Tampere University of Applied Sciences

Refactoring legacy website styles

Gleb Tsurakov

BACHELOR'S THESIS
December 2021

Media & Arts
Interactive Media

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Media and Arts
Interactive Media

Gleb Tsurakov:
Refactoring legacy website styles

Bachelor's thesis 34 pages, appendices 0 pages
December 2021

The purpose of this bachelor's thesis was to identify problems in working on styles for the default version of the Oscar Software eCommerce website and solve these problems by creating a new structure for style sheets. In the course of work on this bachelor's thesis the project was created and default styles that are used in eCommerce's web-shop were copied into it. After the problems were identified, a new structure was created for style sheet files and style rules inside them were refactored.

In computer programming, refactoring is the process of restructuring the existing code without changing its external functionality. In this case the goal was to refactor CSS without changing the visual part of the website.

The existing style rules were distributed into separate components and then structured using a modern approach called Nested CSS.

This project did not have any effect on creating custom websites for clients, but it will potentially affect the work of developers with versions of the default website. The work on the dissertation was commissioned by Oscar Software Oy.

Key words: website, style sheets, web-shop, refactoring

Table of Contents

ABBREVIATIONS AND TERMS

CSS	Cascading Style Sheets.
HTML	Hypertext Markup Language, a standardized system for tagging text files to achieve a font, colour, graphic, and hyperlink effects on World Wide Web pages.
Legacy code	A source code inherited from someone else or inherited from an older version of software. It can also be any code that is hardly understandable and is difficult to maintain.
Repo	A repository or a storage location for software packages.
OOCSS	Object-oriented Cascading Style Sheets.
BEM	Block Element Modifier.
SASS	Syntactically Awesome Style Sheets.
CP / M	Originally standing for Control Program/Monitor and later Control Program for Microcomputers.
DOS	Disk Operating System.
eCommerce	Commercial transactions conducted electronically on the internet.
Git	A software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing a source code during software development.

Misc.	Stands for Miscellaneous or something that includes many things of different kinds.
Code refactoring	In computer programming, refactoring is the process of restructuring the existing code without changing its external functionality.
Pre-processor	In program that processes its input data to produce output that is used as an input to another program.

1 INTRODUCTION

Well-structured code is easy to read and work with. But it is rarely possible to write a perfect code right away. Developers are often in a hurry, task requirements may change in the process pf development, testers find bugs that need to be quickly fixed. As a result, even an initially well-structured code becomes messy and incomprehensible.

Any piece of software becomes obsolete over time: programming languages are improving, new frameworks, libraries, operators come to market. New technologies make it easier and faster for developers to write a code. What a year ago required twenty lines, today can be solved with just one. Therefore, even a once ideal program eventually requires new updating obsolete code fragments. In computer programming code refactoring is the process of restructuring the existing computer code without changing its external functionality. It is intended to improve the structure of a code or a style sheet file while preserving its functionality.

In this bachelor's thesis I will do the refactoring of a default style sheets of a website by separating style rules into smaller components and creating a new file architecture. This might significantly reduce the amount of time developers spend on working with a default version of the website.

2 Oscar Software

Oscar Software is a software company based in Tampere. The history of the company begins in 1984 when a version of the Oscar software was released for the CP / M and DOS operating systems (Oscar Software n.d.). Now, the main product of the company is the provision of various ERP "Enterprise Resource Planning" (Oscar Software n.d.). The list of services also includes eCommerce.

2.1 eCommerce

The main tasks of the eCommerce are to create online stores, as well as provide a content management tool to simplify the maintenance of website content for client. “Oscar Online Store is an online store software that in addition to comprehensive e-commerce functionalities also contains a content management tool for easy content maintenance of the website. Oscar Online Store is an online store and extranet solution that is genuinely fully integrated in customers enterprise resource planning system” (Oscar Software n.d.).
To create a web shop eCommerce team has developed a default website that includes already made templates and default styles. Instead of creating a web shop from scratch, a developer can simply upload already made files into a new project. This approach helps developers to save time on creating websites.
Please start the second paragraph here.

3 TECHNOLOGY USED BY ECOMMERCE

3.1 CSS

CSS (Cascading Style Sheets) is a coding language that gives a website its look and layout. Along with HTML, CSS is fundamental to web design. Without it, websites would still be plain text on white backgrounds. CSS allowed several innovations to webpage layout, such as the ability to:

1. Specify fonts other than the default for the browser
2. Specify colour and size of text and links
3. Apply colours to backgrounds
4. Contain webpage elements in boxes and float those boxes to specific positions on the page

(Bigcommerce n.d.)

It is an important tool for eCommerce because it allows developers create custom websites in most cases without changing pre-made HTML templates. With CSS it is possible to affect elements on a website globally instead of changing them one by one.

3.2 Bootstrap 4

Bootstrap 4 is a HTML, CSS and JavaScript library that helps developers to simplify the process of creating websites. When you add bootstrap to the project it gives basic styles to the elements on the website via HTML classes. As a result, you get a uniform appearance for elements on a website, for example: Tables, forms, inputs, checkboxes. Developers can also take advantage of CSS classes defined in Bootstrap to further style the appearance of their contents (REDAD 2021). The default website developed by eCommerce team was created using Bootstrap 4. This library allows developers to save time on styling websites as it has built-in style rules by default.

3.3 Sass CSS

Sass (Syntactically Awesome Style Sheets) is a pre-processor scripting language. It is compiled into CSS (Cascading Style Sheets). There are several reasons why Sass is used by eCommerce developers.

3.3.1 Variables

The first reason is that it allows to use variables. Variables in web development allow a value to be stored in one place, then referenced in multiple other places. It is quite important to have variables when working on a project with a large amount of CSS, often with a lot of repeated values, because it saves a lot of time.

3.3.2 Nested CSS

The second reason it that Sass pre-processor allows developers to write nested CSS. Nested CSS does not only reduce the amount of CSS lines in a file but also makes it possible to follow the same visual hierarchy of HTML.

3.3.3 Mixins

“Mixins” are the third reason why Sass pre-processor is used. They allow developers to define patterns of property value pairs, which can then be reused in other rules. In other words, “mixins” help developers not to repeat themselves by writing the same properties repeatedly.
The “mixin” name is a class selector that identifies the “mixin” being declared. Using “mixins” can potentially reduce the amount of time spent on writing style rules for elements.

3.4 Gulp commands

To create a new project developer runs a Gulp command. Gulp is a cross-platform runner that allows programmers to automate many development tasks, so developers do not have to do them manually (Dev 2021). It helps in web development processes automatization. In our case it helps to minify files and avoid uploading CSS files and HTML templates manually.

3.5 Git

Repository or simply Repo is a storage space where developers can keep their projects. Repository can be local to a folder on a computer, or it can be a directory space on GitHub or another online host. The all sorts of files can be kept inside a repository (Readwrite 2013). Besides that, it can be used for tracking changes in any set of files. Often it is used for to coordinate work among developers collaboratively working on source code during software development. Its main goals include data integrity, speed and support for distributed, non-linear workflows (Git-Scm n.d).

4 DEFAULT STYLE STRUCTURE

In the eCommerce website default styles, the module folder consists of “variables”, **“colors”** and a “component” - file. In our case file **“varables”** includes variables for breakpoints, fonts and some properties for buttons, for example its height. Variables allow developers to significantly reduce time on writing and changing styles of a website. The file called **“colors”** includes variables that are connected to a colour used on the website. For example, variables to change the colour of text, elements, background, and borders. The **“components.scss”** - file includes additional style rules for random components used on website, **“mixins”** and media queries. This file was refactored further in a project.

4.1 Modules folder

In eCommerce website default styles, the module folder consists of “variables”, **“colors“** and a “component” file. In our case file **“variables“** includes variables for breakpoints, fonts and some properties for buttons, for example its height. Variables allow developers to significantly reduce time on writing and changing styles of a website. The file called **“colors”** includes variables that are connected to color used on the website. For example, variables to change the colour of text, elements, background, and borders. The **“components.scss”** - file includes additional style rules for random components used on website, **“mixins”** and media queries. This file was refactored further in a project.

4.2 Components folder

The component folder consists of files with CSS classes that define properties of a particular component. The components are reusable, self-contained, and independent user interface elements. For example, cart, footer, header, checkout, carousel, and so on. Each of the listed components can be also created from other components, for example, a carousel on a website can include several product cards, which in turn consist of an image, a buy button, a line with a product code or a price. When working on a big project it is a good practice to separate styles into smaller parts to have a code that is more understandable, readable, and easier to work with.

4.3 "styles.scss” - File

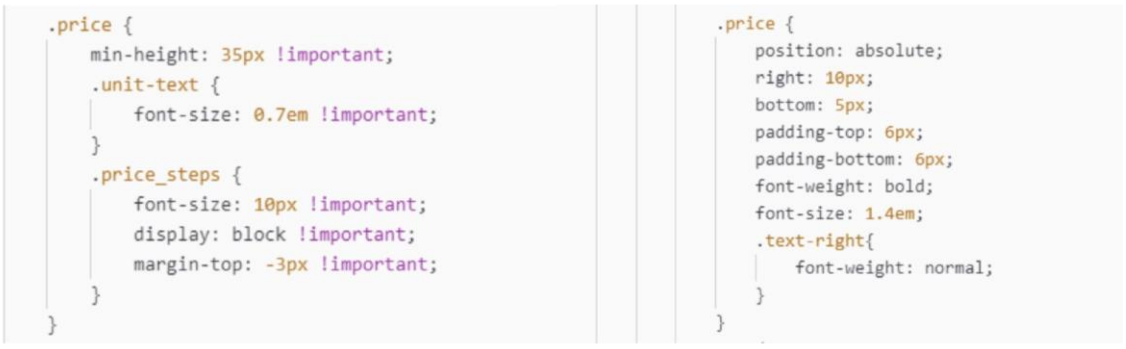
The main purpose of this file is to bring up all the default styles together in one. To achieve that all the components and modules are inserted in that file using @import rule. The CSS rule “@import” is used to import style rules from other style sheets. It is a common practice to keep all the importing files in one to connect them all and keep an easy and clear file architecture.

5 DEFINITION OF PROBLEMS IN DEFAULT STYLES

The development and maintenance of the default eCommerce website requires constant improvement for example adding new features, changing the styles or correction of errors. This process requires a well thought architecture of files and well-structured style sheets.
A well-structured architecture saves developers from spending time on finding a correct line of code and thinking where to add a new property. A bad structure of files can cause conflicts, overlapping and errors. CSS doesn't have conflicts because multiple rules can be applied to the same elements and their order reflects an “importance”. For example, if it appears last, it will override previous rules. But in order to keep a good structure it is recommender to separate styles into smaller components. It is also important to avoid using global styles that can affect all the elements on a website.
There are several problems in eCommerce website’s default styles due to which it cannot be called well-structured.

5.1 Overlapping styles

Each of the elements on a website usually has a class or an id. In order to style the element, you need to refer to the element by writing its class or an id and then giving properties to it. Usually, this process does not cause any problems. However, if there are multiple elements with same classes or ids that have different properties it might cause a styles overlap (Figure 1).



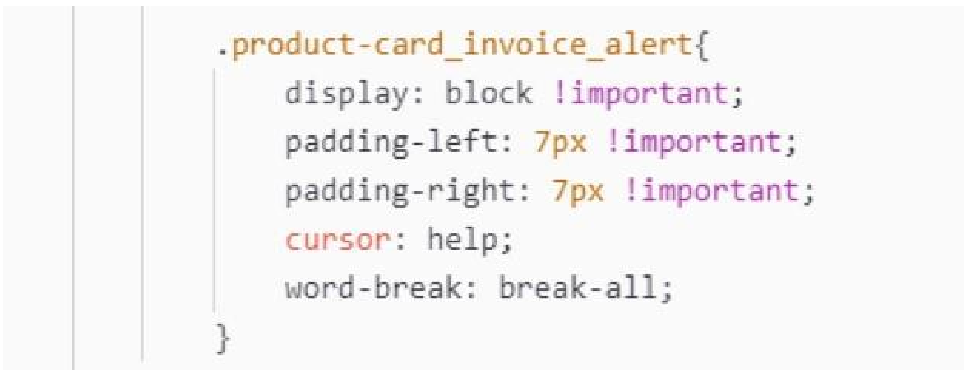
```
.price {
    min-height: 35px !important;
    .unit-text {
        font-size: 0.7em !important;
    }
    .price_steps {
        font-size: 10px !important;
        display: block !important;
        margin-top: -3px !important;
    }
}
```

```
.price {
    position: absolute;
    right: 10px;
    bottom: 5px;
    padding-top: 6px;
    padding-bottom: 6px;
    font-weight: bold;
    font-size: 1.4em;
    .text-right{
        font-weight: normal;
    }
}
```

FIGURE 4. Different properties under the same (price) class name.

5.1.1 Solving overlapping using elements “importance”

There are several ways to avoid an overlapping in styles. The first one is to put an “!important” tag after the value after the property (Figure 2). It is used to add more importance to a value and override the styling rule. In some cases it is necessary to put the “!important” rule, for example if a developer wants to override the built-in styles of a library. In our case this library is a Bootstrap. The Bootstrap built-in styles have more “importance” than styles from connected CSS file, that is when developer might need to add the “!important” property in order to override the Bootstrap styles.



```
.product-card_invoice_alert{
    display: block !important;
    padding-left: 7px !important;
    padding-right: 7px !important;
    cursor: help;
    word-break: break-all;
}
```

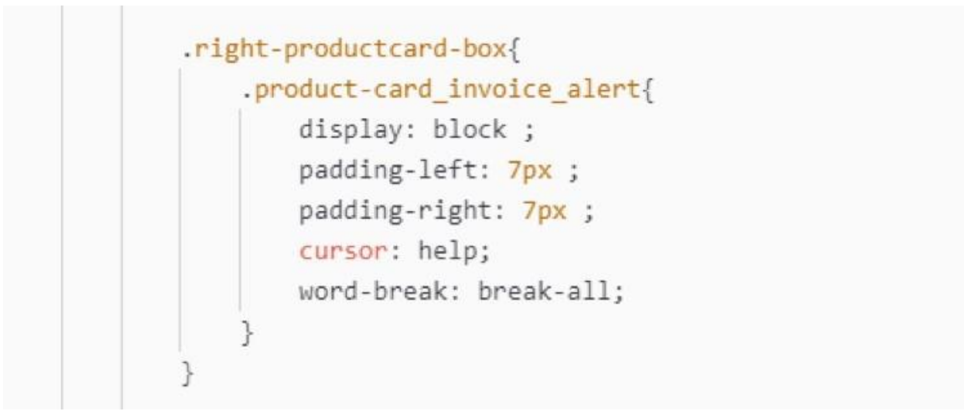
FIGURE 2. Overridden style properties with “!important” rule.

However, it is best to avoid the “!important” property because it makes debugging and maintenance of the styles hard for developers. To change the style of an element It is a better practice to find a line of code that needs to be changed and replace its value. That way developers can reduce the amount lines in CSS files and keep it cleaner.

5.1.2 Solving overlapping by defining a parent element

The second way to avoid conflicts in styles is to specify their parent element. That can be achieved by simply writing the parent element's class or id before the actual element's name. However, that approach can also be improved by nesting one style rule inside another. In our case this method is possible because of Sass.

Nested CSS increases the modularity and maintainability of style sheets and significantly reduces the amount of written CSS. It also helps to keep all styles related to an element directly under its parent element (Figure3).



```
.right-productcard-box{
    .product-card_invoice_alert{
        display: block ;
        padding-left: 7px ;
        padding-right: 7px ;
        cursor: help;
        word-break: break-all;
    }
}
```

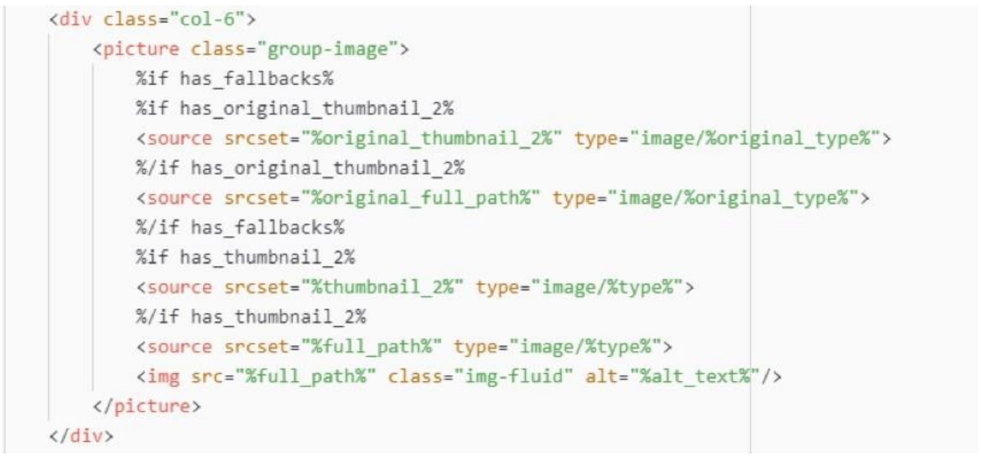
FIGURE 3. "product-card-invoice_alert" - class and its styles properties inside the parent element - “right-productcard-box”.

5.2 Different screen sizes

Nowadays it is essential to create versions of websites not only for desktops but also for mobile phones and tablets. In order to achieve that, the elements used on a website should have different properties based on a screen size.

5.2.1 Bootstrap classes

There are several methods that eCommerce uses to create states of elements for different devices. First is using the Bootstrap library. Bootstrap has built-in classes, for example “**col-6”** that allow developers to set a specific width of an element based on a screen size without writing CSS (Figure 4).



```
<div class="col-6">
    <picture class="group-image">
        %if has_fallbacks%
        %if has_original_thumbnail_2%
        <source srcset="%original_thumbnail_2%" type="image/%original_type%">
        %/if has_original_thumbnail_2%
        <source srcset="%original_full_path%" type="image/%original_type%">
        %/if has_fallbacks%
        %if has_thumbnail_2%
        <source srcset="%thumbnail_2%" type="image/%type%">
        %/if has_thumbnail_2%
        <source srcset="%full_path%" type="image/%type%">
        <img src="%full_path%" class="img-fluid" alt="%alt_text%"/>
    </picture>
</div>
```

FIGURE 4. Bootstrap class “col-6” used in default HTML template.

5.2.2 Media Queries in CSS code

The second method used by the eCommerce team to define the behaviour of an element based on a screen size is writing Media Queries in CSS files. “Media Queries are a feature in CSS3 which allows you to specify when certain CSS rules should be applied. This allows you to apply a special CSS for mobile view. The advantage of this method is that only the valid CSS is downloaded” (CSS Media queries n.d.).
Media queries can be kept in a separate file as well as in a file with other CSS rules, but it is important to keep them easily maintainable. In our case media queries are located randomly in most of the component files. It is a better practice to keep media queries under their parent element. This way it is easier and faster to find a specific media query rule and change its value if needed.

5.3 Massive files

In order to keep the CSS file clean and well-structured it is a good practice to separate styles into smaller components. In eCommerce default styles there are several files that include properties of different elements of the website. This not only causes the overlapping of styles but also makes these files very large and hard to maintain.

To fix this issue I will refactor these files and distribute code inside them into separate components. It will help to keep a better architecture of style files and make maintenance of CSS code easier for developers.

5.4 CONCLUSION

There are several problems in default styles of eCommerce website that interfere with developer’s work and do not allow developers to comfortably maintain the version of a website’s styles. Large files, bad structure of code and overlapping styles affect the productivity of developers and increase time of work. These problems can be solved by refactoring the default style files and creating a better architecture of the files. In order to do that I will have to create a new branch from a Misc. repository, create a new project folder, refactor the existing files in it and restructure the CSS. The main task is to save the default look of the website.

6 LEGACY STYLES REFACTORING

6.1 Definition of a good code

The main reason why HTML and CSS are not considered programming languages is because they only determine the structure and the style of the webpage you are building. They do not contain any instructions like the other front-end languages (FutureLearn 2021). However, CSS code and HTML files should still be easy to work with for developers. Before code refactoring it is important to define the “good” code.
“Good code is written so that is readable, understandable, covered by automated tests, not over complicated and does well what is intended to do” (IntentHQ 2015). In order to make CSS more understandable for developers there are several methodologies that can make working with style sheets easier.

6.2 CSS methodologies

In order to avoid coming up with you own rules of writing CSS, the developer can adopt one of the approaches that design community has tested in many projects. These approaches are called CSS methodologies. These methodologies are basically coding guides that help developers to write an organised and structured code. They generally tend to display CSS in more detailed way than developer would if he/she wrote and optimized each selector for a custom ruleset for that project. Since many of these systems are widely used, other developers are more likely to understand the approach you are using (MDN Web Docs 2021).

6.2.1 BEM methodology

BEM stands for Block Element Modifier. In that methodology a block is a standalone entity for example a product, button or a link. A modifier is a flag on a block. It changes the styling or a behaviour. BEM It is a CSS methodology for creating a reusable CSS code. It provides developers with component naming patterns that make the relationship between styled component classes and markup more obvious.

In our case BEM methodology cannot be applied because it required changing the HTML code in order to change the class names.

6.2.2 OOCSS methodology

Most of the methodologies developers come across have something related to the concept of Object-Oriented CSS (OOCSS). The main idea of Object-Oriented CSS is to split your CSS into several reusable elements that can be used anywhere on your site (MDN Web Docs 2021). For example, if we have two elements that share most of the styles, it is best to avoid writing the same style rules several times. The best solution would be to combine the styles of elements in one rule and then separately extend the styles of elements if needed. That way we can avoid repeating ourselves and keep the style sheets clearer. That methodology was partially used in the refactoring process as it does not require any HTML changes.

6.3 Project set up

Before code refactoring, I have created a new folder for this project. I decided to name my project folder ‘test-styles’. After that I have created a separate branch from a Misc. repository. I have named this branch ‘thesis-styles’ (Figure 5). By creating a separate branch, I have made sure that my work will not affect the original files.

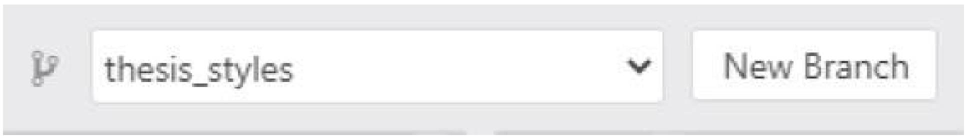


FIGURE 5. A new branch “thesis_styles”.

6.4 Massive files refactoring

When a developer stars working on a big CSS files and big projects he/she understand that maintaining and working with a big CSS file can be complicated. In some cases when you have a lot of different style rules for elements across the website, you might want to separate your stylesheets into smaller files. It makes it easier to work with and to keeps your CSS file organized (MDN Web Docs 2021).

6.4.1 "_main-contents.scss" - file

The first step in the refactoring of a default styles was separating CSS in large files into already existing components. One of the files with the biggest amount of CSS in it was "_main-contents.scss". It included 1330 lines of style rule of multiple components. There were style rules for such classes as "page-2", "product-list", "ajaxContainer", "share-buttons". Most of the styles did not fit to any of the already existing components so I had to create new component files. Only from the "_main-contents.scss" - file refactoring I had to create such component files as:

1. "_product-list-item.scss"
2. "_product-group-list.scss"
3. "_product-list.scss"
4. "_product-related.scss"
5. "_product-variation.scss"
6. "_product-page.scss"
7. "_group-page.scss"
8. "_product-related.scss"
9. "_buttons.scss"

The reason why all the file names start with an underscore is because it is important to keep the same naming system throughout the whole project. The already existing components had this naming system, so I decided to keep it and name the new components the same way.

After separating all the style rules into different components the **"_maincontents.scss"** file was empty. There was no need for this file anymore, so I deleted it. During the project I have deleted most of the origin files after dividing CSS inside them into different components.

6.4.2 "_cart.scss" - file

The second biggest file was the **"_cart.scss"**. This file included all the style rules for shopping carts on a website. There are several places on the website where shopping carts can be used. A shopping cart is an element that facilitates the purchase of a product or service by showing a customer the products he is about to buy. It can be displayed as a drop-down on a top of the website or as a table on a separate page. There are three different places eCommerce uses to show the cart on a website: a drop-down on a header, a table on a shopping-cart page and as a table on a check-out page. I decided to create new components based on different places the shopping cart used. After refactoring the **"_cart.scss"**-file I got three components:

"_cart-checkout.scss" - includes styles of a shopping cart used only on a checkout page.
"_cart-dropdown.scss" - includes styles of the added products in a small dropdown.
"_cart-table.scss" - includes styles of the added products in the form of a table.

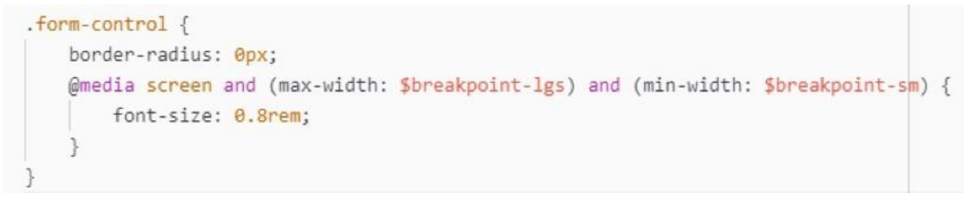
6.5 Common components

Almost in every original component file a common or a global CSS rule could be found. It could be a button, an icon, a font, or a bootstrap override rule. A common or a global style rule that affects all the elements throughout the website should be kept in a separate file. In a case when there are a lot of common styles it is a good practice to separate them into smaller components so it would be easier to maintain them. I have decided to distribute all the common styles from components according to their purpose. In the end I got sixteen common components

“_alert.scss" - styles for pop-up messages on a web page.
“_amount-box.scss“ - styles for amount inputs.
“_body.scss“ - includes general style rules for ‘body’ of every used HTML file.
“_bootstrap-overrides.scss“- styles that override the built-in libraries style rules.
“_brochures.scss“ - includes styles of brochures of the products.
“_buttons.scss“ - includes styles of basic buttons of the website.
“_console.scss“ - includes styles of an element and tab content in it.
“_events.scss“ - includes styles of event elements on a front-page.
“_filters.scss“ - includes styles of product lists filters.
“_font.scss“ - includes styles of basic fonts and their states (hovered and active).
“_icons.scss“ - includes styles of basic icons used on a website.
“_label.scss“ - includes styles of basic product labels.
“_main-content.scss“ - includes style rules of an index-page (template that every page uses except for a front page.)
“_maintenance.scss“ - includes styles that did not match any of the common components (only maintenance-wrapper class).
“_pagination.scss“ - includes styles of arrows and page numbers that are used to navigate between pages.
“_video.scss“ - includes styles of video container.

6.6 Bootstrap overrides

There are a lot of bootstrap classes used in eCommerce default website HTML templates. Each of the Bootstrap classes has its own properties. In some cases, these properties should be extended but sometimes only few of them need to be saved. There are several cases in default styles where the bootstrap classes need to be overridden or changed. For example, **“form-control” -** class. If we inspect that class in a browser, we can see that by default it has multiple styles including border radius. This property was overridden in eCommerce’s default styles (Figure 6).



```
.form-control {
    border-radius: 0px;
    @media screen and (max-width: $breakpoint-lgs) and (min-width: $breakpoint-sm) {
        font-size: 0.8rem;
    }
}
```

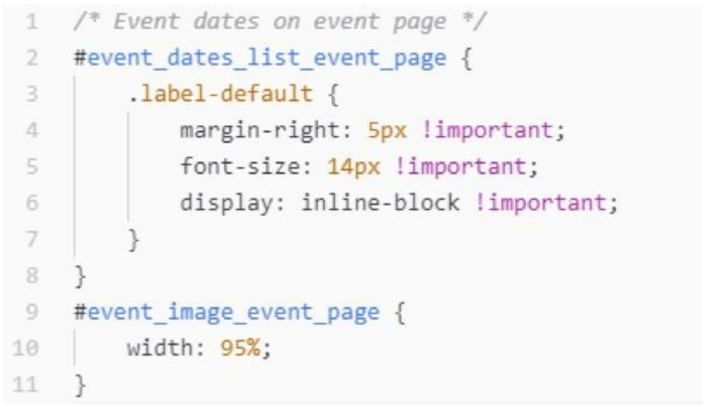
FIGURE 6. Overridden properties of a Bootstrap class.

This and all the other bootstrap overrides that globally affect the default styles I have put into a separate file called "**_boostrap-overrides.scss**". This file is considered as a common component because they affect every element throughout the default website.

6.7 Comments in CSS

When developer adds comment lines to a CSS code it helps other developers to work with stylesheets. The comments also help developers when they come back to the project after a long break. It is a good practice to add comments between logical sections in your CSS code. Comments are very useful for navigation in CSS file. It helps to locate different sections quickly when going through them. (MDN Web Docs 2021).

After separating all the original files into smaller components, I got 43 new component files. For better navigation I have left a short comment on every file that explains the purpose of the styles inside it (Figure 7).



```
/* Event dates on event page */
#event_dates_list_event_page {
    .label-default {
        margin-right: 5px !important;
        font-size: 14px !important;
        display: inline-block !important;
    }
}
#event_image_event_page {
    width: 95%;
}
```

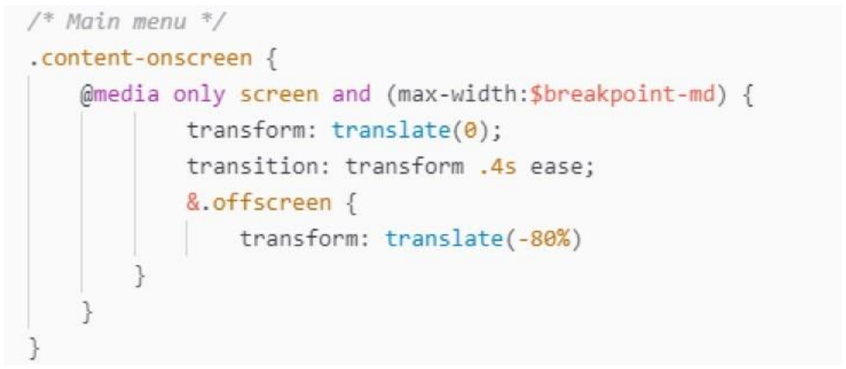
FIGURE 7. A short comment in a CSS file for a better navigation.

6.8 Media Queries

During the code refactoring one of the biggest challenges was to distribute all the media queries under the correct component. In the original styles there was a large CSS file called “**_media-queries.scss**“ that included most of the media queries of the default styles. It is not the best way to keep all the media queries in a separate file because it might be complicated for a developer to understand which component it is related to because components can share the same classes and ids.

The better way is to keep media queries under the component they are related to (Figure 8). That will save developers time on looking for a correct line of CSS to change the property.

The original “**_media-queries.scss**“ file had 387 lines of CSS. After distributing media queries under the correct classes, the file was empty, and I removed it.



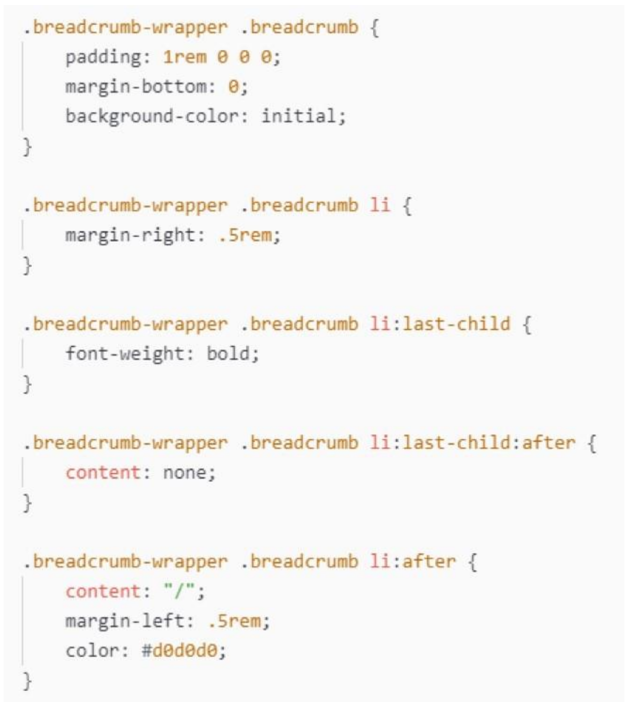
```
/* Main menu */
.content-onscreen {
    @media only screen and (max-width:$breakpoint-md) {
            transform: translate(0);
            transition: transform .4s ease;
            &.offscreen {
                transform: translate(-80%)
            }
    }
}
```

FIGURE 8. Elements Media Queries are under the class they belong to.

6.9 Nested CSS

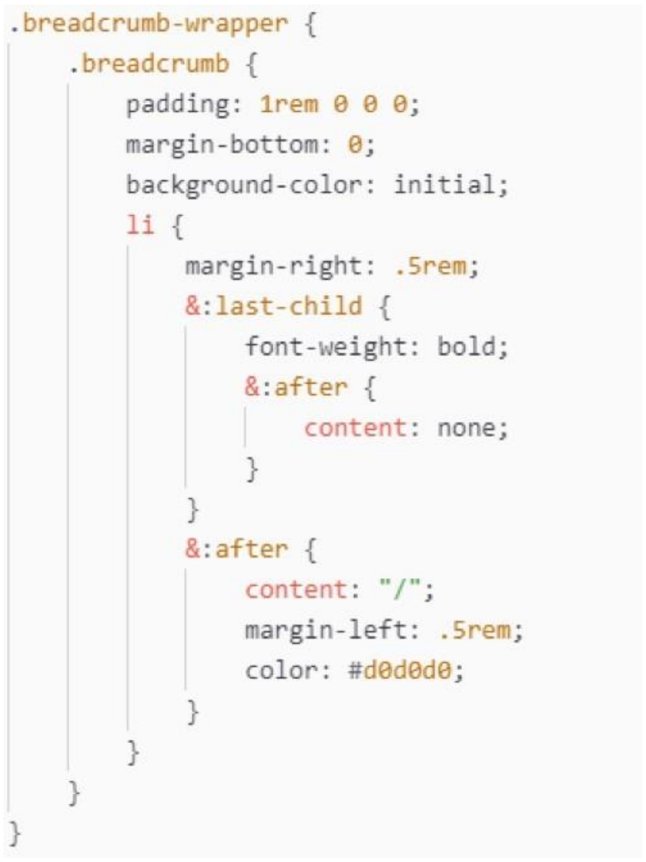
The next step after adding media queries under the correct components was reducing the amount of CSS lines by writing nested CSS. Nested CSS in that case does not only reduce the amount of code but also makes it clearer because there is no need in repeating class names to call their daughter elements. The Figure 9 has repeating classes when Figure 10 has Nested CSS.

Going through all the component files and nesting their CSS structure was the most time-consuming process during the refactoring process.



```css
.breadcrumb-wrapper .breadcrumb {
    padding: 1rem 0 0 0;
    margin-bottom: 0;
    background-color: initial;
}

.breadcrumb-wrapper .breadcrumb li {
    margin-right: .5rem;
}

.breadcrumb-wrapper .breadcrumb li:last-child {
    font-weight: bold;
}

.breadcrumb-wrapper .breadcrumb li:last-child:after {
    content: none;
}

.breadcrumb-wrapper .breadcrumb li:after {
    content: "/";
    margin-left: .5rem;
    color: #d0d0d0;
}
```

FIGURE 9. Style sheet includes a lot of repeat classes e.g. - “breadcrumwrapper”. This might cause distraction in file navigation.



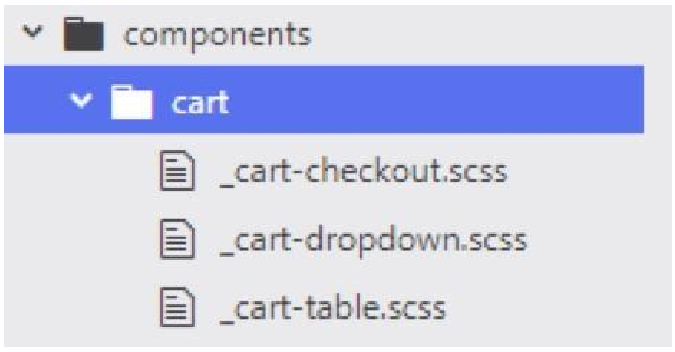
```
.breadcrumb-wrapper {
    .breadcrumb {
        padding: 1rem 0 0 0;
        margin-bottom: 0;
        background-color: initial;
        li {
            margin-right: .5rem;
            &:last-child {
                font-weight: bold;
                &:after {
                    content: none;
                }
            }
            &:after {
                content: "/";
                margin-left: .5rem;
                color: #d0d0d0;
            }
        }
    }
}
```

FIGURE 10. All the elements and their properties are under one class. Nesting makes CSS code more readable.

7 NEW FILE ARCHITECTURE

7.1 Combining components in folders

The next step was to create a well-structured architecture that will allow developers to easily navigate between components. To achieve a good structure of files I have created multiple folders and distributed all the components between them. All the component files were put into folders based on their purposes (Figure 11). For example, footer related components were put into a folder called" footer" common components were put folder" commons", product related components were put into a "product" folder. In the end I got 10 folders with related components in them. There is no one correct way to distribute components between folders. The way of file distribution should be based on a specification of a project.



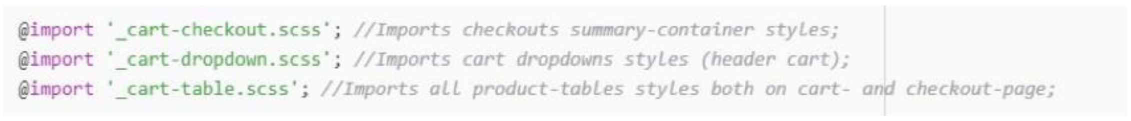
```
components
  cart
    _cart-checkout.scss
    _cart-dropdown.scss
    _cart-table.scss
```

FIGURE 11. Cart related component files are sharing the mutual folder named "cart".

7.2 Combining file

In order to avoid a big amount of imported files in "**style.scss**" - file I have decided to combine all the components in folders in one file. For that purpose, I have created a new file in each component folder and imported all components in there (Figure 12). This structure allowed to significantly reduce the amount of imported files into a "**style.scss**" - file and keep a good hierarchy of the files.

Potentially it can save some time of a developer if he/she would want to add a new component into a folder. You only need to import a new component to a combining file on the bottom of a component file. It will automatically connect a new file to the mutual “**style.scss**” – file, not affecting it structure.



```
@import '_cart-checkout.scss'; //Imports checkouts summary-container styles;
@import '_cart-dropdown.scss'; //Imports cart dropdowns styles (header cart);
@import '_cart-table.scss'; //Imports all product-tables styles both on cart- and checkout-page;
```

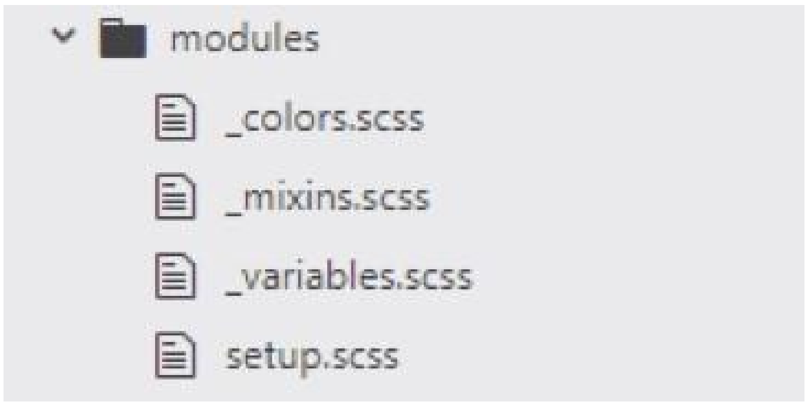
FIGURE 12. All the component files imported into one combining file.

8 MODULES

I decided to keep “variables” and **“**colors**”** files in the modules folder. Modules in this project are the values that all components should have access to. For example, variables are used in media queries to indicate the size of the screen in pixels. Since at this stage there are media queries in almost every component file - every component should have access to a **“**variables**”-** file. Colour is also a variable that is used among most of the component files and should be accessible to all the components.

Another values that components can share are the **“mixins”**. In the original version of default styles most of the **“mixins”** were located in the **“_components.scss”** file among some random styles. I have renamed this file and transferred all the styles that are not inside **“mixins”** to component files that they belong to. Eventually I got three files in a **“modules”** folder that need to be connected to components.

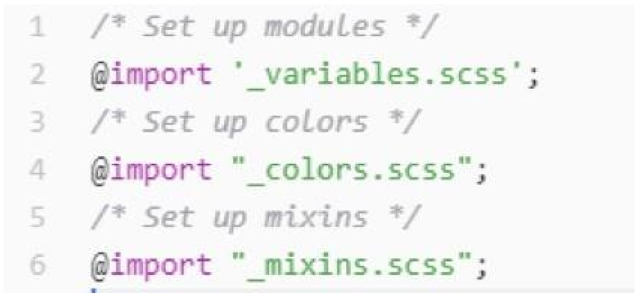
In order to give the components an access to modules I have created a new file - **“set-up.scss”** (Figure 14)



modules
_colors.scss
_mixins.scss
_variables.scss
setup.scss

FIGURE 14. Module – folder new structure.

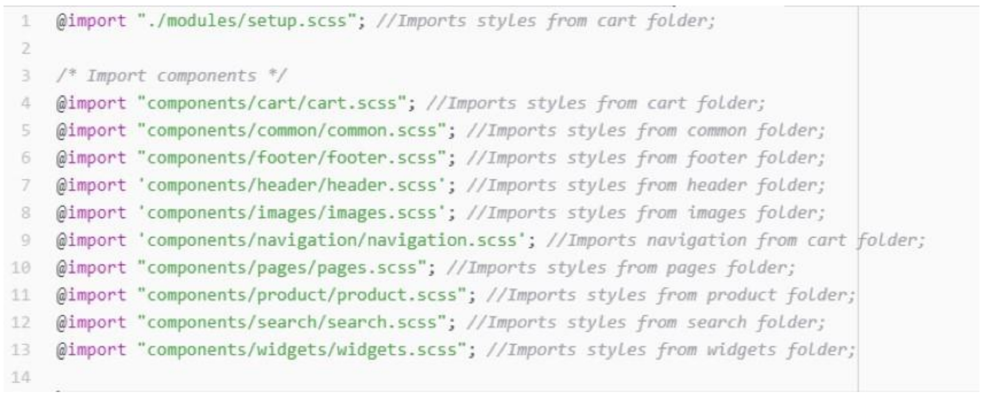
In that file I have included all the modules via the CSS **"@import"** rule (Figure 15).



```scss
/* Set up modules */
@import '_variables.scss';
/* Set up colors */
@import "_colors.scss";
/* Set up mixins */
@import "_mixins.scss";
```

FIGURE 15. Modules imported into one **"set-up.scss"** - file.

After that I have included the **"set-up.scss"** file into every combining file in components folders (Figure 16). The final stage was to import components to the **"style.scss"** file to bring all the components and modules together.



```scss
@import "./modules/setup.scss"; //Imports styles from cart folder;

/* Import components */
@import "components/cart/cart.scss"; //Imports styles from cart folder;
@import "components/common/common.scss"; //Imports styles from common folder;
@import "components/footer/footer.scss"; //Imports styles from footer folder;
@import 'components/header/header.scss'; //Imports styles from header folder;
@import 'components/images/images.scss'; //Imports styles from images folder;
@import 'components/navigation/navigation.scss'; //Imports navigation from cart folder;
@import "components/pages/pages.scss"; //Imports styles from pages folder;
@import "components/product/product.scss"; //Imports styles from product folder;
@import "components/search/search.scss"; //Imports styles from search folder;
@import "components/widgets/widgets.scss"; //Imports styles from widgets folder;

```

FIGURE 16. The set-up file imported into "commons" - combining file.

9 EVALUATION OF PERFORMANCE

After the code refactoring was done the next step was to test it and check if it affected the original look of the website. To do that I had to install new default styles for my project from a **“thesis-styles”** branch. To automatically install new styles to the project folder I had to write a Gulp command,
After the default styles were uploaded into the new project folder, I had to run a second gulp command **“watch-customer-styles – customer thesis-styles”**. This command supposed to track all the changes in a custom SCSS file and minify or compress all the stylesheets into one CSS file.
After saving the custom file, a build folder was created automatically inside the **thesis-styles”** folder. The **“build”** folder had a minified CSS file that was ready to be put on a server. The website was the same after the code refactoring, no changes were noticed.

9.1 SUMMARY

Refactoring did not fix bugs or add a new functionality to the default website. But it made the code more understandable and readable for developers. In our case code refactoring did not have any effect on the creating custom websites. However, a better structure of CSS and a new file architecture will potentially save a lot of time for developers when working on the version default websites styles.

REFERENCES

Oscar Software. N.d. Historia. Read 20.11.2021. https://www.oscar.fi/historia

Oscar Software. N.d. English. eCommerce. Read 20.11.2021. https://www.oscar.fi/web/eng/e-commerce

IntentHQ. 2015. What is good code? A scientific definition. Read 17.11.2021. https://intenthq.com/blog/it-audience/what-is-good-code-a-scientific-definition/

Bigcommerce. N.d. What is CSS, and why is it important? Read 15.11.2021. https://www.bigcommerce.com/ecommerce-answers/what-css-and-why-itimportant/

REDAD. 2021. REDAD© has been using Bootstrap Version 5.0.l to customize your website. Read 19.11.2021. https://red-advertising.com/en/official/news/detail/50-REDAD©-has-been-using-BootstrapVersion-50l-to-customize-your-website

Dev. 2021. Setting Up Gulp On Node.js. Read 19.11.2021. https://dev.to/oyedeletemitope/setting-up-gulp-on-node-js-1082

Readwrite. 2013. GitHub For Beginners: Don’t Get Scared, Get Started. Read 19.11.2021. https://readwrite.com/2013/09/30/understanding-github-ajourneyfor-beginners-part-1/

Git-Scm. N.d. 1.2 Getting Started - A Short History of Git. A Short History of Git. Read 21.11.2021. https://git-scm.com/book/en/v2/Getting-Started-A-ShortHistory-of-Git

MDN Web Docs. 2021. https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Organizing

CSS Media Queries n.d. What are CSS Media Queries and how to implement them. Read 21.11.2021. http://cssmediaqueries.com/what-are-css-mediaqueries.html

FutureLearn. 2021. What are HTML and CSS used for? The basics of coding for the web. Read 19.11.2021. https://www.futurelearn.com/info/blog/what-arehtmlcss-basics-of-coding