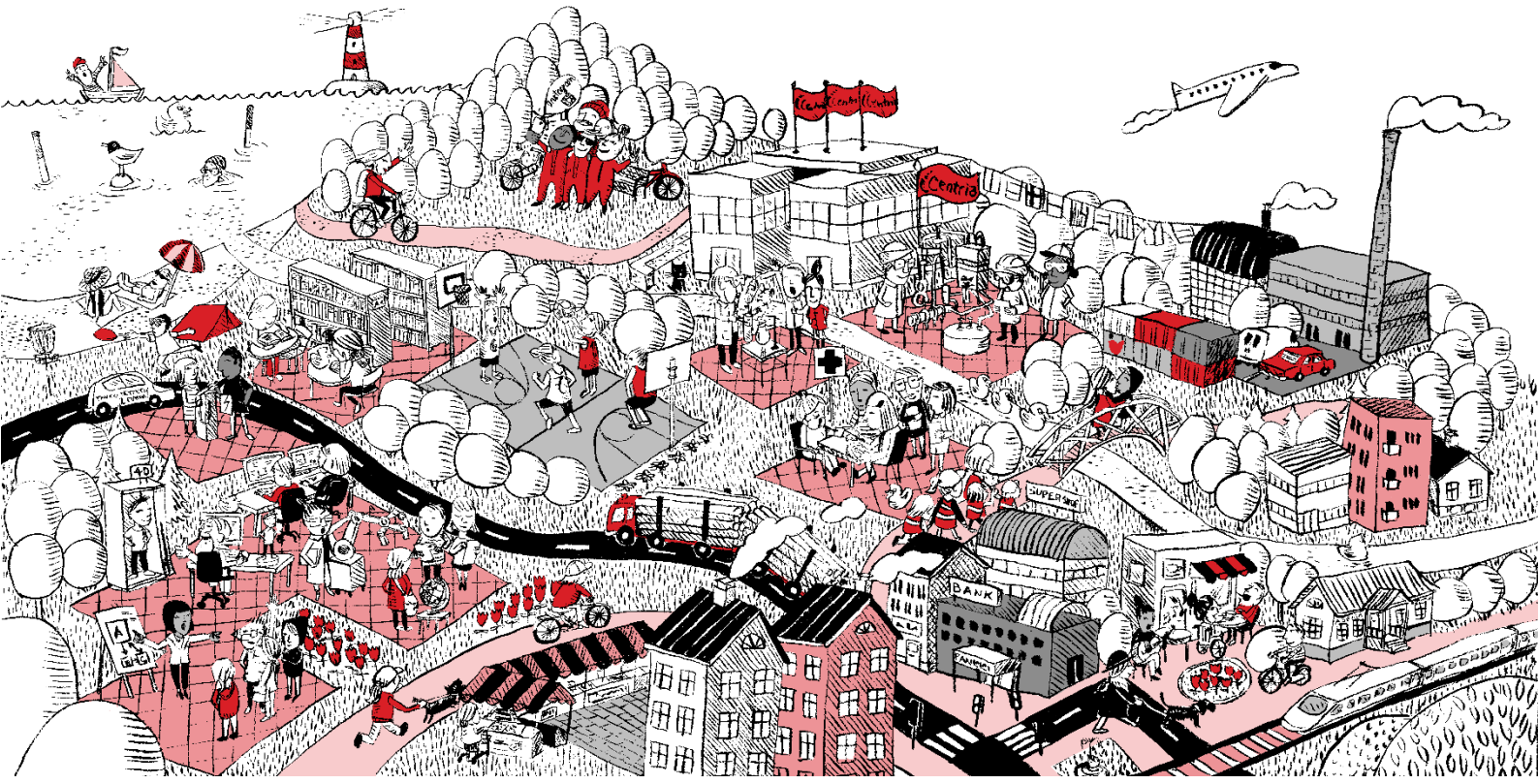
Vy Huynh

VASTE WEBSTORE – FRONT-END

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
IT Engineering
January 2021



ABSTRACT

Centria University of Applied Sciences	**Date** January 2021	**Author** Vy Huynh
Degree programme IT Engineering		
Name of thesis VASTE WEBSTORE – FRONT-END		
Centria supervisor Kauko Kolehmainen		**Pages** 57 + 14
Instructor representing commissioning institution Jyri Mäkelä		

Online booking of delivery helps both companies and customers to optimize their work. Most delivery companies need a platform to support their customer demand. Customers can also easily choose their options and find a suitable method to drive their order to the right place. The whole process of booking service, payment, and delivery management is fast and convenient by using a platform with API and a cloud database. It will save time and cost for both sides.

Vaste Webstore provides the most variable catalogue of products with smart order, allows the shop owner to add, edit and update their product conveniently, supports the customer in viewing, selecting and buying products with fast delivery and easy payment. It serves the delivery which is environmentally friendly and supports all product you want, gives the most detail and quality of product.

For this thesis, the process of building the Vaste webstore front-end would be presented. At first, the requirement of the platform must be gone through step by step and appling technical knowledge to analyse. UI/UX was performed from the prototype to the actual website. The combination of knowledge of HTML, CSS, JS, and the support of Bootstrap support on the success of the website. However, the functional and non-function requirements must be taken into account during the website development.

Key words
Bootstrap, CSS, Database, Front-end, JS, HTML, Platform, Vaste Webstore, Webstore.

CONCEPT DEFINITIONS

API

Application Programming Interface

CSS

Cascading Style Sheets

JS

Java Script

HTML

Hypertext Markup Language

UI/UX

User Interface, User Experience

ABSTRACT
CONCEPT DEFINITIONS
CONTENTS

FIGURES

TABLES

APPENDIXES

1 INTRODUCTION

Web development comes with a huge set of rules and techniques. Since computers can't communicate with each other the way people do, they require codes instead. Web technologies are the markup languages and multimedia packages computers use to communicate. The main markup language is HTML, the main style setting element for web is CSS, and programming languages such as JS, PHP, Python, so on. There are many other technologies shall be known to develop a website such as browsers, frameworks, libraries, databases, client, server, protocols, API, data format. The most famous thing when mention about website are front-end and back-end. Front-end is comprised of HTML, CSS and JSS which is the process how and where the website is shown. Back-end is comprised of server and database where functions, methods and data manipulation happen. Besides technology, website usability is also important. Usability refers to the quality of UX which is about effectiveness, efficiency and satisfaction of the user when interacting with the website.

A grand delivery platform will economically support for the company and customer. There the customer can see clearly and understand how the delivery services are offered and quickly choose the service. Based on that demand, I and my team in Summer Workshop, Annastasiia and Santeri received a demand from Vaste project to build a Webstore which is a website directly integrated into the Vaste delivery Platform. The main goal of the Vaste project is to to increase the cargo fill rate with parcels and lowering the need of individual deliveries with the use of route optimization, by trying to fill the vehicles with parcels and lowering the need for smaller deliveries with increased planning. The Vaste Webstore will be mainly catered to customers and shop owners, who will interact thereby purchasing and selling products, which will be then delivered by the drivers assigned from the Vaste delivery Platform.

The webstore is a website where customers can purchase products from shop owners, and then have them delivered to their desired location in a economical manner. The platform will require a good user experience for both customers and shop owners. The deliveries are made via an integration to the Vaste Delivery system through its API. The main way of delivery is to design an optimized route for delivery with the lowest cost and fastest time. By taking advantage of technology such as GPS, API, delivery industry put another step in the future. Consequently, the company can provide their service and customer can easily choose the service they want in a convenient way.

The logic behind the development is that data will be implemented and displayed by order. All the data of the company services will be saved in the cloud with multiple choices of privacy. The information would be shown to the customer after it has been verified from the system. The user when accessing the website can see and choose the favourite product and options of delivery. This website implementation is a constant menu of services and products, however, the orders are different based on the filter options.

In this thesis, the main part is to solve the technical requirements and develop the front-end for the platform. The thesis contains a mockup database for testing, Interface Design, and code implementation using HTML, CSS, and JavaScript.

2 PRODUCT VISION

A product vision, or product vision declaration, defines the product's ultimate long-term mission. Vision statements are optimistic and concisely communicate where the item wants to go and what it hopes to accomplish in the long term. The declaration should serve as a reference and reminder for all stakeholders interested in the production of sales about the common aim that is sought to be accomplished with the product. (Robin.2017).

In this thesis, the product vision is to develop a website that helps small and mid-sized delivery companies to find an efficient method to support their delivery plan and customer. A webstore is to be designed where customers can purchase products from shop owners, and then have them delivered to their desired location in an economical manner. The platform will require a good user experience for both customers and shop owners. The deliveries are made via integration to the Vaste Delivery system through its API.

The main way of delivery is to design an optimized route for delivery with the lowest cost and fastest time. By taking advantage of technology such as GPS, API, delivery industry put another step in the future. Consequently, the company can provide their services and customer can easily choose the service they want in a convenient way.

Web development is expediting at an aggressive rate. Better and user-friendly interfaces are in demand. When it comes to developing a successful web application there are a number of factors defining that success. Customers are eager to know different aspects of your product such as its cost, look and feel, and value for money. (Paul. K, 2018).

Businessmen and their customers are the main users of Vaste Webstore. In modern life, improving the webstore and services play an important role and is a nonstop mission. By collecting information and analyzing the delivery router and time, The Vaster delivery service will optimize the service and satisfy customer needs. Besides that, the purpose of the webstore is to save the environment as a result of lower carbon emissions of logistics. Vaste Webstore will be mainly catered to customers and shop owners, who will interact thereby purchasing and selling products, which will be then delivered by the drivers assigned from the Vaste delivery Platform where the company tries to fill the vehicles with parcels and lowering the need for smaller deliveries with increased planning.

3 SPECIFIC REQUIREMENTS

In this chapter, Vaste Webstore's specific requirements will define external interface requirements, functional, non-functional requirements which is the most important part of the planning and building of the Vaste platform.

3.1 External Interface Requirements

Integration to Vaste delivery Platform via their API. Delivery information will be sent to the platform whenever an order is made. Integration with third-party payment service, through which customer pay is extracted.

3.1.1 User Interfaces

This chapter summarises the identified high-level user interface requirements from specific Vaste platforms in mobile and web browsers.

TABLE 1: Registration Requirement.

UIREQ-01-Registration	
Description	The Vaste platform shall permit users to register and login. The platform must require registration for the shop owner while the customer can be considered as optional.
Priority	High
Dependency	
Risk	The lack of registration information will cause the shop owner to be unable to use the service, additionally, the information is required for the payments and shipment of the system.
Rationale	Shop owner must register to create their online shop and upload their products. Registration is not mandatory for customers, but it eases their shopping experience.
Additional attributes	The UI shall allow user to edit their information

As can be seen in table 1, the registration is required with high priority, especially, the shop owner must register to be able to use the platform. However, this registration is an optional choice for user.

TABLE 2: Customisation Requirement.

UIREQ-02- Customisation	
Description	The Vaste platform shall permit users to customise their privacy settings according to their personal needs, as well as to organisational or legal constrains, respectively. It shall allow the user to specify if they want to share the information publicly.
Priority	High
Dependency	UIREQ-01-Register
Risk	If information can't be customised according to the needs of an individual user, it might prevent that user from effectively using the platform. Lack of customisation may lead to lost customer data and decrease the security.
Rationale	Some information required for registration might not be something the users wish to share in public. Therefore, you must be able to choose which parts of your account information are visible for others.
Additional attributes	

According to table 2, the customisation has a high priority that decide the effectively using platform of users. The flexible in customisation will give the safe and comfortable for the user.

TABLE 3: Explanation Requirement.

UIREQ-03-Explanation	
Description	The Vaste platform shall display the terms related to the object that customer needs to fill in.
Priority	Middle
Dependency	UIREQ-01-Register
Risk	Some objects would be unfamiliar with the user, lack of explanation would lead to misunderstanding and input of the wrong type of data.
Rationale	There are different input formats in different systems, to unify the format, an explanation shall be provided.
Additional attributes	An error will be shown whenever the user inputs the wrong type of data.

The requirement in table 3 is explanation, that must be in the high priority as it gives the instruction to the user when using the website. If lack of of explanation would lead to misunderstanding and input of the wrong type of data.

TABLE 4: Comprehensibility Requirement.

UIREQ-04 - Comprehensibility	
Description	The UI of Vaste platform shall be designed in such a way that it is a comprehensible and understandable by the intended user group. It means that the user can understand the visualisation of each operating step of the underlying process.
Priority	High
Dependency	UIREQ-03-Explanation
Risk	If the user is unable to understand the UI, they might be unable to use it in the way it is supposed to be used.
Source	
Rationale	In order to be comprehensible, UI must be designed as simple as possible and give clear explanation.
Additional attributes	None

As in the table 4, the comprehensibility is very important, so that the UI design must be easy to comprehend and use. If the user is unable to understand the UI, they might be unable to use it in the way it is supposed to be used.

TABLE 5: Visibility Requirement.

UIREQ-05 - Visibility	
Description	The Vaste platform shall be designed in such a way that relevant information can be grouped and easily accessible and visible at all time. The information must be displayed in a way that does not disturb the user.
Priority	High
Dependency	None
Risk	If the user is unable to access the information in a convenient way at any time they want, it may lead to risky behaviour.
Additional attributes	The shop owner's name, the product's name shall be order in accessing, priority also in favourite shops and products. The product the customer has ordered will be grouped by shop owner group and type of products. The transaction history shall be ordered by date and be able to filter according to use purpose.

Another high priority requirement is visibility which is mentioned in table 5. It requires the Vaste platform shall be designed in such a way that relevant information can be grouped and easily accessible and visible at all time. The information must be displayed in a way that does not disturb the user.

TABLE 6: Payment Requirement.

UIREQ-06 - Payment	
Description	The platform shall use third-party payment services in order to avoid dealing with sensitive customer information.
Priority	High
Risk	With lacking payment information and methods, the customer cannot use the service to purchase the products. The main purpose of the platform is to help the customer and shop owner can do business, without payment the main purpose cannot be achieved.
Additional attributes	The third-party payment service integration requires in order to function properly.

Payment is a very significant requirement, as it can be seen in table 6, platform shall use third-party payment services in order to avoid dealing with sensitive customer information.

TABLE 7: PC and Mobile Display UI Requirement.

UIREQ-07 – PC and Mobile Display UI	
Description	The Vaste platform must be displayed on both computer and mobile browsers. In both browsers it must be clear, have a nice look and be easy to use.
Priority	High
Dependency	UIREQ-04 – Comprehensibility UIREQ-05 – Visibility
Risk	If the platform is not designed with PC and mobile versions, it will reduce the quality of UI when using.
Source	
Rationale	Two separate designs UI for Mobile and PC must be carried to create the suitable layout for two versions.
Additional attributes	None

According to table 7, PC and Mobile Display UI play an important role with high priority. The Vaste platform must be displayed on both computer and mobile browsers. In both browsers it must be clear, have a nice look and be easy to use. If the platform is not designed with PC and mobile versions, it will reduce the quality of UI. Additionally, Two separate designs UI for Mobile and PC must be carried to create the suitable layout for two versions.

TABLE 8: Product Filtering Requirement.

UIREQ-08 – Product Filtering	
Description	The Vaste platform navigation bar contains categories for products. These categories will include things like products by type, products by company and products by storage location. The search bar will search for products based on name and description.
Priority	High
Dependency	
Risk	Without product filtering, navigating the site would be difficult. It will lead to inconvenience for the customer while using the website.
Source	
Rationale	Product filtering helps the customer easily find what they are looking for.
Additional attributes	None

Product Filtering Requirement is a function allows user to filter the grouped information of products by type, products by company and products by storage location. Without product filtering, navigating the site would be difficult. It will lead to inconvenience for the customer while using the website. Product filtering helps the customer easy to find what they are looking for.

3.1.2 Other Interfaces

The hardware interface holds an important role to run the website in the right places. Mobile phones and computers serve as the hardware interfaces for the UI, thus not giving us many limitations to design, Pad UI support can be considered. Another significant factor is the software interface which decides the success of the website according to the requirements. In order to itegrate to Vaste delivery platform via their API, delivery information will be sent to the platform whenever an order is made. The 3rd Party payment processing services will be used for a safer overall system and fewer problems with money. Last but not least, communications protocols as HTTPS will be used for the website itself as the main communication protocol for users.

3.2 Use cases Diagram

A use case diagram represent user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. To develop the Vaste web store, use case diagram helps developer understand the relationship and interaction of each use case. This thesis just gives some basic use case to have a general view on the project as the figure 1 shows on the next page.

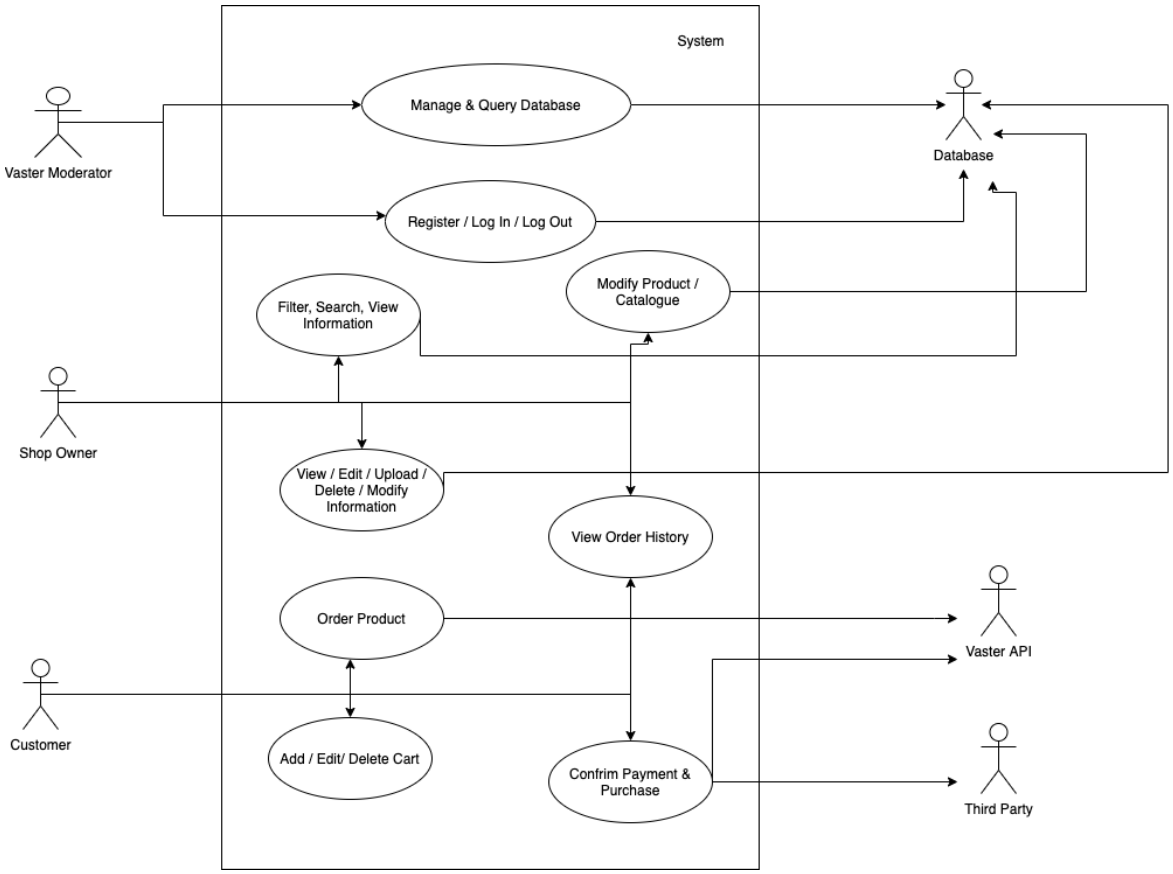


FIGURE 1: Vaste Main Use Case Diagram

There are some main Actors for the Vaste Website which play the role in the management and interacting with the system by the user involving. The figure 2 shows about actors of the Vaste web store: The customer is the user who enters the site to buy products, the shop owner is the user who posts products to the site for customers to buy. Vaste Moderator is the user who is in charge of moderating the webshop. Vaste Delivery API is the interface through which order deliveries are handled. Vaste Webshop Database is where customer, shop owner, etc, information is stored. The use case diagram also shows the main use case of the project according to each actor. In detail, the shop owner can be able to filter, search, view information; they are able to view, upload, delete, modify their own information; they are able to modify product and catalogue; they are able to view order history. The customer can be able to order product; they are able to add, edit, delete cart; they can view order history; they can confirm payment and purchase. The Vaste moderator can be able to manage and query database; they can register, log in, log out.

3.3 Functional requirements

The definition of a functional requirement is: “Any requirement which specifies what the system should do.” In other words, a functional requirement will describe a particular behaviour function of the system when certain conditions are met, for example: “Send email when a new customer signs up” or “Open a new account” (Eriksson, 2020).

3.3.1 ShopOwner functional requirements

Vaste webstore requires that Shop Owners will be able to register themselves as an online shop. After registering, they are able to login and set up their online shop with general information. The important thing is that they shall be able to modify their product as add, list, edit, delete or select which ones to feature. Checking statistics like how many products have been sold, or when the product was originally added, and last edited is the highest priority function to help the shop owner manage their online business. In addition, shop owners shall have the option to close their account when they do not want to use the platform.

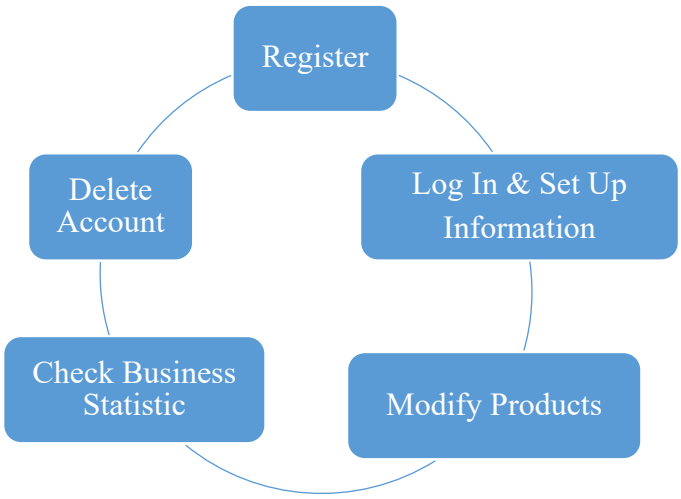


FIGURE 2: Shop Owner Functional Requirements

This Figure shows the functional requirements for the shop owner. There would be the register functions, Log In & Upload Information functions, Delete Account functions, Check Busniness Statistic functions and Modify Products functions.

3.3.2 Customer functional requirements

Customer's functional requirements set the role for the customer when using the website. In details, the customer can only see the product details, such as price or description. They are able to register themselves; set up their information; search for the product or the shop they want to find; filter the product by type or company name; edit, update and set private setting for their information; see the explanation of the objects; pay their service with a 3rd party payment method; and delete their registered account. These are the main requirement for a webstore according to the project requirements. In a real project, other functions can be added into the project base on the situation and other extend requirements.

3.3.3 System functional requirements

System functional requirements set the role and activities of the system when the website is run. In detail, there are some principal functions that must be included: the system will send the confirm information to the user's email address. When the customer makes an order, the system automatically sends a delivery order to the Vaste Platform and marks it down in transaction history. Simple "How To" pages, where it is instructed how to perform each action must be included. Shopkeepers must make a request to moderators before they can open a shop, before that thy cannot be seen on the site. Moderators will be able to approve, edit, and delete shops and customers. Moderators will be able to create new user accounts, and assign user roles (user roles: customer, shopkeeper, moderator). Customer and Shop Owner accounts will be verified by email.

3.4 Non- Functional requirements

The definition of a non-functional requirement is: "Any Requirement That Specifies How The System Performs A Certain Function." In other words, a non-functional requirement will describe how a system should behave and what limits there are on its functionality. Non-functional requirements cover all the remaining requirements which are not covered by the functional requirements. They specify criteria that judge the operation of a system, rather than specific behaviours (Eriksson, 2020). Load times need to be preferred as quickly as possible, somewhere in between near-instantaneous and one second. Customer satisfaction with the site needs to be focused on, and, nowadays, people are used to fast internet and quick loads.

3.4.1 Reliability

Customer information will be checked before it is accepted by the order form in order to avoid possible misspellings. In addition to the receipt the customer receives to their email, they will also receive a confirmation mail afterward for their delivery having been started. All of these actions are to protect the customer rights and also help the shop owner to check the order status or process whenever errors occur. The reliability is very important to build the webstore, especially, for The Vaste platform which contains both webstore and delivery services.

3.4.2 Availability

Both customers and shop owners are free to create accounts for the site. Shop owners can also list their products available for purchase and delivery through the site. Since the product is a webstore, anyone with a computer or a phone, and an internet connection can access it via a web browser. The availability to access in multiple devices give a good feedback from customer and encourage them to use the website more frequently.

3.4.3 Security

Customer data is to be handled safely and securely. Any credentials need to be encrypted and sent via secure, encrypted transmissions only. Additional measures should also be taken to ensure that all shop owner accounts created are legitimate, in order to avoid false shops and products from appearing on the website. Customers are also able to choose what information they wish to give out.

3.4.4 Maintainability

The system will be maintained by the Vaste team, who take in support requests and handle future updates to the site. They are able to verify shop accounts, delete accounts, alter account information at customer requests, and generally make sure that the site is kept in operation. To obtain satisfactory maintainability the following factors must be considered: The error of equipment or machine; maintenance displays, check points, gauges, meters and the position of one assembly with

respect to others; the human frame limitations; the maintenance environment; and the design of test equipment.

3.4.5 Portability

Vaste Webstore is a website, so portability will mostly be a non-issue between browsers. Some portability problems with specific commands might need addressing between Chrome, Edge, and Firefox, however. Mobile use will require its UI settings as well. Pad portability can be considered.

3.4.6 Performance

The website needs to be able to handle multiple customers browsing and buying products at the same time. Shop owner traffic will not be quite as frequent but might have spikes of activity when they add multiple products in a row. Any decent enough web server should be able to handle it. The landing page supporting 1 thousand users per hour must provide 3 seconds or less response time in a Chrome, Firewall desktop browser and 4 seconds on the Mobile browser, including pictures and text. The PC and Mobile versions must assess the highest workload to meet the performance requirements. Considering the maximum load that the system can operate in the future when the workload is increased.

3.4.7 Scalability

The webstore will be built with scalability heavily in mind, as it will be a growing platform. Registration is made as automatic as possible, alongside with adding more products. Scalability can be measured by the comparison of the increase in device output to the increase in used capital. Scalability also includes the ability to incorporate new services while maintaining the core node's function unchanged. Adding resources only leads to a small improvement in efficiency in a system with low scalability, and after hitting a certain level, improving the resources would not have any impact at all. There are a variety of similar principles, in addition to scalability, that help explain the problem better. Recoverability is one of them.

4 VASTE WEBSITE ANALYSIS

This chapter contains the analysis of the Vaste website with main diagrams such as activity, class, and database. There are three important logics to set for website functions, information connection, and data modification. Deeply analyzing the algorithms and data structure helps to clear the vision of building a website.

4.1 Activity Diagrams

In this chapter, the activity diagram will be presented. It describes the main activities of the user when access to the website. The step by step that the system will modify according to the user's actions. In each activity, the system will react in a different way, therefore, activity diagram helps to easily understand how the action are processed.

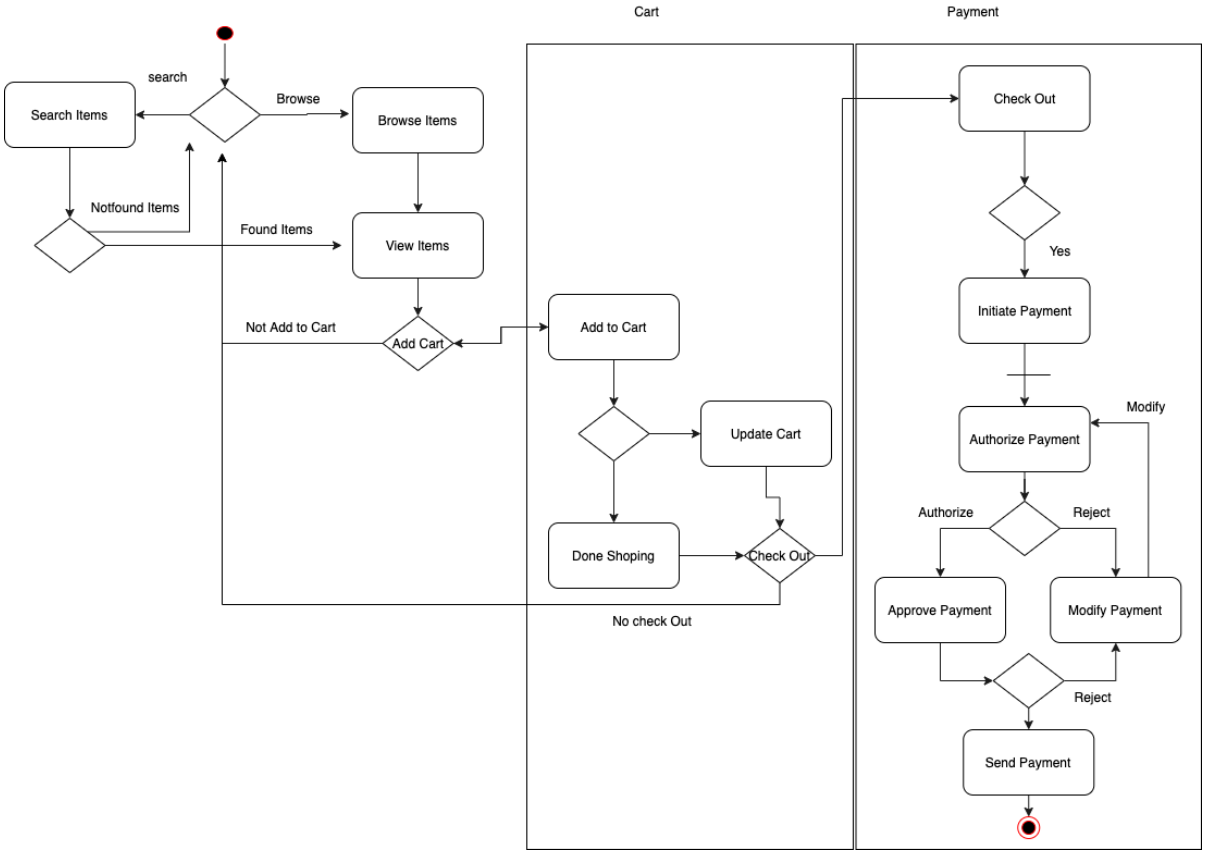


FIGURE 3: Vaste Browsing and Buying Diagram

As presented in figure 33 Browsing and Buying process is how a customer adds items to shopping cart and buys them. This activity diagram presents the logic for the Customer (who is registered or not) to Browsing and Buying, that is on the product page, the user can choose to browse the Item or search for the item they are looking for. If they can find the items, the view state will be active and they can decide to add the product to the cart or not. If not, they can go back to search or browse for other items. When the function “Add to Cart” is active, the Cart class will modify the information about the items that the customer has added to the cart and display an option for the customer to “Update Cart” or “Done Shopping” then lead them to “Check Out” section. In case the customer does not want to checkout, they can go back to shop, otherwise, Payment will be activated. In the Payment section, the customer needs to input their cart information on the Initiate Payment State, after that, the Authority must be confirmed by the third party; If the customer fail to verify for 3 times, the system requires to modify again until verify and payment will be processed. Products show how the shop owner adds new products and edits products to the site after they have logged into their account.

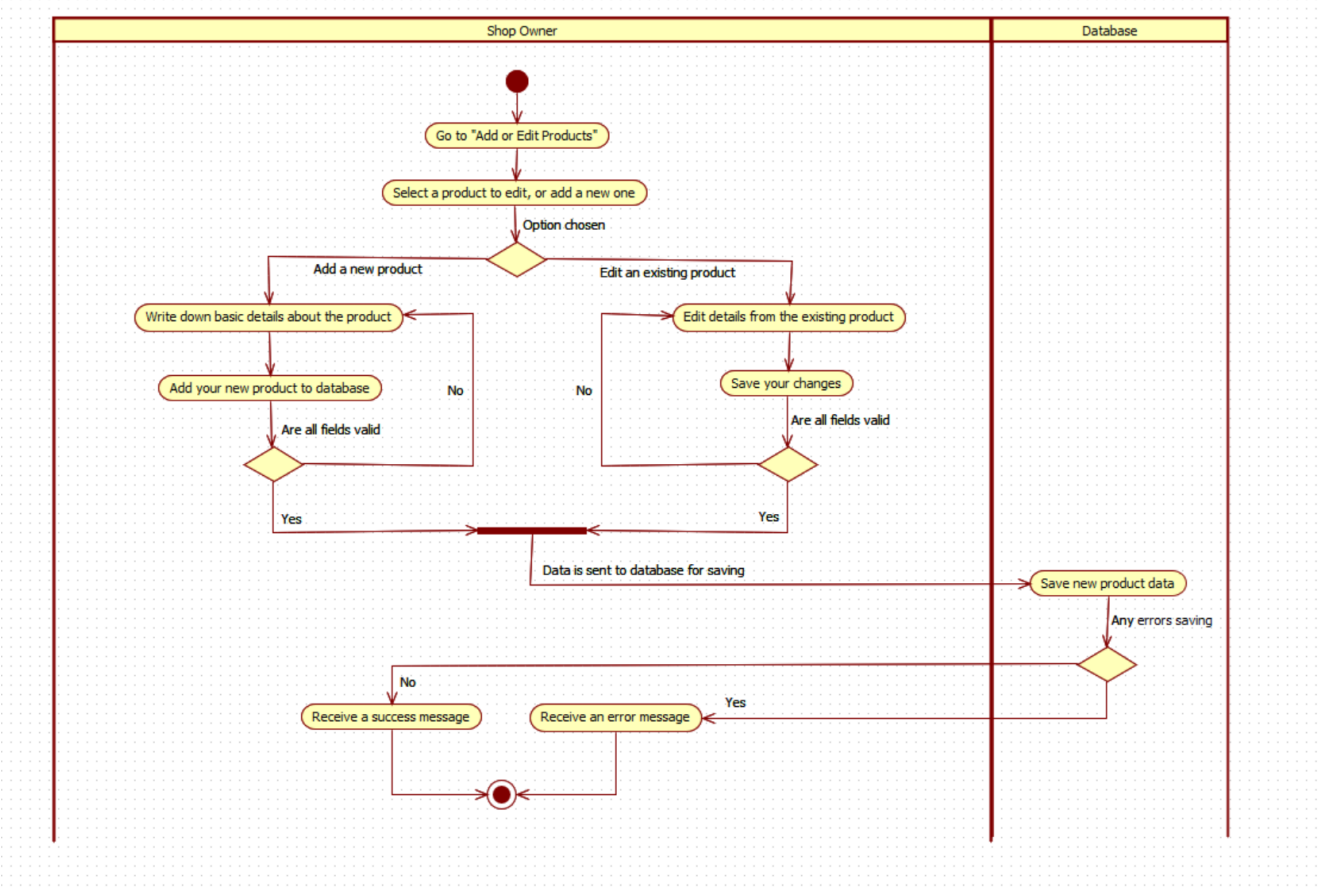


FIGURE 4: Vaste Add and Edit Product

By clicking on the button “Add or Edit Products”, Shop owners can choose the product they want to add or edit. If they want to add a new product, they need to fill in the information and upload pictures for the product to the database, and the product also has been checked if it already existed. Otherwise, when the Shop owner wants to edit their product they can choose the product from their database and start to edit and save all-new changes. When the button “Save” is clicked, the System Database will be uploaded, and an error check will be sent to the Shop owner.

Customer Activities:

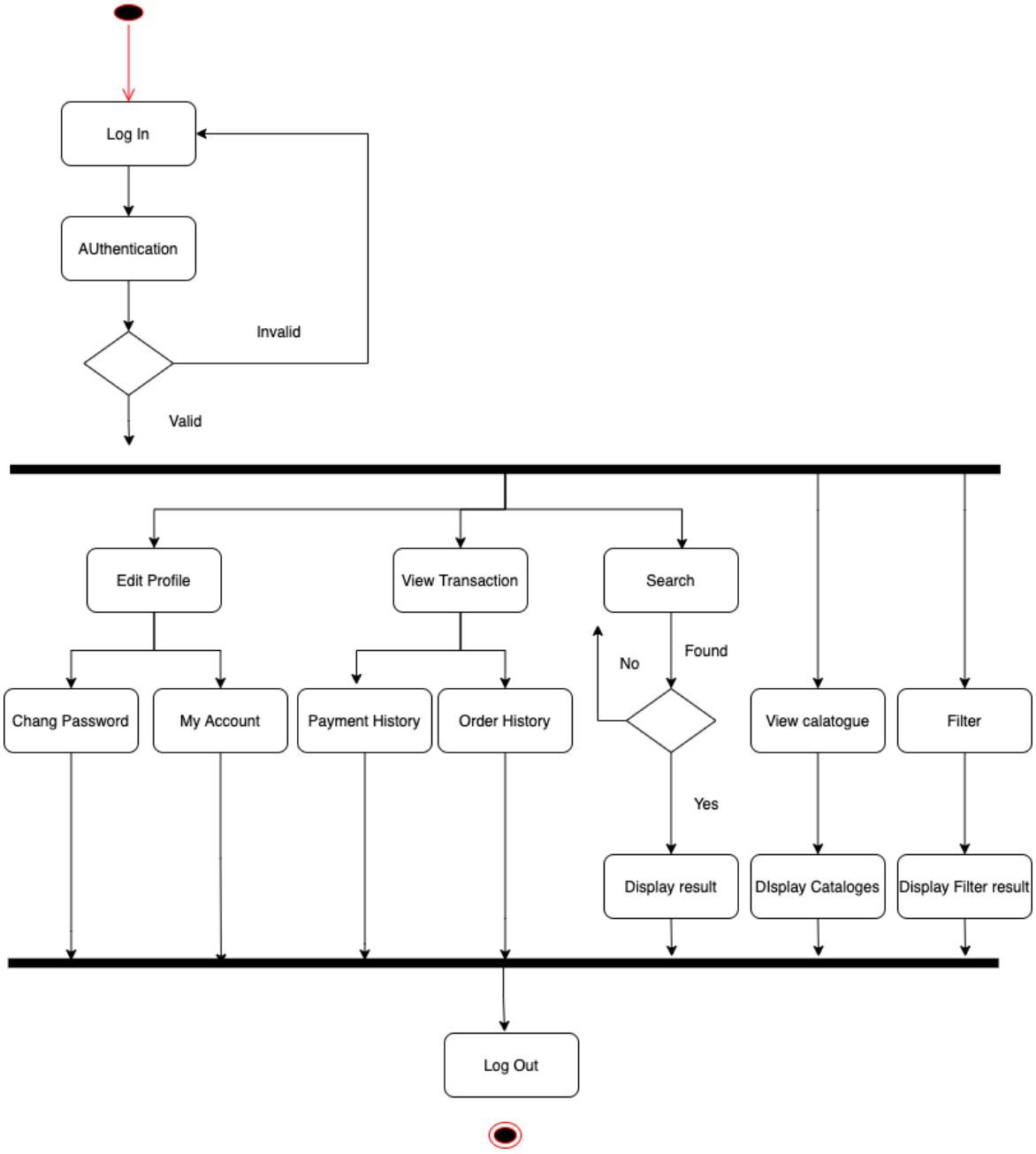


FIGURE 5: Vaste Customer Activities Diagram

For customers who have registered with a user account and password, they have the basic activities as shown in the Figure 3: Vaste Browsing and Buying Diagram. In addition, they have more activities as they can check and Edit their profile and View Transactions. The system will support them to change password and edit account information. In advantage, they can check their payment history and Order history to track their delivery status.

4.2 Entity-Relationship (ER) Diagrams

This chapter will present the entity-relationship diagrams which describes interrelated things of database from Vaste webstore. It shows the structure of the website's database and the connection of them. The ER diagram helps developers to understand the algorithms and data structure of the website, also how to implement functions according to the description. The ER diagram in figure 6 mostly supports for backend development, however, it creates an image for front-end developer to test and implement elements for the website.

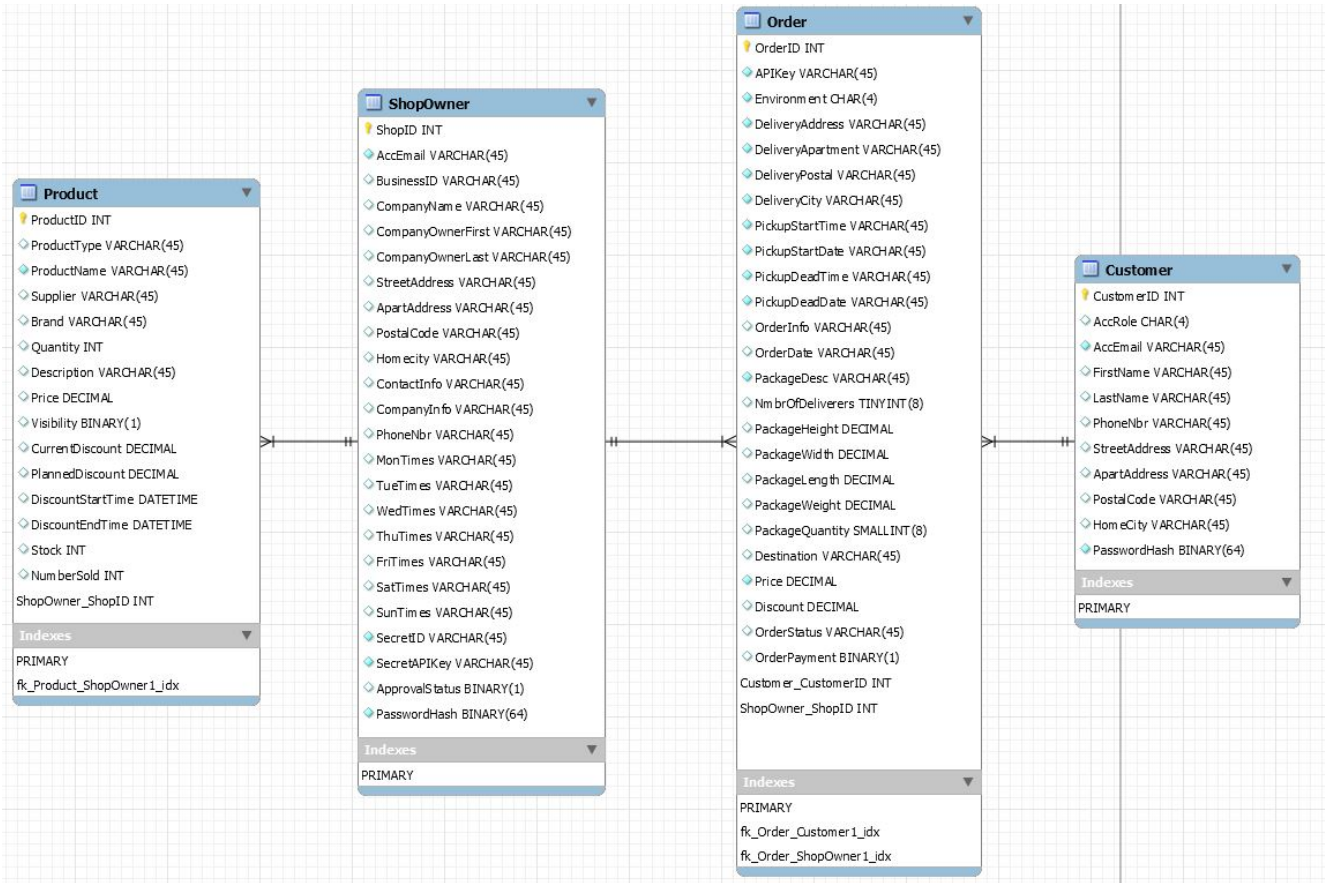


FIGURE 6: Vaste ER Diagram

4.3 Database Tables

This chapter will introduce about the database structure of the website. It will create an image about how the data is managed and linked to each other. The chapter focuses on four main database that are product, shop owner, order, and customer.

Database tables include: “Customer” table which is linked to “Order” table by order ID; “ShopOwner” table which is linked to the “Product” table by product ID; “Order” table which is linked to “Customer”, “ShopOwner” and “Product” by order ID and product ID; “Product” table which is linked to “ShopOwner” by product ID. The detail connection between tables is shown in the figure 6 - ER diagram below.

Vaste Webstore will require multiple databases that work in unison. There will be a customer database, which contains the customer information such as: last name, first name, phone number, email, address, apartment number, postal code, city. Additionally, if the customer has registered to the platform the database also needs email such as the username and password. The Shop owner database contains required information as the figure 8 shows.

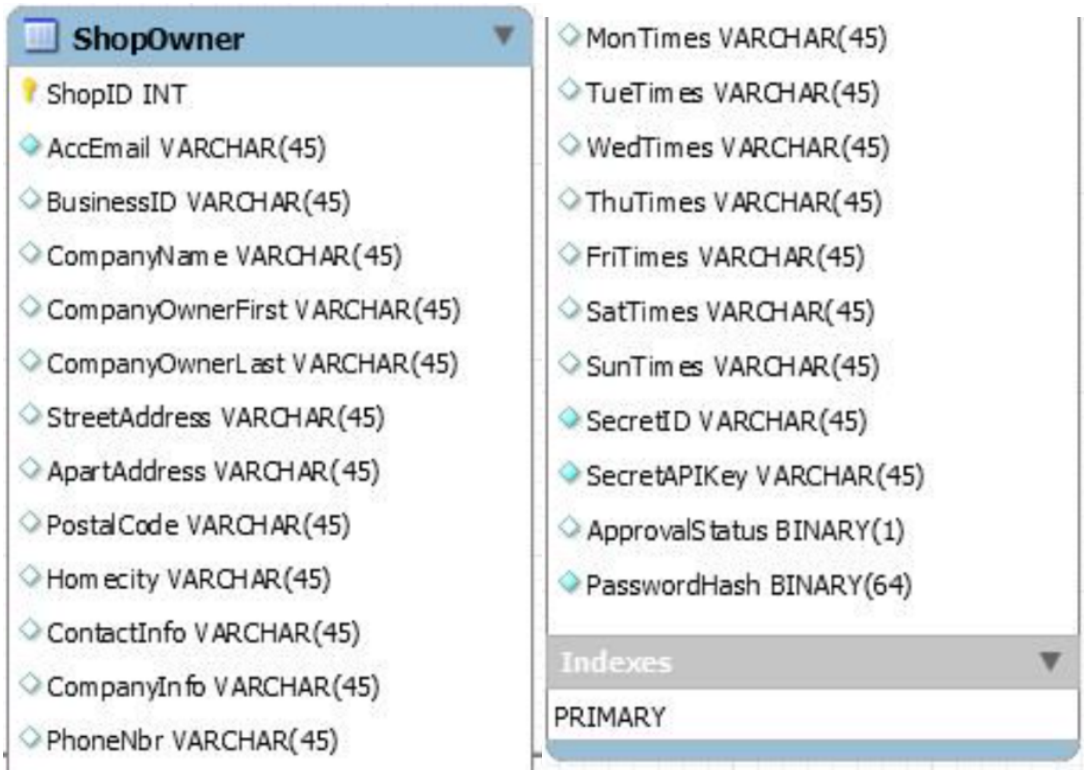


FIGURE 7: Shop Owner Table Database.

Shop ID will be the main key which is unique to verify the Shop Owner, an email account is necessary to verify and recover their account. Business ID must be provided for security purpose. Additionally, basic information of the company such as: company's name, owner's first & last name, address, for instance, street, apartment, postal code, city, phone number, contact, general information and opening hours from Monday to Friday.

Order database must include the following information: secret ID; API key; approval status. The order database must include the information as Figure 5 display: order ID; API key; environment (test or prod); sender's information such as: last name, first name, phone, email, pickup address, pickup apartment, pickup postal code, pickup city; receiver's information such as: last name, first name, phone, email, delivery address, delivery apartment, delivery postal code, delivery city; pick up start date (format yyyy-mm-dd); pickup start time (format hh:mm, mandatory); deadline for pickup date (format yyyy-mm-dd); deadline for pickup time (format hh:mm, mandatory); order information note; package description; count of people in the delivery; package's information such as: height, width, length, weight, quantity; destination contains value: "address", "box", "depot", "driver", the default value is "address"; shipping method; price; discount; order confirm status ("invalid", "confirm"), order payment ("unpaid", "paid").

The transaction history database will contain all the data from the previous purchases on the Vaste webstore as: transaction ID, product ID, company name, order ID, customer ID, customer name, product type, product name, quantity, price, payment amount, date of payment, payment method. finally, product database contains the specific information as: product ID product type, product name, supplier, brand, quantity, description, picture, price, discount, stock (amount or "out of stock", if the amount equals 0), sold quantity.

Customer and shop owner databases will be edited automatically as they create, edit or delete their accounts. Transaction history database will only get new transactions added to the list and will not be edited or deleted. The Product database will have the products and their descriptions added and updated by the shop owners. The Order database includes customers and is edited by admin (in case the order information has been confirmed and the customer wants to change their order information, they need to contact the staffs to make changes.)

The customer and shop owner databases will not be linked directly, but they will be linked through transaction history. Products will be linked directly to shop owners and transaction history, and to customers via transaction history. The order database links directly to other databases as: customer, shop owner, payment, transaction, product.

5 WEBSITE FRONT-END DEVELOPMENT AND IMPLEMENTATION

This chapter is the main part in the thesis where the website front-end development and implementation will be presented. The SQL prototype with databases mockup will display how the data is queried. The UI interface design shows how the website look like. The detail of coding for the HTML, CSS file and implement function by JS are also be included in this chapter.

5.1 SQL prototype

This chapter will create a picture of the SQL for database mockup which includes the EER diagram – the structure of the database and the example code of the mockup SQL of: customer, order, product and shop owner.

5.1.1 MySQL database mockup

The 3rd party payment service: payment service used to collect pay from purchased products. The mockup has been constructed with tables and variables we at the time believed to be necessary for the creation of a functional database for the Vaste webstore. There might still be some missing variables, and some variables might not have the correct data type or size.There are in total 4 primary keys and 3 foreign keys in the database. The primary keys are customerid, orderid, productid, shopid; and the foreign keys are: customerid from customer database is the foreign key for order table, and shopid from shopowner table is foreign key for both order table and product table.

Most VARCHAR(45) variables are meant to have that data type as temporary placeholder, and some variables are missing the proper or minimalistic variable size. The SQL dump contains a couple mock customers, mock companies, products and orders, which were created for testing purposes.

5.1.2 SQL dump

This chapter will present the SQL dump which is the test for data query. The purpose for the SQL dump is to check if the set data will be access correctly according to the primary set up. The example set ting and query request will show how the variables are set inside each table and the connection between each table.

Part 1: Vaste Schema Customer SQL

```
--
-- Table structure for table `customer`
--
CREATE TABLE `customer` (
  `CustomerID` int NOT NULL,
  `AccRole` char(4) NOT NULL DEFAULT 'cust' COMMENT 'Account role, default value is "cust" for customer, but can be changed to "mode" for moderator, by system administrators.',
  `AccEmail` varchar(45) NOT NULL,
  `FirstName` varchar(45) DEFAULT NULL,
  `LastName` varchar(45) DEFAULT NULL,
  `PhoneNmbr` varchar(45) DEFAULT NULL,
  `StreetAddress` varchar(45) DEFAULT NULL,
  `ApartAddress` varchar(45) DEFAULT NULL,
  `PostalCode` varchar(45) DEFAULT NULL,
  `HomeCity` varchar(45) DEFAULT NULL,
  `PasswordHash` binary(64) NOT NULL,
  PRIMARY KEY (`CustomerID`),
  UNIQUE KEY `CustomerID_UNIQUE` (`CustomerID`),
  UNIQUE KEY `AccEmail_UNIQUE` (`AccEmail`)
```

Part 2: Vaste Schema Order SQL

```
--
-- Table structure for table `order`
--

CREATE TABLE `order` (
  `OrderID` int NOT NULL AUTO_INCREMENT,
  `CustomerID` int NOT NULL,
  `ShopID` int NOT NULL,
  `APIKey` varchar(45) NOT NULL,
  `DeliveryAddress` varchar(45) NOT NULL,
  `DeliveryApartment` varchar(45) NOT NULL,
  `DeliveryPostal` varchar(45) NOT NULL,
  `DeliveryCity` varchar(45) NOT NULL,
  `PickupStartTime` varchar(45) NOT NULL,
  `PickupStartDate` varchar(45) NOT NULL,
  `PickupEndTime` varchar(45) NOT NULL,
  `PickupEndDate` varchar(45) NOT NULL,
  `OrderInfo` varchar(45) DEFAULT NULL,
  `OrderDate` varchar(45) DEFAULT NULL,
  `PackageDesc` varchar(45) NOT NULL,
  `NmbrOfDeliverers` varchar(45) NOT NULL DEFAULT '1',
  `PackageHeight` decimal(10,2) unsigned DEFAULT NULL,
  `PackageWidth` decimal(10,2) unsigned DEFAULT NULL,
  `PackageLength` decimal(10,2) unsigned DEFAULT NULL,
  `PackageWeight` decimal(10,3) unsigned DEFAULT NULL,
  `PackageQuantity` int unsigned DEFAULT NULL,
  `Destination` varchar(45) DEFAULT NULL,
  `Price` decimal(10,2) unsigned NOT NULL,
  `Discount` decimal(10,5) NOT NULL DEFAULT '1.00000',
  `OrderStatus` varchar(45) NOT NULL,
  `OrderPayment` binary(1) NOT NULL DEFAULT '0',
  PRIMARY KEY (`OrderID`),
  UNIQUE KEY `OrderID_UNIQUE` (`OrderID`),
  UNIQUE KEY `APIKey_UNIQUE` (`APIKey`),
  KEY `CustomerID_idx` (`CustomerID`),
  KEY `ShopID_idx` (`ShopID`),
  CONSTRAINT `CustomerIDOrder` FOREIGN KEY (`CustomerID`) REFERENCES `customer`
(`CustomerID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `ShopIDOrder` FOREIGN KEY (`ShopID`) REFERENCES `shopowner` (`ShopID`)
ON DELETE CASCADE ON UPDATE CASCADE
```

Part 3: Vaste Schema Product SQL

```
--
-- Table structure for table `product`
--
CREATE TABLE `product` (
  `ProductID` int NOT NULL AUTO_INCREMENT,
  `ShopID` int NOT NULL,
  `ProductType` varchar(45) DEFAULT NULL,
  `ProductName` varchar(45) DEFAULT NULL,
  `Supplier` varchar(45) DEFAULT NULL,
  `Brand` varchar(45) DEFAULT NULL,
  `Description` varchar(45) DEFAULT NULL,
  `Price` decimal(10,2) DEFAULT NULL,
  `Visibility` binary(1) NOT NULL DEFAULT '1' COMMENT 'Whether the product can be seen in the shop or not, 0 stands for invisible, 1 stands for visible. By default 1 is selected, but it can be toggled on product creation.',
  `CurrentDiscount` decimal(10,5) DEFAULT '1.00000',
  `PlannedDiscount` decimal(10,5) DEFAULT '1.00000',
  `DiscountStartTime` datetime DEFAULT NULL,
  `DiscountEndTime` datetime DEFAULT NULL,
  `Stock` int unsigned DEFAULT '0',
  `NumberSold` int unsigned DEFAULT '0',
  PRIMARY KEY (`ProductID`),
  UNIQUE KEY `ProductID_UNIQUE` (`ProductID`),
  UNIQUE KEY `ShopID_UNIQUE` (`ShopID`),
  KEY `ShopID_idx` (`ShopID`),
  CONSTRAINT `ShopIDProduct` FOREIGN KEY (`ShopID`) REFERENCES `shopowner` (`ShopID`) ON DELETE CASCADE ON UPDATE CASCADE
```

Part 4: Vaste Schema ShopOwner SQL

```
--
-- Table structure for table `shopowner`
--
CREATE TABLE `shopowner` (
  `ShopID` int NOT NULL,
  `AccEmail` varchar(45) NOT NULL,
  `BusinessID` varchar(45) DEFAULT NULL,
  `CompanyName` varchar(45) DEFAULT NULL,
  `CompanyOwnerFirst` varchar(45) DEFAULT NULL,
  `CompanyOwnerLast` varchar(45) DEFAULT NULL,
  `StreetAddress` varchar(45) DEFAULT NULL,
  `ApartAddress` varchar(45) DEFAULT NULL,
  `PostalCode` varchar(45) DEFAULT NULL,
  `HomeCity` varchar(45) DEFAULT NULL,
  `ContactInfo` varchar(45) DEFAULT NULL,
  `CompanyInfo` varchar(45) DEFAULT NULL,
  `PhoneNmbr` varchar(45) DEFAULT NULL,
  `MonTimes` varchar(45) DEFAULT NULL,
  `TueTimes` varchar(45) DEFAULT NULL,
  `WedTimes` varchar(45) DEFAULT NULL,
  `ThuTimes` varchar(45) DEFAULT NULL,
  `FriTimes` varchar(45) DEFAULT NULL,
  `SatTimes` varchar(45) DEFAULT NULL,
  `SunTimes` varchar(45) DEFAULT NULL,
  `SecretID` varchar(45) NOT NULL,
  `SecretAPIKey` varchar(45) NOT NULL,
  `ApprovalStatus` binary(1) NOT NULL DEFAULT '0' COMMENT 'Checks whether the shop has been
approved or not, 1 stands for approved, 0 stands for not approved.',
  `PasswordHash` binary(64) NOT NULL,
  PRIMARY KEY (`ShopID`),
  UNIQUE KEY `ShopID_UNIQUE` (`ShopID`),
  UNIQUE KEY `AccEmail_UNIQUE` (`AccEmail`),
  UNIQUE KEY `SecretID_UNIQUE` (`SecretID`),
  UNIQUE KEY `SecretAPIKey_UNIQUE` (`SecretAPIKey`)
)
```

5.2 User Interface Design

The user interface contains all the components of the website from the beginning to the end of a process when the user accesses the website. In other words, the elements will be implemented in a coherent set of functionalities according to the action that customers take when using the website. The Vaste Web-store uses a component-based approach, with the first step as the decomposition of the website into several components.

The first page when they access the link would be the Create Account Options page – which allows the user to create an account as a company or customer, the user also visits the home website without creating an account by clicking the #Vastepi Logo.

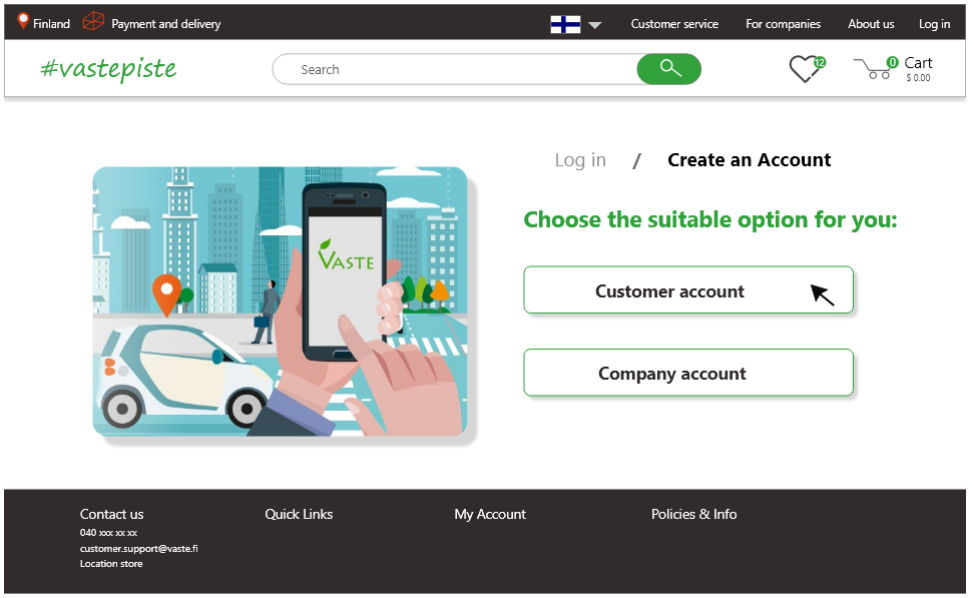


FIGURE 8: Create an Account Page.

There are two options: Customer and Company accounts which lead to two other sets of interfaces. The customer account will be created for the normal customer where their basic information shall be provided. The Company's account will only be selected for the store or company who wants to sell their product on the website. According to the requirement, company information must be verified before they can post their product and information on the platform. Details about the design of front-end interfaces of Company and Customer accounts will be shown in chapters 5.2.1 and 5.2.2.

5.2.1 Company Account Actions

This chapter introduces about the company account action. In other words, it shows how to create a company account steps by steps and edit company account. That actions support the shop owner to create their account to sell their products and communicate with their customers.

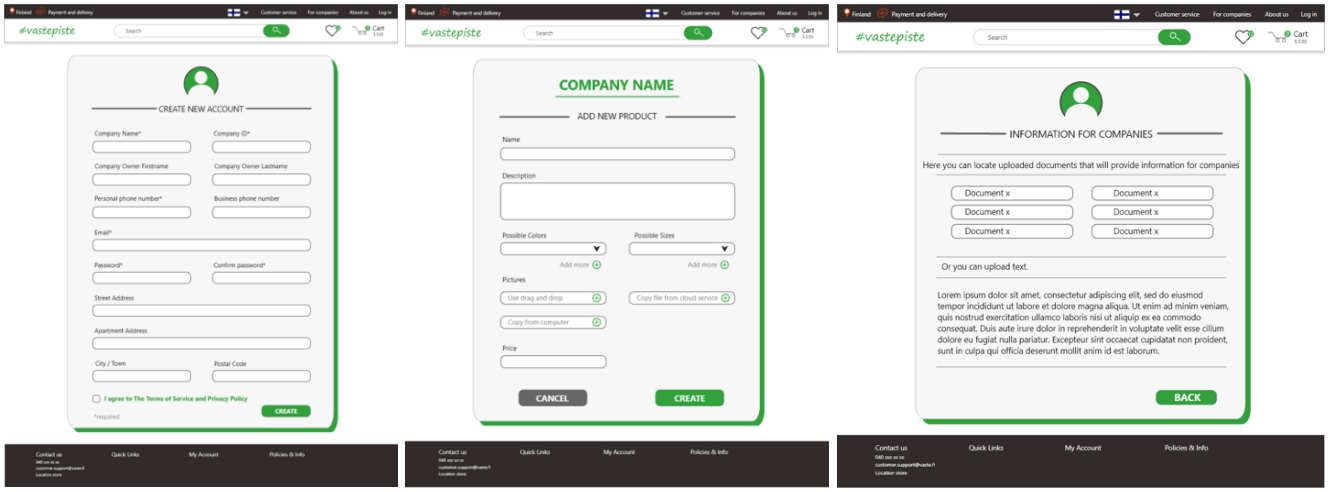


FIGURE 9: Create Company Account

As in the figure 10, in the beginning, the company must provide detailed information in a form, then click “Create” to create an account. As mention before, the company account must provide the authenticity of their information like name, business id, address, phone number, email, and other documents related to the company. By filling in information and attaching the required document, the registration information will be automatically sent to the administrator, a notification email will be sent to the company to the registered email. After checking the authenticity then the company account must be created and notified by email. This process aims to protect the customer and the company's information.

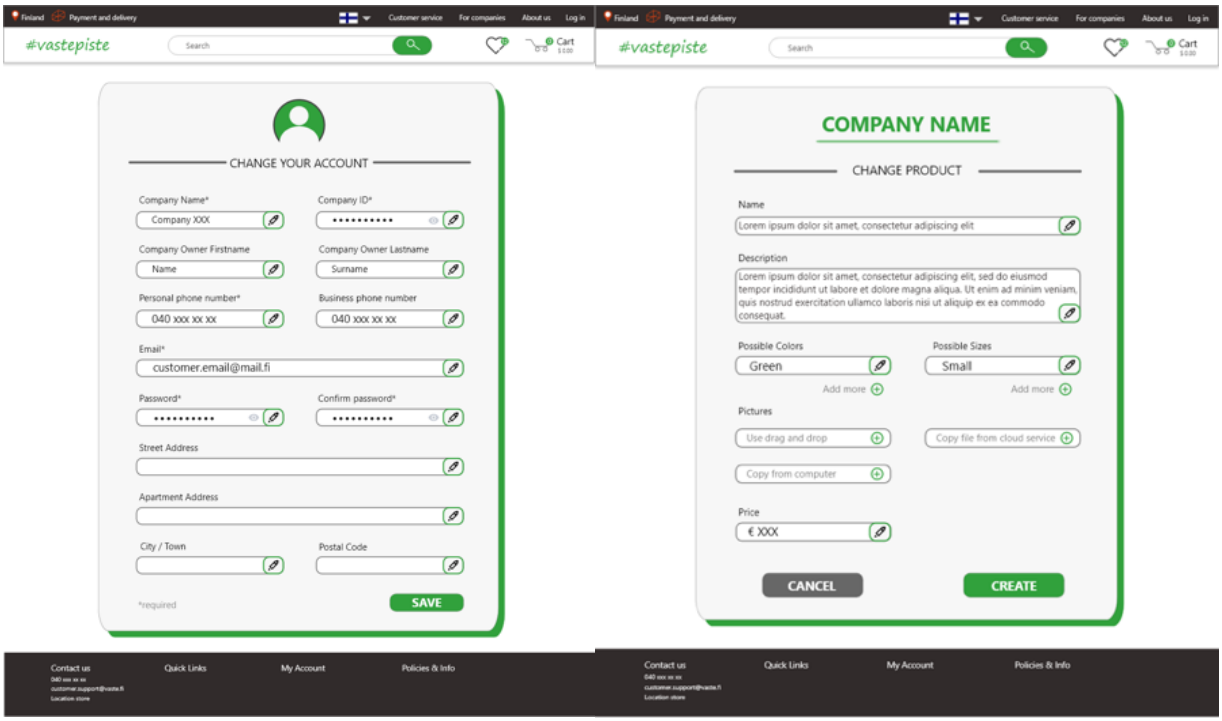


FIGURE 10: Change Company Information

A company is allowed to change their information after having created an account like the figure 10 shows, with this function, a company can update their profile and product in a convenient way. The selected information can be changed by click on the pen icon, then the shop owner can type the new information to the text box and press save button to save the change.

5.2.2 Customer Account Actions

This chapter introduces about the customer account action. In other words, it shows how to create a customer account steps by steps and edit the account. That actions support the customers to create their account to save their information and access to other functions of the website.

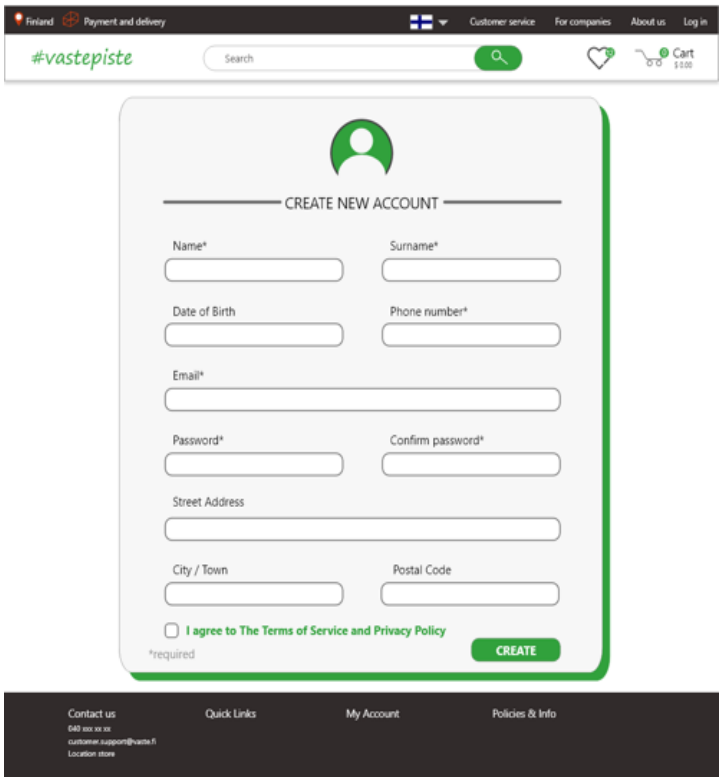


FIGURE 11: Create Customer Account

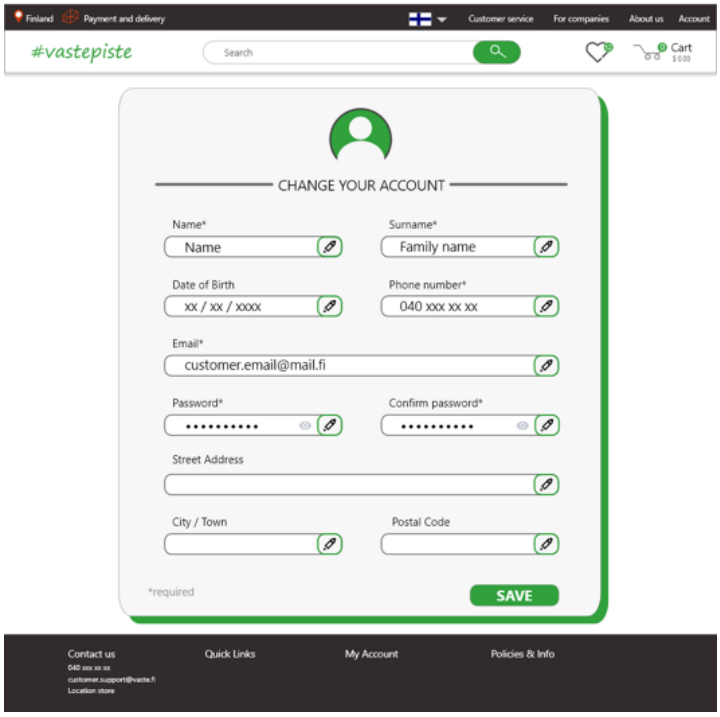


FIGURE 12: Change Customer Information

A customer account is much easier to create than a company account, customers only need to fill in their information and after creating the account an auto-email will be sent to their registered email to verify their account. By clicking the email, the customer can login into their account and use other functions of the platform. The figure 12 and 13 show how to create customer account and change information respectively.

5.2.3 General Users Interfaces

This chapter introduces about the general user interface which designed how the website look like and which position the elements are placed. User interface, which includes text-based UI and graphical UI, plays a main role in user interaction.

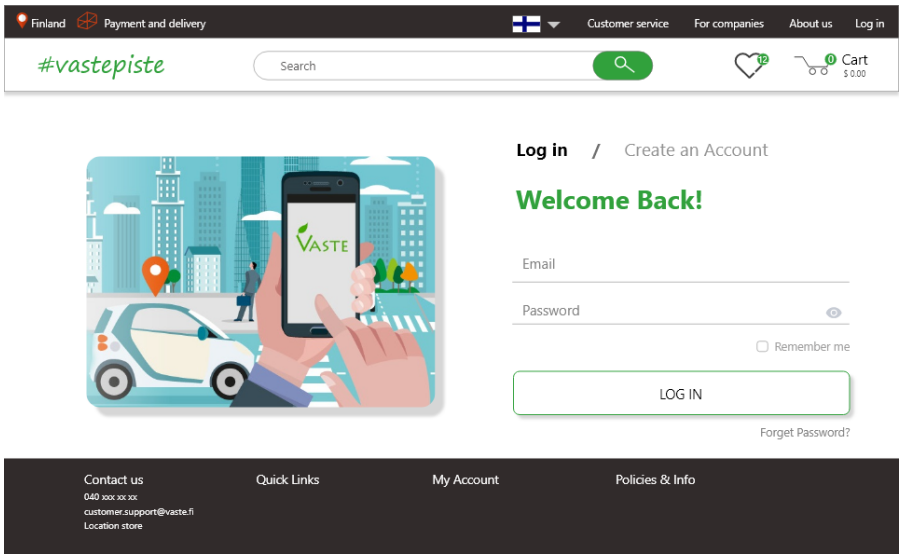


FIGURE 13: Login.

The main page of the platform will display the logo, picture, and other functions as: search, view interest, cart, log in and create an account. By choosing any function it will lead to the responded page. On the top of the page – header - will display the basic functions: location, payment, and delivery information link, language options, customer services, for company information, about us, log in, Vaste logo (lead to the main market page), search, favorite items, cart. At the bottom of the page will be the rooter which contains the contact information, quick links, my account management, policies & info. Both the header and rooter will be displayed on all pages of the design.

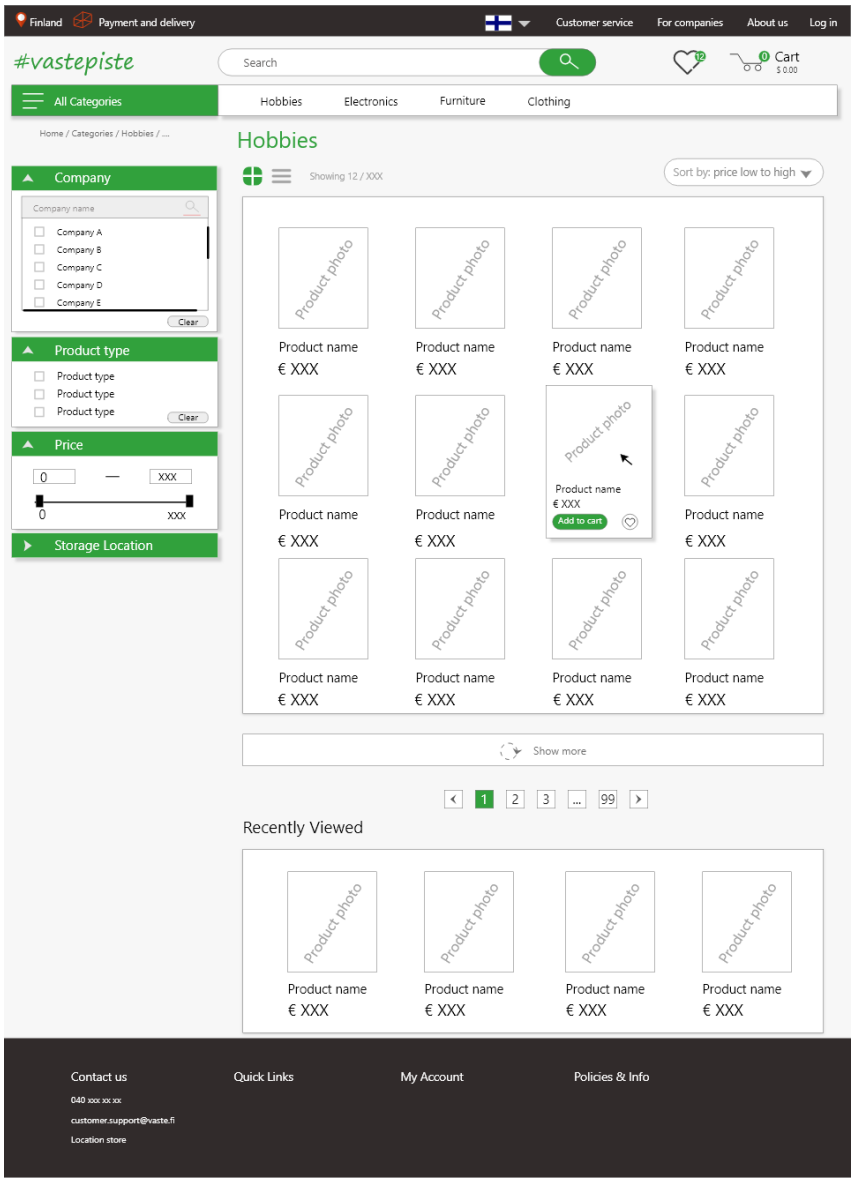


FIGURE 14: Market Place.

After Log In to the account, the platform will lead the user to the market main page where all products of companies are listed. There are tabs which display the group information of a corresponded group of the product such as hobbies, electronics, furniture, clothing. The user can see the picture, name, price of items, products, also they can sort by company name, product type, and price. When a user clicks any product, the product page will be shown in the figure 16. All the details such as name, size, color, price, delivery, payment method, description are presented in an attractive way.

FIGURE 15: Product Page.

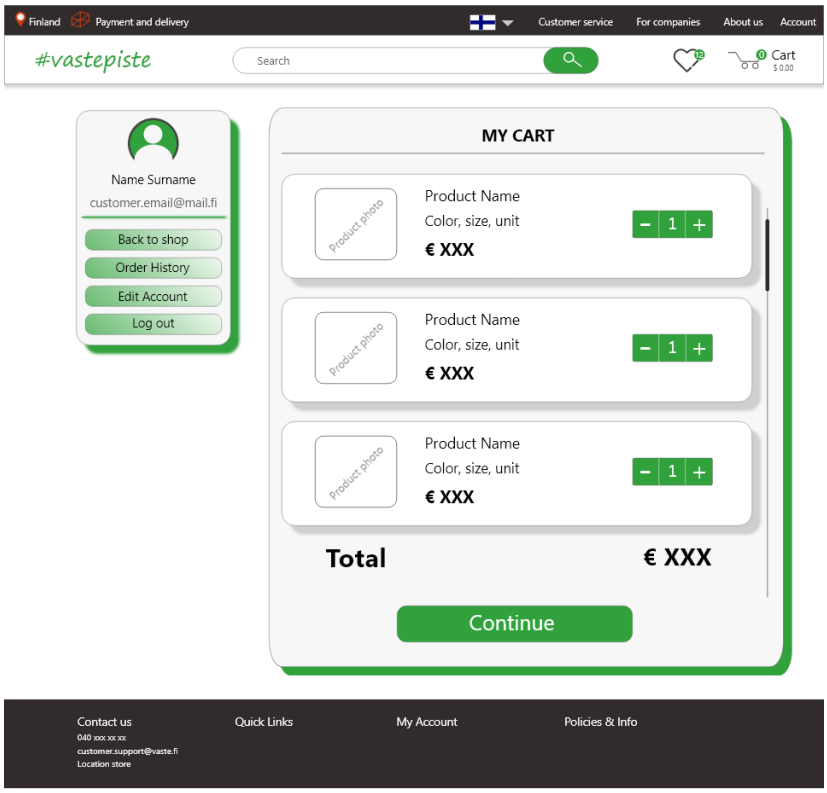


FIGURE 16: My Cart

The user can check their cart and edit their cart whenever they want. There are also some functions like: back to shop, order history, edit account, log out or continue to payment are support in this page design.

In “Order history” function, there are two options that the user can view the “Active” or “Complete” history. In “Active” the user can see the order that they have paid for and which is on the way to delivery. In “Complete” mode they can see the order that has arrived in the past.

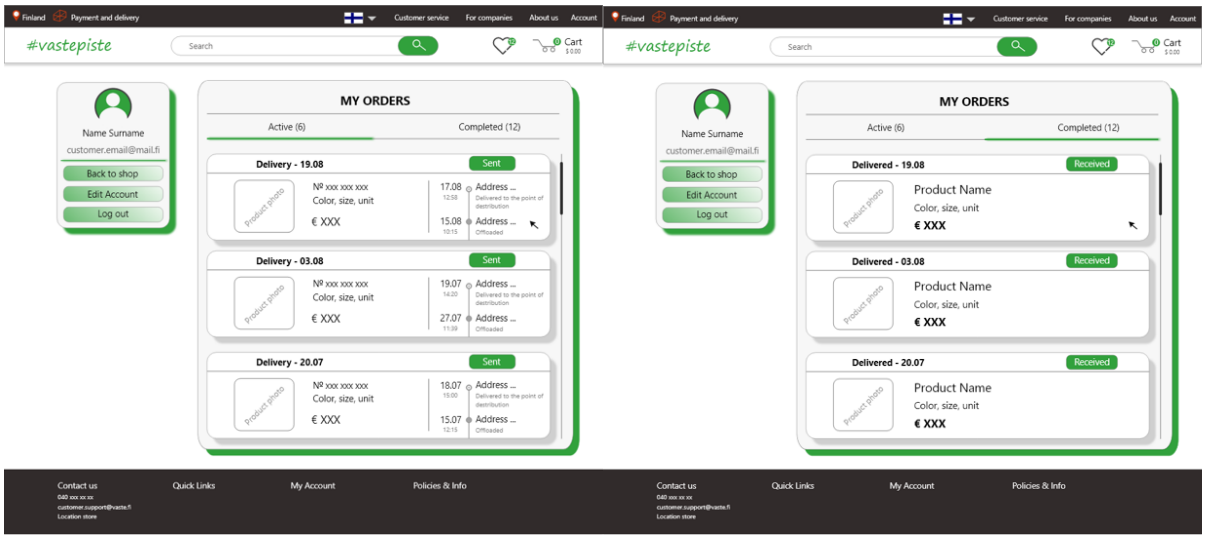


FIGURE 17: Order History Options

The order process will be designed based on the http://www.vastepiste.fi/ website - all the orders will be sent to Vaste Delivery Platform by API key. Therefore, there will be five steps when the user comes to the order section. According to the requirement, this part was designed in Finnish language.

Step 1: The user selects the delivery menu, chooses the size of the package and fills in the destination information.

The delivery menu includes “Ovelta ovelle-kuljetuspalvelu” which means door to door shuttle service; “Ovelta Vastepisteeseen” which mean from the door to the point of response; “Vastepisteestä ovelle” which means from the response point to the door; “Vastepiste-kuljetuspalvelu” which means counter-point shuttle service; and “Vastepiste säilytyspalvelu” which means counterpoint storage service.

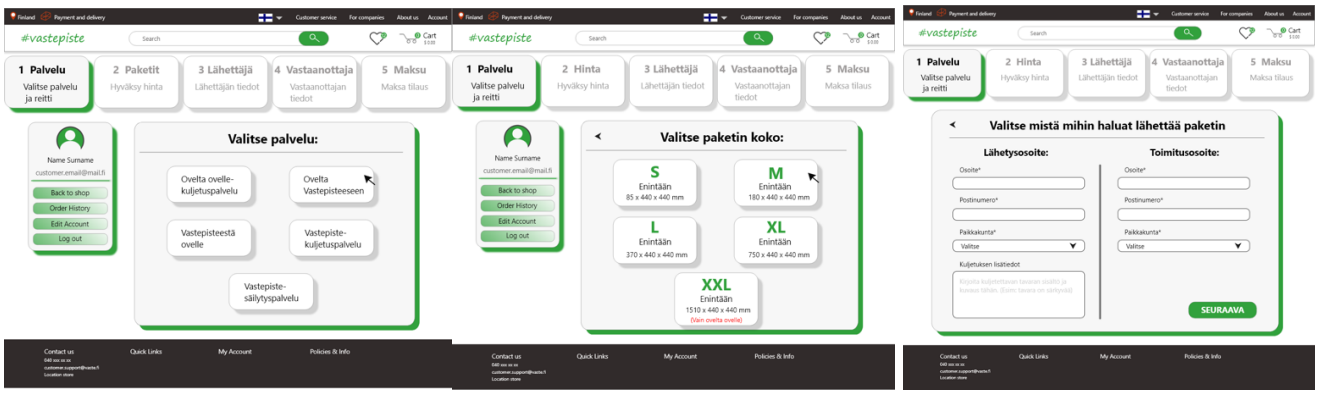


FIGURE 18: Delivery Menu - Package Size – Destination Information

This step includes three small steps, first, the user chooses the suitable delivery service they want based on the menu, then chooses the size of the package and finally fills in the shipping and delivery address. In the second step, the user chooses package information such as pickup and delivery estimate time.

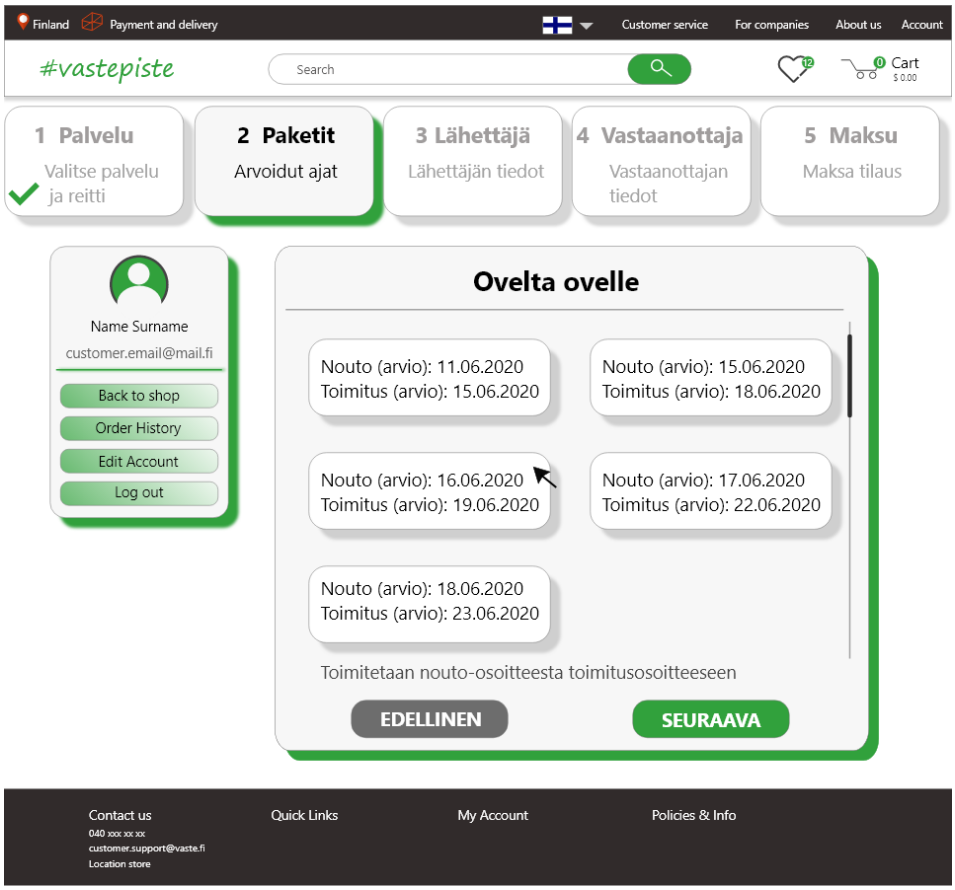


FIGURE 19: Delivery and Pickup Estimate Time

In this step, the system will display the estimated time of pickup and delivery and the user can choose the time suitable for them to send and receive the package. Then the user chooses “Seuraava”- to continue the process to step 3 and 4. In the third step, the user shall fill in the sender information according to the form in figure 21. In the next step, the user fills in the recipient information with the detail is shown in the figure 22. The last step is show in the figure 23, the payment information will be processed.

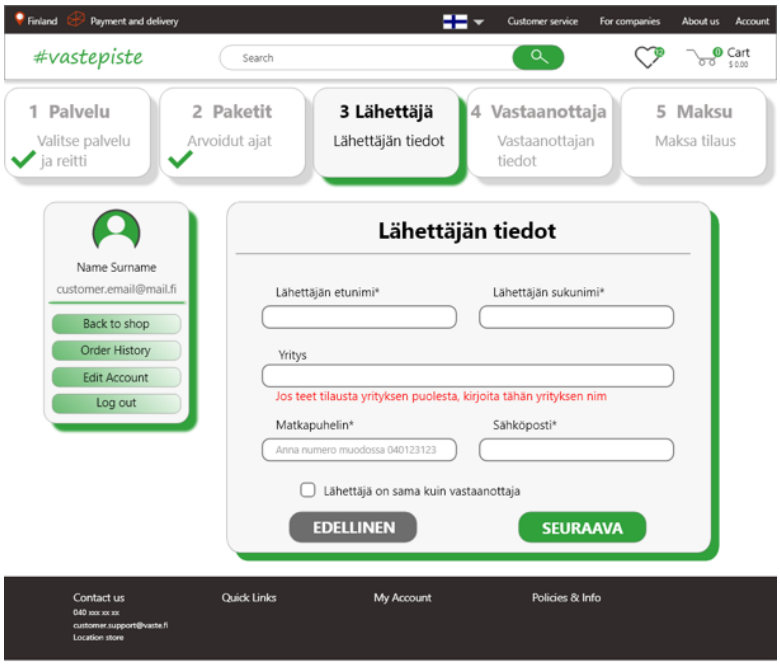


FIGURE 20: Fill Sender Information

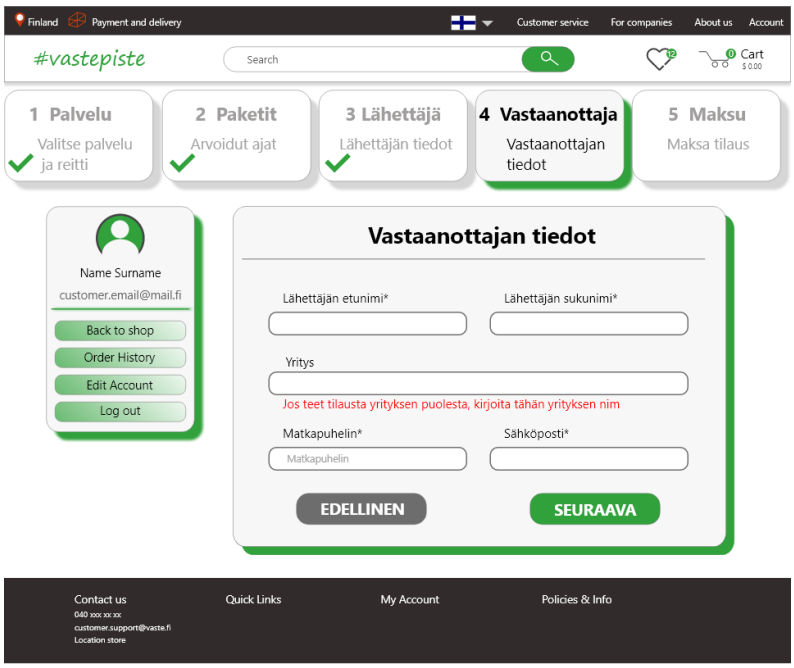


FIGURE 21: Fill Recipient Information

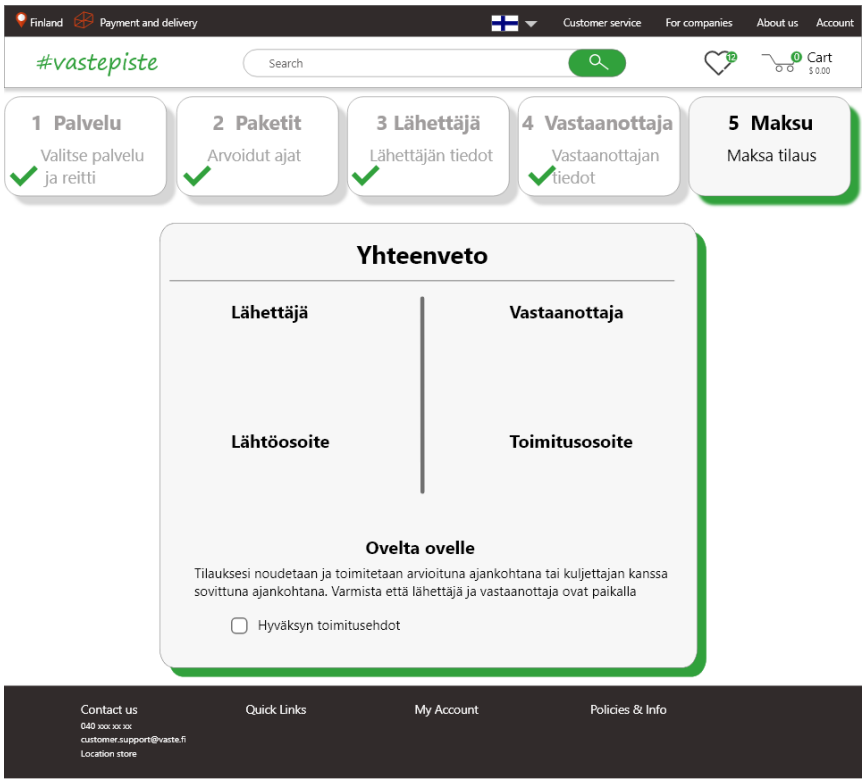


FIGURE 22: Payment

The final step is payment. In this step, the summary of the information that has been filled in the previous steps as: Sender, Recipient, Sending Address, Recipient Address will be displayed to help the user to check the information again before paying. Then the payment will be processed by clicking “Ovelta Ovelle”.

Above are some of the main UI of the platform, there are other designs of UI on the main project which is not included in this thesis. Only the most important part is selected.

5.3 Front-end Implementation

In this part, HTML, CSS and JS will be used to implement the important elements to build the front-end for the Vaste Platform. There will be some main elements that are used in the implementation, for example, header, rooter, tabs, picture, flask card, board, placeholder, icons. Most of the implementation will be applied by Bootstrap.

“Developers are getting excited, but they can be hard to build and hard to do well.” (Kinlan. P, 2018). According to Kinlan, building a website can be excited but making it run well is very hard. Therefore, a web developer must have the basic and advance knowledge and skills for professional work.

HTML (HyberText Markup Language) is the most basic building block of the Web. It describes the definition of web content and its structure. In addition to HTML, other technologies are commonly used to define the appearance/presentation (CSS) or functionality/behavior of a webpage (JavaScript). Cascading Style Sheets (CSS) is the language used to define the display of a document written in HTML or XML format (including XML dialects such as SVG, MathML or XHTML). CSS specifies how the elements on the screen, on paper, invoice, or in other media should be made. JavaScript (JS) with first-class features is a lightweight, read, or just-in-time compiled programming language (Developer Mozilla).

Bootstrap is the world’s most popular framework for building responsive, mobile-first sites, with a template starter page – GetBootstrap. Bootstrap is also known as a free open source that provides the template for most element implement using CSS and JS function. These useful tools help to save time to implement and make work easier, however, user knowledge of HTML, CSS, and JS are generally required to fully understand and apply Bootstrap to web development.

By using Bootstrap, developers can build complex website layout quickly, design layout over a 12-column grid system, use prewritten JavaScript for dynamic site features, create device-specific layouts, create layouts that work on a variety of web browsers (Codecademy, 2020).

> Understanding HTML and CSS can help anyone who works with the web; designers can create more attractive and usable sites, website editors can create better content, marketers can communicate with their audience more effectively, and managers can commission better sites and get the best out of their teams. (HTML&CSS Design and Build Websites, 2011).

The Vaste Front-end was built with 16 HTML files, 1 CSS to define the style, and main.js to test the action based on JavaScript implementation. The figure below shows the list of files was built for the project.

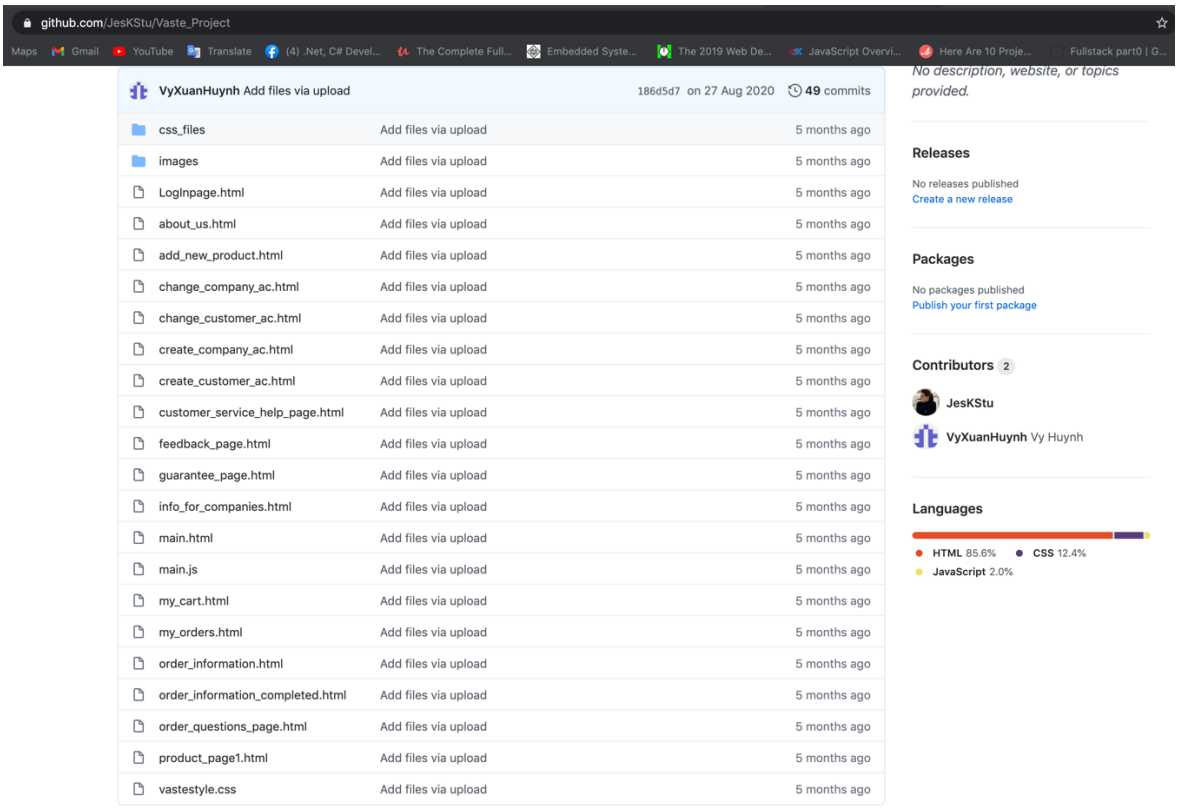


FIGURE 23: Vaste Front-end Files

5.3.1 HTML file Header

The HTML header in each file has the same contents and order as defined by the HTML file and language.

```
<!DOCTYPE html>
<html lang="en">
<html>
```

In the Header, the title of the web shop “Vaster Web-shop” will be set with the link to the Vaste CSS file to set the display and other CSS link of the stylesheet. The link to CSS files will be put inside the tag <link> as the code in the next page shows.

```
<head>
    <title>Vaste Webshop</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <style>
      table, th, td {
        border: 1px solid black;
        border-collapse: collapse;
        width: 50%;
        border-radius:10px
      }
      th, td {
        padding: 5px;
        text-align: left;
      }
    </style>
    <!--Vaste CSS-->
    <link rel="stylesheet" type="text/css" href="css_files/vastestyle.css">
    <link rel="stylesheet" type="text/css" href="css_files/teststyle.css">

    <!--CSS Links-->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" integrity=
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
```

5.3.2 HTML body

The body of the HTML file contains all elements that the website will display and the function takes the response to the user actions. It plays the main role in web content building. This section will show how to implement the header, footer, form, pictures, links, icons, and other important elements of the Vaste Platform.

Part 1: Implement Header on the body in HTML file.

FIGURE 24: Top Header in Body of HTML File

To implement this part, a container is used to hold a navigation bar. The navigation collapse put the #vastepites, search section, favourite and card on the one row. The top of the design will display the location and payment and delivery on the left column. Other element as language – using drop list, customer service, for Company, About Us, Login are place on the right column.

“When creating a web page, you add tags (known as markup) to the contents of the page. These tags provide extra meaning and allow browsers to show users the appropriate structure for the page.” (Duckket. 2011).

The division , navigation <nav>, button, span, form, input, icon, list <l>, un order list <ul>, anchor<a>, image <img> were used to implement the correct order of the top Header in this body section. Details of the setting order will be shown in the code below.

CODE 1: Setting order

```
<body>
    <!--Top Header-->
    <div class="container-fluid">
        <nav class="navbar navbar-expand-lg navbar-light fixed-top">
            <!--Nav Button-->
            <button class="navbar-toggler ml-auto" type="button" data-toggle="collapse" data-target="#navbarToggler
                        <span class="navbar-toggler-icon my-toggler"></span>
                    </button>
            <div class="collapse navbar-collapse" id="navbarTogglerDemo02">
                <a class="navbar-brand mr-auto" href="#"><span class="vastelogo"><i>#vastepiste</i></span></a>
                <a class="nav-item mr-auto" href="#">
                    <form class="form-inline"> <!--Search Section-->
                        <input class="form-control mr-sm-1" type="search" placeholder="Search" aria-label="Search">
                        <button class="btn btn-outline-success my-2 my-sm-0" type="submit" style="border-radius: 15
                    </form>
                </a>
                <ul class="navbar-nav mt-2 mt-lg-0">
                    <li class="nav-item"> <!--Heart Icon for favourite product-->
                        <a class="nav-link" href="#" style="margin-top: -7px">
                            <span class="heart_cart"><a href="#"><i class="fa fa-heart-o" style="font-size: 30px; m
                        </a>
                    </li>
                    <li class="nav-item"> <!--cart icon-->
                        <a class="nav-link" href="#" style="margin-top: -7px">
                            <span class="heart_cart"><a href="#"><i class="fa fa-cart-arrow-down" style="font-size:
                        </a>
                    </li>
```

Part 2: Implement Footer on the body in HTML file.

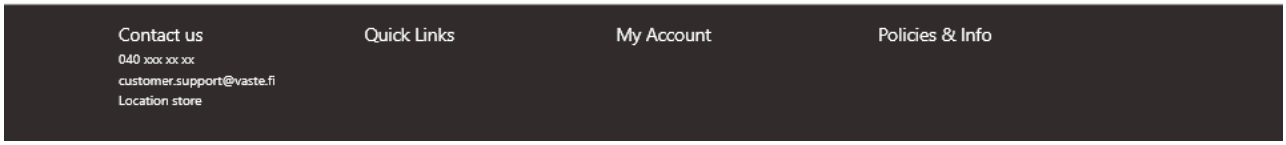


FIGURE 25: Footer of HTML File

The Footer will contain the Platform information as Contact information, Quick Links, My Account, Policies& Info are managed in 4 columns with the <footer> tag and class column according to the suitable position.

CODE 2: Footer coding

```
    <!-- Footer -->
<footer class="page-footer font-small pt-4" style="margin-top: 60px; background-color: #312B2B;
    <div class="container-fluid text-center text-md-left">
        <div class="row">
            <div class="col-md-5 mt-md-0 mt-3">
                <!-- Footer Content -->
                <h6 class="text-uppercase">Footer Content</h6>
                <p style="font-size: 12px">Lorem ipsum dolor sit amet, consectetur adipiscing el
                    Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore
            </div>
            <div class="col-md-1"></div>
            <hr class="clearfix w-100 d-md-none pb-3">
            <div class="col-md-3 mb-md-0 mb-3 col-sm-6">
                <h6 class="text-uppercase">Contact Us</h6>
                <ul class="list-unstyled">
                     <!-- Contact List -->
                    <li><a href="#!">Link 1</a></li>
                    <li><a href="#!">Link 2</a></li>
                    <li><a href="#!">Link 3</a></li>
                    <li><a href="#!">Link 4</a></li>
                </ul>
            </div>
            <div class="col-md-3 mb-md-0 mb-3 col-sm-6">
                <h6 class="text-uppercase">Policies & Info</h6>
                <ul class="list-unstyled">
                     <!-- Policies & Info List -->
                    <li><a href="#!">Link 1</a></li>
                    <li><a href="#!">Link 2</a></li>
                    <li><a href="#!">Link 3</a></li>
                    <li><a href="#!">Link 4</a></li>
                </ul>
            </div>
        </div>
    </div>
</footer>
```

Part 3: Implement Forms.

For example, to implement the LogIn form as the figure below, the code in Html file would be:

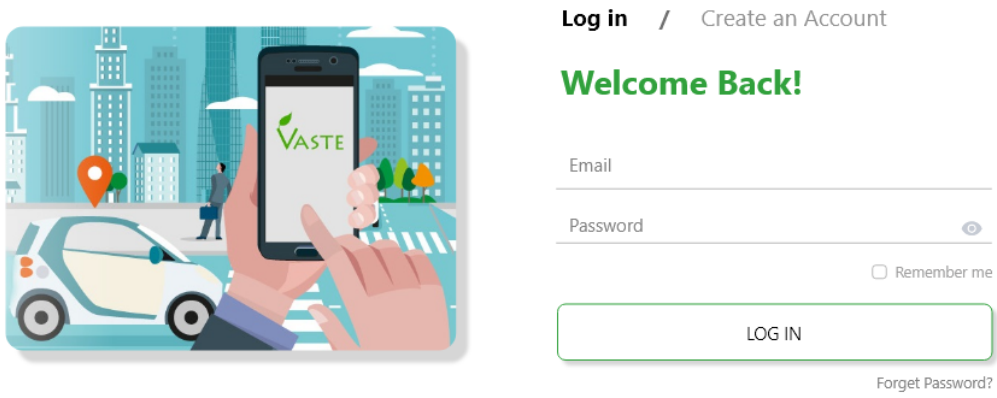


FIGURE 26: LogIn Form

CODE 3: LogIn Form code

```
<!--Main Part - Log In Form  -->
<div class="col-md-7 col-sm-12" style="margin-top: 70px">
    <div class="col-sm-12 col-md-12 text-center">
        <input type="button" id="login_button" class="hide_show" value="Log In " onclick="show_login()" style="c
        <span class="hide_show">/</span>
        <input type="button" id="create_button" class="hide_show" value="Create an Account" onclick="show_create
    </div>
    <div class="col-md-12 col-sm-12">
        <div id="login_page">
            <p style="text-align: center; font: Bold 47px/55px Segoe UI; color: #31A13C; margin-top: 2%">Welcome
            <form>
                <div class="form-group">
                    <div class="input-box">
                        <i class="fa fa-envelope"></i>
                        <input type="email" placeholder="Email">
                    </div>
                </div>
                <div class="form-group">
                    <div class="input-box">
                        <i class="fa fa-key"></i>
                        <input type="password" placeholder="Password" id="myInput">
                        <span class="eye" onclick="eye()">
                                                                <i id="hide1" class="fa fa-eye"></i>
                                                                <i id="hide2" class="fa fa-eye-slash"></i>
                                                            </span>
                    </div>
                </div>
                <div class="form-group form-check" style="position: absolute; left: 11%;">
                    <div>
                        <input type="checkbox" class="form-check-input" id="exampleCheck1">
                        <label class="form-check-label" for="exampleCheck1">Remember me</label>
                    </div>
                </div>
                <button type="submit" class="login-btn" style="margin-top: 70px">LOG IN</button>
            </form>
        </div>
    </div>
```

Also, the form for customer register or company register have the same structure; therefore, the code can be used for both register forms. Developers only need to change the content according to the information of customers and the company. The figure and the code of the Customer registration form will be as follows below in figure 28.

FIGURE 27: Customer Register Form

CODE 4: Register Form code

```
<!--Registration Form-->
<div class="row">
    <div class="col-md-6 offset-md-3 col-sm-12 main_block">
        <div class="col-md-12 col-sm-12 text-center">
            <span class=" user_icon"><i class="fa fa-user-circle"></i></span>
            <div class="separator" style="font-size: 24px">CREATE NEW ACCOUNT</div>
            <form style="margin-top: 15px">
                <div class="form-row text-left">
                    <div class="form-group col-md-6">
                        <label for="name">Name:*</label>
                        <input type="text" class="form-control" id="name" required>
                    </div>
                    <div class="form-group col-md-6">
                        <label for="surname">Surname:*</label>
                        <input type="text" class="form-control" id="surname" required>
                    </div>
                    <div class="form-group col-md-6">
                        <label for="dob">Date of Birth:</label>
                        <input type="text" class="form-control" id="dob">
                    </div>
                    <div class="form-group col-md-6">
                        <label for="phonenum">Phone Number:*</label>
                        <input type="text" class="form-control" id="phonenum" required>
                    </div>
                </div>
                <div class="form-group text-left">
                    <label for="email">Email:*</label>
                    <input type="email" class="form-control" id="email" required>
                </div>
                <div class="form-row text-left">
                    <div class="form-group col-md-6">
                        <label for="password">Password:*</label>
                        <input type="text" class="form-control" id="password" required>
                    </div>
                    <div class="form-group col-md-6">
                        <label for="conf_password">Confirm Password:*</label>
                        <input type="text" class="form-control" id="conf_password" required>
                    </div>
                </div>
                <div class="form-group text-left">
                    <label for="address">Street Address:</label>
                    <input type="text" class="form-control" id="address">
                </div>
```

Part 4: Implement Tabs.

The platform requires the possibility to have multiple lists of products and sort functions according to Company, Product Type, and Price. The most effective way to do this is using tabs.

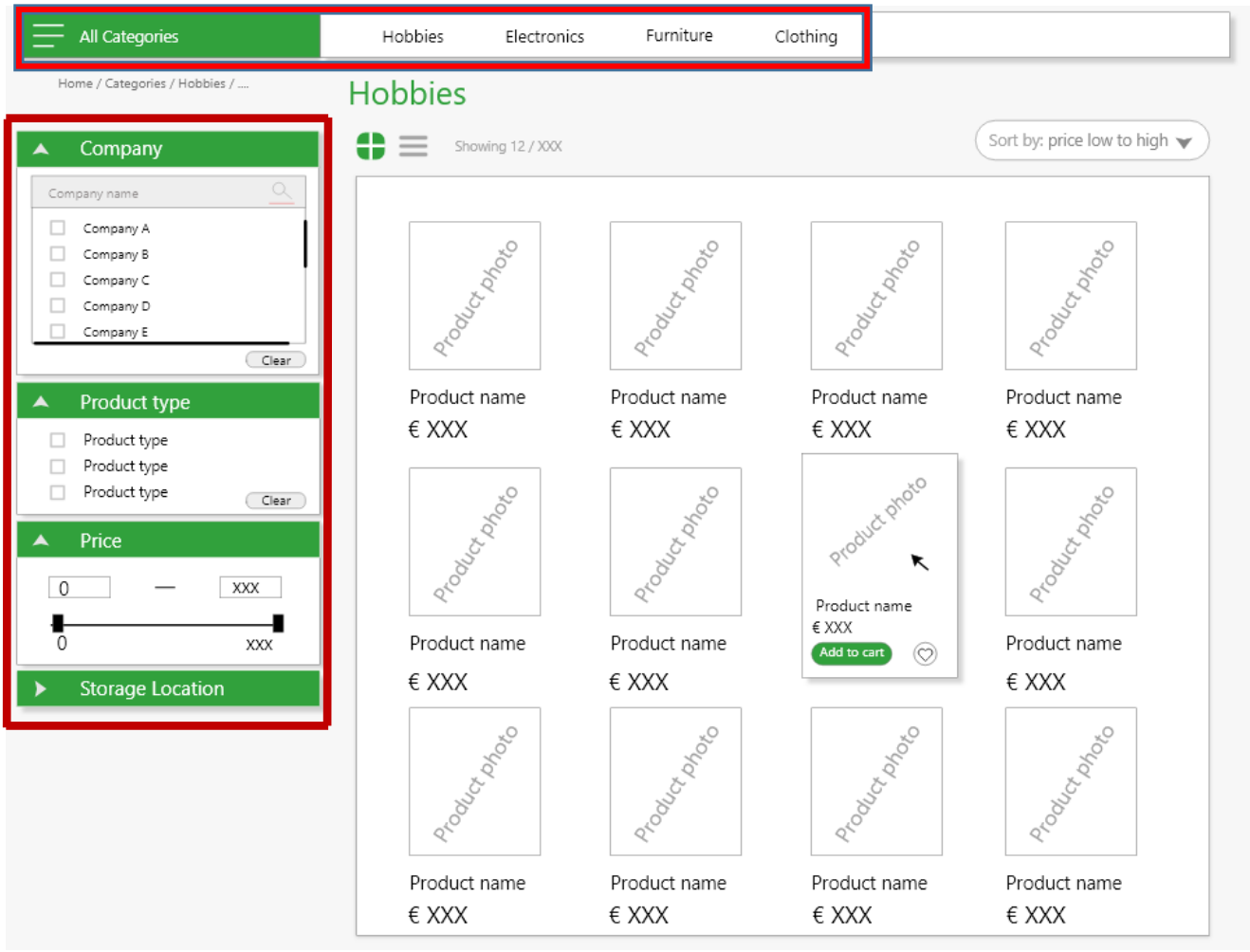


FIGURE 28: Categories Tabs

CODE 5: Categories Code

```
<!--Categoties-->
      <div class="row" style="margin-top: 2%">
          <div class="col-3 all_categories_nav d-none d-md-block" onclick="alert('hi')">
              <h4 style="color:white">All Categories</h4>
          </div>
          <div class="col-9 d-none d-md-block" style="background-color: white; border-radius: 0 8px 8px 0">
              <div class="row">
                  <div class="col-2 offset-1 category_nav"><a href="#">Category #1</a></div>
                  <div class="col-2 category_nav"><a href="#">Category #2</a></div>
                  <div class="col-2 category_nav"><a href="#">Category #3</a></div>
                  <div class="col-2 category_nav"><a href="#">Category #4</a></div>
               </div>
          </div>
      </div><!--class="row" style="margin-top: 2%"-->
          <div class="filter-content collapse" id="category_search"> <!-- Category Filter -->
              <div class="card-body">
                    <div class="box_inside">
                          <form class="search_inside form-inline my-2 my-lg-0">
                                                <input class="form-control mr-sm-2" type="search" placeholder="Search"
                                                      </form>
                          <ul class="list-menu"> <!-- Category List-->
                         <li>
                                    <input type="checkbox">
                                                                  <span class="checkmark">Some Text</span>
                                                              </li>
                                                              <li>
                                                                  <input type="checkbox">
                                                                  <span class="checkmark">Some Text</span>
                                                              </li>
                                                              <li>
                                                                  <input type="checkbox">
                                                                  <span class="checkmark">Some Text</span>
                                                              </li>
                                                              <li>
```

Part 5: Implement Images.

Images can be implemented as card like the code below.

FIGURE 29: Product Images

CODE 6: Image code

```
<div class="row"> <!-- Display product block-->
            <div class="product_block">
                        <div class="column">
                           <div class="card" >..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
                        <div class="column">
                           <div class="card">..</div>
                        </div>
```

Or using

```
<div class="imgContainer">
  <img class="thumbnail active" src="images/ball.jpeg" height="80%" width="80%"

style="margin-top: 10px; margin-left:5px; " href="images/ball.jpeg">


</div>
```

“Src” will provide the link to the image we want to displays and set the width, height for the image. Also the “href” must be provide to access the image.

Part 6: Implement Icons and Buttons.

Icons can be placed in the html file by using the CSS Prefix fa and the icon's name. Icons are designed to be used with inline elements <i> tag for brevity, or a for more semantically correct. To increase icon sizes relative to their container, use the fa-xs, fa-sm, fa-lg (33% increase), fa-2x, fa-3x, fa-4x, fa-5x, fa-6x, fa-7x, fa-8x, fa-9x, fa-10x classes.

FIGURE 30: Icon Size (MBD 2021)

For example, the Cart and heart Items were added by the code:

```
<li class="nav-item"> <!--Heart Icon for favourite product-->
    <a class="nav-link" href="#" style="margin-top: -7px">
        <span class="heart_cart"><a href="#"><i class="fa fa-heart-o" style="font-size: 30px; margin-left: 15px"></i></a></span>
    </a>
</li>
<li class="nav-item"> <!--cart icon-->
    <a class="nav-link" href="#" style="margin-top: -7px">
        <span class="heart_cart"><a href="#"><i class="fa fa-cart-arrow-down" style="font-size: 30px; margin-left: 15px"></i></a></span>
    </a>
</li>
```

Bootstrap buttons component for actions in tables, forms, cards, and more. Bootstrap 4 provides various styles, states, and size. When the user clicks or touches it, buttons show what action will happen. Bootstrap buttons are used in the background or foreground of an encounter to initialize operations (CoreUI, 2021).

Buttons will be defined with type “button”, class “btn” with multiple parameters for instance: button shape: pill, square, circle; button action: Primary, success, Danger, Warning. There are some buttons that are used in Vaste Platform such as navigation button and search button will be defined with the setting int the next page.

```
<button class="navbar-toggler ml-auto" type="button" data-toggle="collapse" data-target="#navbarTogglerDemo02"
<span class="navbar-toggler-icon my-toggler"></span>
</button>

<button class="btn btn-outline-success my-2 my-sm-0" type="submit" style="border-radius: 15px !important;">Search
</button>
```

Part 7: Define CSS style file.

The CSS file takes responsibility for the appearances of all elements and content in the web page.

> CSS controls the fonts, text, colors, backgrounds, margins, and layout. CSS offers several significant advantages over alternative approaches to web design. CSS modifies the site designs to look how you want them to look, save money, earn money, redesign quickly, build more diverse websites. (Kyrnin, 2020).

To build Vaste platform, CSS also plays an important role. Every element inside the HTML file must have a set of design that is implemented in the file vastesstly.css. The code for CSS file can be found below.

```
/* Navbar style */

.navbar {
    background-color: #F6F6F6;
    box-shadow: 2px 4px 3px #888888;
    margin-left: -14px;
    margin-right: -14px;
}

.navbar a {
    color: #31A13C;
    text-decoration: none;
    font-size: 20px;
}

.navbar a:hover {
    color: black !important;
}
```

Part 8: Implement JS and CSS style.

The CSS file and links must be included in the Header part of the HTML file:

```
<head>
    <title>Vaste Webshop</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <!--Vaste CSS-->
    <link rel="stylesheet" type="text/css" href="css_files/vastestyle.css">
    <!--CSS Links-->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css" integrity='
    <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
</head>
```

This part will show which style the html file must follow to set the appearance of all elements in this file.
However, the JavaScript must be included inside the BODY of the html file:

```
<!--Javascript Links-->
<script src="main.js"></script>  <!--Implement javascript function form main.js file-->
<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBn
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaE
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js" integrity="sha384-OgVRvuATP1z7JjHLkuOU7X
```

JavaScript is a client-side programming language that, by implementing custom client-side scripts, allows web developers to do web application development and create dynamic and interactive web pages. Cross-platform runtime engines like Node.js can also be used by developers to write server-side code in JavaScript (Judi, 2018).

JS is used to add interactive behavior to web pages. In Vaste platform, some JS functions were added to modify the action according to the requirement showed in the code below.

CODE 8: JS Implementation.

```
// Main Page
const selected = document.querySelector(".selected");
// Returns the first Element within the document that matches the "selected" value. If no matches are found, null is returned.
const optionsContainer = document.querySelector(".options-container");
// Returns the first Element that matches the "option-container" value.
const searchBox = document.querySelector(".search-box input");
// Returns the "input value" that user add to the search box.
const optionsList = document.querySelectorAll(".option");
// Returns the value in the "option" section that user has chooseen.
selected.addEventListener("click", () => {
        optionsContainer.classList.toggle("active");
        // When the "click" action works, the option changes to "active" status.
        searchBox.value = ""; // Take the search value in search box
        filterList(""); // fillter the output according to the search box value.
        if (optionsContainer.classList.contains("active")) {
        searchBox.focus(); // search action works when the option is active.
        }
        });

// For each option in the optionList, when onclick display the suitable label and active status.
optionsList.forEach((o) => {
        o.addEventListener("click", () => {
        selected.innerHTML = o.querySelector("label").innerHTML;
        optionsContainer.classList.remove("active");
        });
        }):
```

6 CONCLUSION

Building a professional image for a business by investing in a website is a sound decision for Vaste Delivery Service and Webstore. Besides, the website supports an easy way for customer access, reduces marketing cost, gives the company opportunities abroad, and creates an easily accessible data center. From the past few decades to the present, the evolution of technology has skyrocketed effortlessly. That evolves the website-industry through multiple programming languages and smart tools like PHP, ASP.NET, JAVA. JS, and React. In facts, the market supports many resources for building a free web-site, for example: Wix, Monday, Webnote, WordPress which are all free. This is a challenge that a web developer must compete with. However, multiple resources help to improve professional skills as a web developer as well. Most of the time during this project, it was researched and learned plenty of new technology and how to apply them to the Vaste project.

Variable API support and vendor preferences make it difficult or impossible to construct consistent experiences. The website also has a high requirement about legacy considerations, platform standards, and combat problems. A web developer shall have good knowledge about offline, accessibility, localization, performance, and security. Additionally, in this thesis and the actual work during the Vaste project, an issue occurred that the working website can not ensure hundred percent similar to the original design. Building a well-functioning experience for user interface and user experience is also a big challenge. Another important requirement the project faces is the language, as Vaste is a Finnish business, the Finnish language is required. In order to understand the main service that Vaste wants to convey, the team have to translate and ask the consultant to deeply understand the requirement.

In this thesis, HTML, CSS, JS, and Bootstrap are used to build a front-end for the Vaste platform based on the API request to connect to the Vaste delivery services site. In general, the project has built a successful image for the platform that meets the requirement. The main page and support pages are well designed and implemented in a good-looking and logical way. All the requirements were researched and focused on, the process was built step by step under the consultancy from the Vaste company and Centria instructors. The knowledge of HTML, CSS, and JS has been improved and practiced in real website building. On the other hand, the limits of the experience also affect the project - experience is the best teacher- with practicing, knowledge and skills are improved. Furthermore, this is necessary to keep learning and adopt new technology to have an evolution in building websites.

REFERENCES

Codecademy, Bootstrap: Using It In a Project, 2021. Available at: https://www.codecademy.com/articles/bootstrap-deprecated-ii. Accessed: 24 January 2021.

CoreUI. Bootstrap Buttons. 6 January 2020.

Duckett. J, HTML&CSS Design and Build Website, 2011. Available at: https://wtf.tw/ref/duckett.pdf. Accessed: 22 January 2021.

Erikson, U. 2012. Why is the difference between functional and Non-functional requirements important?

Judi, T. 2018. Why Millions of Developers use Javasript for Web Application Development. Available on: https://torquemag.io/2018/06/why-millions-of-developers-use-javascript-for-web-application-development/#:~:text=JavaScript%20is%20a%20client%2Dside,implementing%20custom%20client%2Dside%20scripts.&text=JavaScript%20allows%20the%20programmers%20to%20build%20large%2Dscale%20web%20application%20easily., accessed: 28 January 2021.

Kyrnin. J, K. 2020. 5 Reasons To Learn CSS.

Mozilla.org, HTML: HyperText Markup Language, 2021. Available on: https://developer.mozilla.org/en-US/docs/Web/HTML, accessed: 23 January 2021.

Paul, K. 2018. Challenges for web developers. England: Chrome DevRel. Available on: https://paul.kinlan.me/challenges-for-web-developers/, accessed: 26 January 2021.

APPENDIX 1: VASTE CUSTOMER SQL

```
--
-- Table structure for table `customer`
--
CREATE TABLE `customer` (
  `CustomerID` int NOT NULL,
  `AccRole` char(4) NOT NULL DEFAULT 'cust' COMMENT 'Account role, default value is "cust" for customer, but can be changed to "mode" for moderator, by system administrators.',
  `AccEmail` varchar(45) NOT NULL,
  `FirstName` varchar(45) DEFAULT NULL,
  `LastName` varchar(45) DEFAULT NULL,
  `PhoneNmbr` varchar(45) DEFAULT NULL,
  `StreetAddress` varchar(45) DEFAULT NULL,
  `ApartAddress` varchar(45) DEFAULT NULL,
  `PostalCode` varchar(45) DEFAULT NULL,
  `HomeCity` varchar(45) DEFAULT NULL,
  `PasswordHash` binary(64) NOT NULL,
  PRIMARY KEY (`CustomerID`),
  UNIQUE KEY `CustomerID_UNIQUE` (`CustomerID`),
  UNIQUE KEY `AccEmail_UNIQUE` (`AccEmail`)

--
-- Dumping data for table `customer`
--
/*!40000 ALTER TABLE `customer` ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

APPENDIX 2: VASTE ORDER SQL

```
/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!50503 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOR-
EIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

-- Table structure for table `order`

CREATE TABLE `order` (
  `OrderID` int NOT NULL AUTO_INCREMENT, `CustomerID` int NOT NULL,
  `ShopID` int NOT NULL,`APIKey` varchar(45) NOT NULL,
  `DeliveryAddress` varchar(45) NOT NULL,`DeliveryApartment` varchar(45) NOT NULL,
  `DeliveryPostal` varchar(45) NOT NULL, `DeliveryCity` varchar(45) NOT NULL,
  `PickupStartTime` varchar(45) NOT NULL,
  `PickupStartDate` varchar(45) NOT NULL,
  `PickupEndTime` varchar(45) NOT NULL,
  `PickupEndDate` varchar(45) NOT NULL,
  `OrderInfo` varchar(45) DEFAULT NULL,
  `OrderDate` varchar(45) DEFAULT NULL,
  `PackageDesc` varchar(45) NOT NULL,
  `NmbrOfDeliverers` varchar(45) NOT NULL DEFAULT '1',
  `PackageHeight` decimal(10,2) unsigned DEFAULT NULL,
  `PackageWidth` decimal(10,2) unsigned DEFAULT NULL,
  `PackageLength` decimal(10,2) unsigned DEFAULT NULL,
  `PackageWeight` decimal(10,3) unsigned DEFAULT NULL,
  `PackageQuantity` int unsigned DEFAULT NULL,
  `Destination` varchar(45) DEFAULT NULL,
  `Price` decimal(10,2) unsigned NOT NULL,
  `Discount` decimal(10,5) NOT NULL DEFAULT '1.00000',
  `OrderStatus` varchar(45) NOT NULL,
  `OrderPayment` binary(1) NOT NULL DEFAULT '0',
  PRIMARY KEY (`OrderID`),
  UNIQUE KEY `OrderID_UNIQUE` (`OrderID`),
  UNIQUE KEY `APIKey_UNIQUE` (`APIKey`),
  KEY `CustomerID_idx` (`CustomerID`),
  KEY `ShopID_idx` (`ShopID`),
  CONSTRAINT `CustomerIDOrder` FOREIGN KEY (`CustomerID`) REFERENCES `customer`
(`CustomerID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `ShopIDOrder` FOREIGN KEY (`ShopID`) REFERENCES `shopowner` (`ShopID`)
ON DELETE CASCADE ON UPDATE CASCADE
```

APPENDIX 3: PRODUCT SQL

```
--
-- Table structure for table `product`
--
CREATE TABLE `product` (
  `ProductID` int NOT NULL AUTO_INCREMENT,
  `ShopID` int NOT NULL,
  `ProductType` varchar(45) DEFAULT NULL,
  `ProductName` varchar(45) DEFAULT NULL,
  `Supplier` varchar(45) DEFAULT NULL,
  `Brand` varchar(45) DEFAULT NULL,
  `Description` varchar(45) DEFAULT NULL,
  `Price` decimal(10,2) DEFAULT NULL,
  `Visibility` binary(1) NOT NULL DEFAULT '1' COMMENT 'Whether the product can be seen in the shop or not, 0 stands for invisible, 1 stands for visible. By default 1 is selected, but it can be toggled on product creation.',
  `CurrentDiscount` decimal(10,5) DEFAULT '1.00000',
  `PlannedDiscount` decimal(10,5) DEFAULT '1.00000',
  `DiscountStartTime` datetime DEFAULT NULL,
  `DiscountEndTime` datetime DEFAULT NULL,
  `Stock` int unsigned DEFAULT '0',
  `NumberSold` int unsigned DEFAULT '0',
  PRIMARY KEY (`ProductID`),
  UNIQUE KEY `ProductID_UNIQUE` (`ProductID`),
  UNIQUE KEY `ShopID_UNIQUE` (`ShopID`),
  KEY `ShopID_idx` (`ShopID`),
  CONSTRAINT `ShopIDProduct` FOREIGN KEY (`ShopID`) REFERENCES `shopowner` (`ShopID`) ON DELETE CASCADE ON UPDATE CASCADE

--
-- Dumping data for table `product`
--

LOCK TABLES `product` WRITE;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

APPENDIX 4: VASTE SHOP OWNER SQL

```
--
-- Table structure for table `shopowner`
--
CREATE TABLE `shopowner` (
  `ShopID` int NOT NULL,
  `AccEmail` varchar(45) NOT NULL,
  `BusinessID` varchar(45) DEFAULT NULL,
  `CompanyName` varchar(45) DEFAULT NULL,
  `CompanyOwnerFirst` varchar(45) DEFAULT NULL,
  `CompanyOwnerLast` varchar(45) DEFAULT NULL,
  `StreetAddress` varchar(45) DEFAULT NULL,
  `ApartAddress` varchar(45) DEFAULT NULL,
  `PostalCode` varchar(45) DEFAULT NULL,
  `HomeCity` varchar(45) DEFAULT NULL,
  `ContactInfo` varchar(45) DEFAULT NULL,
  `CompanyInfo` varchar(45) DEFAULT NULL,
  `PhoneNmbr` varchar(45) DEFAULT NULL,
  `MonTimes` varchar(45) DEFAULT NULL,
  `TueTimes` varchar(45) DEFAULT NULL,
  `WedTimes` varchar(45) DEFAULT NULL,
  `ThuTimes` varchar(45) DEFAULT NULL,
  `FriTimes` varchar(45) DEFAULT NULL,
  `SatTimes` varchar(45) DEFAULT NULL,
  `SunTimes` varchar(45) DEFAULT NULL,
  `SecretID` varchar(45) NOT NULL,
  `SecretAPIKey` varchar(45) NOT NULL,
  `ApprovalStatus` binary(1) NOT NULL DEFAULT '0' COMMENT 'Checks whether the shop has been
approved or not, 1 stands for approved, 0 stands for not approved.',
  `PasswordHash` binary(64) NOT NULL,
  PRIMARY KEY (`ShopID`),
  UNIQUE KEY `ShopID_UNIQUE` (`ShopID`),
  UNIQUE KEY `AccEmail_UNIQUE` (`AccEmail`),
  UNIQUE KEY `SecretID_UNIQUE` (`SecretID`),
  UNIQUE KEY `SecretAPIKey_UNIQUE` (`SecretAPIKey`)
)
--
-- Dumping data for table `shopowner`
--
/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

APENDIX 5: JAVA SCRIPT FUNCTION IMPLEMENTATION CODE

```
// Main Page
const selected = document.querySelector(".selected");
// Returns the first Element within the document that matches the "selected" value. If no matches are found, null is returned.

const optionsContainer = document.querySelector(".options-container");
// Returns the first Element that matches the "option-container" value.
const searchBox = document.querySelector(".search-box input");
// Returns the "input value" that user add to the search box.
const optionsList = document.querySelectorAll(".option");
// Returns the value in the "option" section that user has chooseen.
selected.addEventListener("click", () => {
  optionsContainer.classList.toggle("active"); // When the "click" action works, the option changes to "active" status.

  searchBox.value = ""; // Take the search value in search box
  filterList(""); // fillter the output according to the search box value.

  if (optionsContainer.classList.contains("active")) {
    searchBox.focus(); // search action works when the option is active.
  }
});
// For each option in the optionList, when onclick display the suitable label and active status.
optionsList.forEach((o) => {
  o.addEventListener("click", () => {
    selected.innerHTML = o.querySelector("label").innerHTML;
    optionsContainer.classList.remove("active");
  });
});
// keyup event when a keyboard key is released.
searchBox.addEventListener("keyup", function (e) {
  filterList(e.target.value);
});

const filterList = (searchTerm) => {
  searchTerm = searchTerm.toLowerCase(); // change the searchTerm input to lowercase
  optionsList.forEach((option) => {
    let label = option.firstElementChild.nextElementSibling.innerText.toLowerCase();

    //incase the value of the label which the index of "searchTerm" exist, return the block display; else display none
```

```
    } else {
      option.style.display = "none";
    }
  });
};
//Function display the filter results
function show_filters() {
  let x = document.getElementById("select-box"); // x is asigned the value of the "select-box"
  // if the value ans style of x value is equal with "none", x's style will change to "block", else its style is "none".
  if (x.style.display === "none") {
    x.style.display = "block";
  } else {
    x.style.display = "none";
  }
}

// My Orders Page
function show_active() {
  // Display the active element with suitable id and style
  // The element with id "active_block" will be display block while the other one with "completed-block"will be none.
  // The element with id "active_btn" will be display with "black"color, "bold" font, with the borderbottom "2px solid #31A13C"
  document.getElementById("active_block").style.display = "block";
  document.getElementById("completed_block").style.display = "none";
  document.getElementById("active_btn").style.color = "black";
  document.getElementById("completed_btn").style.color = "gray";
  document.getElementById("active_btn").style.fontWeight = "bold";
  document.getElementById("completed_btn").style.fontWeight = "normal";
  document.getElementById("active_btn").style.borderBottom = "2px solid #31A13C";
}

function show_completed() {
  // Display the completed element with suitable id and style
  // The element with id "completed_block" will be display block while the other one with "active-block"will be none.
  // The element with id "completed_btn" will be display with "black" color, "bold" font;
  document.getElementById("active_block").style.display = "none";
  document.getElementById("completed_block").style.display = "block";
  document.getElementById("active_btn").style.color = "gray";
  document.getElementById("completed_btn").style.color = "black";
  document.getElementById("active_btn").style.fontWeight = "normal";
  document.getElementById("completed_btn").style.fontWeight = "bold";
  document.getElementById("active_btn").style.borderBottom = "none";
```

```
}

// Log in Page
function eye() {
  // allow to hide the text value (password)
  var x = document.getElementById("myInput"); // asign the input value with the id "myInput"to x
  var y = document.getElementById("hide1"); // asign the input value with the id "hide1"to y
  var z = document.getElementById("hide2"); //asign the input value with the id "hide2"to z

  // If x's type and value are equal with "password"
  if (x.type === "password") {
    x.type = "text"; // asignx's type to text
    y.style.display = "block"; // show y
    z.style.display = "none"; // hide z
  } else {
    x.type = "password"; // asign x's type to "password"
    y.style.display = "none"; // hide y
    z.style.display = "block"; // show z
  }
}

function show_login() {
  // display the Login's element
  document.getElementById("login_page").style.display = "block"; // Return the element with the id "login_page" and display block
  document.getElementById("create_an_account").style.display = "none";
  document.getElementById("login_button").style.color = "black"; // Return the element with the id "login_button" and display black color
  document.getElementById("create_button").style.color = "gray"; // Return the element with the id "create_button" and display gray color
  document.getElementById("login_button").style.fontWeight = "bold"; // Return the element with the id "login_button" and display bold font
  document.getElementById("create_button").style.fontWeight = "normal";
}

function show_create() {
  // display the Create's element
  document.getElementById("login_page").style.display = "none";
  document.getElementById("create_an_account").style.display = "block"; // Return the element with the id "create_an_account" and display block
  document.getElementById("login_button").style.color = "gray"; // Return the element with the id "login_button" and display gray color
```

```
  document.getElementById("create_button").style.color = "black"; // Return the element with the id "create_button" and display black color
  document.getElementById("login_button").style.fontWeight = "normal";
  document.getElementById("create_button").style.fontWeight = "bold"; // Return the element with the id "create_button" and display bold font
}

// Add New Product Page
function add(x) {
  // allow to add value to element
  const div = document.createElement("div"); // return new element with type div
  div.innerHTML = `
    <input style="margin-top:10px; " type="text " class="form-control ">
      `; // sets the HTML markup contained within the element.
  if (x == "color") {
    document.getElementById("new_product_color").appendChild(div); // if value of x is "color" append a "div" node as the last child of the element with the id "new_product_color"
  } else if (x == "size") {
    document.getElementById("new_product_size").appendChild(div); // if value of x is "size" append a "div" node as the last child of the element with the id "new_product_size"
  }
  return false;
}
```