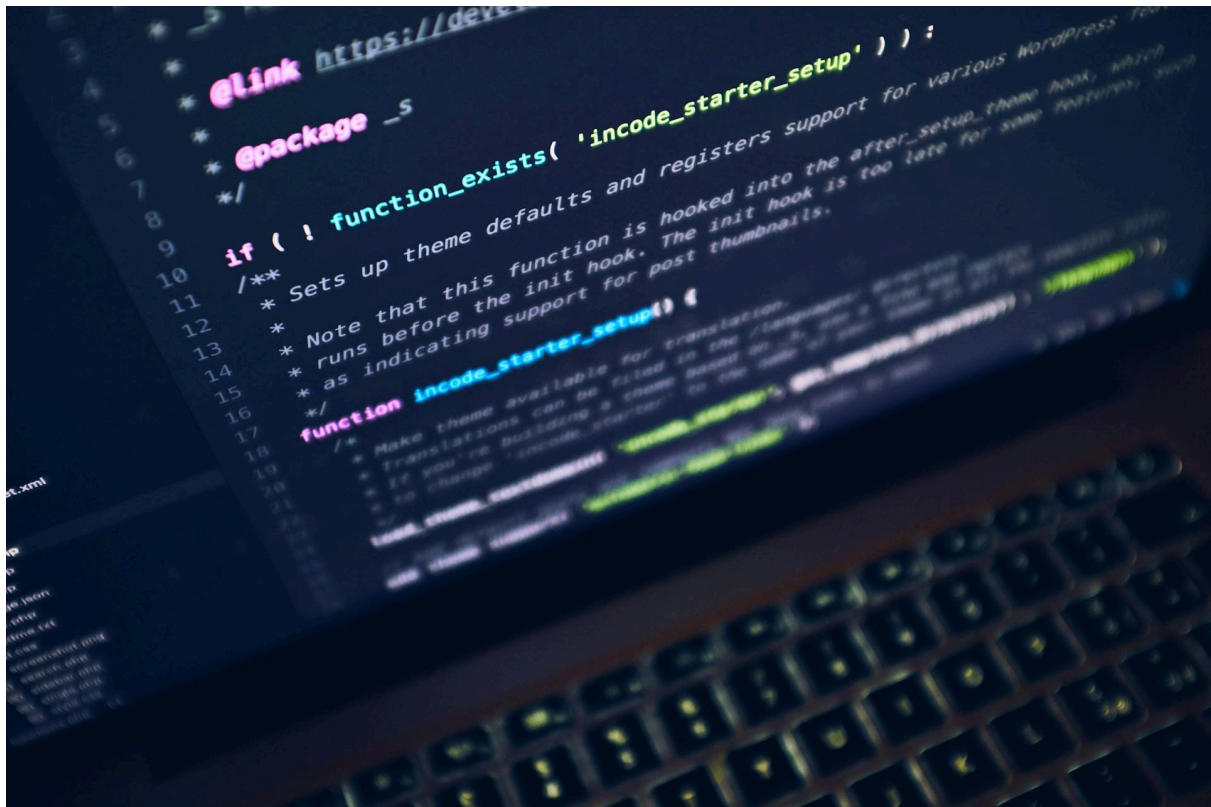




Personal website

Niklas Johansson

Introduction

This document aims to describe the design and development of my personal résumé and portfolio website. The website was created to give potential employers a more thorough overview than a traditional CV. It also provided an opportunity to improve and showcase my web development skills, as no website builder was used.

The document begins with this introduction, followed by a description of the design and design system that was used to create the website. This includes things like color, spacing, typography and how they were used to create a cohesive website. It then continues by explaining the development and implementation of the design. Here, the tools and process will be described along with some of the most interesting parts of the implementation. The document concludes with some reflections about the project.

Design

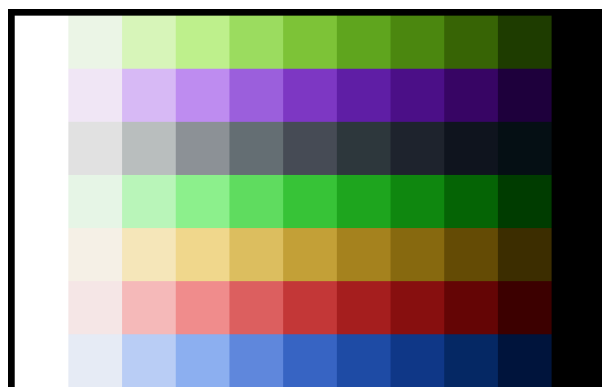
As I am first and foremost a programmer and not a designer, I had to do some research when it comes to design. The first problem was the choice of software to work with. One of the most popular choices and the one that appealed to me the most was Figma Design [1]. It works by creating GUI elements that can be arranged using layout rules. This is a major step up from my previous method of designing using image editors. It allows designers to work with the actual design instead of the pixels of a rendered image. Figma also makes it very easy to follow a design system as it supports global variables, which can be used as values in the design.

Design systems were not something that I had used before. I came across several sources that described them when I was doing the research. Since they seemed to be almost necessary to achieve a cohesive design, I decided to use one. It included color, scale (size, space and thickness) and typography. Each of these will be described in the sections below.

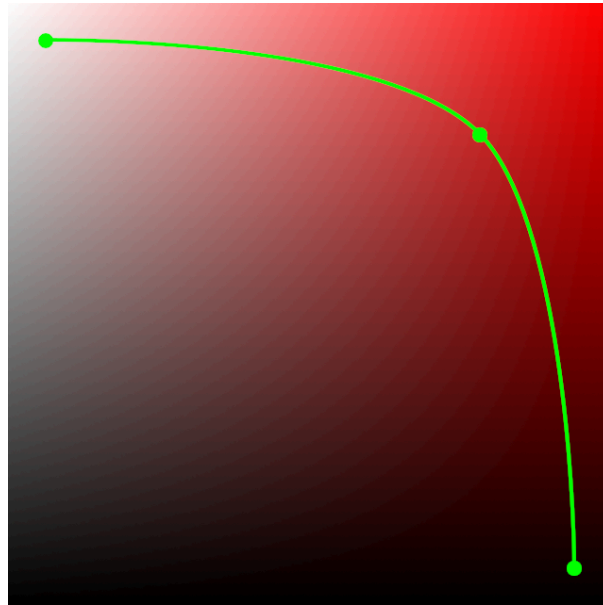
Finally, to make the design a bit more interesting, I used images from Unsplash [2], a great source of free images. I also used FontAwesome [3] for icons and a Google font called Quicksand [4].

Color

The color palette used in the design can be seen below. It features the three main colors: primary, secondary and neutral. It also contains four side colors: success, warning, error and info. All of the colors come in nine shades, ranging from bright to dark. Total white and black have been added to the image for reference.



To create this palette, a guide [5] was used. It explains the colors needed to create a palette and how to select the shades for each color. The guide uses a curve segment to select these shades rather than selecting them linearly. This curve method applies a curve segment to a brightness/saturation plane with the same hue as the base color. An example can be seen below, where the base color is almost pure red.



The curve is determined by the constraints that it must start near the top left corner, pass through the base color and end up near the lower right corner. The curve segment is then determined by selecting the brightest and darkest color on this curve that will be a part of the shades. To select a color for a new shade, a point on the curve segment is chosen between two existing shades.

When choosing which shade of a color to use for an element in the design, contrast is important, especially for accessibility. This is why the contrast was continually checked during development. The contrast was tested with a tool [6] that determines whether it complies with WCAG AA/AAA. To the best of my ability, I believe the website complies with WCAG AAA. There might be some contrasts that do not comply, but still pass, given the context they are in.

Scale

Most design systems enforce a scale of values that are used for sizes and spacing. These scales consist of multiples of a base value, often a power of two (4 and 8 are common). This effectively puts the components on a grid and makes the design less random and more cohesive. To avoid having too many options, the scale is often not linear and does not include all possible multiples. This is fine as the need for granularity is proportional to the magnitude of the values. It does not make sense to have large sizes with only a few pixels of difference between them.

To construct such a scale, the following formula can be used. It is based on powers of two with only four values per granularity. The formula was devised based on common scales found during research.

$$f(n) = 2^{\lfloor n/4 \rfloor} + (n \bmod 4) \cdot 2^{\lfloor n/4 \rfloor - 2}, n \in \mathbb{Z}$$

The scale that was used also removed values with fractions, as that level of granularity was not needed. The resulting scale can be seen in the table below, where the cells without a value were skipped since they were not integers. For each cell, the above formula was applied with the n from the header row/column. I chose to stop at the arbitrary n of 33.

n (Binary)	00	01	10	11
0000xx	1	-	-	-
0001xx	2	-	3	-
0010xx	4	5	6	7
0011xx	8	10	12	14
0100xx	16	20	24	28
0101xx	32	40	48	56
0110xx	64	80	96	112
0111xx	128	160	192	224
1000xx	256	320	-	-

This scale was then multiplied by three different values to produce a scale for size, space and thickness. The values used to produce the final scales differ since they need to provide different ranges. Thickness, which is used for lines such as borders, needs a narrow range, while sizes need a much wider range. I used 1/2rem for size, 1/4rem for space and 1/16rem for thickness.

The idea of size and spacing systems and how to use them was found during research in multiple guides and tutorials [\[7\]](#) [\[8\]](#) [\[9\]](#) [\[10\]](#).

Typography

The final part of the puzzle is typography. Just like sizes and spacing, typography should also be consistent across the design. To achieve this, a scale of font sizes needs to be established. The scale that was used for this website was created following a guide on typography [11]. First, a base font size is picked, in this case 18px. Then, the larger and smaller sizes are picked based on the base size. I chose to follow the same scale as previously used to create the size, space and thickness scales.

This should cover normal text, but headings need to be included in this system. The guide recommended a tool [12] that can be used to create geometric font size scales with previews. I chose the perfect fifth scale, which scales by 150% each time. I then multiplied this scale by a constant to adjust it to fit the design. The result is a consistent and cohesive typography system.

Development

The first step towards implementing the design was to get to know the field of web development a little better. I had not done any serious projects for quite a while and needed to see what new things had turned up since last time. I explored front-end frameworks like Svelte [13], build tools like Vite [14] and much more. In the end, I decided to go with vanilla HTML/CSS + JavaScript. Even though these new technologies were nice to see, they seemed a bit overkill and out of scope for this project. That being said, I will definitely consider using them for future projects.

The next step was to set up the development environment. An editor for HTML, CSS and JavaScript was needed. I chose Visual Studio Code [15], which is one of the most popular editors when it comes to web development. Like most editors and integrated development environments (IDEs), it provides more advanced features compared to a regular text editor. It makes development that much easier.

The project was then version-controlled by creating a Git [16] repository hosted on GitHub [17]. Using version control is essential in any software project, even when solo developing. It means that no code is lost, and you can always go back in time to look at how things were before. Version control can also act as a safety net. Should things break, you can always check what has changed since the last time it worked. These are only a few of the many reasons why I cannot live without version control.

The last tool that was used is ChatGPT [18]. It can be very helpful when problem-solving, especially since my knowledge was a bit outdated. It can suggest solutions that I can then verify by looking at the relevant documentation. Another use case is debugging hard-to-spot bugs. The AI can review my code to help me spot the error. It is, however, important to note that no HTML/CSS nor code was directly copied from the AI. It was exclusively used as an advanced search tool and not a crutch.

Semantics

When writing the HTML for the website, I tried to use as many semantic tags as possible. This has no effect on the website itself, but rather helps with things like screen readers and other automated processing of the website. To improve this support even more, ARIA [19] information could have been added to the tags.

Security

To provide some basic level of protection against malicious users, both a content security policy (CSP) and referrer policy were created. The CSP protects users from cross-site scripting (XSS) attacks by specifying what resources are allowed to load. Finally, the referrer policy prevents leakage of sensitive information by restricting what the information referrer header can contain, depending on various factors.

Reset

After the initial HTML was finished, I started to look at CSS resets/normalizers. A CSS reset resets all default styles to a bare minimum. This forces you to provide your own styles instead of relying on defaults, which might not be the same across different browsers. A CSS normalizer, on the other hand, does not remove styles. It instead tries to modify them to look the same across browsers. Both a CSS reset and normalizer should be applied before any other styles and provide a clean slate to build upon.

When researching the topic, it seems that modern CSS and browser differences have improved to the point that a full reset or normalizer is not necessary. The popular choice is instead to create a small reset with very specific styles. This is the path I chose for this project. The most notable styles are the removal of borders, outlines, padding and margins. The box sizing was also set to border-box for easier sizing of elements.

Theme

To transfer the design system to CSS, I chose to create a theme CSS file with all the variables. The variables were implemented using custom CSS properties as seen below.

```
:root {  
  
  --scale: 1.0;  
  ...  
  --scale-2: calc(var(--scale) * 2);  
  ...  
  
  --size: calc(1.0rem / 2);  
  ...  
  --size-2: max(2px, var(--size) * var(--scale-2));  
  ...  
  
  --space: calc(1.0rem / 4);  
  ...  
  --space-2: max(2px, var(--space) * var(--scale-2));  
  ...  
  
  --thickness: calc(1.0rem / 16);  
  ...  
  --thickness-2: max(2px, var(--thickness) * var(--scale-2));  
  ...  
  
  ...  
  --primary-100: #eff9e6;  
  --primary-200: #d8f6bb;  
  ...  
  
}
```

I could then use these custom properties instead of constants when specifying values. This has many benefits, such as being able to change the scale and the entire website changes accordingly. If I had simply used the values of the variables directly in the styles, I would have had to change all occurrences when changing the design.

I also ended up using this scaling system when dealing with smaller screen sizes. Since the text of the website does not scale with the screen size due to accessibility reasons, it will eventually not fit within the window for very small screens. To solve this, the scale is taken down a notch for smaller screen sizes. The result is a website that is readable on a wider range of screen sizes. This type of automatic propagation would not have been possible without the use of custom properties or similar constructs.

Navbar

One of the most dynamic parts of the website is the navbar. It features animations for hover effects as well as current section indicators. The indicators are animated using CSS and triggered using JavaScript. They are absolutely positioned and moved by setting the corresponding top, right, bottom or left properties. The indicators are also aware of the location history and will animate even if stepping backwards/forwards through it. The navbar is also responsive (like the rest of the website) and will transition to a vertically stacked list of buttons with a hamburger button to open it.

In total, there are about 130 lines of JavaScript to achieve all the functionality. The majority deals with the current section indicators, as there is a lot of logic behind them. The current section must be matched to one of the buttons in the hierarchy. All current indicators must then be updated accordingly. This requires some clever recursive looping of the button hierarchy and must be done each time the browser changes location or is resized. The reason for updating the indicators when the window is resized is due to the size of the indicator being set during the update, and not automatically by using CSS.

Timeline

The second most interesting part of the website is the CSS-only dynamic timeline used by the background section. The technical part of the implementation was inspired by a tutorial [\[20\]](#) showing how such a timeline can be created. The timeline combines a fixed progress bar and sticky time and pin elements. The progress bar is hidden under most of the website and only appears on top for the timeline. Since the height of the progress bar is 50vh, it gives the illusion of filling the timeline as the user scrolls.

To make the time and pin move along the progress bar, they are set to be positioned sticky. Their container's height is also expanded to match the height of the card content on the other side. This results in the time and pin moving along the card's edge when scrolling. They both stop at the top and bottom if the user scrolls past the card in either direction.

Templating

After implementing all of the website design, I decided to have a look at adding a server backend to split up the monolithic HTML page into simpler templates. I also looked at adding some form of content management system (CMS) to better handle the content of the website. After looking at both, I decided that a CMS would probably be overkill for a small static website like this. Even if choosing a smaller CMS, the amount of work to integrate it would take a lot longer than simply editing it directly in place.

As for splitting the HTML file into smaller templates, I decided to give it a try using PHP. To do this, I used XAMPP [\[21\]](#) as a local development server since PHP cannot be executed directly in the browser. I also decided against using an existing template system like Twig [\[22\]](#) since I did not need advanced features and wanted to refresh my vanilla PHP knowledge. The result was a simple template system using vanilla PHP that simply rendered a page by reading a data structure.

The data was populated by the page and then passed to the template engine, which recursively added templates and populated their values. This worked ok, but in the end, I decided against it. The templates were harder to manage than the previous HTML, and the benefits were minimal. I therefore reverted to using a single monolithic HTML page.

Reflections

Overall, I think the project went well. It took roughly seven weeks to complete from start to finish. This includes the time spent on research, design and implementation. Since the project only consisted of a single static web page without interactive content, this seems acceptable. Future projects of similar scope will probably be completed faster since much of the research can be reused. Especially the knowledge of design systems and the current state of web development.

One thing that I will definitely consider using in future projects is Figma [1]. It was such a breath of fresh air compared to using image editors like GIMP [23] or Inkscape [24]. The fact that you can create automatic layouts to arrange the elements instead of doing it manually makes all the difference. Figma also blends very well with design systems, which is another thing that I would consider using in future projects. They reduce the number of options for values, which is always a plus for non-designers.

I was also pleasantly surprised by all of the new CSS functionality that I discovered during research and problem-solving. It gives developers more options to solve their problems. In some cases, the new functionality even provided solutions that were simply not possible before. I definitely need to read up on all the available properties and concepts before starting any new projects.

Besides CSS, there also exist several tools and technologies that I would be very interested in trying out. I chose to go vanilla for this project as I wanted to complete the project in a timely manner, without lots of steep learning curves. This does not mean I cannot have a go at something new in the future. One particular stack of interest is Svelte on top of a headless CMS, either running PHP or Node.js. This would be something very different from what I am used to and would undoubtedly lead to a fun learning experience.

The final part of the project was the attempt at making a templating system. Even though it was successfully implemented, it did not offer any substantial benefits. This is not a fault of the idea itself, however, but rather the implementation of it. I will have to investigate other options and try some of them out to find a better solution. One of the most popular PHP templating systems is Twig [22]. Perhaps a modern and ready-made system will provide the missing pieces. If I cannot find any good systems, I could always make another attempt at making my own. It would also be wise to search for a good CMS to serve as the source of the data to be inserted into the templates.

References

1. Figma Design
<https://www.figma.com/design/>
2. Unsplash
<https://unsplash.com/>
3. FontAwesome
<https://fontawesome.com/>
4. Quicksand Google Font
<https://fonts.google.com/specimen/Quicksand>
5. How I make UI color palettes
<https://youtu.be/yYwEnLYT55c>
6. Contrast Checker
<https://webaim.org/resources/contrastchecker/>
7. How To Get The Perfect Spacing In Web Design
<https://youtu.be/Dp99tztXdOA>
8. Perfect Spacing for Web Designers
https://youtu.be/_V7HAKjYrW
9. Space, grids and layout
<https://www.designsystems.com/space-grids-and-layouts/>
10. Space in Design Systems
<https://medium.com/eightshapes-llc/space-in-design-systems-188bcbae0d62>
11. How to set up your typography sizing and line height for web design
<https://youtu.be/eW2-ghXuYC8>
12. Typescale
<https://typescale.com/>
13. Svelte
<https://svelte.dev/>
14. Vite
<https://vite.dev/>
15. Visual Studio Code
<https://code.visualstudio.com/>
16. Git
<https://git-scm.com/>

17. GitHub
<https://github.com/>
18. ChatGPT
<https://chatgpt.com/>
19. Accessible Rich Internet Applications (ARIA)
<https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA>
20. Build a Responsive Timeline with HTML & CSS (No JavaScript!)
<https://youtu.be/NVg5g8-gP8M>
21. XAMPP
<https://www.apachefriends.org/>
22. Twig
<https://twig.symfony.com/>
23. GIMP
<https://www.gimp.org/>
24. Inkscape
<https://inkscape.org/>