Noroff
School of technology
and digital media

Project Exam 1

Torbjørn Haukås

Word count

Introduction

I want to preface this report to explain why I haven't checked in on Wednesdays during this exam. The truth is, I didn't really start working on this project until the last week. The first week I did do a few small things, like trying out some designs, drawing things on post-it notes etc, but eventually I just stopped and slacked off for 3 weeks. I just got stuck while figuring out what theme to go with and procrastinated. So yeah, I just didn't have anything to check in with, so I just forgot about it. This is not an excuse, just bowing my head in shame.

As I did all of this in 10 days (I started working on it the 20th of May), my work process has been less thorough than it was on the semester project in December. I don't have a long-winded explanation for every little piece of my design, I just sort of did things on a whim. Also, at the time of writing this the deadline for this project is in a mere 3 hours. As such, this report will be brief. I'll try to only cover the most important aspects of my design process, technical process and optimization process.

Here's the link to my Figma workspace:
https://www.figma.com/file/hbYKh8DGnnhZQDQDipOYbh/Untitled
And my Github repo for good measure:
https://github.com/Noroff-FEU-Assignments/project-exam-1-Hawkas
Netlify:
https://dreamy-shaw-314388.netlify.app/index.html

Design

What went well on the project

This time around, I skipped ahead in the wireframing stage of the planning process and went straight for a high-fidelity prototype. In my opinion, this actually worked out quite well for this particular project, given how simple the layout of a blog really is.

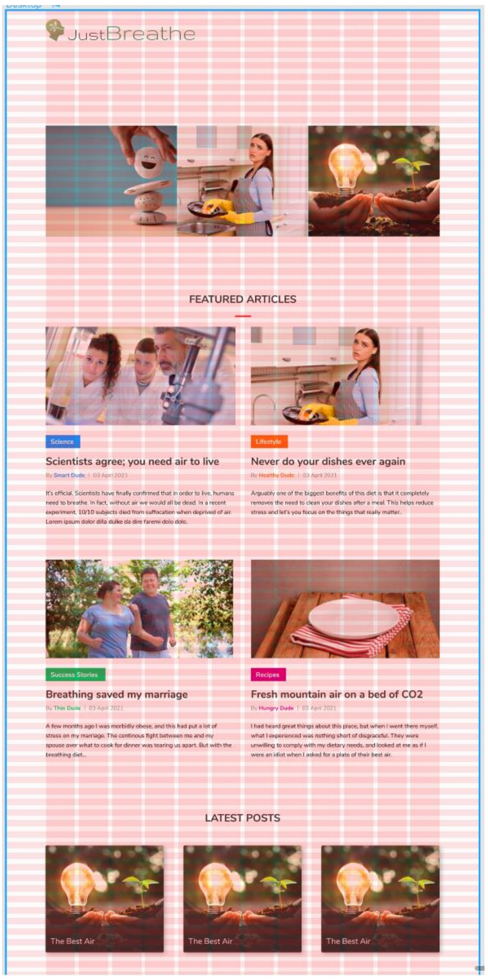
I went mostly by intuition in terms of the overall content distribution and layout, and used most of the same concepts I did in the previous semester project. I did not use other sites for inspiration. Given the theme of my blog is something I know rather well and have seen so much of already, I didn't need to.

For my homepage, I would have a header/landing area that hopefully gives a good first impression of the site. Here I'd also add a CTA that leads to the main content of the website, in this case the 'blogs' page. Further down the page I'd have some featured articles with excerpts, and below that the carousel for the latest posts.

I spent the most time designing this page as per usual. All the other pages were done within a few hours, while I spent over an entire day on the homepage. I made sure to have good contrasting colors wherever appropriate, and paid attention to my text in terms of visual hierarchy and general readability. Designing the hero banner took up a lot of time, as I had a

hard time deciding on what to go with. I tried many different approaches, which can all be seen in my Figma workspace.
My overall workprocess here, in order, was as follows:

1. Overall site layout. I laid out boxes and lined text and boxes up against layout grids and 12 column layouts. Text is not aligned on the image below. I hadn't done that yet at the time.



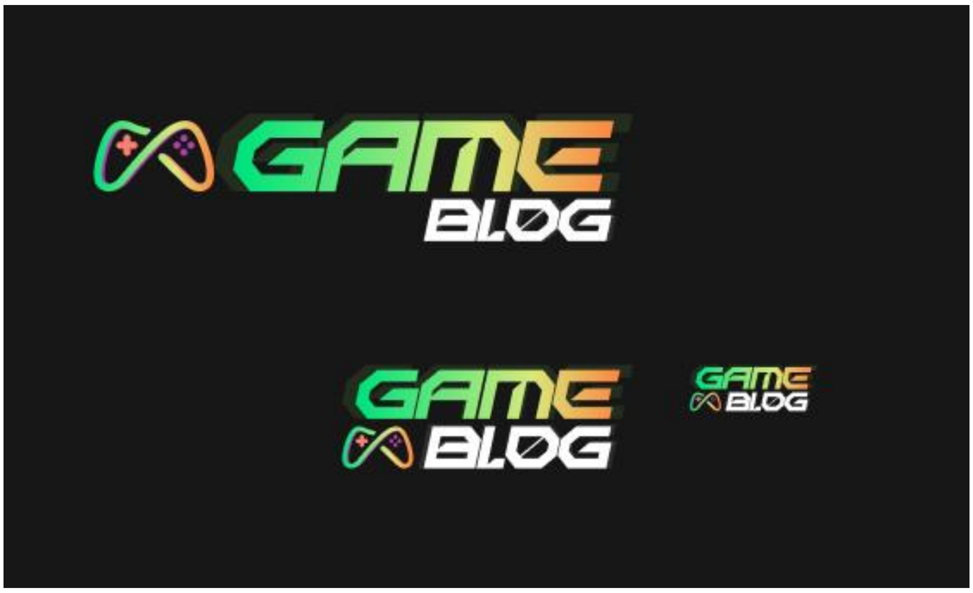
Scientists agree; you need air to live
By Smart Dude | 03 April 2021
It's official. Scientists have finally confirmed that in order to live, humans need to breathe. In fact, without air we would all be dead. In a recent experiment, 10/10 subjects died from suffocation when deprived of air. Lorem ipsum dolor dilla dulke da dire faremi dolo dolo.

Never do your dishes ever again
By Healthy Dude | 03 April 2021
Arguably one of the biggest benefits of this diet is that it completely removes the need to clean your dishes after a meal. This helps reduce stress and let's you focus on the things that really matter..

Breathing saved my marriage
By Thin Dude | 03 April 2021
A few months ago I was morbidly obese, and this had put a lot of stress on my marriage. The continous fight between me and my spouse over what to cook for dinner was tearing us apart. But with the breathing diet..

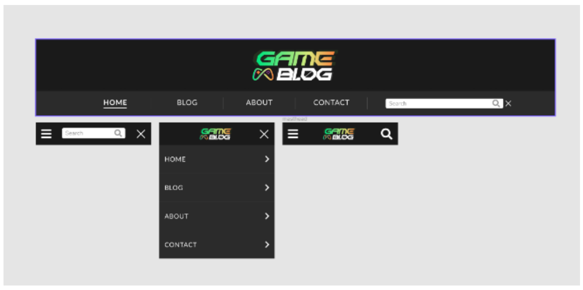
Fresh mountain air on a bed of CO2
By Hungry Dude | 03 April 2021
I had heard great things about this place, but when I went there myself, what I experienced was nothing short of disgraceful. They were unwilling to comply with my dietary needs, and looked at me as if I were an idiot when I asked for a plate of their best air.

2. Logo design. I found a vector alphabet on Adobe Stocks and aligned the letters the way I wanted them to be in illustrator. I duplicated the letters, made them opaque and upscaled

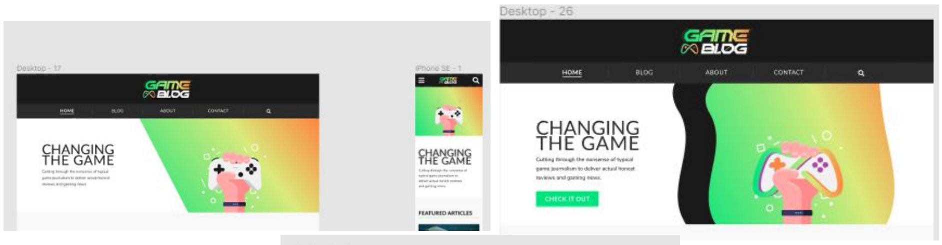


them to add 3D effect to the text. I then threw in another vector of a gamepad in the corner and called it a day.

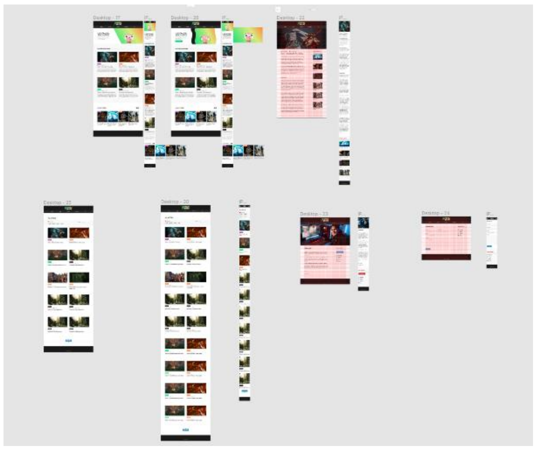
3. Logo and navigation layout at its most basic, deciding how to structure it both on mobile devices and laptops. I also picked my colors for the site at this stage. I wanted some bright, energizing colors, and picked a couple at random using https://www.coolors.co



4. Hero banner design, which is something I value greatly. I want to give a good first impression, so I put in a lot of extra effort to polish it.

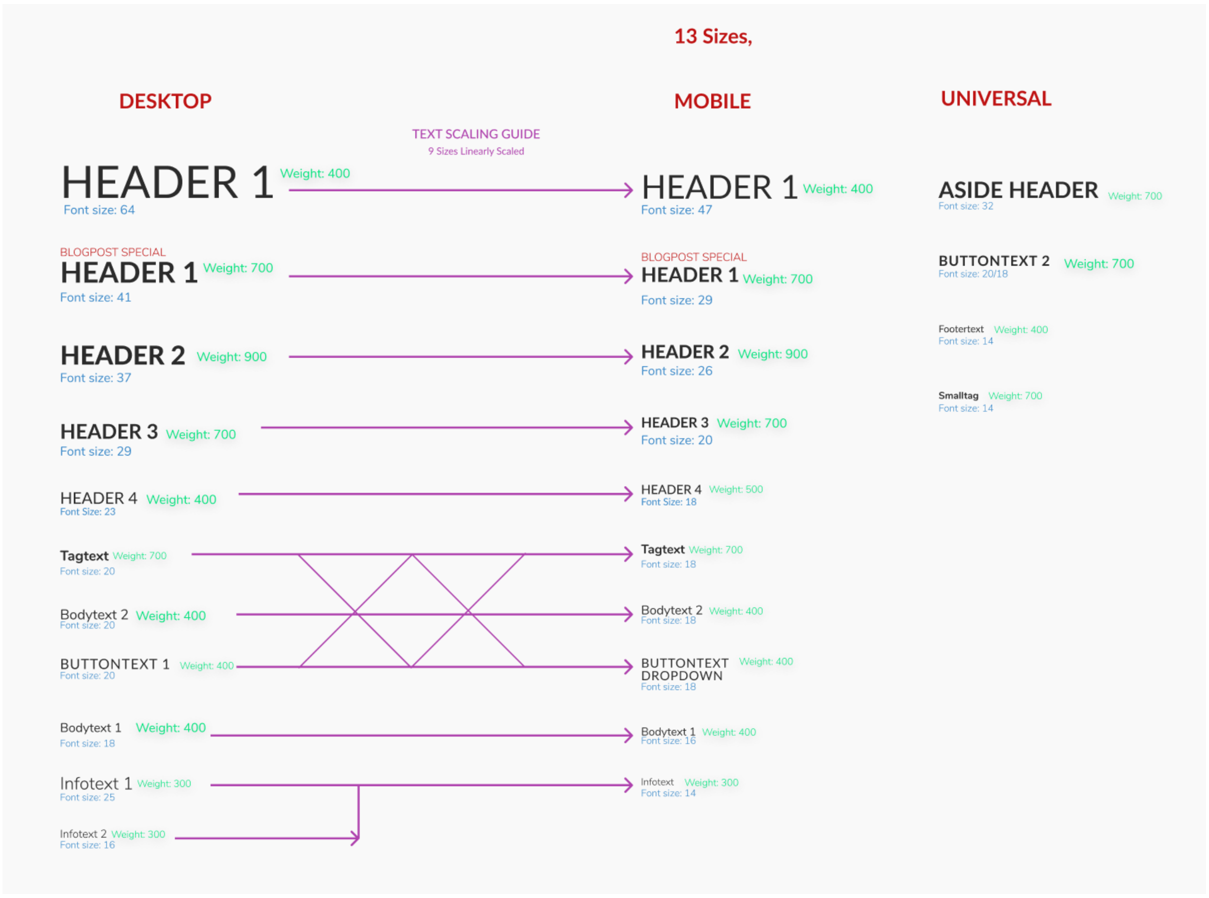


5. The rest of the website. Very little thought went into the remaining pages. I finished designing them for mobile and desktop in around 4-6 hours, give or take. I re-use my components across my entire site. I'd say the design is very much 'atomic'.



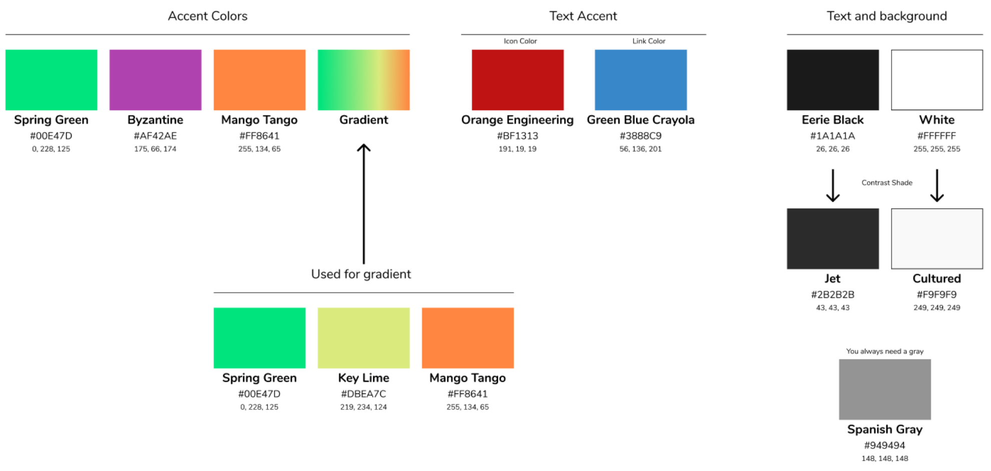
I only decided on fonts at the end, and ended up going with the sharper sans-serif, Lato, for my headers. To contrast this, I went with the more rounded sans-serif Nunito for my body text. I then compiled all my sizes into a guide so I could easily make variables in CSS and sort out how many font-sizes, weights, etc I'd need to account for.

Do note that some of these values were changed later on to reduce the number of sizes.



13 Sizes,
DESKTOP
MOBILE
UNIVERSAL
TEXT SCALING GUIDE
9 Sizes Linearly Scaled
HEADER 1 Weight: 400 Font size: 64
BLOGPOST SPECIAL HEADER 1 Weight: 700 Font size: 41
HEADER 2 Weight: 900 Font size: 37
HEADER 3 Weight: 700 Font size: 29
HEADER 4 Weight: 400 Font Size: 23
Tagtext Weight: 700 Font size: 20
Bodytext 2 Weight: 400 Font size: 20
BUTTONTEXT 1 Weight: 400 Font size: 20
Bodytext 1 Weight: 400 Font size: 18
Infotext 1 Weight: 300 Font size: 25
Infotext 2 Weight: 300 Font size: 16
HEADER 1 Weight: 400 Font size: 47
BLOGPOST SPECIAL HEADER 1 Weight: 700 Font size: 29
HEADER 2 Weight: 900 Font size: 26
HEADER 3 Weight: 700 Font size: 20
HEADER 4 Weight: 500 Font Size: 18
Tagtext Weight: 700 Font size: 18
Bodytext 2 Weight: 400 Font size: 18
BUTTONTEXT DROPDOWN Weight: 400 Font size: 18
Bodytext 1 Weight: 400 Font size: 16
Infotext Weight: 300 Font size: 14
ASIDE HEADER Weight: 700 Font size: 32
BUTTONTEXT 2 Weight: 700 Font size: 20/18
Footertext Weight: 400 Font size: 14
Smalltag Weight: 700 Font size: 14

I also laid out my choice of colors in the same manner:



Overall, the entire design process lasted 2 days. Which, when compared to the last time around where I spent 2 weeks fiddling with colors and texts, is a major improvement I'd say.

Design has become more intuitive to me in terms of colors, layout, text and images. I don't need to consider every single choice I make as much which saves me so much time.

After building my design I sent it to some friends & family and told them to nitpick it to death. I ended up changing several things in my design according to their feedback.

What was difficult/didn't go well on the project

Deciding on a theme was particularly hard for me. My first ideas for a theme were ridiculously stupid; a blog for pet rock owners, a lifestyle/fad diet blog for a 'breathing' diet. Those are just the few I actually started building for, I had a long list of other stupid ideas.

I really struggle to engage myself in my work when the purpose of the website is my own idea and not based on someone else's needs.

Eventually I ended up going with something really generic that I know well; video games. Which in hindsight is so boring I'm rather disappointed in myself. I wanted to pick something more intriguing.

There were also quite a few measurement errors done during design that I had to correct during the development stage, but that's almost to be expected given how rushed this whole process was.

What would you do differently next time

Firstly I would use the time that I've been given efficiently and not bumrush everything the last week of the exam. Secondly I would pick a theme that doesn't make me feel like the most boring person on earth. And thirdly, I'd probably go back to dark mode. I wanted to go with a white backdrop this time around to differentiate it from my earlier projects, but I honestly just have a personal preference for dark-mode.

Technical

What went well on the project

Okay, now there's a lot to go over but I'll just summarize it for each part of the development, in order of HTML/CSS, then Javascript.

HTML/CSS

I started out by throwing in the usual CSS Resets/Normalizers, listing out my variables for colors, fonts, and sidepadding/margins. I stuck with the BEM naming convention for my classes, and as I already knew from the last project how I should structure my HTML to get it the way I want it, this part went smoothly. I also went straight out the gate using the proper semantic terms for each element.

When it came to styling in CSS it was a bit rough to start with given how easily you forget the ten-thousand different CSS properties, but once I got into the rhythm of it things fell into place rather quickly. I prefer having my media queries near the relevant declarations rather than all in one big query as it's easier to maintain and work with that way.

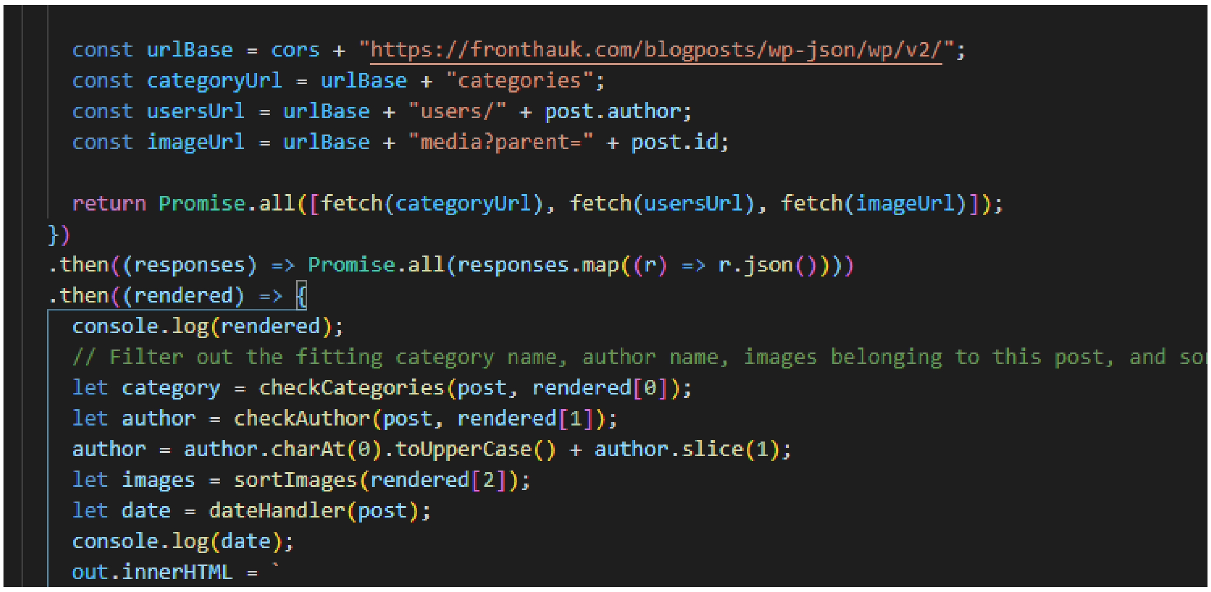
Since we now know JavaScript I could do more fun stuff with the navigation than previously, so I tried animating my bars. It looks pretty good I'd say.

After I'd finished my hero banner and looked at it for way too long, I decided to flip it over to something else entirely. So I made the background of it transparent, sliced a wavy curve around it, threw in an animated gradient along with a patterned background, and faded all of this with a transparent white overlay behind the SVG to give it an extra 3D effect.

Like during the design process, the homepage took up most of my time. As soon as it was done, I had the components for all the other pages at hand, so it was pretty much just copying and pasting.

JavaScript – API Handling

As we'd already done this sort of thing several times before, it wasn't much of a hurdle. I made sure my functions were what I consider readable, though comprehending others code is usually a struggle anyway so what do I know. I also added categories to the query string to show articles belonging to the same category as the article being read in the sidebar.
I did get a bit ahead of myself though and set up fetches for category, images and authors in a Promise.all based upon the ID in the querystring.



```
    const urlBase = cors + "https://fronthauk.com/blogposts/wp-json/wp/v2/";
    const categoryUrl = urlBase + "categories";
    const usersUrl = urlBase + "users/" + post.author;
    const imageUrl = urlBase + "media?parent=" + post.id;

    return Promise.all([fetch(categoryUrl), fetch(usersUrl), fetch(imageUrl)]);
  })
  .then((responses) => Promise.all(responses.map((r) => r.json())))
  .then((rendered) => {
    console.log(rendered);
    // Filter out the fitting category name, author name, images belonging to this post, and so
    let category = checkCategories(post, rendered[0]);
    let author = checkAuthor(post, rendered[1]);
    author = author.charAt(0).toUpperCase() + author.slice(1);
    let images = sortImages(rendered[2]);
    let date = dateHandler(post);
    console.log(date);
    out.innerHTML = `
```

I felt there had to be a better way since making that many HTTP requests for so little seemed a bit extreme. So I pulled up the REST API specs and discovered embeds, and felt like a complete idiot for not going there sooner.

I then modified my code to use embeds instead, and only set up an additional fetch on my posts page to acquire attached media files I'd uploaded to each post for better scaling images on any screen width.

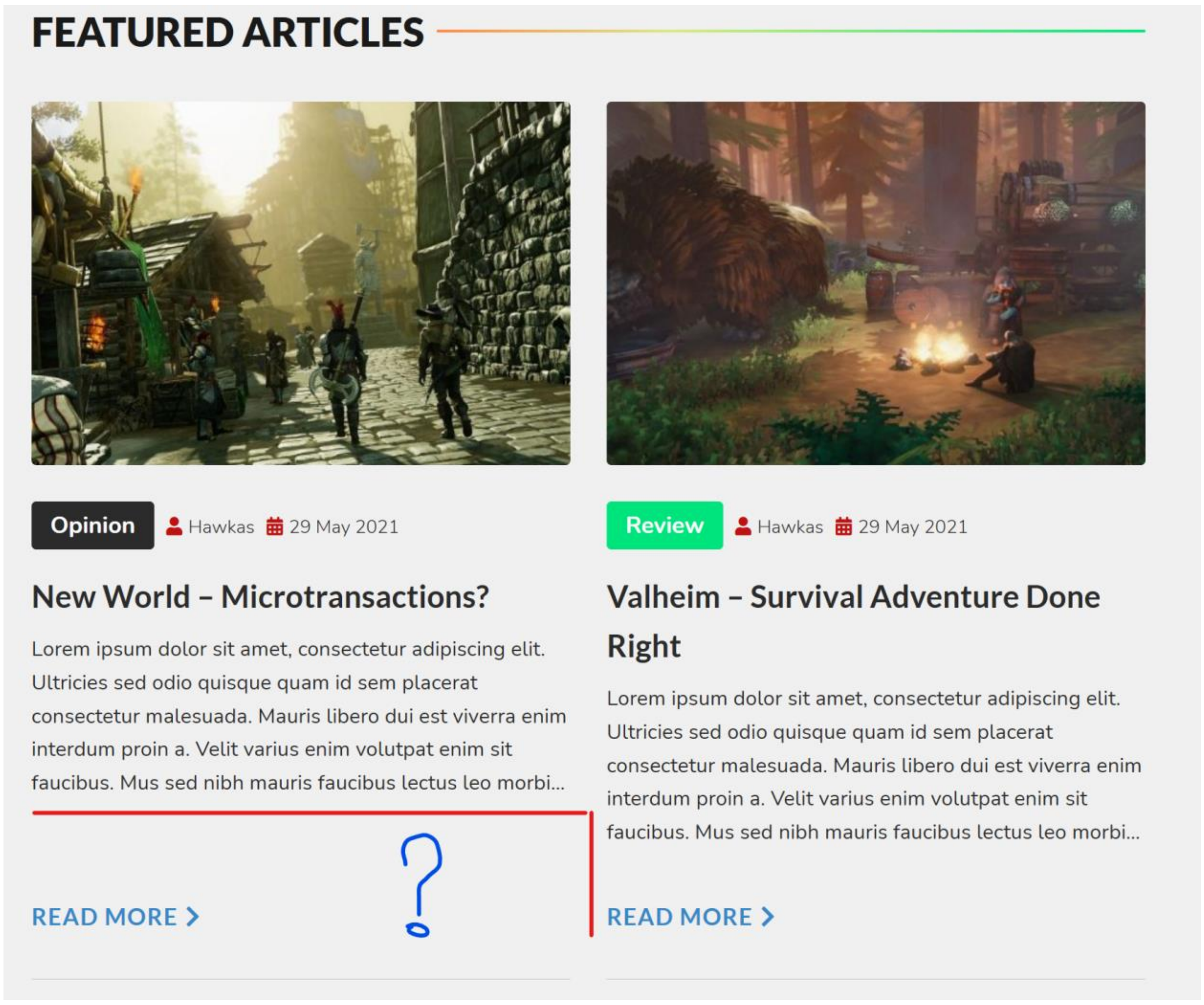
I also added some level 2 features, namely sort and filter. The filter currently removes all posts and adds them back in based on the category being filtered, while the sort uses datasets with the time and title to sort the DOM elements directly rather than rewriting them. I would want to do this with the filter functions as well, but as I only got around to doing it like this towards the end, I didn't have the time for it.

Javascript – Form Validation

This also went well. I set up the validation as requested, and instead of updating it on input as it would be annoying to have the box scream at you as soon as you start typing, I changed it to only validate on 'blur', but to correct itself once the input validates. Mostly just a copy/paste from the JS1 CA.

What was difficult/didn't go well on the project

HTML / CSS



FEATURED ARTICLES

Opinion Hawkas 29 May 2021

New World – Microtransactions?

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ultricies sed odio quisque quam id sem placerat consectetur malesuada. Mauris libero dui est viverra enim interdum proin a. Velit varius enim volutpat enim sit faucibus. Mus sed nibh mauris faucibus lectus leo morbi...

READ MORE >

?

Review Hawkas 29 May 2021

Valheim – Survival Adventure Done Right

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ultricies sed odio quisque quam id sem placerat consectetur malesuada. Mauris libero dui est viverra enim interdum proin a. Velit varius enim volutpat enim sit faucibus. Mus sed nibh mauris faucibus lectus leo morbi...

READ MORE >

A minor annoyance at best maybe, but I spent way too much time trying to figure out how to align these boxes properly without the titles wrapping making it look silly when lined up against each other. I wish there was a rule of thumb to follow on this matter, but I couldn't find anything.

JavaScript – The Carousel

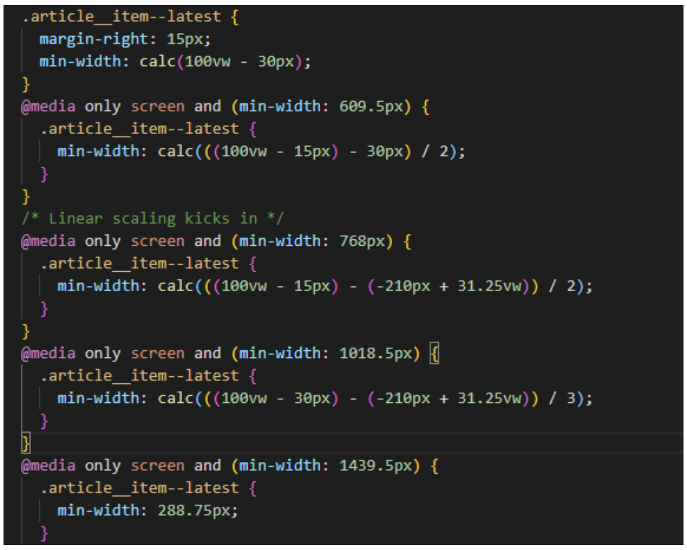
As I was short on time, I made sure to first tackle the one thing that could disrupt my entire workflow, the carousel. I had no idea how to make one after all, and as I didn't know how long it would take, I knew it had to be my first objective in JavaScript.

I figured you would just put all the content in a horizontal flexbox absolutely positioned and set to overflow then slide it with position or transform using JavaScript. I wasn’t certain this was how you should go about it though, so I did some research.

I must’ve used the wrong search terms, because I struggled to find carousels with multiple items visible at once. Most of what I found were single column carousels, but they followed the same idea I had of sliding an absolutely positioned overflowing container. I was planning on having multiple elements at a time in mine though, so I had to do it differently. I first attempted sizing it the same way you would a single item gallery or carousel, but then fitting multiple items within that single container instead, and just swapping between each box.

This was obviously rather rigid, and it didn’t feel good. It was also poor semantically, as ideally, you’d want all these items in the same unordered list. So instead of having each slide be in seperate containers, I tried to slide the carousel by transforming it using the width of the visible container, multiplied incrementally each time. So width x 1 > width x 2 and so on.

The carousel would now slide properly, but I had trouble making the carousel scale responsively. To avoid having clipped boxes in the visible portion of the carousel I had to make my items scale properly. I tried doing it through JS by scaling them to a percentage of the visible container, but while this worked, it wasn't a very elegant solution. I eventually had the idea of scaling each element by the viewport width instead(((100vw - margins) - sidepadding) / columns) with the amount of columns changing in media queries.

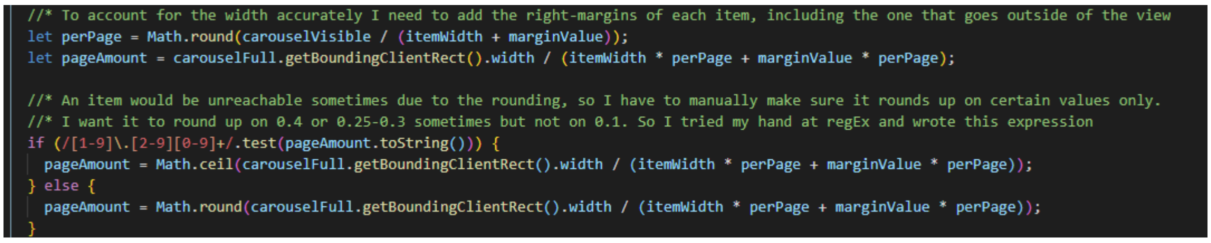


```css
.article__item--latest {
  margin-right: 15px;
  min-width: calc(100vw - 30px);
}
@media only screen and (min-width: 609.5px) {
  .article__item--latest {
    min-width: calc(((100vw - 15px) - 30px) / 2);
  }
}
/* Linear scaling kicks in */
@media only screen and (min-width: 768px) {
  .article__item--latest {
    min-width: calc(((100vw - 15px) - (-210px + 31.25vw)) / 2);
  }
}
@media only screen and (min-width: 1018.5px) {
  .article__item--latest {
    min-width: calc(((100vw - 30px) - (-210px + 31.25vw)) / 3);
  }
}
@media only screen and (min-width: 1439.5px) {
  .article__item--latest {
    min-width: 288.75px;
  }
```

After making the items within scale responsively, it still didn't work quite as I had hoped due to the way my layout scales. It was still clipping the items within after 1 or 2 slides when shrinking the browser window outside of responsive mode, which I found strange.

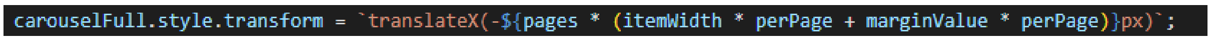
So I experimented with a bunch of different solutions and changed my formula for sliding the carousel to use the width of all items present + their right margins, multiplied incrementally as before, and it all worked smoothly.

So I had to calculate the amount of columns for any screen size, which was a tad bit overcomplicated, but it made setting up a page counter and macro navigation system later on quite a lot easier.



```
//* To account for the width accurately I need to add the right-margins of each item, including the one that goes outside of the view
let perPage = Math.round(carouselVisible / (itemWidth + marginValue));
let pageAmount = carouselFull.getBoundingClientRect().width / (itemWidth * perPage + marginValue * perPage);

//* An item would be unreachable sometimes due to the rounding, so I have to manually make sure it rounds up on certain values only.
//* I want it to round up on 0.4 or 0.25-0.3 sometimes but not on 0.1. So I tried my hand at regEx and wrote this expression
if (/[1-9]\.[2-9][0-9]+/.test(pageAmount.toString())) {
  pageAmount = Math.ceil(carouselFull.getBoundingClientRect().width / (itemWidth * perPage + marginValue * perPage));
} else {
  pageAmount = Math.round(carouselFull.getBoundingClientRect().width / (itemWidth * perPage + marginValue * perPage));
}
```



```
carouselFull.style.transform = `translateX(-${pages * (itemWidth * perPage + marginValue * perPage)}px)`;
```

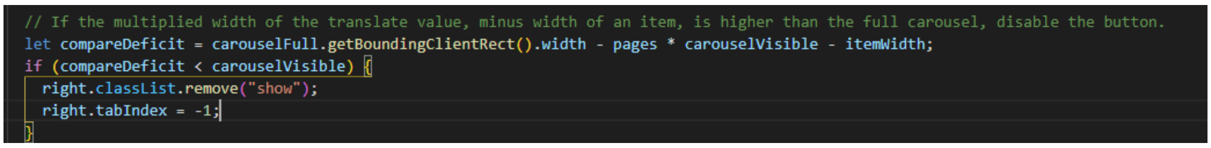
But there was another problem. The carousel would just keep going, I had to somehow stop it. Math has never been my strongest suit, so figuring this out was a bit harder than it should have been. Disabling the left-scrolling button was easy, I just had it disabled whenever the 'page' count was at 0. But the right-scrolling button on the other hand had to be scaled to the width of the overflowing carousel container to work with an infinite number of items.

I found this CodePen: https://codepen.io/agrayson/pen/BodYjx

It is written in JQuery rather than vanilla JS and does things quite differently to my own approach, but the way he controls the buttons was what gave me the idea of how it should be done.

By comparing the value the carousel is being translated with to the overflowing full-width of the carousel, I could more or less accurately determine when it reached the end. I use the full width of the carousel itself and subtract the current translateX value every time the carousel is moved to check the deficit between the two. If the deficit between this value and the full width of the carousel is lower than the visible width, i.e it doesn't have any content that goes beyond what's currently in view, the button is disabled.

There was a slight problem, however. The trigger was exact to the point where if the carousel extended just a few pixels beyond this threshold it wouldn't trigger, so it would still let you go one slide too far in some cases. Due to the margins of each item, the full carousel was just a few pixels wider when the last slide had all its columns filled. To account for this, I simply subtracted the width of an individual item from the deficit.

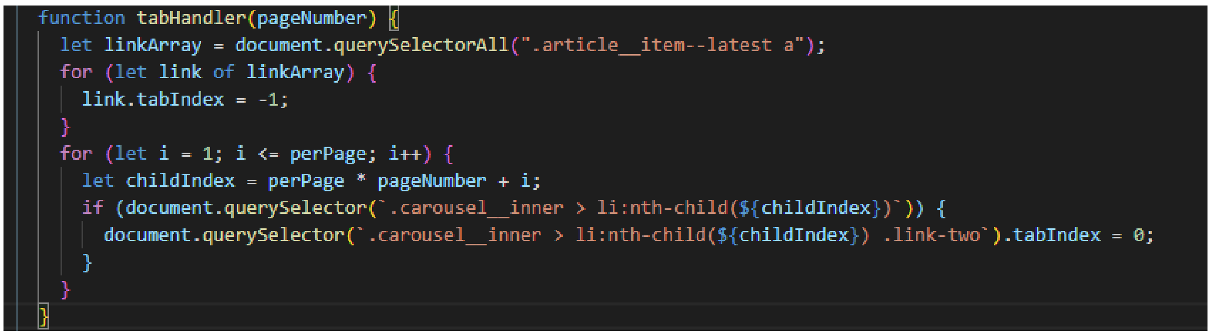


```
// If the multiplied width of the translate value, minus width of an item, is higher than the full carousel, disable the button.
let compareDeficit = carouselFull.getBoundingClientRect().width - pages * carouselVisible - itemWidth;
if (compareDeficit < carouselVisible) {
  right.classList.remove("show");
  right.tabIndex = -1;
}
```

This way, the carousel would only keep going as long as the remaining length of the full track was MORE than the width of a single item, which it always would be if there was an item left over, due to its margin.

I expanded the carousel further by adding a pagecounter with navigation below. This was all rather simple to set up compared to the carousel itself as I had already calculated most of what I needed for it to work. To disable the left/right buttons depending on which 'page' you jump to was quite a lot easier, as I could just determine it by the total amount of 'pages'. If the page was 0, disable the left arrow, if it was the last page, disable the right arrow. In fact, I considered changing the math in my left/right buttons to this solution but figured it didn't really matter.

I also disabled tabbing on any anchors out of view to prevent trapping anyone trying to tab through the website. Furthermore, as I made the mistake of having too many links to the same URLs for each article, I made sure that only one of these links were tabbable and disabled the others through tabindex=”-1”.



```
function tabHandler(pageNumber) {
  let linkArray = document.querySelectorAll(".article__item--latest a");
  for (let link of linkArray) {
    link.tabIndex = -1;
  }
  for (let i = 1; i <= perPage; i++) {
    let childIndex = perPage * pageNumber + i;
    if (document.querySelector(`.carousel__inner > li:nth-child(${childIndex})`)) {
      document.querySelector(`.carousel__inner > li:nth-child(${childIndex}) .link-two`).tabIndex = 0;
    }
  }
}
```

All of this was rather difficult, but in hindsight I suspect I might’ve just overcomplicated this to hell and back.

Important Clarification

There are a few things I want to note in my code. For my background animation, pattern, as well as loading animation, I used open-source auto-generated code rather than make it myself from scratch. I’m unsure if this counts as plagiarism, but in my book it is a tool like any other so I figured it would be alright.

I also included a snippet of code from CSStricks that would disable any animations upon resize, just to remove the janky dropdown menu popping up every time you cross the breakpoint. https://css-tricks.com/stop-animations-during-window-resizing/

What would you do differently next time

Approach the problem with the knowledge I now have and make a much more elegant solution. For instance, use the formulas from the page counter to control the left/right arrows rather than vomiting math I don’t understand all over the place.

I’d also set up authorization through the wordpress REST API so I could get attached media embedded and add comments and stuff like that. I managed to figure out how to do this with JWT authorization by the end, but it’s a bit late to do any further modifications, so it’ll have to be something for the future.

WCAG guidelines, content management and SEO

What went well on the project

I dare say my HTML is semantic at the very least, and I also added some aria-label/aria-labelledby where appropriate. I tested my site extensively with the various filters, and did a contrast check. There's one tag in particular that has a slightly low contrast, but with the added legibility of the shadows and whatnot I think it should be just barely within the acceptable range.

User Testing

As soon as I'd finished the site at least functionality-wise, I sent it to several of my friends and watched them navigate the site through livestreaming. I didn't install HotJar on my site as I figured I wouldn't get much useful feedback in a short time that way.

As the site itself is rather simple, there's not much actionable feedback I can get through observing someone complete a task anyhow, but I did notice a few things.

I'll use one case as an example. I had someone stream from discord while on their phone, and then accessing my website on the phone. I wanted to check if there were any immediate usability concerns for mobile specifically in this instance.

The first thing I observe is how this person navigated the website. They opened the navigation menu almost right away, just to check what was in there I assume, but then when they went to close it, they tapped at the transparent black overlay underneath the menu rather than the cross in the corner. So based on this feedback, I made sure that tapping anywhere outside the menu would close it.

Something else I picked up from another person was that they were often inclined to press the image to access the post rather than the title/read more buttons I had set up, so I added an anchor tag around the image as well based on this.

After all of this, I asked the person a few questions about what they thought of the site. Particularly, I asked what their first impression of the site was, and if there was anything they liked, or didn't like.

The process was similar for all 6 people I observed, although on different devices. But as these people are my friends, their feedback wasn't very 'harsh'. It was mostly just compliments. Two of them gave me actual critique. One said to slow down the color shifting on the hero banner as it was rather overwhelming, and the other told me to remove the shadows from the H1 as it was messing with their eyes.

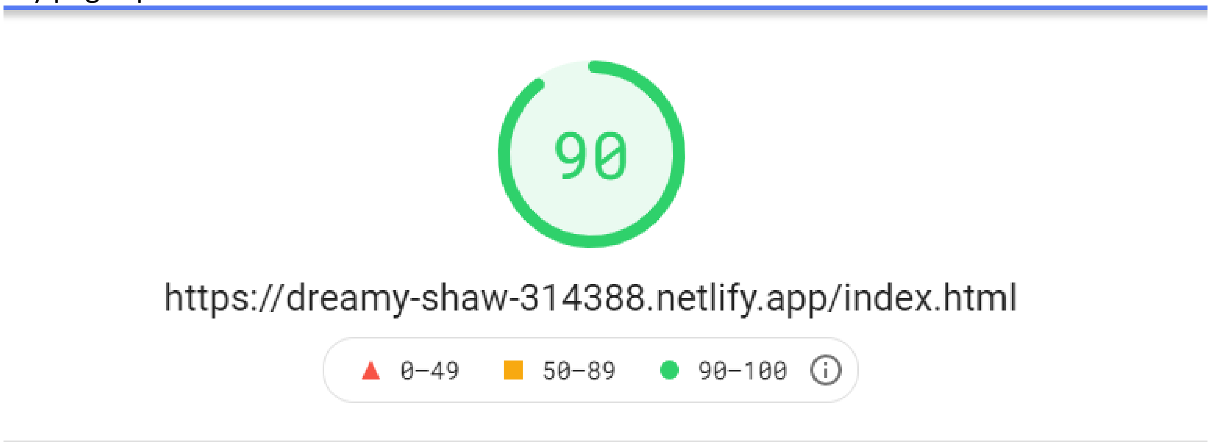
SEO

All my image tags have alt text, and I use the appropriate semantic tags wherever I could to the best of my ability. My document outline is clean, and all of my pages have a single H1, unique titles and meta descriptions.

Moreover, my page speed is pretty good even with the API.

I eliminate the problem of render blocking resources entirely by using this snippet, just like last time: https://csswizardry.com/2020/05/the-fastest-google-fonts/

My page speeds for mobile:

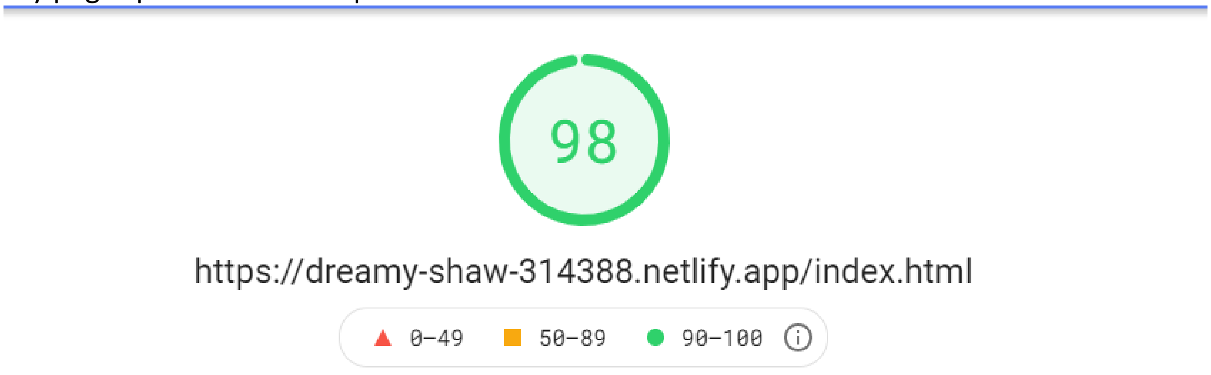


90

https://dreamy-shaw-314388.netlify.app/index.html

▲ 0–49 ■ 50–89 ● 90–100

My page speeds for desktop:



98

https://dreamy-shaw-314388.netlify.app/index.html

▲ 0–49 ■ 50–89 ● 90–100

WCAG Guidelines

I tested my site with Lynx, but as all the text was lorem ipsum I didn't bother going over it with a screen reader. I also went over it using an accessibility validator I found online, and fixed any serious oversights I could find.

What was difficult/didn’t go well on the project

I could optimize my images better, but they are all below 200kb as required in this assignment. For mobile users I’ve even made sure image sizes are below 100kb, but I’ve also included a 2x version of the images for screens with high pixel density.

Another problem was finding out what images were okay to use. As any image associated with a video game is technically not publicly licensed. The publishers allow people to use them freely as it’s free marketing, and in most cases covered under ‘free use’.

All of my images were acquired from press kits, either auto-generated ones from www.igdb.com or official ones directly from the publishers site. I’ve included a list of these sources in the references just in case.

To improve the site further I would definitely focus on the images though, but image optimization is so demanding and with the sheer amount of images I had to go through I purposefully limited myself to three aspect ratios and used the downscaled images from the API where appropriate.

What would you do differently next time

Optimize my images better, do more thorough contrast checks for my colors during the design process.

References

(place references to websites, books, forums etc. that helped you in the project)

- Image sources:
    - Risk of Rain 2: https://www.igdb.com/games/risk-of-rain-2/presskit
    - They Are Billions: http://www.numantiangames.com/press-kit/
    - Loop Hero/Enter The Gungeon: (got access to a google drive with all the publishers media assets from their influencer page: https://influencers.devolverdigital.com/) https://drive.google.com/drive/folders/1bXgOej3TlHI7gtD2ZT46i3NsWRH6Os77
    - Hades: https://www.igdb.com/games/hades--1/presskit
    - New World: https://www.amazon.com/gp/help/customer/display.html?nodeId=GNX7GA7HXVL9V8XZ&pop-up=1 (Content Usage Policy, confirming that I am allowed to use their media)
    - Valheim: https://www.valheimgame.com/ - Presskit downloaded directly from site
    - Bannerlord 2: https://www.igdb.com/games/mount-and-blade-ii-bannerlord/presskit
    - Phasmophobia: https://www.igdb.com/games/phasmophobia/presskit
    - Cyberpunk: https://www.igdb.com/games/cyberpunk-2077/presskit
    - Biomutant: https://www.igdb.com/games/biomutant/presskit

- Design sources
    - www.mycolor.space
    - www.stock.adobe.no
    - www.coolors.co
    - www.fontjoy.com
    - www.grtcalculator.com

- Development sources
    - https://css-tricks.com/stop-animations-during-window-resizing/
    - CSS background patterns: https://www.magicpattern.design/tools/css-backgrounds
    - Problem with floating numbers: https://stackoverflow.com/questions/1458633/how-to-deal-with-floating-point-number-precision-in-javascript

- Changed from validating with if (item.classList.contains) due to this: https://stackoverflow.com/questions/51214589/is-there-any-harm-in-adding-a-class-to-an-element-that-already-has-that-class-us
- Multiple API Calls, Promise.all and .map array functions: https://javascript.info/promise-api
- Handling Date objects: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date
- https://loading.io/css/
- https://www.gradient-animator.com/
- https://www.magicpattern.design/tools/css-backgrounds
- https://csswizardry.com/2020/05/the-fastest-google-fonts/
- https://css-tricks.com/linearly-scale-font-size-with-css-clamp-based-on-the-viewport/