FED 1
HTML & CSS
Course Assignment

Egil Johnsen Dahle

Foreword

This document contains more than was required in the assignment, and is therefore structured to first contain what the assignment asked for, and then afterwards the rest. The not-required parts where a part of my planning process. I wanted to keep a record of this, which is why I include these parts in this document. This document can be used freely for any educational purposes.

URL to project resources

GitHub repo
https://github.com/LektorDahle/html-css-egil-dahle

GitHub Pages
https://lektordahle.github.io/html-css-egil-dahle/

Figma design
https://www.figma.com/design/LeNq2ENZmtea0SEthGu8CQ/RainyDays?node-id=0-1&p=f

Figma desktop prototype
https://www.figma.com/proto/LeNq2ENZmtea0SEthGu8CQ/RainyDays?node-id=0-1&t=ZExwRDoOcWatV5Ai-1

Figma phone prototype
https://www.figma.com/proto/LeNq2ENZmtea0SEthGu8CQ/RainyDays?node-id=239-1152&t=ZExwRDoOcWatV5Ai-1

HTML & CSS CA Report

Reflection

I have been teaching HTML, CSS, JavaScript, TypeScript, Node.js (with Express), Python, and MySQL for the last four years. This is the first time I have sat down, not to make something look really cool and awesome, but to make something functional and compliant with best practices. The result is probably not fully compliant, and the CSS ended up being quite messy.

My prior experience made me sceptical of the "purely HTML and CSS" requirement of the task, and in the beginning, I considered breaking it. After all, programming (and templating) usually makes the world much simpler. I am glad I decided to follow the task instructions. This gave me an opportunity to learn much more about HTML and CSS, and to test things I had seen or read about, but never really felt the need to use.

This is not my best work, but it represents me at my best - exploring a new world. Here, I am faced with the reality that I, in my life, might rely too much on TypeScript, even for the simplest things. I have never spent so much time researching best practices or reading documentation for HTML and CSS to truly understand what I'm using, rather than just making things work.

If I had more time, I would have spent most of it not completing the site, but refining this document. I have put effort into it so that I can use it in my teaching. Hopefully, it will help both me and my students in the future. Further refining this document would make it clearer how the webpage should be written, and refactoring the HTML and CSS, along with cleaning up the style sheet, would have been a natural extension of this work.

I would never use some of the checkbox hacks you find in the project in actual production. This is the only limitation where I think using JavaScript wouldn't have stopped me from learning - it might even have helped me learn more.

Inspirations & Sources

1. MDN documentation has been used extensively throughout this project to understand and implement many of the features on this page. It is therefore not possible to distinguish which parts were created with help from MDN or to recall exactly when it was used.
https://developer.mozilla.org/en-US/

2. Book: CSS in Depth by Keith J. Grant (ISBN:978-1-61729-345-0)
An “older” book, but with internet it is easy to find something, once you know what to look for; this book helped me understand and gain vocabulary so that I knew what to look for.

3. Book: UI/UX Design for Professionals by Sharanpreet Kaur (ISBN:978-93-481073-8-1)
I have been reading this book during the period I worked on this CA, and know I took inspiration from this, even though it does not teach HTML or CSS.

4. Webpage: https://www.w3schools.com/howto/howto_css_switch.asp
This page helped me make the dark/light switch, hamburger menu and the payment page. Even though I find my implementation “hacky”, it was educational.

5. Webpage: https://www.w3schools.com/css/css_combinators.asp
I have used “>” and “ “ in CSS many times, but this page helped me understand why they work the whey they do. It also made me aware of “+” and “~” combinators. All quite useful.

6. Webpage: https://en.wikipedia.org/wiki/IPhone_(1st_generation)
I decided that this should represent the smallest smartphone anyone might use on my page. Finding extremes can be useful.

7. Webpage: https://javascript.plainenglish.io/the-golden-rule-of-css-layouts-flexbox-or-grid-026e9fef08ed
When to use flexbox or CSS Grid - an attempt to find a best practice, where I argue around information found on this webpage.

8. Webpage: https://www.opengraph.xyz/
Used this to test my Open Graph implementation, to see what it looked like and control that it worked as intended.

9. Webpage: https://validator.w3.org/
Validating my markup.

10. Webpage: https://wave.webaim.org/
Checking WCAG compliance. This did not like some of the “hacks”.

11. AI: ChatGPT co-wrote a lot of the content visitors will see, but not code. I did use ChatGPT as a code reviewer during these weeks, asking questions about my ideas, and some corrections in the code come from input given by ChatGPT. I also have access to GitHub Copilot, but do not use this service. These last years I have become increasingly annoyed at builtin AI that writes code for my students. They learn nothing, and my help holds less value.

Screen-sizes

As a maths-teacher, I have used the document below to describe pixels to my students. This is a style-updated and translated version, where I also included the MDN documentation that I might have referred to when originally creating this resource for my classes.

Theory - Pixel

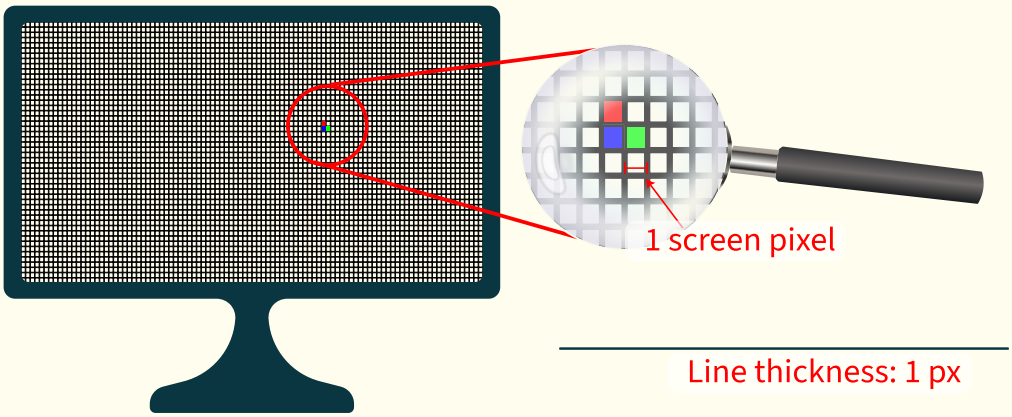
A pixel is often used as a unit of length in web development, especially for the width of elements. It is important to distinguish between *physical screen pixels* and the CSS *unit of length* called a pixel.

Pixel as a variable length:
Technically, a pixel does not have a fixed physical size. It is sometimes defined as thin but visible.

Pixel as a fixed length
In CSS, one pixel is defined as **1/96 of an inch**, which equals exactly 0.75 points (pt) or approximately 0.26 mm.

(https://developer.mozilla.org/en-US/docs/Glossary/CSS_pixel)

There is no direct relationship between the screen's physical pixels and the pixel units used in CSS. Modern browsers scale CSS pixels so that content appears at a consistent visual size across devices.



As we can see, it doesn't make sense to use 4k resolutions like 3840 x 2160 or HD resolutions like 1920x1080 as guide when designing with CSS, for example when setting up @media rules. We have to use the viewport width (using CSS pixels), since this can be measured physically and should be equal between screens with different resolutions or shapes.

Even though a webpage is often designed with a phone version and a desktop version in mind, it makes more sense to build one version that adapts dynamically when needed. The phone version is the desktop version — it simply adjusts when there is less horizontal space.

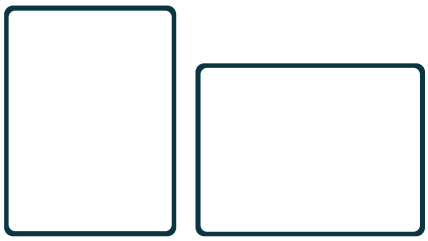
The reverse is also true: the desktop version is the phone version, just with extra space as a luxury.

In order to create a cohesive design, the layout should be studied carefully, and the different hinge points in the design identified. These hinge points mark where the design changes its structure to fit the available space. The illustrations below show how viewport width for phone, tablet and desktop. This is a resource I made and have used in my classes, and I always stress that these values are just guides, not authoritative numbers.

Screen

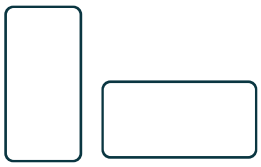
Width: 1280 - 3840 px

Tablet



Width: 600 - 1366 px

Phone



Width: 320-960 px

Font Size

The font-size in the document should adjust some between screen sizes. If we decide a largest and a smallest font-size we use clamp(min, preferred, max) to make it adjust between screen widths. The root em should therefore be written in a way that makes preferred "max out" just before the largest screens and "min out" before the smallest screen widths.

```
:root {
    font-size: clamp(0.8em, 1.5vw, 1.2em);
}
```

0.8em means 0.8 of 16px - the default font size. This is 12.8px.

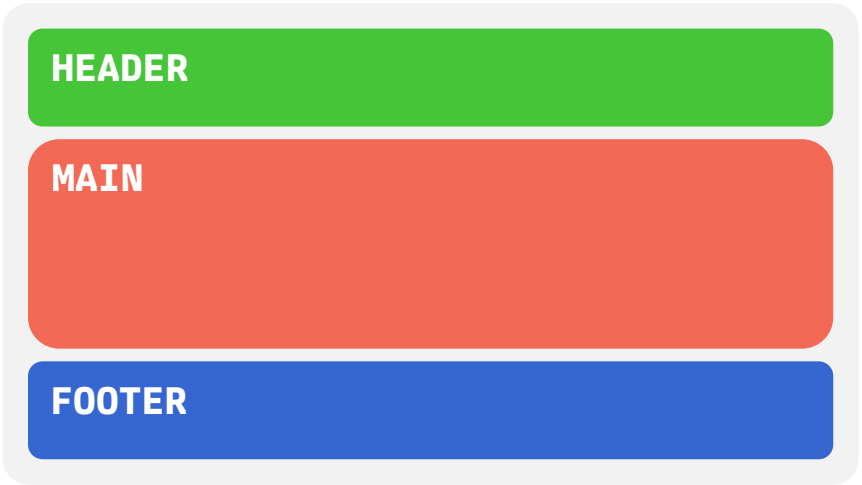
1.2em means the same, and results in 19.2px. These are our max-values.

The 1.5vw means 1.5% of the screen width. The clamp will max out at 1280px view width and min out at 853px view width (1280*0.015=19.2 and 853*0.015=12.8)

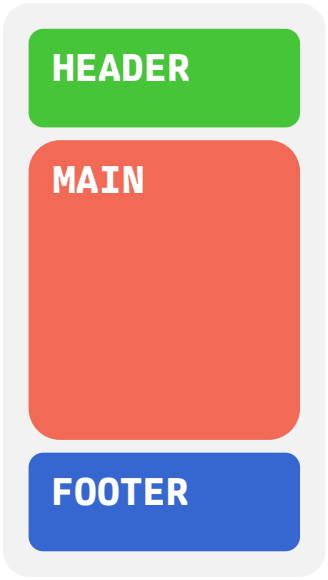
The HTML structure

To study the design, we draw each element as a child placed inside its parent box. This was done in InkScape and I exported these diagrams to PDF so that I could embed them in this document (Affinity Publisher, and Word for that matter, does not support SVG properly).

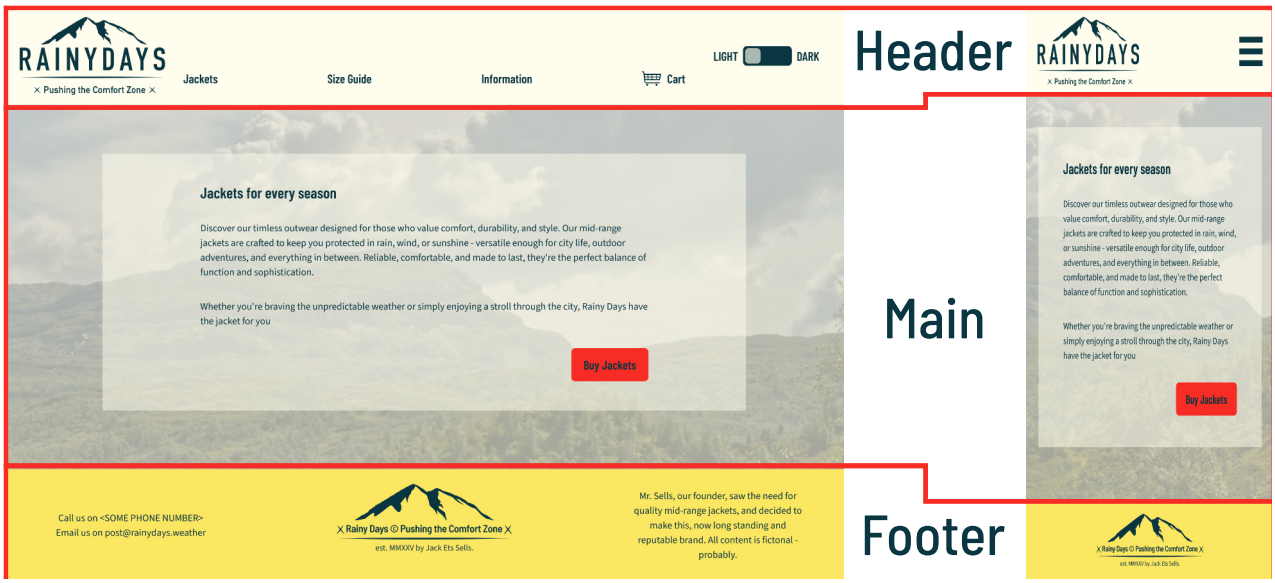
When doing this, it becomes obvious what happens at the hinge points. The most significant change is that the navigation menu is compressed into a hamburger menu when space becomes limited.



DESKTOP

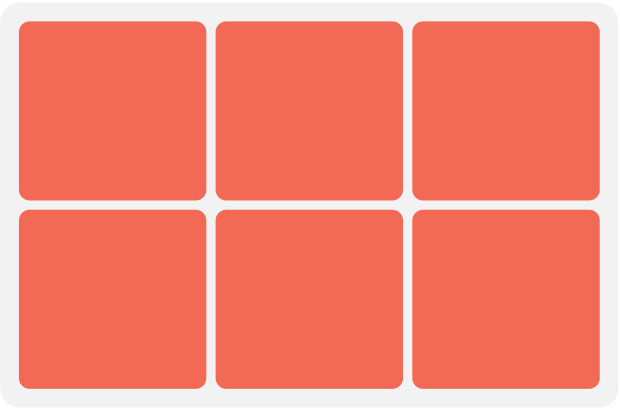


PHONE



CSS Grid vs. Flexbox

This is the time to discuss when to use display: grid and when to use display: flex. I've had the privilege of working closely with highly skilled, self-taught web-developers. They taught me to use grid on the base page layout (left figure), and to keep using flex on layouts illustrated to the right - even when grid is tempting here.



The body can be built quite easily with grids, and it will work in predictable ways. We do have two main choices - one that I found online as "best practice" and one that also works quite well. The best practice uses "auto 1fr auto" for template rows, and the other uses "10rem auto 10rem". The difference is whether it is the child or the parent that decides the height of the children. In the first, the children scale to what is necessary for them, in the second we fix their height, for example at 10 rem.

```
<body>
    <header>Some content</header>
    <main>Some content</header>
    <footer>Some content</footer>
</body>
```

To the left we see the HTML-structure in question. Below we see the two different methods for structuring the body; what is needed to make these methods work.

```
body {
  min-height: 100dvh;
  display: grid;
  grid-template-rows : 10rem auto 10rem /* header, main, footer */
}

main {
  min-height: 100%;
}
```

Method 1

Method 2

```
body {
  min-height: 100dvh;
  display: grid;
  grid-template-rows : auto 1fr auto /* header, main, footer */
}
```

But what is best practice? I honestly don't know, and often self-taught developers find good solutions, and then stick with them, even if there is better approaches exist. So I then decided to use Google to explore other developers' thoughts and ideas about this.

I found this article: https://javascript.plainenglish.io/the-golden-rule-of-css-layouts-flexbox-or-grid-026e9fef08ed, which offered clear advice with solid reasoning behind it. I chose to use the two tables from this page, but apply my own reasoning for my choices. What I found on this page suggests that the practices I've been using during my four years of teaching HTML and CSS are in fact sound.

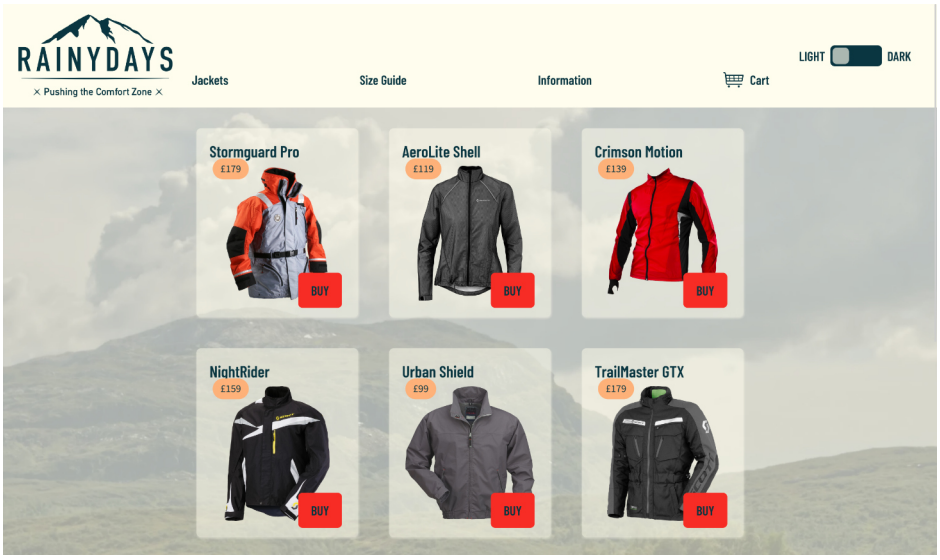
Flexbox (1-Dimentional)	CSS Grid (2-Dimentional)
Perfect for components & layouts in one direction	Complex layouts with rows AND columns
Navigation bars and menus	Page layouts (header, sidebar, main, footer)
Centering content	Image galleries
Equal-height cards	Card layouts with different sizes
Form controls and alignment	Magazine-style layouts
Content-based sizing	Grid-based designs

Choose Flexbox For:	Choose CSS Grid For:
Navigation menus	Page layouts
Button groups	Image galleries*
Media objects (image + text)	Magazine layouts
Centering content	Dashboard interfaces
Distributing space between items	Complex card arrangements
Form control alignment	Responsive layouts
Card components	Calendar layouts*

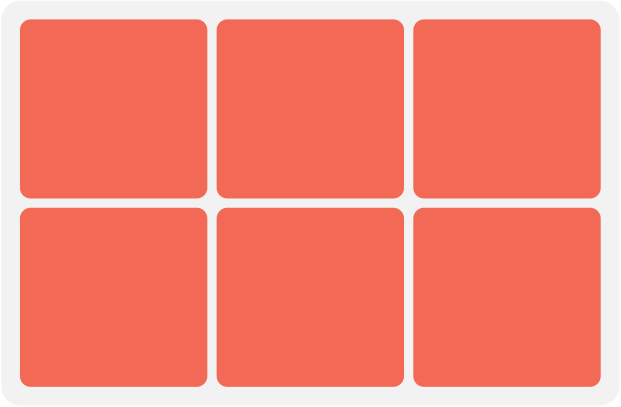
I marked the red * in the tables. This summer I built a media server whit a corresponding webpage for my images, so that family members can access them. I found the Flexbox much better suited for this, since an image gallery is typically a one-dimensional list of images that simply wraps inside its container.

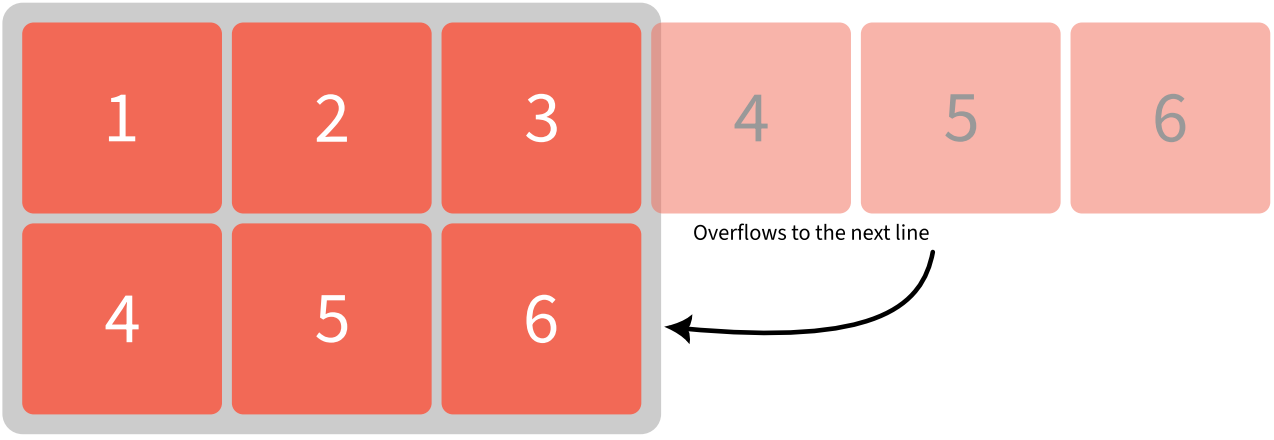
I had a similar experience when I made my own calendar. I'm not arguing that these lists are wrong, just that Grid is alluring in its power, and we should aim to use the simplest solution for each case. The calendar that I made was simple and had only one function: a record of all my plans and reflections for each class I taught, allowing me to track my progress and improve upon the next year.

Why is this one-dimensional?



This is a list. Lists are inherently one-dimensional. They flow in one direction, and when combined with max-width rules and wrapping, they may appear as a grid. However, looking like a grid is not the same as being a grid in HTML and CSS.



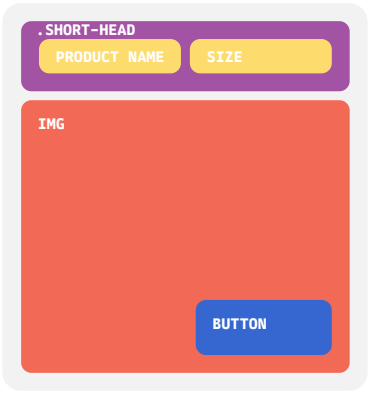


Both grid and flexbox can be written to ensures a responsive design without @media rules in this case. Since the layout is one-dimensional, and because this is the approach I'm accustomed to using, I'll apply flexbox here.

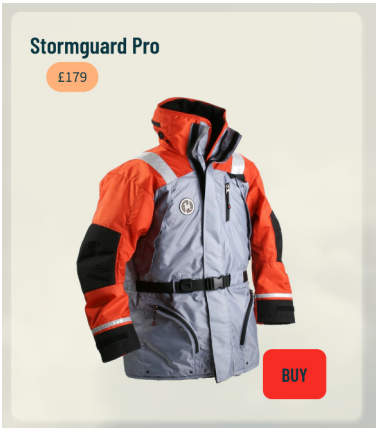
The entire structure plan

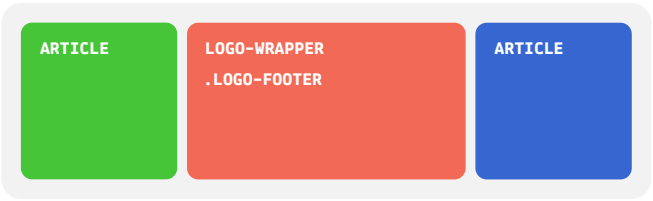
I drew up (in InkScape) the HTML structure for the entire webpage. Below are all the sketches not shown so far. These where instrumental in building the site.

With more time I would have reworked these, and gone through the HTML and CSS to ensure these where followed. This would have resulted in a more consistent naming strategy and ensured no unnecessary code.

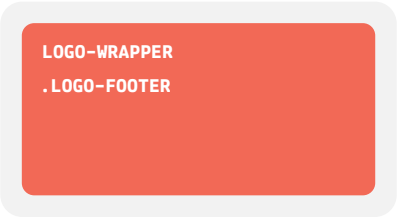


CONTENT SHORT



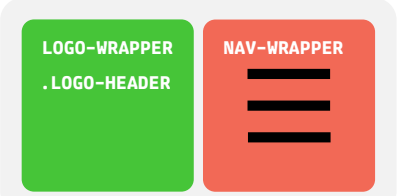


FOOTER - DESKTOP

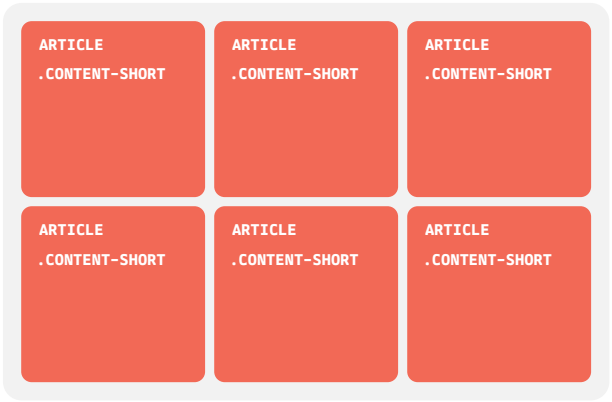


FOOTER - PHONE

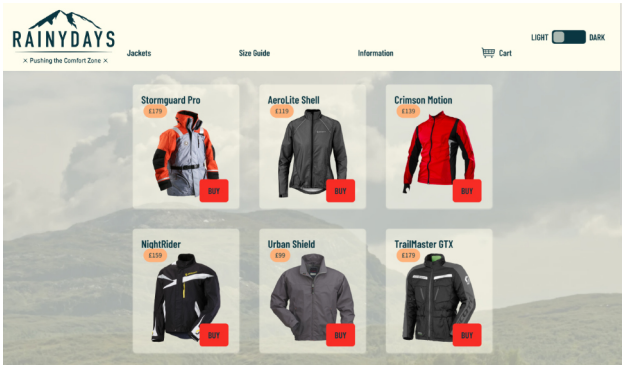
HEADER - DESKTOP



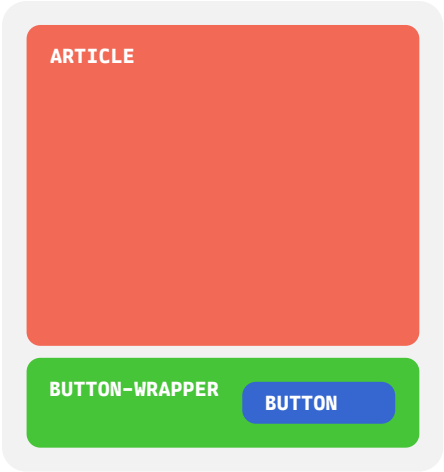
HEADER - PHONE



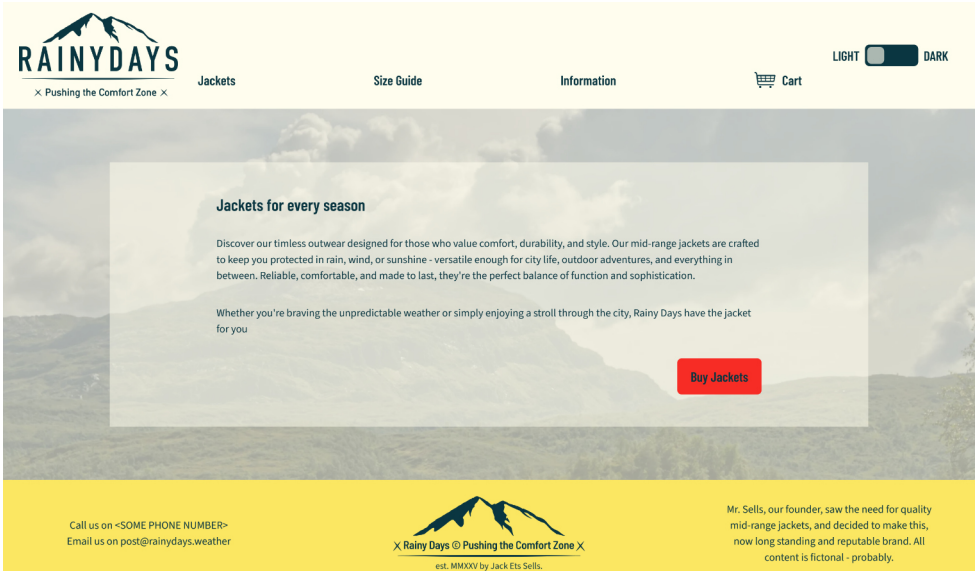
MAIN - CONTENT OVERVIEW - DESKTOP

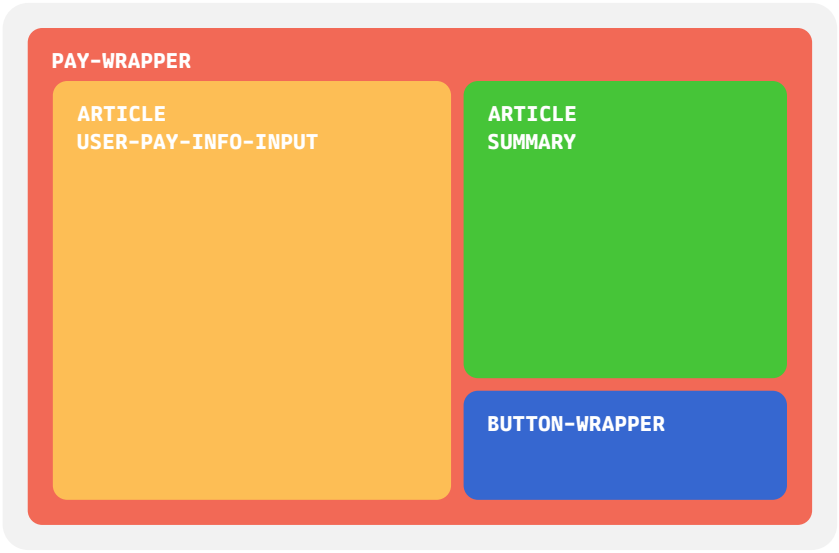


MAIN - DESKTOP

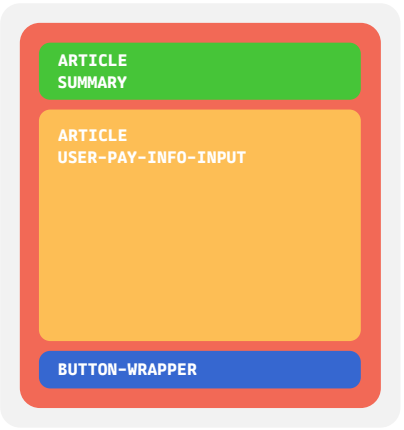


MAIN - PHONE

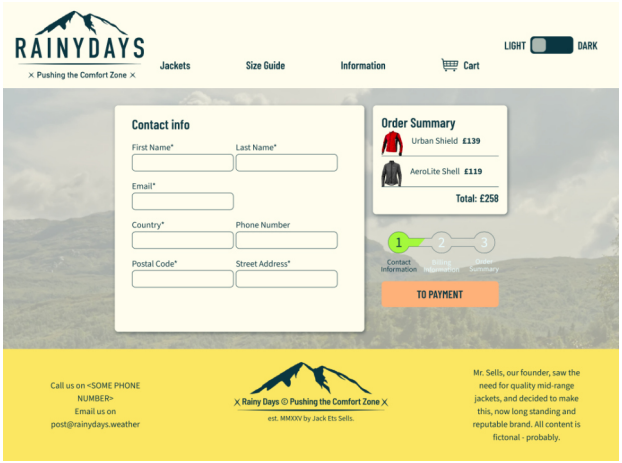


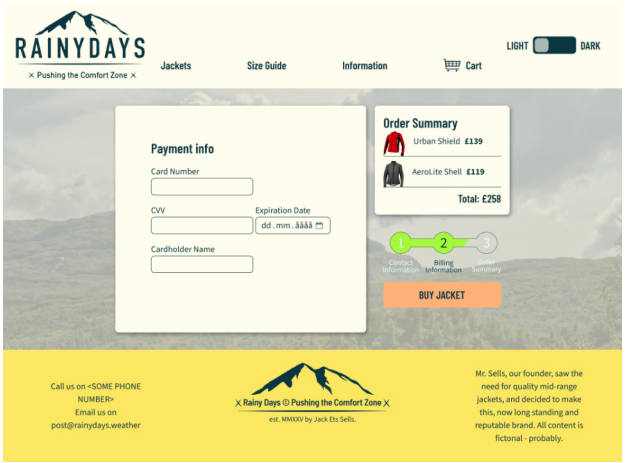


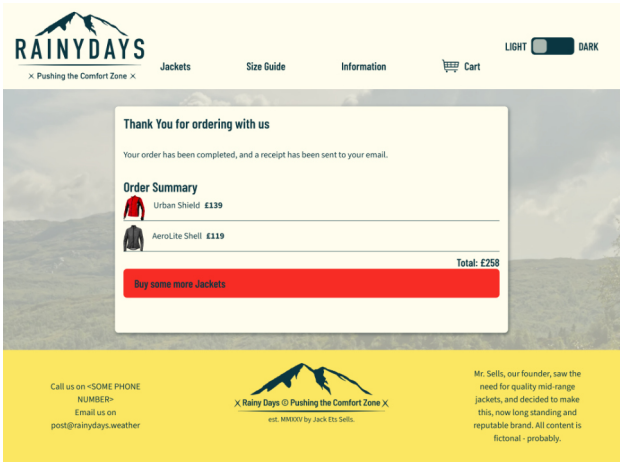
MAIN - PAYMENT - DESKTOP

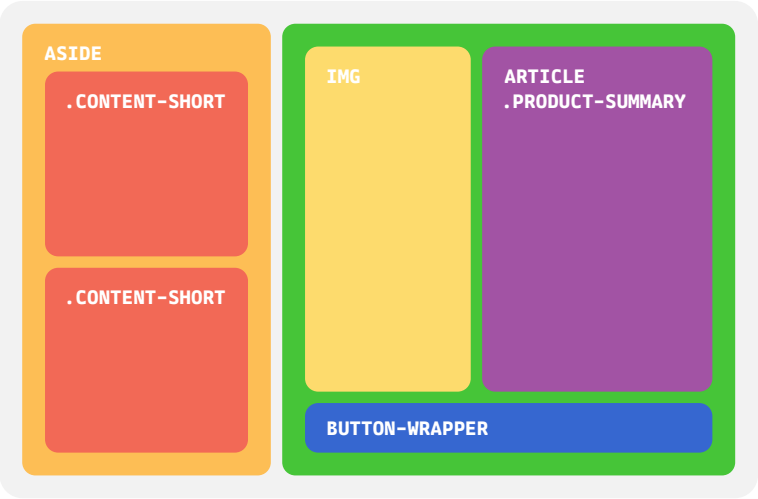


MAIN - PAYMENT - PHONE

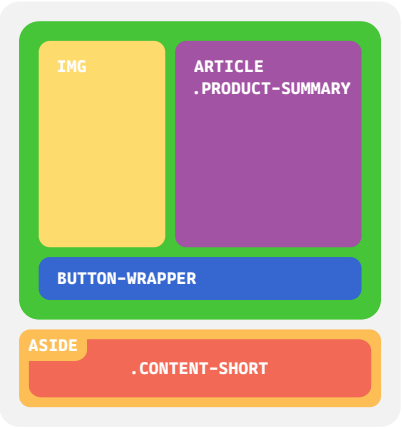




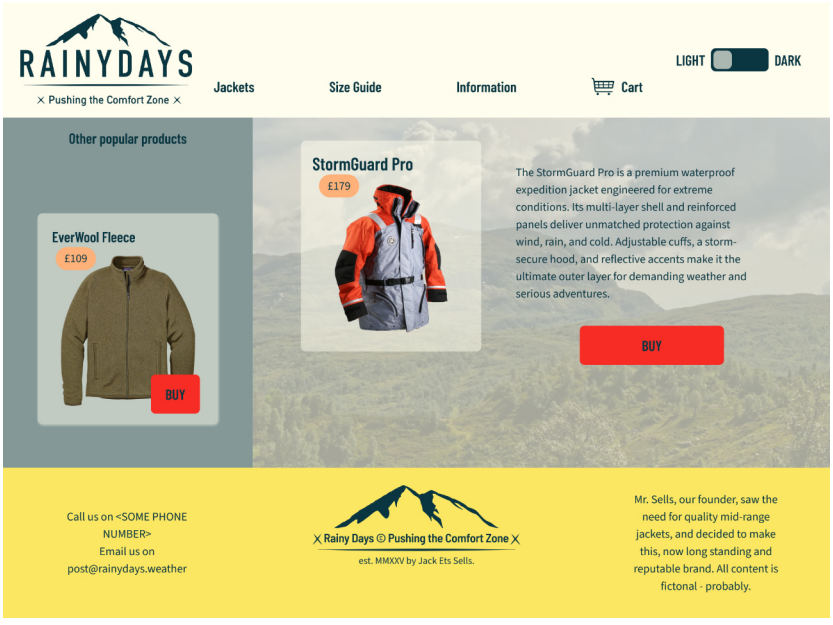




MAIN - PRODUCT DETAIL - DESKTOP



MAIN
PRODUCT DETAIL
PHONE



Working Tree

LektorDahle.github.io/html-css-egil-dahle

.git	The git-repositiory base
.vscode	Rules for VSCode, spesifically to prevent formatting of svg code
_docs	My notes and sketches during this project
about-us	The about-us page
content-overview	The content page that displays all the jackets on sale
images	The images used on the site.
payment	The payment-page
product	The spesific product pages show extended jacket information
sizes	The size guide
styles	All the CSS files
.gitignore	The files not to be included in version controll
index.html	The home page of Rainy Days
LICENSE	The license file
README.md	The markdown readme file