tools out there. Since Laravel already has PHPunit integrated we decided that the best course of action was to use that.

* What is PHPunit?

**PHPunit** is a unit testing framework for the PHP programming language. It is an instance of the xUnit design for unit testing systems that began with SUnit and became popular with JUnit.

## **Feature Testing**

**Feature Testing** allows us to test the way individual units (features) work together and pass message. Feature tests will comprise the majority of our testing done on this application.

The next step is setting up the testing environment, including the hardware and software configurations, network infrastructure, and security requirements. It was important to ensure that the test environment is identical to the production environment, which enabled us to test the site under real world-conditions.

For our first set of feature tests, we decided to test the reliability and functionality of the **authentication system**. Authentication is a key component of the site's security infrastructure. By testing the authentication process, we can ensure that it is functioning correctly and that the site is protected against unauthorized access. Also, if the authentication process is difficult or confusing, users may abandon the site or become frustrated.

Graphical user interface, text, application

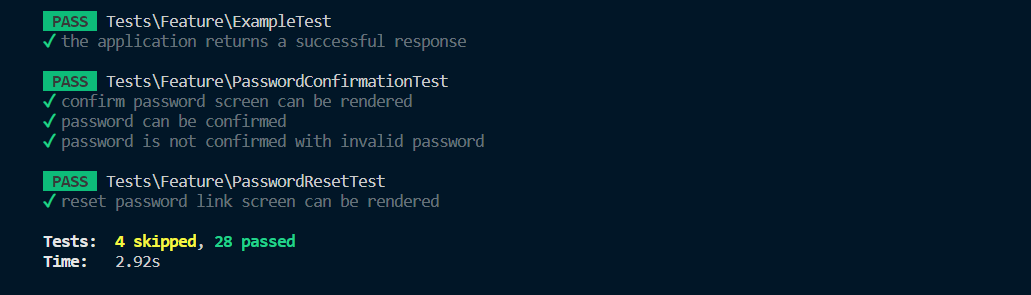
Description automatically generated Text

Description automatically generated

1. **Feature tests done** b) **Example of a Feature test**

After deciding which features to test and setting up the testing environment, we must generate dummy test data that we will use to perform the tests. This data was representative of the types of data that will be entered by real users of the site.

Next, we executed the tests we planned, using the test data generated. It’s important to record the results of each test, for example at the beginning the tests were failing because of an issue with one of the fields in the registration screen. After finding the issue and correcting it, all the tests passed successfully.



## **Unit Testing and Test-Driven Development**

**Unit testing** is a critical part of the software development process, and it is especially important for e-commerce sites, where reliability and functionality are crucial for the success of the business. Unit testing involves testing individual components of the site, such as functions and classes, in isolation from the rest of the system.

The way we incorporated unit testing into our application was through **Test-Driven Development**. Essentally, every time we added a unit of significance into our website we also wrote tests for that unit and made sure that it worked 100% before moving on to the next unit.

Graphical user interface, diagram, application, Teams

Description automatically generatedThis algorithm describes the way we organized our unit testing. Tests were added and then run, if passed we decided which unit to add next to the website. Once the new unit was added, we would write a test for that unit and the process repeated itself until the end. Once a unit was added and tested, all the previous tests were conducted again to make sure that new code did not interfere with implemented code. The practice of **Test-Driven Development** helped us keep the code clean and up to date, also the constant refactoring meant that the code was simplified making it easier to understand and maintain.

## **User Testing**

A form of user testing was also conducted where an early version of the system was released under a controlled environment in order to get the opinion and feedback of both some of the stakeholders and a few customers of the store. The small size of the testing audience meant that the release would be considered an **alpha test**. While we did receive some feedback telling us to adjust some of the features of the website by some of the users, most of the feedback was positive and most of the stakeholders who obtained a version of the system were left happy with the experience.

# **DEPLOYMENT & MAINTENANCE**

**The deployment phase** is the process of making the software system available for use by end users. This phase typically involves moving the software from a development environment to a production environment, where it will be used by the intended audience.

The domain name that will be used for the website will be <https://tessa.com.mk/>.

The website will be hosted online through [mkhost](https://mkhost.com/en/hosting/web-hosting/).

During the deployment phase, several tasks were performed, including:

1. **Installing software dependencies:** This involves installing any software libraries, frameworks, or other components that are required for the software to function properly.

The Laravel framework has a few system requirements. We had to ensure that our web server had the following minimum PHP version and extensions:

* **PHP >= 8.0**
* **Ctype PHP Extension**
* **cURL PHP Extension**
* **DOM PHP Extension**
* **Fileinfo PHP Extension**
* **Filter PHP Extension**
* **Hash PHP Extension**
* **Mbstring PHP Extension**
* **OpenSSL PHP Extension**
* **PCRE PHP Extension**
* **PDO PHP Extension**
* **Session PHP Extension**
* **Tokenizer PHP Extension**
* **XML PHP Extension**

1. **Configuring the environment:** This involves setting up the production environment with the necessary configurations, such as server settings, database settings, and security settings.
2. **Deploying the software:** After acquiring the domain name the website will be hosted and will be accessible to users online when writing the address in the URL.
3. **Testing the deployment:** This involves testing the software in the production environment to ensure that it is functioning correctly and meets the requirements specified in the design phase. This part is very important because even though the functionality was tested on the development environment, it’s important to assess that the website features were carried over to the online setting.
4. **Documenting the deployment:** This involves documenting the deployment process and the configuration of the production environment, so that the deployment can be replicated in the future if necessary.

**The maintenance phase** is the process of ensuring that the software continues to function correctly over time. This phase involves fixing bugs, adding new features, and making performance improvements to the software.

During the maintenance phase, several tasks will be performed, including:

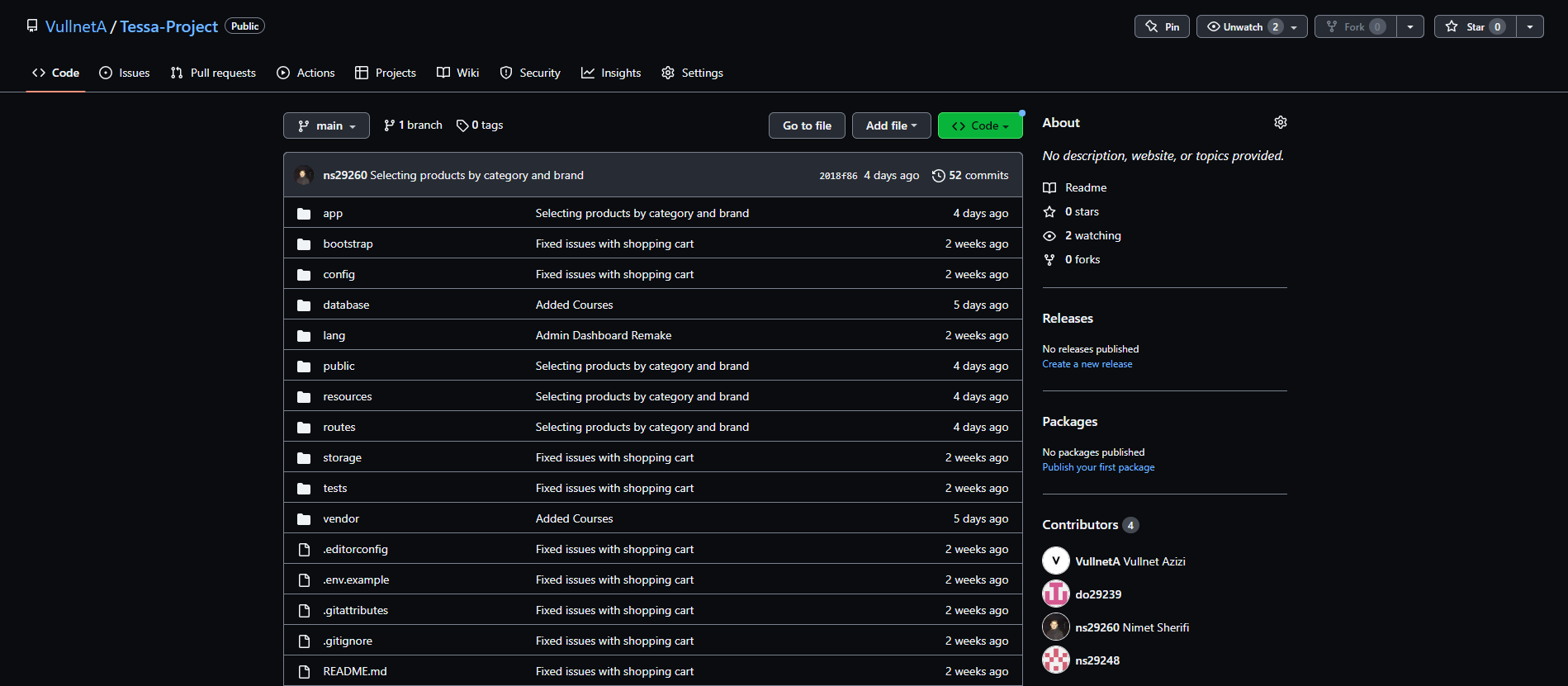
1. **Fixing bugs:** This involves identifying and fixing errors or unexpected behavior in the software.
2. **Adding new features:** This involves adding new functionality to the software to meet changing user requirements.
3. **Improving performance:** This involves making performance improvements to the software to ensure that it is running efficiently and meeting performance requirements.
4. **Updating documentation:** This involves updating the documentation to reflect any changes made to the software during the maintenance phase.
5. **Monitoring the software:** This involves monitoring the software for any issues and taking action to resolve them if necessary.

Overall, the maintenance phase is an ongoing process that is necessary to ensure that the software continues to meet the needs of its users over time. The development team will continue monitoring the site even after deployment.

# **GITHUB**

During the development of the website, we utilized Git and GitHub. GitHub provided a centralized repository for code and other project assets. This central repository was used to engage all members of the team on the project, allowing developers to contribute code and ideas, without interfering with each other’s work.

**Git version control** allowed us developers to track changes to the code over time, revert to previous versions, and collaborate with other developers on the project.



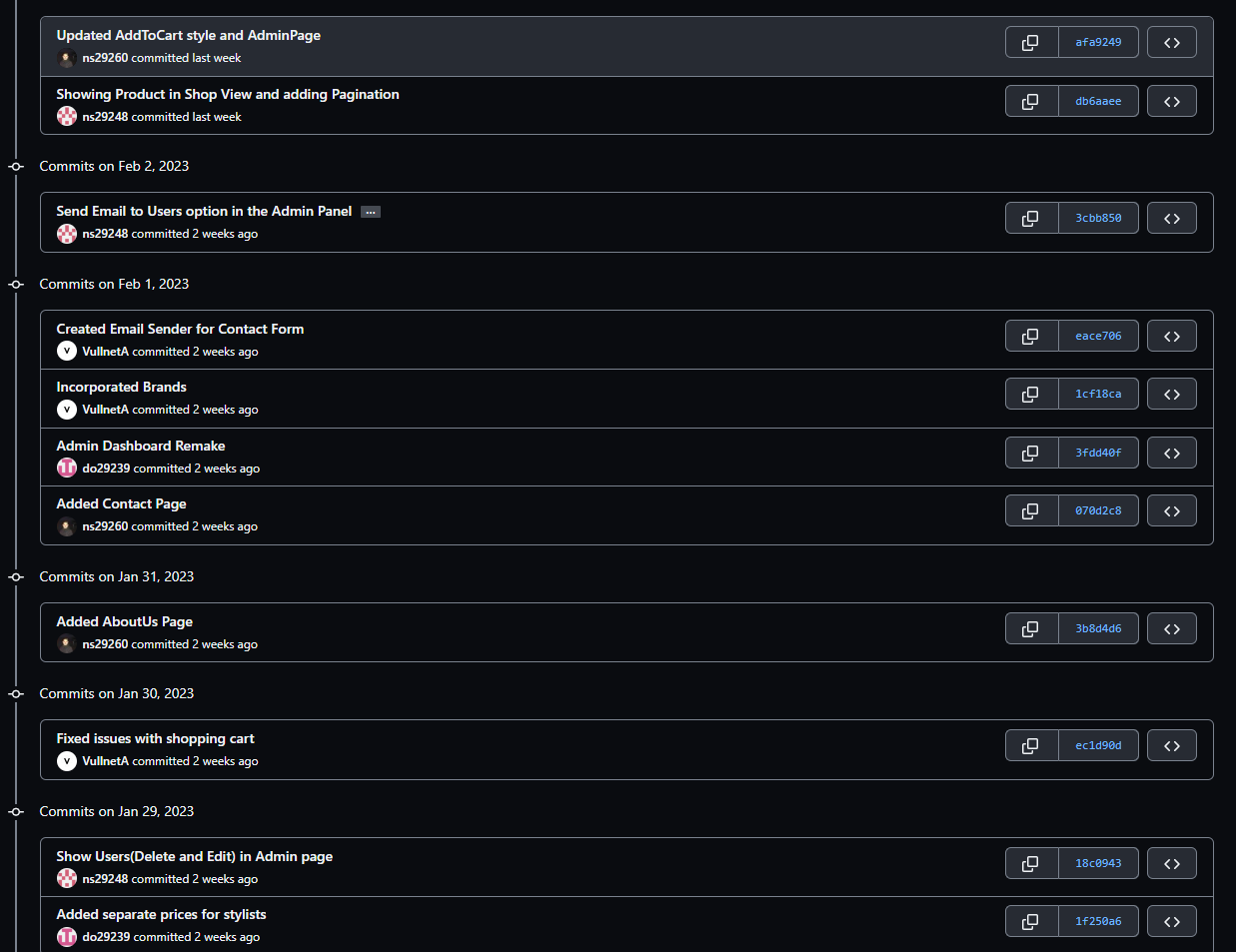
**An overview of our GitHub repository**

Pushing the code to the repository and pulling it from the repository to the individual workspaces of the developers was done with the help of GitHub Desktop, which made the procedure far easier than writing commands yourself on the terminal. Occasionally developers would branch of the project in order to attempt a feature to see if it works. If it did work, then they would merge the branches into one.

Capabilities like these made us feel safe and gave us a platform to go and attempt different ideas and not make us worry about the integrity of the code.

GitHub integrates with other tools such as **continuous integration (CI)** and **continuous deployment (CD)**, platforms which allowed us developers to automate the build, test, and deployment of the code.

Below is a screenshot of the various pushes and commits made to the project via the GitHub Repository. The repository also contains details such as a README about the website and the various UML Diagrams designed for it.



The GitHub Repository link is: <https://github.com/VullnetA/Tessa-Project>