Noroff
School of technology
and digital media

Project Exam 1

Jøran Engelund

Word count
2969

Design

Wireframe: Low-Fidelity Wireframe

- In the process of drawing the low-fi wireframe, I quickly managed to vision what kind of layout I wanted to have and how to make the website look clean, non-noisy and have a easy and good overview of the content and a easy, natural user-flow for the end-user in regard of good UI and UX.

- I found out that the best layout for the website would be to have a grid column layout troughout all the pages, to further improve on a good overview of the content and consistensy. The layout structure will be further explained in the High-Fidelity Prototype phase in the report.

- I managed to stick mainly to the original design for the blogsite from the wireframe, when designing the high-fidelity design and prototype, as I didn't meet on any difficulties regarding the structure of each of the pages. There were some small changes that occured in the prototype that I saw was needed. To be able to change and implement changes troughout the design phase, helped me to not have a static webdesign and stuck on one idea on how it would structured. Even tho the changes were small, they could still indirectly change the structure and the feel of the design.

- The user-flow of the website was also decided on and drawn in the Low-Fidelity wireframe, trough site navigation in the header and to the CTA's and links around the website. Having the principle of "maximum 3 clicks" for the end-user to reach the targeted goal, it was easy to make the user-flow natural and easy to navigate. Also implementing IxD helped on the user-flow overall.

Prototype:
High-Fidelity Prototype: Desktop
High-Fidelity Prototype: Mobile

- The color choice of the website is a warm white-cream colour, giving a relaxing and warm welcome to the users. Making them feel at home. The font style of the headings, cta, logo, and navigation elements is called "Chennai", giving a little "playful" look and not a strict hard feeling. As the theme of the page is "Life".
- **Theme:** The theme of the page is not centered around one theme, but mainly all of the themes. Life is a blogging platform for many people. Where they can share their life experiences.

- **Index:** I decided that the most natural placement for the website logo was in the top left corner of the header. This is to make it easier for the end-user to quickly see the logo and the brand-name of the website they've landed on. Centered in the middle of the header, the website's navigation is placed, horizontally, making it easy to get a good overview of what the website consists of in depth of content. The navigation elements has a "non-active" styling, where the page name is only rendered in standard font and no styling, while the active navigation element, the page the user is currently on, is styled with a background color of dark-greyish. The hexcode is "#415D5D". This is to make the user easily understand what page they're currently on and it gives a good overview, as the active navigation styling is easy to spot. When the user hovers over the "unactive" navigation elements, the same styling for the "active" navigation elements gets displayed. This to make the experience easier for the user to understand that they can click on the links to navigate. The active navigation styling for the navigation elements will be changed depending on what page the user is currently viewing.

- Below the header, we find the main content of the index-page. The main structural layout of the page is grid with 1 column. Here I decided that the first thing the user will see is the "Latest Posts" section. Starting with a heading, "Latest posts" and below it a gallery of 4 posts, consisting of the post image, post-title, "quote" from the posts and a CTA giving the user a option to og directly to the selected posts. This dramatically reduces the amount of clicks for the end-user to reach the targeted goal to just 1 click. The post-cards are styled with a gap of 16px of white-space between each post-card, giving some breathing room. The gallery is to be developed to a carousel, making the user able to scroll either left or right to view through different posts that has recently been posted. This is to make the user instantly find the targeted goal, reducing frustation for the user to need to scroll down on the page or navigate through different pages to find what they're looking for. It'll make it easier for new users to quickly get comfortable with the content and aswell for returning users to find what they came back for.

- Scrolling below the "Latest Post" section, we'll find the section "preview-info" with the secondary heading "Life". Here the user can read a body copy consisting of a short preview paragraph information on what the website is about and what you can expect of it. Below the copy of the "preview-info" section, the user is presented with a CTA, that reads "About us", which takes the user directly to the "About" page on the website. This is to make the user to not need to scroll up to the header again to find the "About" page, and therefore makes better user experience and an easier user-flow.

- **Posts:** Following the navigation link "Posts" in the navigaton, the user now gets directed to the "Posts" page on the website. The user is presented with all the blog posts, structured in a layout where the container of all the posts is a grid with 1 column, and the nested post-cards has a layout grid with 2 columns, centered on the page. This gives a clean look and a good overview of the posts on the page. Each post-card has a image on top, followed by the title of the post, then a quote from the copy in the blog posts and then a CTA which has a good contrast compared to the rest of the page design, making the user not look hard for a button to read the desired post.

- **Specific-Posts:** Following one of the CTA's from either "Latest Posts" gallery on the index page or the CTA's in the post-cards on the "Posts list" page, the user gets directed to the specific post they want to read. The blog image is at the top, centered on the page, followed by its blog title, then the body-copy, the paragraphs of the posts. Beneath each post, there is a "Comment Section" where the user may read posted comments, and beneath that, a form for the user to fill out and post their own comment on the current viewed post. At the end of the page, the user is presented with a CTA that takes them directly back to the list of posts, again making the flow and navigation easier for the user and giving a better experience by presenting a "shortcut"

- **About:** Following the navigation links "About" in the navigation or the "About Us" CTA from the index page's "Preview-info" section, the user gets directed to the websites "About" page. Here there is 1 main heading, "About Life" and a informative section consisting of a body copy with paragraphs that informs the user of what the Life blogging plattform is about. Following the website main structure, the page has a 1 column grid layout, the content centered on the page and a "shortcut" option in form of a CTA at the bottom, giving the user the option to get direct navigation to the "Posts" page.

- **Contact:** The page is structured with a 1 grid column layout, where the user gets presented with a contact form, on how to get in touch with the administration of the Life blogging platform. Where they may write their inqueries. After filling out the form and submitting it, the user gets presented with a CTA shortcut that takes the user to the posts list page.

Technical

Note: *I will mainly focus on the important technicalities revolving around JS and some CSS styling, aswell as leaving out repetetive functions. The structured layout for html and CSS styling of the individual pages are very much consistent throughout. Some functions are also reused on various pages*

- Index: Starting with the header, I struggled a little with the positioning layout of the logo and <nav> elements using only display: flex;. I ended up using display: grid; with grid-template-areas: “logo nav nav”; to be able to have the correct layout placement for the elements. Using grid for the container and then further using display: flex;, for more finer adjustments proved crucial.
- **Styling of the carousel:** container has only a set width of pixels on 964px to get the desired width and design I was looking for. I have a child container, that consists of all post-cards created with dynamic HTML in JS using DOM. I struggled to either use display flex on this or grid. But I found out that display: grid with columns and using grid-gap: 16px to get the desired layout. It is obviously much easier to structure a big layout with grid than with flex. Thus I use display: flex; in each post-card for finer adjustments. Added overflow: auto on the carousel container to hide the posts that are “overflowing” on the sides of the container.

- **javaScript(index):** Retrieving blog posts from the WP REST API, I used a async function, inside the try{} code block, I fetched the arrays from the API using await fetch with the REST API URL as argument, then I used javaScript Object Notation (json()) to read the fetched arrays from the api. Retrieving the title property from the objects wasn’t difficult. But selecting the blog images proved to be a problem, as I had added the images in the post on WordPress admin panel, together with the post paragraphs. This resulted in the images having a hardcoded styling from wordpress assigned to it and it proved to be difficult to make the image responsive, aswell as the paragraph and the post images from the post were both assigned in the same property (post.content.rendered). The paragraphs wasn’t meant to be displayed on the post-cards or blog list, but on the specific post page. Because of this, it showcased that the paragraphs and the images had to be split in the Wordpress admin panel to be able to retrievie them on separated properties, which I will further explain in Content Management section in this report.

- After assigning the post image to “featured-media”, the properties was embedded and it proved difficult to retrieve the images. I had to add “&_embed” on the end of the REST API url, to be able to get access to the source_url of the image. The use of “&_embed” solved the issue. Further I used forEach method with an inline function and set parameter to post. Then I used innerHTML to create a dynamic html using template literals to fetch the properties from the api and display it on the HTML page. I also added catch{} block to catch any errors in the API fetch function.

Carousel Function: I declared a variable named gap and initialized it with the number value of 16, to mimick the grid-gap: 16px from the css styling. After this I declared another variable named width, initialized with the carouselContainer.offsetWidth; to get the whole width of carousel container, including border and padding aswell. Further I have two functions:

```
function nextSlide() {
    carouselContainer.scrollBy(width + gap, 0);
}
```

- I use the scrollBy method which scrolls the document, or element by the given amount.
  `scrollBy(x-coord, y-coord)`
- As I only want to move the cointainer horizontally to the right, I set the first parameter of the method to width + gap, telling the function to move the whole width of the assigned container + the gap of the cards. As we don't want to move vertically, I set the parameter to the value of 0

```
function prevSlide() {
    carouselContainer.scrollBy(-(width + gap), 0);
}
```

- In this function I want to it to do the exact same as the function above, but now it should subtract from the width and gap variable, this way it moves backwards or left. We still keep the vertical parameter at 0.
- Assigning these two functions to an element.AddEventListener("click", function); makes the carousel either move left or right when clicking on the assigned elements:

```
prevBtn.addEventListener("click", prevSlide);
nextBtn.addEventListener("click", nextSlide);
```

Blog posts list: After setting up the API Call on this page, more or less the same as on the index page. I started on the view more function. I had 2 solutions on this. The first was a rather static solution, until some teacher helped with a header.property from the api to get the total amount of pages from the WP. First try I added a secondary API FETCH and used this url `` const url = `https://life-api.engelund.site/wp-json/wp/v2/posts?page=2&_embed`; ``

- This helped me get access to the two last posts. I then assigned the API fetch call to a button.addEventListener and fetched the last two posts. But again static, what if there are more than two pages?

This is where `const maximumPages = response.headers.get("x-wp-totalpages");`

This helped me fetch the amount of pages that were in the REST API for the posts. Utilizing this I added a variable with value 0, `let pageNumber = 0;` and added this variable to URL using template literals `` const url = `https://life-api.engelund.site/wp-json/wp/v2/posts?page=${pageNumber}&_embed`; ``

- Variable starts at 0, so added then `pageNumber++;` on the start of the api fetch before getting the url. This makes it add +1 to the page number, each time the fetch has been called.

and if statement

```
if (Number(maximumPages) === pageNumber) {
        viewMoreBtn.style.display = "none";
      }
```

- This checks value number of maximumpages with the variable pageNumber value. If it is the same value, the viewMoreBtn is set to display: none; this way, if it is 3, 4 or more pages, it will be more dynamic and less static.

- **Search function:** To be able to retrieve the data from the api fetch, I set a renderPosts function that utilizes the innerHTML that creates the dynamic html from the API. With a forEach() method. I further write another function that grabs the event.target.value (target here is the value from the search input field) to a variable called searchQuery. And uses the methods trim() to remove all characters from before and after the value of the input string and toLowerCase() method to make all the string values to lowercase.

```
function filterPosts(searchQuery, postList) {
  postContainer.innerHTML = "";
  const filteredPost = postList.filter(function (blogPost) {
    if
(blogPost.title.rendered.toLowerCase().trim().includes(searchQuery)) {
      return true;
    }
  });
```

- I used in this function a .filter() method with inline function to filter in the arrays from the api fetch, and check the value of the search input against the value of the searchQuery is the same as in the property blogPost.title.rendered. Using the method includes() I check wether any of the string values match any string values in the property, if it does, return true

- **Specific Post:** The api fetch is the same here as in the other API FETCH functions. Tho here I wanted to have a dynamic meta description in the header, as the page changes based on the id and content. So to optimize SEO i added used querySelector() with setAttribute = “content” and the property post.title.rendered

```
const metaDescription = document
      .querySelector(`meta[name="description"]`)
      .setAttribute(
        "content",
        `Read and comment on ${details.title.rendered}, posted on Life`
      );
```

- With the setAttribute(“content”, “value”) method I managed to change the value of the content to change according to the title of the post. Much like the dynamic document.title to change the documents title to correspond to the posts title currently viewed.

- **MODAL:** Making the modal work. I created a modal container in the html, with an empty img tag. Retrieved it with a querySelector. I also added a ID to the post original Image. This way to be able to get the original image from the dynamic html. I then assigned the original image to and addEventListener with an inline function using the methos of .src and this.src

- When the original image is clicked. I take the modal image's empty img tag from html and with element.src. This targets the empty img tags source attribute.
- `modalImage.src = this.src;` - This method replaces the left hand elements source attribute with the right hand source. As "this" is the original image, that is assigned to the addEventListener("click", function() {})

```
postImage.addEventListener("click", function () {
      modalContainer.style.display = "flex";
      modalImage.src = this.src;
      modalImage.alt = this.alt;
    });
```

- **Form Validation:** The form validation uses a function that checks the value and length against each other to exams criteria of the form validation input lengths. I also use trim() to remove space characters from the value in the inputs, so those doesn't count. I also disable the submit button to prevent the user from spamming the submit button. Submit.disabled = true; changes to false; if all the input validations passes as true, which is checked in an another function. This checks if all the values of the input is more than the criteria. If it equals to true, then submit.disabled changes to false;
-

```
const buttonDisabled = () => {
  if (
    lengthChecker(inputName.value, 4) &&
    lengthChecker(subject.value, 14) &&
    lengthChecker(message.value, 24) &&
    emailValidator(email.value) === true
  ) {
    submit.disabled = false;
  } else {
    submit.disabled = true;
  }
};
```

- *Note: I've tried explaining the functions and technicalities the best I can, as this proves to maybe be the most difficult thing about the project.*

WCAG guidelines, content management and SEO

- WCAG: Following the WCAG guidelines for accessibility, I've added alt="" attributes on all images. All pages have a unique title in the <head> aswell as a unique <h1> element on each page, telling what the page is about. I made sure the colour contrasts of the background-colours and the foreground elements had sufficient compliant contrast – in accordance with color-blindness for WCAG Guidelines. I also made sure that the hierarchy of the heading titles like h1, h2, h3 was compliant in order of what the hierarchy of the content on each corresponding page.

- **Following SEO guidelines:** I added keywords for the webpage in the meta description of each page, with each page having its unique description and keywords corresponding to content of the page. Aswell as each page has it's own unique title in the element, displaying in search engine results what this page is and what it is about.

- Content Management: Content management system was used with Wordpress Admin panel with the webhost One.com. Added plugin for headless mode. To be able to use the WP CMS as a REST API. I created posts that had 1 unique h1 title and a following paragraph for the post. As described earlier, there were some problems adding the images straigth into the post. Doing this added hardcoded styling on the images and also added the images in the same property of the paragraphs. Post.content.rendered. I had to remove the images and add them in the right-hand menu in the blog post panel called "featured media". This way I managed to assign the image to it's own property. I also wanted to use a "quote" from the posts. So I added it to the excerpt menu, assigning it to it's own property, to be able to divede content, image and quotes.

- **Testing:** For testing I used various tools. User testing was done with Hotjar, specifically using screen-reader. This way I managed to notice that the users struggled to see the difference on the "view post" cta and the "View more" cta in the blog posts list page. This proved crucial, so I decided to change the styling of the "view more" cta to create better contrast betweent the various CTAs on the page. I also got feedback that the body copy's typography on the paragraphs was hard to read, so this was also changed.
- For WCAG and SEO I used google lighthouse and Wave (I will list these links in the references for testing)

References

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

References for Wireframe:

1. Wireframe drawing tool

References for Prototype:

1. Color palette creator

2. WCAG Contrast Checker
3. Font "Chennai"
4. Font "Ofelia Display"
5. Inspiration for webdesign
6. Inspiration for webdesign (2)
7. Inspiration for webdesign (3)
8. Inspiration for webdesign (4)
9. Image compressor
10. Shutterstock - Archive for stock images

References for Technical:

1. FontAwesome
2. Hamburger menu Icon - FontAwesome
3. Typography - font.adobe
4. Font "Chennai"
5. Font "Ofelia Display"
6. Learning of scrollBy() method
7. Learning of this.src method
8. .oninput event
9. learning how to get width or height from .window or .document
10. References to Moodle videos in lectures can't be given because of currently not able to access Moodle, this because of OneLogin app problems
11. References to teachers as Connor and Oliver for tips along the way.
12. References to fellow students in JS1 chat on discord for tips along the way

References for Testing and tools for WCAG and SEO:

1. Google Lighthouse
2. Wave
3. Hotjar