HCI Course PWA Web Site

Michael Hebesberger
Diego Jacobs Tercero
Christoph Koch
David Mischak

Institute of Interactive Systems and Data Science (ISDS),
Graz University of Technology
A-8010 Graz, Austria

01 Feb 2021

Abstract

This paper reports the process of revamping the original Human Computer Interaction (HCI) course website into a Progressive Web App (PWA) by using a static site generator called Eleventy. Therefore this paper is divided into three major parts, starting with a brief overview about static sites and their advantages and disadvantages. The second part describes how Eleventy works, including the installation of a project, the configuration, the basic workflow and the usage of templating-languages, especially Nunjucks. Finally, the last chapter explains the development process of the HCI course website project. This includes the configuration for the project and the defined structure of folders and files. The PWA component integration plays a major role in this chapter, since it was important for the new version of the website. The paper shows the most important steps on how to create the manifest file and states how the service worker is configured in order to make the website offline available.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

The aim of this project was to rework the website of the Human Computer Interaction (HCI) course at the Graz University of Technology. This website contains static content about the HCI course like the schedule, exercises, materials and so on. The main goals for the new website were to realize it using a static site generator called Eleventy, make it into a simple Progressive Web App (PWA) and avoid using JavaScript. These measures ensure that the site is lightweight, robust and easy to maintain.

The first Chapter 2 explains what static sites and static site generators are. It also addresses the advantages and disadvantages of them. At the end of the chapter examples of static site generators are named.

The following Chapter 3 focuses on the static site generator Eleventy which was used in the project. It explains a few basics and how to use it. Furthermore, examples of important files are shown throughout the chapter.

Finally, all the details of the project are explained in Chapter 4. The first sections in this chapter are about the configuration, structure and templates of the project. The next section is about Relative URLs which were important to create reliable and sustainable code. Then a section about the newly included PWA features, most notably the offline functionality implemented with service workers. Lastly sections about the navigation bar and the styling of the website are addressed. Noteworthy in these sections are the realization of the navigation bar purely with stylesheets.

The final Chapter 5 concludes the contents of the paper and the experience of working on the project. The project offered an exciting challenge for the authors of this paper. The used technologies are relatively new and up-to-date which is exciting, but also introduces problems when it comes to documentation or existing precedents. Nonetheless, the project was realized with all the features which will be explained in the following chapters and pages.

Chapter 2

Static Sites

Static sites consist of pre-built files like HTML, JS or CSS. This type of site also does not have changes from a server. It means that they will display all the content in the built files. There is no possibility for dynamic changes from a server. The site can still be interactive, but to have some dynamic changes in it, the client-side would have to be the responsible one to manipulate the content on the site in reaction to an action from the user. Since there is no server behind doing operations or validating actions from the user. The site will display the same information for all the users. In other words, all users can access the same content. [Kalkman 2020]

2.1 Advantages of static sites

One of the advantages of static sites is that it will have improved performance for end-users. Since it is only displaying content and does not do any calculations or processing in the background when the user requests a page. Another advantage of a static site is being able to have fewer dependencies. It will depend on how the site is developed because it can still have more dependencies if its developers used a diverse type of sources to style and build the site. Last but not least, it has the advantage of cost savings. Since static sites show only pre-built content, which means that there is no need to spend a great amount of money in developing it or hosting it. [IONOS 2017]

2.2 Disadvantages of static sites

The first disadvantage of a static site is that dynamic behavior on the site will have to be done on the client-side. Another disadvantage of static sites is the need to maintain many pages if no tool, which automates the generation, is used. of it. [IONOS 2017]

2.3 What is a static site generator?

Static site generators (SSG) are frameworks which create static sites. They usually support content written in markdown languages such as markup and HTML. What this tool does is it generates the pages of the website ahead of time. It means that the page will be generated before the user requests it. This is because the pages get built on the developer's computer and then published on the server. For this reason, most generators have a template engine, which is used to create template layouts. [IONOS 2017]

2.4 Examples of static site generators

Here are some of the most popular static site generators [Rinaldi 2021]:

- Eleventy

- Hugo
- Next.js
- Gatsby
- Nuxt.js
- Jekyll

Chapter 3

Eleventy

Eleventy is a static site generator and since it is written in JavaScript, it gives you access to all npm packages. In this chapter the installation and configuration process and how Eleventy works will be explained.

3.1 Setup

Setting up Eleventy is a very straight-forward task, which only requires Node.js as a prerequisite.

3.1.1 Installing

Eleventy can easily be installed with either npm or yarn using the commands from Listing 3.1. After the simple installation process, it is ready to be used in a project [Digital Ocean 2020].

3.1.2 Configuration

Once the installation is done, Eleventy can be configured flexible. Without a configuration, it will just take all markup files inside the working directory and compile them to static HTML files. However, setting up a configuration file is fairly simple and offers great benefits.

The file itself is written in JavaScript and lets one configure which templating formats should be transformed, where the input, output and include folders are located and many more things. Configuration options that are very valuable are for example the option to set files to be passed through to the built site folder and also registering plugins as well as adding shortcode, which is being used in the project. In Listing 3.2 the configuration of the HCI website project can be found, which uses all the above mentioned features. A full overview of all possible configuration options can be found in the Eleventy documentation [Zach Leatherman and Contributors 2021].

3.2 Working with Eleventy

It does not really matter whether there is already an existing project, which should be further developed with Eleventy, or if a completely new project should be started. Eleventy covers both, as it is very likely to work with an existing project structure as soon as the configuration is set up correctly. From this point on, the strengths shine through and there are very few limitations to creating static sites. One of the most useful features is without a doubt the option to run the project on a localhost server with the possibility of hot browser reload, which means as soon as a file gets changed and saved, the site will be rebuilt and the browser will be refreshed. This is a very convenient feature to speed up the development process and can be achieved with the command 'eleventy --serve'. As Eleventy allows to use many different markup languages and template engines in a single project, the best out of both worlds can be used.

It supports [Zach Leatherman 2021]:

```
# npm
npm install -g @11ty/eleventy

# yarn
yarn global add @11ty/eleventy
```

Listing 3.1: The commands for NPM and Yarn to install Eleventy globally.

```
module.exports = config => {

    // navigation
    config.addPlugin( require('@11ty/eleventy-navigation') );

    // shortcodes
    config.addShortcode('navlist', require('./lib/shortcodes/navlist.js'));

    config.addPassthroughCopy("src/styles");
    config.addPassthroughCopy("src/manifest.json");
    config.addPassthroughCopy("src/sw_register.js");
    config.addPassthroughCopy("src/service-worker.js");

    // Images
    config.addPassthroughCopy("src/icon.png");
    config.addPassthroughCopy("src/images");
    config.addPassthroughCopy("src/guides/images");

    // Materials
    config.addPassthroughCopy("src/hci.pdf");
    config.addPassthroughCopy("src/test");
    config.addPassthroughCopy("src/materials/en");

    return {
        templateFormats: ["html", "njk"],
        dir: {
            input: "src",
            include: "_includes",
            output: "_site"
        },
        passthroughFileCopy: true
    };
};
```

Listing 3.2: The Configuration of the HCI website in Eleventy.

- HTML
- Markdown
- JavaScript
- Liquid
- Nunjucks
- Handlebars
- Mustache
- EJS
- Haml
- Pug

3.3 Template Engines

Template engines are used to combine separated parts of design and content of an application. Templates include static designs that can be fed with different content in certain predefined areas. The strength of a template engine is for creating uniform looking websites without needing to build and reinvent the design over and over, because it only needs to be built once and it then can be reused in every single page [SEO-Analyse 2021].

3.3.1 Nunjucks

Nunjucks is a high performance JavaScript templating language and can be installed with npm. Nunjucks makes it easy to build templates for websites as it is possible to write normal HTML code with variables which get injected, including other templates, defining blocks which can be extended by all other files and it also offers the possibility to use macros, which basically are imports with parameters [Chris Coyier 2017].

Nunjucks was a very important part of the HCI website project and more about it and the experience using it in the project can be read in Section 4.3.

Chapter 4

HCI Course Website in Eleventy

The Human Computer Interaction (HCI) website source was to be updated to use a state of the art static site generator (SSG). Eleventy was chosen because it is a well known, simple and highly adaptable SSG. Furthermore, existing content can be easily transferred into an Eleventy project. Nunjucks was chosen as templating language to generate templates for the web pages to use.

4.1 Configuration

The Eleventy configuration file ".eleventy.js" was adapted to fit the project structure. Mainly there were many files and folders which just needed to be passed through to the output folder. Other than that a plugin and a shortcode[1] were added to the project needed for the generation of the navigation bar of the website. In order to install the plugin with npm it also had to be added into the "devDependencies" in the "package.json" file. Section 4.6 offers more information about the plugin and shortcode regarding the navigation bar.

4.2 Structure

The main hierarchy of the project consists of a root and four subfolders. The folder `src` includes all the project files for the build, `node_modules` contains the dependencies,`lib` holds the javascript files and `_site` is created automatically through the build process. The last mentioned folder is not included in the following tree since it is generated. The `_includes` folder inside of `src` is crucial, because all essential template files are located there.

[1]Shortcodes are small bits of code which generate content in a template [Leatherman and matt-auckland 2021].

```
project
├── lib
│   └── shortcodes
│       └── navlist.js
├── node_modules
│   └── ...
└── src
    ├── _includes
    │   ├── partials
    │   │   ├── footer.njk
    │   │   ├── htmlhead.njk
    │   │   ├── navbar.njk
    │   │   └── pagetitle.njk
    │   └── page.njk
    ├── exercises
    ├── guides
    │   ├── images
    │   └── ...
    ├── images
    │   └── ...
    ├── materials
    │   ├── en
    │   │   ├── he
    │   │   │   ├── logs
    │   │   │   └── videos
    │   │   └── heplans
    │   │       └── logs
    │   └── ...
    ├── schedule
    │   └── ...
    ├── styles
    │   └── ...
    ├── test
    │   └── ...
    └── index.html
```

The proposed site structure can be seen in Figure 4.1.

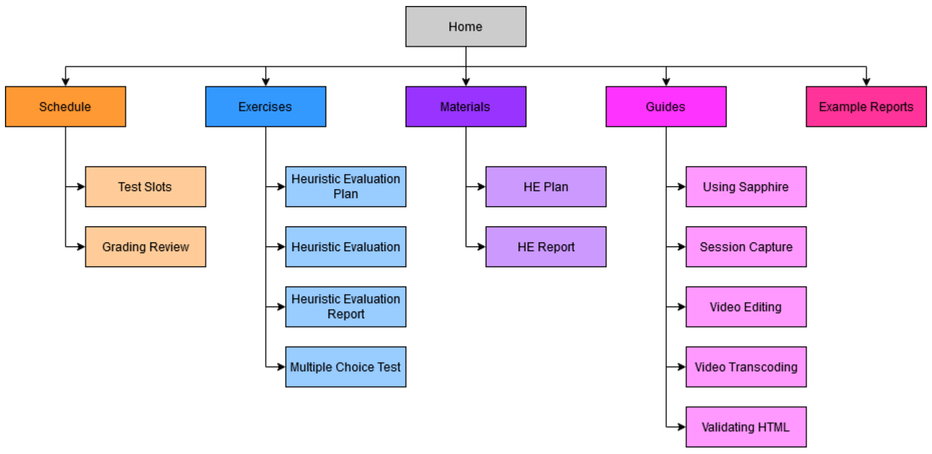


Figure 4.1: HCI Site Structure [The image was created by the authors of this paper]

```
<footer>
<span class="valid-xhtml">
<a href="http://validator.w3.org/check/referer">Polyglot XHTML5</a></span>
<span class="copyright">Copyright © Keith Andrews 2020.</span>
<span class="valid-css">
<a href="http://jigsaw.w3.org/css-validator/check/referer">CSS3</a></span>
</footer>
```

Listing 4.1: The content of the footer in our Nunjucks file.

4.3 Templates

For the templates Nunjucks was used as the templating engine. It enables the injection of a header, navigation bar, footer and also the whole content in the form of HTML files into the final static HTML page, which are generated.

The main template consists of standard HTML code and contains smaller template injections, which are other templates like the site header and footer. In Listing 4.1 the template for the footer can be seen.

Each template file contains a block which refers to its HTML file. The content of the corresponding HTML file inserted into this position during the build process. If there are more sections which should be modified, it would also be possible to define more blocks for different parts of the content. This project only needed one content block. An example of how to define such a block can be seen in Listing 4.2.

4.4 Relative URLs

Getting relative URLs in the projects to work required a small workaround. The relative URLs referring to other pages needed to be based on the root URL of the website in order simply navigate out of a subfolder. Of course, relative URLs could also dependant on the current path, but then the linked URLs are hard to read and understand while editing the content. This requirement for the relative root to be based on the domain root was a problem when deploying the website to a subfolder of a domain. In this case the relative URL root, normally indicated by a slash ("/"), would lead to the domain's URL which would not include the subfolder the source is actually in.

Based on this problem a HTML base tag had to be added to the project. This base tag modifies what the relative links use as the root URL [Mozilla and individual contributors 2020]. For example, if the website is located in a subfolder then the path to it can be entered into the base tag which later results

```
{% block content %}

  {{ content | safe }}

{% endblock %}
```

Listing 4.2: Snippet of the block where the HTML content is later inserted.

in the usage of the correct URL for the relative links in the project. To change this parameter easier a variable ("baseURL") was declared in the header of the main template file. In order for the path to work correctly the folder path has to be complete with a leading and trailing slash ("/"). The default value if there is not subfolder path is just the slash as it would be without the base tag.

4.5 PWA

One of the most important requirements was to add two features of a progressive web app (PWA) to the course site: Add to home screen (A2HS) functionality and offline availability. The A2HS functionality was achieved by adding a manifest to the project and the caching for offline availability was made using a service worker.

4.5.1 Manifest

The manifest delivers important information in form of a JSON file. It is the most basic component of a PWA, but necessary for every single website, which wants to have PWA features. As Eleventy would not know what to do with such a file by default, adding the manifest as a pass-through-file in the configuration of Eleventy must be done. After doing so the file is copied into the root of the generated website.

In Listing 4.3 the content of the file can be inspected.

Some properties are required to be added to the manifest to activate the A2HS functionality of the browsers, which support it:

- background_color
- display
- icons
- name / short_name
- start_url

4.5.2 Service Worker

The service worker for the project was used for the caching on the main site content, namely the HTML pages and their included content like images. This file is the only JavaScript (JS) code in the project, which is used in the website itself. It is registered in the main template (see Section 4.3) at the bottom of the body.

The service worker code was adapted from the example found on the Service Worker Cookbook [Mozilla and individual contributors 2019]. This specific one retrieves the content from the server as normal if the timeout, which was set for it, was not exceeded or the content is not in cache. If this is not the case then the content is served from cache by the service worker.

A small problem arose regarding the setting of the base URL tag as described in Section 4.4. As the service worker operates in a separate setting it can not retrieve the base URL settings in the current

```
{
  "short_name": "HCI Course Website",
  "name": "Human Computer Interaction Course Website",
  "description": "Website of the HCI Course on the Graz University of Technology",
  "start_url": "index.html",
  "background_color": "#ffffff",
  "display": "minimal-ui",
  "scope": "/",
  "theme_color": "#3367D6",
  "icons": [
    {
      "src": "icon.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ]
}
```

Listing 4.3: The content of the manifest, which enables PWA features and also the A2HS functionality.

```

...
    return (
        '<li class="navbar__item"' +
        (classList.length ? ` class="${ classList.join(' ') }"` : '') +
        '>' +
        (active ? `<a class="navbar__item--strong" href="${ entry.url.substring(1)
            }">` : `<a href="${ entry.url.substring(1) }">`) +
        entry.title + '</a>' +
        (childPages ? `<ul class="navbar__dropdown">${ childPages }</ul>` : '') +
        '</li>'
        );
    }
...
```

Listing 4.4: Generating HTML Navigation Bar by javascript

HTML file. To fix the URLs for the service worker as well the path of the service worker file had to be extracted and the base URL reconstructed from that. This extracted path is then added to each item in the list of files which are to be cached and ensures the correctness of the path to the files regardless of the root URL.

4.6 Navigation Bar

As described in the introduction a requirement was to build a navigation bar. The original site has no navigation bar. The navigation bar was created by the plugin "@11ty/eleventy-navigation", which provided a javascript code to generate the needed HTML code for a static menu. See Figure 4.4 to find the related javascript-code.

Finally, it was important to include the template at the correct position in the page.njk template. By using the correct embedding position the template could be written once but used repeatedly. The implementation is shown in Figure 4.5.

```
...

{% include "partials/htmlhead.njk" %}

<body>
  {# Header #}
  <header>

    {% include "partials/navbar.njk" %}

    {% include "partials/pagetitle.njk" %}
  </header>
  <div id="whole">
  ...
```

Listing 4.5: Embedding the Navigation bar in main layout file

```
 1  <!DOCTYPE html>
 2  <html lang="en">
 3      <head>
 4          ...
 5
 6          {# Base URL #}
 7          <base href="{{ baseURL }}" />
 8
 9          {# Stylesheet #}
10          <link rel="stylesheet" type="text/css" href="styles/standard.css"/>
11          <link rel="stylesheet" type="text/css" href="styles/report.css"/>
12          <link rel="stylesheet" type="text/css" href="styles/schedule.css"/>
13          <link rel="stylesheet" type="text/css" href="styles/navbar.css"/>
14
15          {# Manifest #}
16          <link rel="manifest" href="manifest.json" crossorigin="use-credentials">
17      </head>
18
19      ...
```

Listing 4.6: Linking all stylesheet files to HTML-header (the navbar.css was linked in line 13, which is below the other stylesheets, hence it will not be overwritten by the other stylesheets)

4.7 Stylesheets

Stylesheets (CSS) were used in order to format the HTML pages. At first, they were done inline, but this was only done due to testing-purposes. Later on, they were implemented using separate files in an own folder called "styles", which is located in the project's source folder. One requirement, regarding CSS, was to use the lecturer's prefabricated stylesheets as much as possible.

4.7.1 Embedding CSS into project files

It was important to write stylesheets as sustainably as possible, which means to write each CSS-class once and reuse it as often as possible in the whole project. In addition to that, since using a templating-engine, it was possible to link all stylesheet-files to the HTML header once and after generating the static site, it will be rolled out for all pages. See Listing 4.6 for the code.

```
 1  ul.navbar {
 2      width: 100%;
 3      display: flex;
 4      flex-wrap: wrap;
 5      position: fixed;
 6      top: 0;
 7      margin: 0;
 8      padding: 0;
 9      background: white;
10      box-shadow: 0px 3px 5px 0px rgba(50, 50, 50, 0.2);
11  }
12
13  ul.navbar a:link, a:visited {
14      color: black;
15      text-decoration: none;
16  }
```

Listing 4.7: Placing navbar container on the page

4.7.2 Navigation Styling

Part of the requirement was to implement a pure CSS solution for the Navigation Bar and not to use javascript. The Navigation Bar was delivered with partially javascript, which was changed to CSS during the progress of the project. In order to keep a clearer structure on the CSS-code, an approximation to BEM was used (See Section 4.7.4 for more information).

Figure 4.7 shows the basic container of the Navigation Bar. Therefore an unordered (bullet) list was used. It was expanded to fill the complete width of the document. Flexbox helped to get the list items in the correct position easier and to handle line-breaks. To make it always stick to the top of the page, the `position:fixed` property was added to the class. A box-shadow should help the user to optically distinguish between the Navigation bar and the actual webpage, since both have the same color theme. As shown from line 13 to 16 links were colored black without the standard underline. This contributed to aesthetics.

The next Figure 4.8 shows the styling of the menu buttons which were given as list-items. The standard bullet point was removed. The first class explicitly centres the text of the navigation buttons of order "1". The class which goes from line 12 to 15 makes the menu buttons interactable by mouse-hover. The pointer type was also considered in order to improve the user experience.

The final Figure 4.9 shows the development of the dropdown functionality in CSS. Lines 1 to 10 show the properties which were firstly necessary to create the dropdown and secondly to place it directly below the appropriate navigation button. The property `z-index` brought the dropdown on the highest layer and `position: absolute` forced each dropdown to stay at the given position. The third css-class, which goes from line 17 to 19, changes the dropdown state from `none` to `block` (visible).

4.7.3 Appearance of Navigation Bar

The final look of the Navigation by is shown in figure 4.2.

4.7.4 Block Element Modifier

Block Element Modifier (BEM) is one of many conventions, which approach to keep the CSS-structure efficient. BEM is considered as a long-term solution and works by splitting each class into three parts. At first the Block (B) what stands for a container, header, menu and so on. Then the individual Element (E) follows. This can be a menu item, list item etc. At last the Element (E) could have a Modifier (M) which could be size, disabled, fixed, checked and so forth. An example could be `button--state-success`.

```css
ul.navbar > li.navbar__item {
    text-align: center;
}

li.navbar__item {
    min-width: 200px;
    margin: 0;
    padding: 8px 0;
    list-style-type: none;
}

li.navbar__item:hover {
    cursor: pointer;
    background: rgb(189, 189, 189);
}
```

Listing 4.8: Formatting the menu buttons

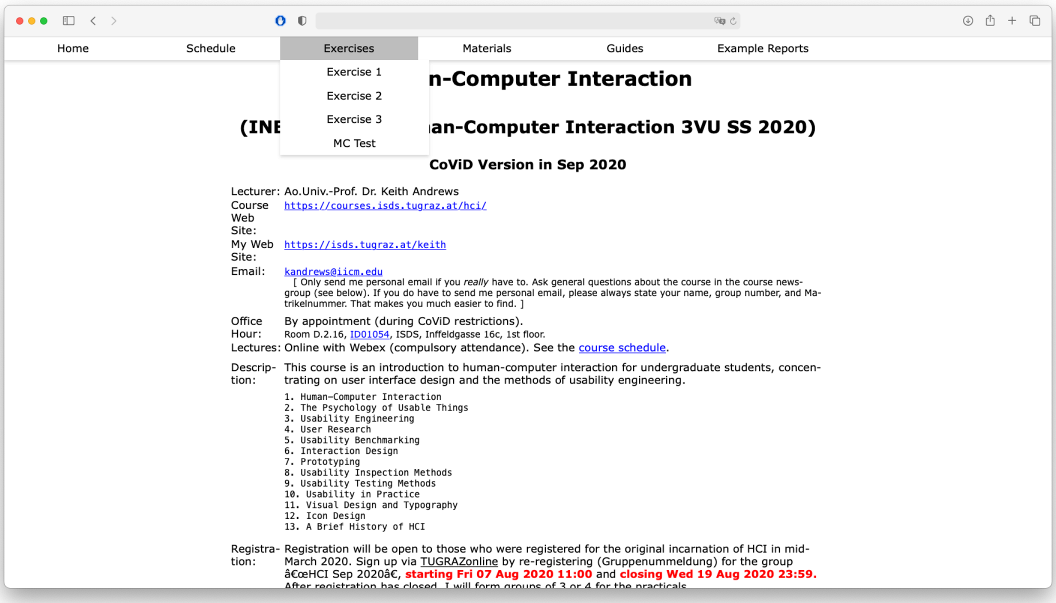
```css
ul.navbar__dropdown {
    display: none;
    list-style-type: none;
    margin: 8px 0;
    padding: 0;
    position: absolute;
    z-index: 1;
    background: white;
    box-shadow: 0px 3px 5px 0px rgba(50, 50, 50, 0.2);
}

ul.navbar__dropdown li.navbar__item {
    padding: 8px;
}

/* opening dropdown */
ul.navbar > li.navbar__item:hover > ul.navbar__dropdown {
    display: block;
}
```

Listing 4.9: The dropdown which appears on hover

The version used in this project was affected by personal preferences [*BEM - Block Element Modifier* 2021].



n-Computer Interaction

(INE an-Computer Interaction 3VU SS 2020)

CoViD Version in Sep 2020

Lecturer: Ao.Univ.-Prof. Dr. Keith Andrews
Course Web Site: https://courses.isds.tugraz.at/hci/
My Web Site: https://isds.tugraz.at/keith
Email: kandrews@iicm.edu
[Only send me personal email if you *really* have to. Ask general questions about the course in the course newsgroup (see below). If you do have to send me personal email, please always state your name, group number, and Matrikelnummer. That makes you much easier to find.]
Office Hour: By appointment (during CoViD restrictions).
Room D.2.16, ID01054, ISDS, Inffeldgasse 16c, 1st floor.
Lectures: Online with Webex (compulsory attendance). See the course schedule.
Description: This course is an introduction to human-computer interaction for undergraduate students, concentrating on user interface design and the methods of usability engineering.

```
1. Human–Computer Interaction
2. The Psychology of Usable Things
3. Usability Engineering
4. User Research
5. Usability Benchmarking
6. Interaction Design
7. Prototyping
8. Usability Inspection Methods
9. Usability Testing Methods
10. Usability in Practice
11. Visual Design and Typography
12. Icon Design
13. A Brief History of HCI
```

Registration: Registration will be open to those who were registered for the original incarnation of HCI in mid-March 2020. Sign up via TUGRAZonline by re-registering (Gruppenummeldung) for the group â€œHCI Sep 2020â€, **starting Fri 07 Aug 2020 11:00** and **closing Wed 19 Aug 2020 23:59.**
After registration has closed, I will form groups of 3 or 4 for the practicals.

Figure 4.2: Navigation Bar Final Look [The image was created by the authors of this paper]

Chapter 5

Conclusion

In conclusion it can be said, that a static site generator is a convenient technology to reduce the work of developers, although some experience is required to use it properly. Especially Eleventy provides many benefits, for instance the usage of various templating languages simultaneously. It is also delivered with the capability of zero-configuration, thus the process of setting up a project can be really fast.

Regarding this project Eleventy turned out to be a suitable lightweight solution to transform the original website into a new one which envelops features like for example an eligible user experience, even if the network connection is unavailable, and the deployment to a web server. Since Nunjucks was used, embedding separate stylesheets was unsophisticated, as it had to be included to the markup only once.

The course of the project also gave a good opportunity to examine the implementation of a navigation bar. Like in other parts of the project, the navigation bar only had to be included to the project once and a useful plugin provided with the package helped to generate HTML content through JavaScript. The stylesheets could be stored separately and were thus easy to manage.

As a result the overall the project involved various technologies which enabled the authors of this paper to explore and learn new technologies and skills. All features were integrated into the website, while also maintaining the integrity of the original site. The new features modernize the site and its look and feel which will hopefully delight future student generations.

Bibliography

BEM - Block Element Modifier [2021]. http://getbem.com/introduction/ (cited on page 18).

Chris Coyier [2017]. *Four Killer Features of Nunjucks*. 03 May 2017. https://css-tricks.com/killer-features-of-nunjucks/ (cited on page 9).

Digital Ocean [2020]. *A Brief Tour of the Eleventy Static Site Generator*. 02 Apr 2020. https://www.digitalocean.com/community/tutorials/js-eleventy (cited on page 7).

IONOS [2017]. *Static-Site-Generatoren: Mit Minimalismus zum eigenen Webprojekt*. 1&1 IONOS SE, 19 Dec 2017. https://ionos.de/digitalguide/websites/webseiten-erstellen/was-ist-ein-static-site-generator/ (cited on page 5).

Kalkman, Ben [2020]. *Static vs. Dynamic Websites: What are They and Which is Better?* 20 Oct 2020. https://rocketmedia.com/blog/static-vs-dynamic-websites (cited on page 5).

Leatherman, Zach and matt-auckland [2021]. *Template Shortcodes*. 30 Jan 2021. https://11ty.dev/docs/shortcodes/ (cited on page 11).

Mozilla and individual contributors [2019]. *Network or cache Recipe*. 11 Aug 2019. https://serviceworke.rs/strategy-network-or-cache.html (cited on page 14).

Mozilla and individual contributors [2020]. *<base>: The Document Base URL element*. 19 Dec 2020. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/base (cited on page 13).

Rinaldi, Brian [2021]. *Choosing the Best Static Site Generator for 2021*. 14 Jan 2021. https://snipcart.com/blog/choose-best-static-site-generator (cited on page 5).

SEO-Analyse [2021]. *Templates Begriffserklärung und Definition*. 31 Jan 2021. https://www.seo-analyse.com/seo-lexikon/t/templates/ (cited on page 9).

Zach Leatherman [2021]. *Eleventy Documentation*. 30 Jan 2021. https://www.11ty.dev/docs/ (cited on page 7).

Zach Leatherman and Contributors [2021]. *Configuration*. 27 Jan 2021. https://www.11ty.dev/docs/config/ (cited on page 7).