Implementing web accessibility to an existing web application

Master's Thesis in Technology
University of Turku
Department of Future Technologies
Software Engineering
2020
Juha-Pekka Jokinen

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

UNIVERSITY OF TURKU
Department of Information Technology

JUHA-PEKKA JOKINEN: Implementing web accessibility to an existing web application

Master's Thesis in Technology, 113 p., 14 app. p.
Software Engineering
November 2020

Web accessibility is becoming more and more important as societies around the world rely more and more on digital services. It enables the services to be used by many different kinds of users. It is starting to become a norm in public sector organisations, which sets up expectations and demands for web applications. When you have an existing educational web application like Sanako Connect, how do you go about implementing web accessibility into it?
That topic is explored through four research questions. First, as different countries and regions have regulations regarding web accessibility, and there are different web accessibility guidelines, what accessibility guidelines should be followed? Secondly, how those guidelines could be transformed into more concrete requirements?
The research questions become more practical from here on out. The third research question asks how to modify the development process so that accessibility will be thought of in future development and revisions of the software. This is vital because the application is in a constant flux of new features and refactoring, which means that web accessibility should be an integral part of development.
The final question is, what are the challenges and solutions when implementing the requirements to an existing application.
Background into web applications and web accessibility is provided to find answers to the first two questions. The last two questions are explored through the starting phases of a web accessibility implementation project. The implementation project will also validate answers for the first two questions, as the guidelines and requirements will guide the development process and implementation.
Even though the implementation details are unique to this one product, the challenges and solutions are general enough that they support previously discussed approaches to web accessibility. The most important takeaway is that there rarely are valid arguments for not including web accessibility into new projects from the start. Including it afterwards is always the harder path to take.

Keywords: accessibility, web accessibility

Contents

Chapter 1

Introduction

Sanako Connect is a language learning platform for schools and universities, developed by software vendor Sanako. It enables the teacher to run classes - divide the class into groups, assign and collect exercises, and speak with the whole class or individual groups or students. The students can do exercises, which primarily consist of reading or listening to something and recording yourself. When assigned to a group, the student can also speak with other group members.

Sanako Connect is a modern web application that runs in the browser. It can be used in computers and mobile devices, allowing the students to use their own devices and the teacher to run the class remotely or as a combination of present and remote students.

As the customers are primarily from public sector organisations around the world, there are different web accessibility requirements the application should fulfil. Broadly speaking, web accessibility means making an application usable to as large a group as possible. Making sure the application can be used through alternative input methods like mouse and keyboard, ensuring the application works with assistive technologies like screen readers and paying attention to design with contrast and colours are all part of this. Several countries have legislation on the subject. Often the legislation is connected to the guidelines and criteria maintained by the World Wide Web Consortium (W3C), which handles the standards related to the World Wide Web.

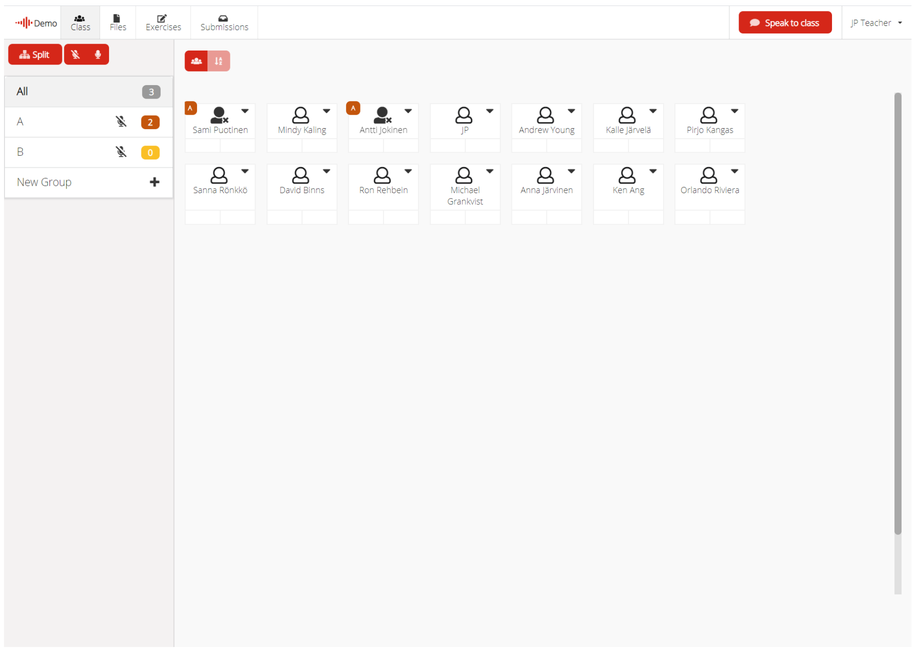
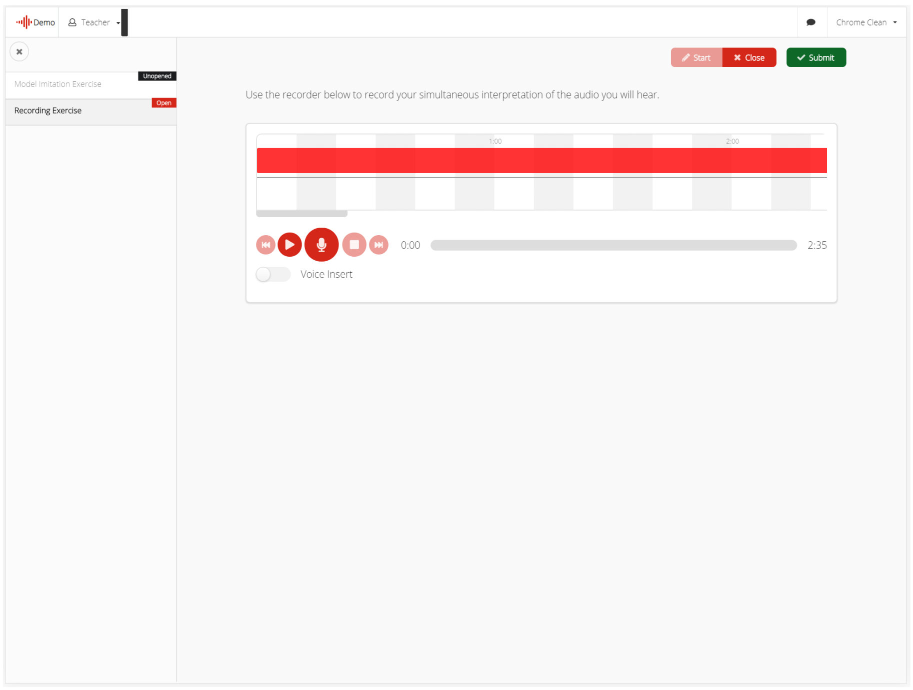


Figure 1.1: Sanako Connect's teacher application

As Sanako Connect is a normal, although complex, web application that ultimately consists of HTML, CSS, and JavaScript, in a perfect world there is nothing that prevents implementing best practices on web accessibility. In practice, this is not as simple. Accessibility was not discussed much at the inception of the project and is not part of the development process. Taking it into account when the product is already released requires resources and could require big alterations. New features are developed at a fast pace, and the software is modified constantly, which means that accessibility should be a part of the development process and not a one-off thing that could be implemented and then forgotten.

Apart from ethical considerations - that software, especially educational software, should be available to as large a group of users as possible - other factors support including accessibility.

First, there are the aforementioned laws and regulations. Sanako Connect is sold globally through a reseller network and is mostly targeted towards public sector organisations, which in some cases are subject to accessibility requirements. Looking at each country's



Use the recorder below to record your simultaneous interpretation of the audio you will hear.

Figure 1.2: Sanako Connect's student application

legislation separately is not feasible, which is why guidelines like W3C's Web Content Accessibility Guidelines (WCAG) [1] are important and something that could serve as the basis for web accessibility implementation.

Secondly, often web accessibility aligns with good, universal design in general. Following the guidelines on navigation and HTML markup, for example, would strengthen the usability of the application for everyone.

Thirdly, all the previous reasons contribute to the sales aspect. Fulfiling the criteria could be a deciding factor in some tenders. Moreover, although a good user interface is not a formal requirement anywhere, making the application more user-friendly is never a bad thing. Simply knowing the state of web accessibility in the application could be used as an argument in sales and marketing, and would allow Sanako personnel to answer questions on it.

However, even when there is a case for web accessibility and guidelines that can be followed, there are still complicating factors. For example, Connect uses several third-party user interface components. If they do not fulfil the criteria, replacing them is not always straightforward. Connect also relies heavily on real-time and recorded audio. Although WCAG discusses audio, some of the use cases are so complex that it is not clear how their criteria could be fulfiled. These use cases will be described in Chapter 4.

This thesis tries to answer the following research questions:

RQ1: What accessibility guidelines a globally sold educational software like Sanako Connect should follow?

RQ2: How can the accessibility guidelines be transformed into more concrete requirements?

RQ3: How to modify the development process so that accessibility will be thought of in future development and revisions of the software?

RQ4: What are the challenges and solutions when implementing the requirements to an existing application?

Chapter 1 is a general introduction to the main problem and motivation for this thesis.

Chapter 2 discusses web applications, starting from the basic elements of HTML, CSS, and JavaScript. Those are important for understanding web accessibility, which is why their history is also briefly explained. After that, the focus is on the features of modern web applications and development processes.

Chapter 3 focuses on web accessibility. After a historical overview, regulations on different regions are discussed. W3C guidelines such as Web Content Accessibility Guidelines and Authoring Tool Accessibility Guidelines are described, along with Web Accessibility Initiative – Accessible Rich Internet Applications, W3C's technical specification on accessibility.

Chapter 4 offers a walkthrough of how Sanako Connect works.

Chapter 5 is about what accessibility guidelines should be followed, and Chapter 6 focuses on the development process and actual implementation.

Chapter 7 discusses the answers and implications of the research questions, and Chapter 8 concludes the thesis.

.

Chapter 2

Web application architecture and development

Over the past three decades, simple and static web pages have morphed into complex and dynamic web applications, and browsers have become an important platform in itself. During the same time period, software development practices have changed also. Understanding web accessibility requires understanding how the web and developing for the web works. This chapter aims to do that.

2.1 Web foundations

This section will go through fundamental technologies that the Web is built upon. The World Wide Web Consortium (W3C) is related to those technologies and web accessibility, so its role is also explained. Finally, this section covers the browser, which is the platform for web apps.

2.1.1 Role of World Wide Web Consortium

When talking about Web technologies and standards, the World Wide Web Consortium comes up time and again. It is the main international standards organisation for the information system known as the World Wide Web (WWW), founded in 1994 and led by Tim Berners-Lee during its whole existence. Berners-Lee proposed the WWW back in 1989 when he was a researcher at European organisation for Nuclear Research (commonly known as CERN, Conseil européen pour la recherche nucléaire).

In addition to writing the first web browser, server, and page[2], he wrote the first three specifications for Web technologies. Those specifications were for Uniform Resource Locator (URL), HyperText Transfer Protocol (HTTP), and HyperText Markup Language (HTML)[3]. URL specifies how to reference a resource on a network. An example of an URL would be https://utu.fi. HTTP specifies a protocol on transmitting and retrieving those resources. Think of a browser loading a web page of the previous example. Finally, HTML specifies the structure of a single web page. HTML will be more thoroughly explained in Section 2.1.2.

The first web page ever is still available[1], although that page is a recreation of the 1992 version of the page. It was the earliest version CERN could find when putting the page back online[4].

That page is written on what is now retroactively called HTML 1.0, and modern browsers can load this 28-year-old web page just fine. For a long time, W3C oversaw new versions of HTML, before transferring that responsibility to Web Hypertext Application Technology Working Group (WHATWG) that operates outside W3C and consists of browser vendors like Mozilla, Apple, and Opera. W3C and WHATWG had been publishing competing standards for years. Originally WHATWG was formed when W3C was not seemingly interested in adding new features to HTML at a constant pace[5].

Apart from HTML, W3C maintains dozens of drafts and standards. Among these

[1]http://info.cern.ch/hypertext/WWW/TheProject.html

is another essential Web technology, Cascading Style Sheets (CSS). They also maintain Web Accessibility Initiative – Accessible Rich Internet Applications (WAI-ARIA), a technical specification meant for improving the accessibility of web pages, and Web Content Accessibility Guidelines (WCAG). Those will be explained in Chapter 3.

2.1.2 Web page foundations

Although recent years have introduced web development tools and abstractions built on top of the HTML, CSS, and JavaScript languages, in the end, what the browser takes in for rendering web pages and applications is still those three languages. The fourth language to natively run in the browser is WebAssembly, which was made into an official standard 2019[6]. Although its use cases in performance-requiring applications are important, it is not nearly as common as the three other languages. No matter what specific technologies or frameworks a developer is using, knowing those three are foundational skills.

HTML

HTML stands for HyperText Markup Language, and it is used to mark the structure of a web page. An HTML document consists of nested elements that define the structure of the page.

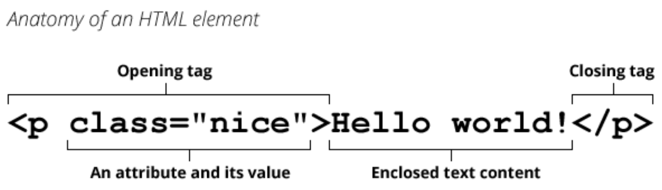

Anatomy of an HTML element
Opening tag
Closing tag
<p class="nice">Hello world!</p>
An attribute and its value
Enclosed text content

Figure 2.1: Anatomy of an HTML Element. From https://developer.mozilla.org/en-US/docs/Glossary/HTML

The element consists of an opening and closing tag. Tags can be extended with attributes, which provide information to the browser as to how it should interpret the element[7]. Besides text, HTML can include other types of content as well, such as links, images, audio, and video content.

An HTML file can also have references to other assets the browser should fetch and process, like CSS and JavaScript files.

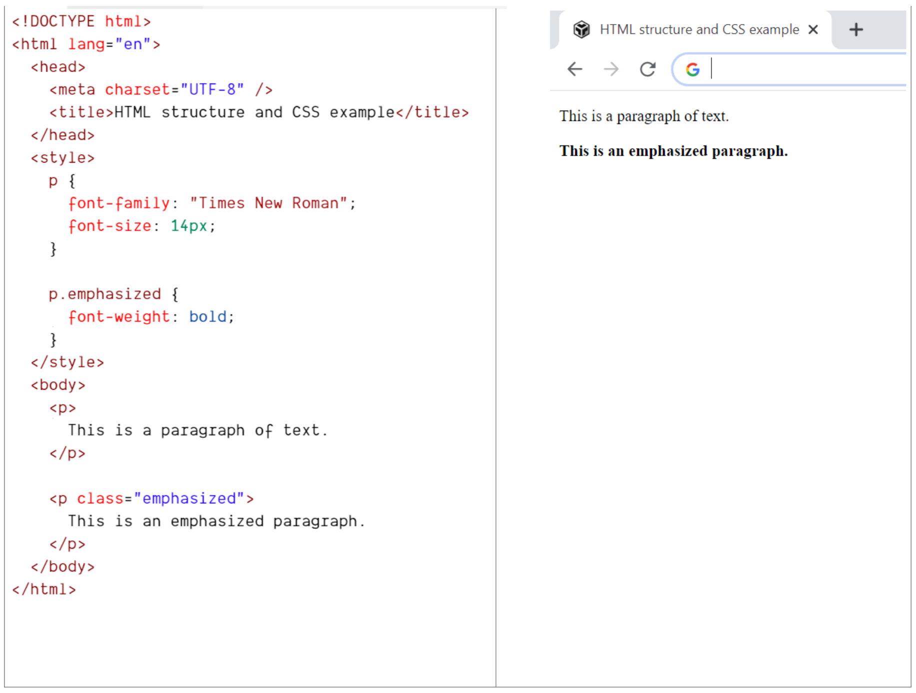
There are many ways to achieve the same outlook with different HTML structures and elements. Along the years HTML has gotten support for different semantic elements that have an intended purpose, like using a <button>element for buttons or putting the main content of the page inside a <main>element. That is called semantic HTML. However, nothing prevents using a different element as a button and maybe using JavaScript for its functionality. For the basic user, the outcome is the same, but this could mean that for those using alternative input or navigation methods cannot interact with the element.

CSS

CSS is short for Cascading Style Sheets. Whereas HTML is used for structure, CSS is used for presentation. CSS rules tell the browser how to display a certain element or a certain kind of element. This separation of structure and layout means that it is possible to alter how the same HTML document looks like by just modifying the associated CSS rules.

CSS rules consist of selectors and properties. Selectors are used to select what elements the following properties should be used for. The properties then will dictate how the browser should display that element.

In the early days of the Web, there was no way to alter how HTML documents looked. During 1994 and 1995 there were competing proposals on how to style HTML. CSS had one distinguishing feature: “It took into account that on the Web, the style of a document couldn’t be designed by either the author or the reader on their own, but that their wishes had to be combined, or cascaded, in some way; and, in fact, not just the reader’s and the author’s wishes, but also the capabilities of the display device and the browser.”[8]



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>HTML structure and CSS example</title>
  </head>
  <style>
    p {
      font-family: "Times New Roman";
      font-size: 14px;
    }

    p.emphasized {
      font-weight: bold;
    }
  </style>
  <body>
    <p>
      This is a paragraph of text.
    </p>

    <p class="emphasized">
      This is an emphasized paragraph.
    </p>
  </body>
</html>
```

This is a paragraph of text.

This is an emphasized paragraph.

Figure 2.2: On the left: a simple HTML document with two inline CSS rules. The lower HTML tag ‘p’ has an attribute ‘class’ with the value of ‘emphasised’. The corresponding CSS rule has a selector that matches all p elements with the class ‘emphasised’. On the right: how the browser displays the page.

JavaScript

The third cornerstone of the Web is JavaScript. With HTML giving the structure and CSS providing the looks, JavaScript is used for interactivity and all kinds of functionality in web pages. Unlike the name says, it does not have anything to do with Java, the programming language[9]. The official standard is called ECMAScript. JavaScript started in 1995 as a scripting language for browsers.

Declaring something as the most popular is often an ambiguous statement. However, on GitHub, the world’s largest source code host, JavaScript has been the most popular language at least for the past five years[10]. Although developed for the browser, nowadays, JavaScript can also be used in a variety of environments. A popular environment for running JavaScript on the server and elsewhere is called Node.js. It is built on Google’s

V8 JavaScript engine, which is also used in Google's Chrome browser.

When used inside a browser, what can be done with JavaScript depends on what features and interfaces the browser exposes for JavaScript.

It is possible to manipulate HTML and CSS in the browser with one of those interfaces. This happens through an API called Document Object Model (DOM). It creates a tree-like structure of the HTML which can be navigated and elements and content added, modified, and deleted[11].

Among other things, JavaScript has popularised a technique in web development where only parts of the web page (or just data) can be reloaded instead of the whole page. Before that, any action the user took, like clicking a link, would mean reloading a new HTML page from the server. With a set of technologies called Asynchronous JavaScript and XML (AJAX), data can be fetched from the server without needing to reload the page itself[12]. As discussed in Chapter 2.2, this is essential for modern-day web applications. This is also relevant for web accessibility since the page user has loaded can change on its own without user interaction.

2.1.3 Role of the browser

For the average user, the browser is the way to access the Web. It brings the aforementioned three first Web specifications together. It is used to fetch resources located at a certain URL. It does this through HTTP. Furthermore, what it fetches are most likely HTML, CSS, and JavaScript files, which it then handles to display a web page for the user.

During the 30 years since Berners-Lee created the first browser, browsers have become vastly more capable, adding new features and APIs that developers can use for a myriad of purposes. This has made it possible to built powerful and complex applications that run in the browser. Mobile devices with their browsers have appeared next to desktop computers and laptops.

Today, browsers compete against each other with aspects like speed, privacy, and security. Currently, Google Chrome is dominating with its 63% global market share. Far behind on the second place is Apple's Safari with 18%[13].

Even though there are bodies for standards, as mentioned in the previous section, browser vendors can still implement those standards in different ways, leave something unimplemented, or develop something on their own. Differing implementations make it harder to develop a feature, and in some cases, all features just cannot be supported in all browsers. An example of this is discussed in Section 4.2.2.

Often the user can use another browser - if they want to take up that effort. Sometimes there are restrictions for doing that, like organisation IT or security policies. Sometimes it is the devices themselves. For example, Apple enforces browser vendors to use Safari's rendering and JavaScript engine on iOS and iPadOS[14],. This means that on those platforms, the user is locked to the features Safari has implemented.

2.2 Modern web applications

Whereas the previous section went through the basics of web and web development, this section will focus on the current state of web app development. Common practises and categories of tools are discussed first. After that, as the term 'web development' encompasses an enormous field, the focus will be on concepts more relevant to web accessibility. Single-page applications, front-end frameworks, and functionality on mobile are gone through in more detail.

2.2.1 Development tools

As the browser has no difficulty opening a web page coded almost 30 years ago, it is perfectly possible to create web pages today with the same method as 30 years ago. All that one needs is a text editor to edit and save HTML.

In practice, there are so many things to consider when developing for the web today that specialised tools are used in every aspect of work.

Package Managers

Package manager is a tool for importing third-party software into your project and for maintaining external dependencies. One of the most common package managers in web development projects is npm, which is the default package manager of Node.js. Not only a package manager, it is also a repository of public and private packages. The npm registry is the largest package registry in the world[15]. Once a package is available in npm's registry, it is possible to add it to your project by using npm's command-line tool.

This makes adding external code easy, as dependencies associated with a project can be automatically installed and updated. There is a good chance that an open-source package exists for required functionality in your project, speeding up development as you do not have to re-invent the wheel. This can be some JavaScript function you need, or a user interface component. The downside is that apart from possible security issues - the package itself can be malicious, or have its own dependencies which might have vulnerabilities - there is no guarantee on the quality of the code in the package. If you need a dropdown menu component, for example, it might or might not fulfil web accessibility standards.

Build and automation tools

Generally speaking, build tools are used to transform and concatenate the code you write and optimise it for the web environment. Like stated in Section 2.1.2, it is HTML, CSS, and JavaScript that the browser uses to construct the web page. But it is possible to write web applications without directly using them. For example, SASS (Syntactically Awesome Style Sheets) is a CSS extension language that has features CSS does not have. A developer could use the features that help them with their work and write SASS instead of CSS, and then use a build tool to compile that SASS code into CSS code.

With JavaScript, it is, for example, possible to use a superset language of JavaScript like TypeScript and then compile that into JavasScript for the browser. As the support for JavaScript features also differs between browsers, it is possible to write your code with the latest standard, and then use a transpiler (source-to-source translator) like Babel that produces code that older browsers can run.

These tasks would be impossible without automation. Build tools can also be used to run other kinds of tasks - testing and formatting the code, for example. Module bundlers are also commonly used with JavaScript. They can take your code and its dependencies and output a single JavaScript file.

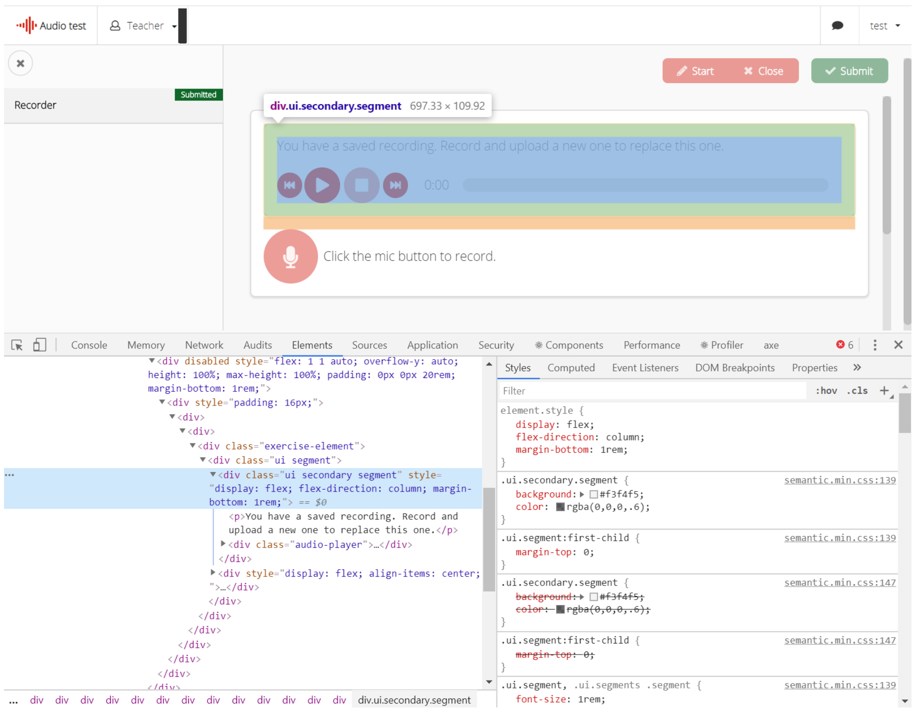
Browser developer tools

The browser itself is also an important web development tool. Since web developers are developing pages and apps for the browser, it is understandable that the result will be viewed on the browser. Nevertheless, apart from that self-evident use, browsers today include a variety of development tools, and can also be extended to incorporate new functionality.

Viewing the source code of any HTML page is possible on any browser, but development tools also help with viewing what the browser has rendered. As will be discussed in Section 2.2.2, these can be two very different things.

2.2.2 Single-page applications

As mentioned in Section 2.1.2, JavaScript popularised a technique for loading parts of the web page from the server instead of loading the whole page. The traditional way of requesting a whole HTML page from the server on each user interaction is the basis of a multi-page application (MPA). This means that any data processing is done on the server, an HTML page is created, and this page is then served to the browser during an HTTP



```
▼<div disabled style="flex: 1 1 auto; overflow-y: auto;
height: 100%; max-height: 100%; padding: 0px 0px 20rem;
margin-bottom: 1rem;">
  ▼<div style="padding: 16px;">
    ▼<div>
      ▼<div>
        ▼<div class="exercise-element">
          ▼<div class="ui segment">
            ▼<div class="ui secondary segment" style=
            "display: flex; flex-direction: column; margin-
            bottom: 1rem;"> == $0
                <p>You have a saved recording. Record and
                upload a new one to replace this one.</p>
              ▶<div class="audio-player">…</div>
              </div>
            ▶<div style="display: flex; align-items: center;
            ">…</div>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
... div div div div div div div div div div div div div div.ui.secondary.segment
```

```
element.style {
   display: flex;
   flex-direction: column;
   margin-bottom: 1rem;
}
.ui.secondary.segment {                    semantic.min.css:139
   background: ▶ #f3f4f5;
   color: rgba(0,0,0,.6);
}
.ui.segment:first-child {                  semantic.min.css:139
   margin-top: 0;
}
.ui.secondary.segment {                    semantic.min.css:147
   ~~background: ▶ #f3f4f5;~~
   ~~color: rgba(0,0,0,.6);~~
}
.ui.segment:first-child {                  semantic.min.css:147
   ~~margin-top: 0;~~
}
.ui.segment, .ui.segments .segment {       semantic.min.css:139
   font-size: 1rem;
```

Figure 2.3: Chrome developer tools. The information is related to the currently open web page.

request. All that is left for the browser is to render the HTML given by the server.

Single-page applications (SPA), on the other hand, run in the browser. The initial HTTP request serves an HTML page, which could otherwise be empty but contains references to the necessary JavaScript and CSS pages. JavaScript then takes over, dynamically rendering the HTML for the application and handling all application logic. User interactions like mouse clicks are handled in the web app inside the browser. They could result in HTTP requests for fetching new data, but the page itself is not reloaded at any point.

2.2.3 Front-end frameworks

The concepts discussed in the two previous sections collide with front-end frameworks, which emerged during the 2010s as a way to build web applications[16]. Single-page applications and the possibility to easily bring in third-party components into your project are a few enabling factors for them.

A framework in software development is a ready-made set of tools that provides a way to build applications or parts of them. Front-end refers to the client, which in this case is the browser. So a front-end framework provides a way to build web applications with JavaScript. Although building a web application is possible without any external packages or frameworks, front-end frameworks provide solutions for many repetitive tasks and offer abstractions over lower-level code.

What any single front-end framework covers depends on the framework, but this could mean things like data and state management and DOM manipulation. Usually, the frameworks can also be extended.

When working with a front-end framework, it is possible to build a whole application without directly writing any HTML or CSS code. React, developed inside Facebook and released as open-source, is currently the most popular front-end framework[17][18].

When writing user interface components with React, instead of HTML the user interfaces are described with JavaScript or more commonly with JSX, a syntax extension to JavaScript[19]. The result will then be modular and nested React components which in the browser will be translated into the DOM. Moreover, although normal CSS files can be used without problems, there are several solutions for writing CSS with JavaScript[20].

As React components can be easily distributed as libraries through npm, this means it is possible to use 3rd party code in your project without knowing anything about the actual HTML markup the library outputs.

An annual accessibility analysis by WebAIM found interesting correlations between front-end frameworks and problems with accessibility. The analysis examines one million home pages. In the 2020 edition, pages using React had 5.7% higher error percentage than the average page [21]. This does not mean that the framework itself causes these errors, as there is nothing intrinsic about React that prevents web accessibility. One possible reason could be React's popularity coupled with the aforementioned abstraction of HTML and use of 3rd party libraries.

2.2.4 Functionality on mobile devices

From the perspective of the web, the last two decades can be seen as the slow rise of mobile devices. During this time, devices, standards, and connection speeds have all evolved. Limited by restrictions like small screen size and lack of common standards, there have been different approaches for bringing the web to mobile devices.

One of the early approaches was Wireless Application Protocol (WAP), a standard introduced in 1999 that made it possible for dedicated browsers on mobile phones to access WAP sites. These sites needed to be coded with Wireless Markup Language (WML) instead of HTML, which meant that the WAP sites needed to be designed separately from normal HTML web sites.

The situation is different now. Globally almost 52% of internet traffic comes from mobile devices[22].

Two platforms dominate the market, Google's Android and Apple's iOS (and its tablet variant iPadOS). Screen sizes and resolutions have increased, as have connection speeds. Touch screens become common with the iPhone in 2007. Web standards like HTML, CSS, and JavaScript can be used to build web sites that display differently on mobile and desktop, and mobile browsers are capable of practically the same things as their desktop equivalents.

Mobile-first design

Although the same web page can be displayed on the desktop and mobile browser, the page still requires some work from the developer so it can work just as well on a small phone and a huge monitor, and also that it can be used effortlessly with a touch screen instead of a mouse. Responsive and adaptive web design are methods for making it possible. The terms are related but slightly different. Responsive design means that the available browser width and height determine how the page will look, and resizing the page will make the content re-adjust itself dynamically. With adaptive design, several pre-made layouts are displayed within a certain width/height range.

Progressive enhancement and graceful degradation are design philosophies that are also related closely to web page functionality on mobile and older browser. With progressive enhancement, the standard functionality of the site is brought to as many users as possible. Then, this baseline experience is enhanced for those users with the most modern and powerful browsers. Graceful degradation starts from these most modern browsers and provides fallbacks and the essential content for users with older browsers[23].

Mobile-first is a strategy related to progressive enhancement. According to mobile-first, the design should start with the mobile and then be expanded to work on other environments like the desktop also.

Progressive Web Apps

Both iOS and Android have their native applications that are distributed through the platform vendors' distribution systems, the application stores. Native applications have access to all the platform APIs and UI components.

Web applications, on the other hand, are always at the mercy of what the browser supports. The situation has gotten better during the last years, as the preceding paragraphs tell.

In some case, web apps can rival native apps in functionality and speed. Progressive Web Apps (PWA) are a category of web apps that can be installed on some platforms when they fulfil certain criteria, like being served over HTTPS (HyperText Transfer Protocol Secure, encrypted version of HTTP) and working offline. On mobile devices, this means that a PWA can be installed to provide a more native-like experience. Once the PWA is installed, it gets its own icon on the device, and once launched through the icon, the PWA launches in full-screen mode without the normal browser interface elements.

Since the same PWA works on both iOS and Android and the desktop, they have in some cases replaced the need to develop native applications. Another possible benefit is that by using them, the application store can be bypassed.

There are several business cases where using a PWA has proven to be a better option than a native app or a regular web app. Twitter started serving a PWA to all its mobile users in April 2017, with the result that data usage was reduced and user engagement increased[24].

Chapter 3

Web accessibility

This chapter goes through the what, why, and how of web accessibility. It starts with 'what', going through accessibility in general before narrowing down to web accessibility. The 'why' consists of going through relevant legislation and reasons outside of legislation to take web accessibility into account. The 'how' is about the most important guidelines and implementing them.

3.1 Accessibility

Accessibility means ensuring people with different kinds of disabilities can live their lives just as well as anyone else. There hardly is any aspect of life or society that could not be thought from this perspective.

The history of accessibility consists of different attitudes towards disabilities, disability rights movements, their efforts becoming legislation and policies, and, inseparably, technology, that both causes and solves accessibility issues.

One definition of accessibility is found from the United Nations' Convention on the Rights of Persons with Disabilities (CRPD). It is a treaty signed in 2007 and "intended to protect the rights and dignity of people with disabilities". Accessibility is one of the guiding principles of the Convention. Article 9 begins:

"To enable persons with disabilities to live independently and participate fully in all aspects of life, States Parties shall take appropriate measures to ensure to persons with disabilities access, on an equal basis with others, to the physical environment, to transportation, to information and communications, including information and communications technologies and systems, and to other facilities and services open or provided to the public, both in urban and in rural areas."[25]

Several countries and regions have legislation on accessibility based on CRPD. This will be discussed more in Section 3.4.

3.1.1 Assistive and adaptive technologies

Assistive technology is an umbrella term for all kinds of assistive products and services that maintain or improve their users' functioning and independence. Examples of assistive products are hearing aids and wheelchairs[26]. Adaptive technology is a subcategory of assistive technology. It refers to technology specifically designed for people with disabilities. In contrast, assistive technology can be anything that improves the life of its user, whether off-the-shelf or custom-made[27].

The line between "normal" and assistive technology is unclear. As Hendren argues in an interview by Rosen (2013), what technology would not be assistive?[28] She further points out that assistive technologies are often borne out of medical aids, and their users are commonly seen as medical cases instead of people with expanded needs. She argues that this means that assistive technologies are often left outside of design attention.

She illustrates this point with eyeglasses, which have crossed the line of being merely a medical aid and become a fashion accessory. As the writer of this thesis could not write or even see anything without wearing glasses, it is hard to argue against her point about the blurry line between normal and assistive technology.

3.1.2 Universal design

The previous section discussed seeing assistive technology as a purely medical aid. The broader implication is the view that accessibility is something extraneous. Although not specifically related to assistive technology, universal design takes the opposite stance. According to universal design principles, things should be designed to be accessible to all people, at the same time thinking about the design and aesthetic values. Considering diverse needs and abilities is not a special requirement. Instead, it is good design that everyone benefits from[29].

Universal design has sprung from architecture, from the accessibility of homes and other buildings, but has since transformed into a broader social movement. It has been applied to education as well. Universal design for learning takes into account the diverse backgrounds and ability levels of the students with schools, classrooms, and curriculums that provide resources for all students. This means making the physical environment as well as information technologies accessible[30].

3.2 What is web accessibility

The web itself is barely 30 years old, and the concept of web accessibility is a few years younger than that. It can be seen as a subcategory of computer accessibility, which means lowering the barriers of computer use in general. Computer accessibility was studied and thought of at least already in the 1980s, like in a study from 1984 that analysed problems that disabled people faced with personal computer hardware and software[31].

Instead of computers in general, web accessibility deals specifically with the ability to access and interact with web pages regardless of disabilities and impairments. This is achievable if the website's technical implementation is flawless, the user interface is easy and intuitive, and the content is clear and easy to understand.[32] This means support from websites and web developers, browser vendors, and makers of assistive technologies is required. It should be noted that technical details are only one part of the equation.

Internet is practically ubiquitous today and still becoming more important to everyone, as measures caused by 2020's corona pandemic have shown. Remote work, remote school, and remote everything is not for everyone if the services are not accessible by everyone. This will be discussed more in Section 3.3.

3.2.1 History of web accessibility

"The emergence of the World Wide Web has made it possible for individuals with appropriate computer and telecommunications equipment to interact as never before. It presents new challenges and new hopes to people with disabilities. As part of its commitment to realise the full potential of the Web, the Consortium has been promoting a high degree of usability for disabled people, by following the development and encourage an ongoing discussion in the area."

- Excerpt from W3C Newsletter, September 1996, written by Tim Berners-Lee[33]

Like with so many things associated with the web, the history of web accessibility is very much related to W3C. As early as 1994, in the Second International Conference on the World Wide Web, Tim Berners-Lee wrote notes on accessibility. A few years later - after the newsletter above - a decision was made to set up a stand-alone initiative on accessibility inside W3C.[34].

In 1997, several actions happened. The aforementioned initiative, Web Accessibility Initiative (WAI), was launched. HTML4.0, the first version of HTML to specifically mention accessibility, was released. And a bill was proposed in the United States to update 1986's ineffective Section 508 Amendment, which was legislation related to electronic and information technologies.[34]

Following trajectories from those three actions, we can track the progress of crucial web accessibility elements from those early days to this moment: guidelines, HTML markup, and legislation. HTML has already been described in Section 2.1.2. Guidelines and legislation will be handled later in this chapter.

3.2.2 Disabilities, assistive technologies, and adaptive strategies

Web accessibility encompasses every disability that can affect access to the web. This includes but is not limited to auditory, cognitive, neurological, physical, speech, and visual disabilities. A typical example of implementing web accessibility is thinking about how blind people could use the site in question. Although they are an important group, they are just a small portion of people who can be helped by web accessibility, and a small portion of people with visual disabilities. Statistically, around 20% of the population need web accessibility in one form or another. The biggest single group benefiting from it are people with cognitive disabilities[35].

However, it is also important to remember that web accessibility benefits everyone, for example, people with temporary disabilities such as a broken arm or situational limitations like bright sunlight[36]. This aligns web accessibility with universal design mentioned in Section 3.1.2 - it is something everyone benefits from.

This is an important point to keep in mind when implementing web accessibility: that the ultimate goal is to help people use the web and not to check off items from guideline checklists. Although there is correspondence between certain disabilities and accessibility features and assistive technologies, everyone - not just disabled people - uses the web in their own way, and the designer or developer cannot know that. Some use assistive technologies, which means using specialised hardware and software, like a screen reader. Some use adaptive strategies, techniques that are possible by adjusting the operating system or browser settings.

The following descriptions of disabilities list common barriers of use, accessibility features, and adaptive technologies and adaptive strategies commonly associated with the disability in question.

Auditory Auditory disabilities refer to hearing losses in one or both ears. The hearing loss can range from hard-of-hearing to deafness. In some cases, people can hear, but not enough to understand all speech, especially with background noise. Possible barriers include audio content without captions or transcripts and media players that do not display captions or allow changing their font size or colour [37]

Accessibility features related to auditory disabilities include captions, which is a text with a verbatim recording of any speech and relevant auditory information in the audio, and transcripts, manuscripts that contain the recording of any speech with important auditory information.[38]

Cognitive Cognitive disabilities contain a wide range of different disorders that may affect how well people hear, move, see, speak, and understand information [37]. Dyslexia, the most common learning disorder, is one example. It affects 10-15% of the school-age population and is characterised with difficulties in accurate word recognition and by poor spelling and decoding abilities.[39]

Possible barriers include complex navigation mechanisms and complex sentences, and long passages of text without images or other ways of highlighting the context[37].

Accessibility features that can help include customisable fonts and colours and easy-to-read text. Assistive technologies and adaptive strategies include pop-up and animation blockers and reading assistants that change the presentation of the content[38].

Physical Physical or motor disabilities include weakness and limitation of muscular control. Examples include arthritis, fibromyalgia, and repetitive stress injury. Possible barriers include missing keyboard support, controls that do not have text alternatives, and insufficient time limits to complete tasks, like filling out a form[37].

Accessibility features include consistency and predictability in labelling functions and controls, multiple navigation mechanisms, and descriptive titles and labels. Assistive technologies and adaptive strategies include alternative input methods - keyboards, custom keyboards, sip-and-puff switches, trackballs, voice recognition and eye tracking, and keyboard navigation[38].

Speech Speech disabilities mean in this context that the individual's speech is hard to understand by others or a voice recognition software. Barriers include web services that offer interaction with voice only[37].

Visual Visual disabilities mean a range of vision loss in one or both eyes. Mild or moderate vision loss is the low vision, while substantial vision loss is blindness. One category of visual disability is colour blindness, the inability to perceive between certain colour combinations or even any colour. Barriers include images and controls without text alternatives, content that cannot be resized, video that does not have an alternative presentation, and missing keyboard support[37].

Examples of accessibility features are audio descriptions, customised fonts and colours, and screen magnification. An example of assistive technology is a screen reader that converts content to other formats like text-to-speech or Braille[1][38].

3.3 Why web accessibility is important

The CRPD defines access to the Web as a basic human right. Apart from legislative demands stemming from CRPD, there is a strong case for web accessibility. One theme that has occurred throughout this thesis is that accessibility benefits everyone, not just disabled people. Temporary disabilities and situational impairments were mentioned in Section 3.2.2. Examples are abundant: people with a slow connection or limited or expensive bandwidth benefit from a fast web site, and people using devices like smartwatches or smart TV's benefit if the site can adapt to their screen and input method[36].

[1]Braille is a system that uses raised dots for letters of the alphabet, allowing a tactile way of reading and writing.

According to Eurostat, 87% of individuals living in the Euro area used the internet in 2019. In 2008, that figure was 62%. Even Bulgaria, the lowest country in the 2019 statistic, now has 68% usage [40]. From the individual's point of view, the Internet might be the only way to access services or contribute to society, depending on where one lives, geographically and mentally. Take voting, for example. In theory, enabling web-based voting makes participating easier and possible to a larger group than traditional voting. A study made in Norway revealed issues in usability and accessibility of several prototype systems[41]. Web-based voting will not help if the voting system itself is the barrier.

There are also valid business reasons in favour of web accessibility. Although the return on investment (ROI) is sometimes difficult to measure, there is also the cost and risk from inaction. Potential benefits include an improved brand image, extended market reach, and less risk for legal actions.[42]

A study conducted in 2014 looked at private sector organisations from three different sectors and tried to find motivations and business impact for their web accessibility projects. [43]. The study divided motivations into three categories: economic, social, and technical. It is noteworthy that the successful projects in the study reported an increase in image, customer loyalty, and web site traffic. In most cases, there were more advantages than disadvantages. Also important is the knowledge that several projects failed, sometimes simply because management or marketing department did not support the project, or it was perceived that web accessibility did not fit into the company's design requirements.

3.4 Laws and policies

163 countries have signed the Convention on the Rights of Persons with Disabilities. It is an important treaty since it is the first legally binding international instrument that aims to guarantee that the signatories will protect the rights of disabled people.[44] The treaty has several obligations, including adopting legislation and administrative measures

to promote the human rights of persons with disabilities and ensuring that the private and public sector and individuals respect the rights of persons with disabilities.[45]

Not all countries have legislation that deals specifically with web accessibility. If it exists at all, it can be on its own or a part of broader discrimination legislation, with different scopes and levels of obligation. For example, the list of international laws and policies on WAI's website has 40 items concerning 23 countries (including the European Union), ranging from recommendations to accessibility laws and mandatory policies. Furthermore, each piece of legislation can cover either the public sector, the private sector, the government, or any combination of those. 17 items are based or derived from WCAG guidelines[46].

To keep things in scope and relevant to Sanako Connect, the focus in this section will be on the United States and European Union, with a relevant example from Finland.

3.4.1 United States

Since the United States is a federal republic, it has legislation on both federal and state/territory level. The following is not a comprehensive list; instead, it goes through the most relevant legislation on web accessibility.

Sections 508 and 504 of the US Rehabilitation Act of 1973 Section 504 covers organisations like postsecondary entities, federally funded projects and K-12 schools, and it is meant to ensure people with disabilities cannot be discriminated against.[47]

Section 508, mentioned in Section 3.2.1, prevents the federal government from leveraging and/or procuring electronic and information technology goods and services that are not designed to be accessible.[47]. Additionally, other institutions and even state governments apply the standards of Section 508 into their own policies[48].

In 2017, the information and communication technology requirements in Section 508 were updated, and Web Content Accessibility Guidelines were incorporated into the legislation[48].

Individuals with Disabilities Education Act (IDEA) IDEA was designed to ensure all schools meet the educational needs of students who have a disability. For schools, this means, among other things that they are purchasing the right products or services for students with disabilities.[47]

Americans with Disabilities Act (ADA) Civil rights legislation meant to keep individuals with disabilities from facing discrimination in all areas of life, including transportation, jobs, schools, and public or private facilities that are open to the general public. [47] ADA was enacted in 1990 when the web was used only by a few people. However, the Department of Justice, which enforces ADA, considers web sites that offer goods and services to consumers to be "places of public accommodation". Since 2018 there has been a surge of lawsuits on web sites and mobile apps that allegedly fail to comply with Title III of ADA.[49] In 2018 there were almost 2000 web accessibility lawsuits related to Title III, whereas before 2015 there was hardly any.[47]

2019 was a record year for ADA Title III lawsuits in general, with 11053 lawsuits filed in the general court, 8.8% increase from 2018. An interesting detail is that 84% of these lawsuits were filed in three states: California, New York, and Florida.[50] Almost a fifth of the Title III cases in 2019 was related to web accessibility. The majority of those cases were about blind plaintiffs claiming that a website does not work with their screen reader software, with a smaller group of deaf plaintiffs suing about the lack of closed captions for online videos.[51]

To mention an education-related example, in 2016 the US Department of Justice investigated University of California at Berkeley's online platform, UC BerkeleyX, and found it to violate ADA Title II, with "significant portions" being "totally inaccessible". Instead of correcting the issues, UC Berkeley reacted by removing over 20 000 videos and podcasts. The university stated that it would have been too costly to caption them[48].

State legislation

Some states have accessibility legislation of their own. The majority of the 50 states seem to base their laws and standards either to Section 508 and/or WCAG guidelines[52]. This is a good thing for Sanako since following and keeping up-to-date on each state's laws would be difficult for a company the size of Sanako. In addition to legislation, many states seem to have policies on state software procurement that include web accessibility[53].

3.4.2 European Union

"Directive (EU) 2016/2102 of the European Parliament and of the Council of 26 October 2016 on the accessibility of the websites and mobile applications of public sector bodies" came into effect in 2016 and it aims to standardise accessibility laws in the European Union[54]. It is a directive, which means it binds the member states to adopt the content into national law.[55]

In Finland, this was done with article 306/2019, which came into effect in March 2019[56]. Both the law and the EU directive behind it reference the CRPD, which shows its importance in practice. The Finnish article is based on WCAG 2.1. It mandates officials, public sector organisations, and certain NGOs to fulfil WCAG 2.1 Level AA criteria. There is a transition period before the law comes fully in effect. For example, public sector organisation web sites published before 23.9.2018 must fulfil the criteria by 23.9.2020. A web site published after 23.9.2018 must fulfil the criteria by 23.9.2019[57].

As a specific example from education, a school can use digital learning material or service that does not fulfil the criteria temporarily, i.e. for one semester. Any service used regularly must fulfil the criteria.[58] As a digital service that is sold to schools and organisations, this directly affects Sanako Connect.

The directive also requires all public sector organisations to publish an accessibility statement, even if the web page in question does not fulfil the requirements. The statement should include information on what parts conform to the WCAG guidelines and what parts do not.

3.5 Guidelines and standards

Thus far, the discussion on web accessibility has been on a general level, without mentions of implementation details and techniques. This is on purpose, to emphasise the point that the principles of web accessibility itself do not change that much, even if the guidelines, techniques and even devices do.

There have been references to WAI's Web Content Accessibility Guidelines (WCAG) throughout this text, especially when discussing laws and policies. Because of their importance, the guidelines will be discussed next. However, they are not the only guidelines produced by WAI. Additionally, there are Authoring Tool Accessibility Guidelines (ATAG) and User Agent Accessibility Guidelines (UAAG). UAAG is aimed at developers of applications that render web content, such as web browsers, browser extensions, and media players. Since it is not relevant to Sanako Connect, it will not be discussed further.

Besides guidelines, WAI also maintains a technical specification, Web Accessibility Initiative – Accessible Rich Internet Applications (WAI-ARIA), that complements HTML5 and is meant to improve the accessibility of web content and applications[59]. This will be discussed after the guidelines. Together the guidelines and WAI-ARIA will provide a way for implementing web accessibility.

3.5.1 Web Content Accessibility Guidelines

The first version of WCAG, 1.0, was published in 1999. The next major revision came in 2008 with WCAG 2.0, and the current version of WCAG is 2.1[60]. Both WCAG 2.0 and WCAG 2.1 are existing standards, but 2.1 is backwards compatible with 2.0. The specification is primarily meant for web content developers, web authoring tool developers, web accessibility tool developers, and others who need or want a standard for web accessibility[1]. As Section 3.4 demonstrated, many accessibility laws are based on WCAG.

Principles, guidelines, and success criteria

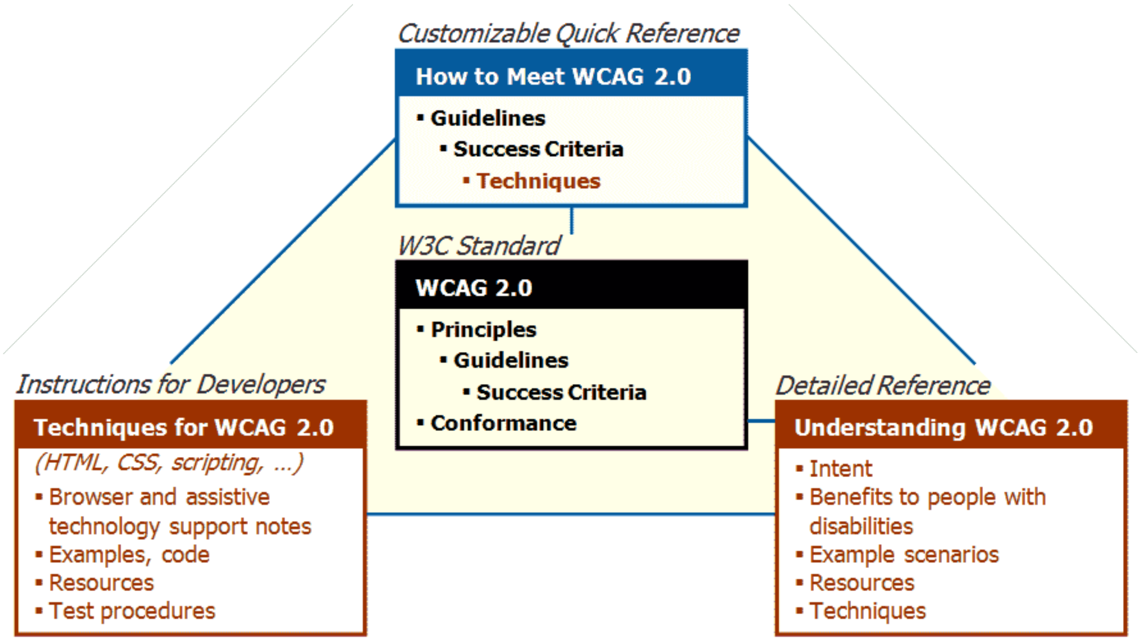


Figure 3.1: WCAG2.0 and WCAG2.1 documents. Copyright © 2010 W3C. From https://www.w3.org/WAI/standards-guidelines/wcag/docs/

WCAG is organised around four principles which contain a total of 12 guidelines. Each guideline, in turn, has success criteria (78 in total) that describe what actions need to be taken to conform to that guideline[61]. The criteria have three levels: A (lowest), AA, and AAA (highest). It is common to use the levels to indicate requirements and conformance to WCAG. For example, the Finnish law 306/2019 requires to conform to A and AA levels, which would mean conforming to every success criteria for those levels. Conforming to a higher level means also conforming to lower levels. W3C recommends against using level AAA conformance as a general policy, because for some content, it is not possible to satisfy all AAA criteria[60].

The success criteria are written to be technology-neutral and also testable, which means that it is possible to give a simple yes or no answer when inspecting if a certain criterion is fulfiled or not[61]. This also means that sometimes the success criteria are somewhat abstract.

Although the principles and guidelines are relatively easy to understand, there is a large number of supplementary documents detailing every possible aspect of the guidelines, from intent and benefits to examples and techniques. If everything were printed, it would take a thousand A4 pages[62].

When going through the guidelines, the standard is at the centre of everything. But instead of using the standard itself when implementing WCAG, WAI recommends using a document called *How to Meet WCAG (Quick Reference).* The document can be filtered according to different criteria and goes through every success criterion. It links to yet another document, *Techniques for WCAG2.1.* That document lists informative techniques for implementing the criteria, and the techniques can be mutually exclusive and be related to more than one success criteria. For example, some techniques use WAI-ARIA as a solution and some focus on making PDF and Flash content.

Yet another document is *Understanding WCAG2.1*, which is referenced both from the standard itself and *How to Meet WCAG* document. That document links to *Techniques for WCAG2.1* document.

To give a comprehensive overview of WCAG, the principles and guidelines are listed below[61]. Going through every success criteria is out of the scope of this work, but examples will be discussed after the list.

Principle 1 - Perceivable

Information and user interface components must be presentable to users in ways they can perceive.

This principle has four guidelines and a total of 29 success criteria: 9 level A, 11 level AA, and 9 level AAA.

1.1 Text alternatives Provide text alternatives for any non-text content so that it can be changed into other forms people need, such as large print, braille, speech, symbols or simpler language.

1.2 Time-based Media Provide alternatives for time-based media.

1.3 Adaptable Create content that can be presented in different ways (for example simpler layout) without losing information or structure.

1.4 Distinguishable Make it easier for users to see and hear content including separating foreground from background.

Principle 2 - Operable

User interface components and navigation must be operable.

This principle has five guidelines and a total of 29 success criterion: 14 level A, 3 level AA, and 12 level AAA.

2.1 Keyboard Accessible Make all functionality available from a keyboard.

2.2 Enough Time Provide users enough time to read and use content.

2.3 Seizures and Physical Reactions Do not design content in a way that is known to cause seizures or physical reactions.

2.4 Navigable Provide ways to help users navigate, find content, and determine where they are.

2.5 Input Modalities Make it easier for users to operate functionality through various inputs beyond keyboard.

Principle 3 - Understandable

Information and the operation of user interface must be understandable.

This principle has three guidelines and a total of 17 success criterion: 5 level A, 5 level AA, and 7 level AAA.

3.1 Readable Make text content readable and understandable.

3.2 Predictable Make Web pages appear and operate in predictable ways.

3.3 Input Assistance Help users avoid and correct mistakes.

Principle 4 - Robust

Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies.

This principle has one guideline and three success criteria: 2 level A and 1 level AA.

4.1 Compatible Maximize compatibility with current and future user agents, including assistive technologies.

Success Criteria Examples

As stated, going through every success criteria is not possible. Instead, success criteria for guidelines 1.1 and 1.2 will be discussed in more detail. They focus on live and recorded audio which is relevant for Sanako Connect, as will be discussed in Chapter 4.

Before delving into the example criteria, it should be noted that not every criterion requires similar scrutiny. For example, criteria 3.1.1, Language of Page, requires that the human language of each Web page can be determined programmatically. Achieving this can be as easy as setting the “lang” attribute of the <html>tag to the correct language, and the criteria provide techniques for Flash and PDF content also.

Success Criterion for guideline 1.1

Guideline 1.1 tells to provide text alternatives for any non-text content. There is only one success criterion for it:

1.1.1: All non-text content that is presented to the user has a text alternative that serves the equivalent purpose, except for the situations listed below.

The criterion then lists six exceptions: Controls and Input, Time-Based Media, Test, Sensory, CAPTCHA, and Decoration, Formatting, Invisible.

The description for Time-Based Media exception is "If non-text content is time-based media, then text alternatives at least provide descriptive identification of the non-text content. (Refer to Guideline 1.2 for additional requirements for media.)"

The description for Test is "If non-text content is a test or exercise that would be invalid if presented in text, then text alternatives at least provide descriptive identification of the non-text content."

Going through the document *Understanding Success Criterion 1.1.1: Non-text Content*[63] provides more context for this criterion. It has intent, additional information, benefits, examples, techniques, and related resources.

Additional information mentions tests and exercises as one situation where a text alternative cannot be provided, as the test must be conducted using a particular sense. An example is a hearing test that would be invalid if the content would be provided as text. In these situations, the recommendation is to describe the non-text content's purpose.

Success Criteria for guideline 1.2

Guideline 1.2 tells to provide alternatives for time-based media. It has nine success criteria:

Level A:

- 1.2.1 Audio-only and Video-only (Prerecorded)

- 1.2.2 Captions (Prerecorded)
- 1.2.3 Audio Description or Media Alternative (Prerecorded)

Level AA:

- 1.2.4 Captions (Live)
- 1.2.5 Audio Description (Prerecorded)

Level AAA:

- 1.2.6 Sign Language (Prerecorded)
- 1.2.7 Extended Audio Description (Prerecorded)
- 1.2.8 Media Alternative (Prerecorded)
- 1.2.9 Audio-only (Live)

Criterion 1.2.1, Audio-only and Video-only (Prerecorded), will be used as an example. The full criterion reads:

1.2.1: For prerecorded audio-only and prerecorded video-only media, the following are true, except when the audio or video is a media alternative for text and is clearly labeled as such:

- Prerecorded Audio-only: An alternative for time-based media is provided that presents equivalent information for prerecorded audio-only content.
- Prerecorded Video-only: Either an alternative for time-based media or an audio track is provided that presents equivalent information for prerecorded video-only content.

The intent behind this criterion is to make audio-only or video-only information available to all users. Several benefits are listed for audio-only: it helps people who have difficulty with visual content, people who are deaf, hard-of-hearing, or just having trouble understanding audio information can read the text presentation, deaf-blind people can read the text in braille. Also, text version allows for searching and repurposing the content[64].

It is not stated explicitly whether the exceptions in Criteria 1.1.1, like doing a test or exercise, apply here also. Nevertheless, since time-based media is mentioned as one type of non-text content in Criteria 1.1.1, it is quite safe to assume that the exceptions apply.

Standards for Real-Time Audio

It should be noted that although WCAG mentions live audio and video on several criteria, standards for Web Real-Time Communications (WebRTC), which enables live audio and video streaming directly from and to the browser, are still developing. WebRTC will be discussed more in Chapter 4.

In March 2020, W3C published the first public working draft of RTC Accessibility User Requirements (RAUR). RAUR aims to inform on potential user needs for people with disabilities, to initiate discussion and gather feedback on how to solve issues with RTC technologies[65]. The potential issues include support for captions and live transcriptions, and being able to route audio output to different output devices. Alternative content like Real-Time Text (RTT) should also be supported.

3.5.2 Authoring Tool Accessibility Guidelines

ATAG is meant for developers of authoring tools. This includes web page authoring tools, courseware tools, multimedia authoring tools, and websites that let users add content. ATAG is for both making the tool itself accessible, and for helping to make the content it produces accessible[66]. There does not seem to be any laws requiring conformance with ATAG. This is probably because its scope is narrower and it relies heavily on conformance with WCAG.

ATAG is divided into two parts. Part A is about the accessibility of the tool itself. Part B focuses on how the tool can help produce accessible content. Otherwise, the structure is similar to WCAG, with eight principles, 24 guidelines, and 63 success criteria.

Part A. Make the authoring tool user interface accessible

A.1. Authoring tool user interfaces follow applicable accessibility guidelines

- **A.1.1.**(For the authoring tool user interface) Ensure that web-based functionality is accessible
- **A.1.2.** (For the authoring tool user interface) Ensure that non-web-based functionality is accessible

A.2. Editing-views are perceivable

- **A.2.1.** (For the authoring tool user interface) Make alternative content available to authors
- **A.2.2.** (For the authoring tool user interface) Ensure that editing-view presentation can be programmatically determined

A.3. Editing-views are operable

- **A.3.1.** (For the authoring tool user interface) Provide keyboard access to authoring features
- **A.3.2.** (For the authoring tool user interface) Provide authors with enough time
- **A.3.3.** (For the authoring tool user interface) Help authors avoid flashing that could cause seizures
- **A.3.4.** (For the authoring tool user interface) Enhance navigation and editing via content structure
- **A.3.5.** (For the authoring tool user interface) Provide text search of the content

- **A.3.6.** (For the authoring tool user interface) Manage preference settings
- **A.3.7.** (For the authoring tool user interface) Ensure that previews are at least as accessible as in-market user agents

A.4. Editing-views are understandable

- **A.4.1.** (For the authoring tool user interface) Help authors avoid and correct mistakes
- **A.4.2.** (For the authoring tool user interface) Document the user interface, including all accessibility features

Part B. Support the production of accessible content

- **B.1.** Fully automatic processes produce accessible content
- **B.1.1.** Ensure that automatically-specified content is accessible
- **B.1.2.** Ensure that accessibility information is preserved

B.2. Authors are supported in producing accessible content

- **B.2.1.** Ensure that accessible content production is possible
- **B.2.2.** Guide authors to produce accessible content
- **B.2.3.** Assist authors with managing alternative content for non-text content
- **B.2.4.** Assist authors with accessible templates
- **B.2.5.** Assist authors with accessible pre-authored content

B.3. Authors are supported in improving the accessibility of existing content

- **B.3.1.** Assist authors in checking for accessibility problems
- **B.3.2.** Assist authors in repairing accessibility problems

B.4. Authoring tools promote and integrate their accessibility features

- **B.4.1.** Ensure the availability of features that support the production of accessible content
- **B.4.2.** Ensure that documentation promotes the production of accessible content

ATAG largely focuses on making the tool and the content it produces compatible with WCAG, and on promoting and documenting the tool's accessibility features. For example, Guideline A1.1's only success criterion is about WCAG:

A.1.1.1 Web-Based Accessible (WCAG): If the authoring tool contains web-based user interfaces, then those web-based user interfaces meet the WCAG 2.0 success criteria. (Level A to meet WCAG 2.0 Level A success criteria; Level AA to meet WCAG 2.0 Level A and AA success criteria; Level AAA to meet all WCAG 2.0 success criteria)

The scale of this single success criterion is enormous, as it encompasses the whole WCAG success criteria. A similar success criterion is for Guideline B2.1:

B.2.1.1 Accessible Content Possible (WCAG): The authoring tool does not place restrictions on the web content that authors can specify or those restrictions do not prevent WCAG 2.0 success criteria from being met. (Level A to meet WCAG 2.0 Level A success criteria; Level AA to meet WCAG 2.0 Level A and AA success criteria; Level AAA to meet all WCAG 2.0 success criteria)

Many other success criteria also refer to WCAG directly or indirectly. For example, **A.2.1.2 Alternatives for Rendered Time-Based Media**, states that

If an editing-view renders time-based media, then at least one of the following is true: (Level A) (a) Option to Render: The authoring tool provides the option to render alternatives for the time-based media; or (b) User Agent Option: Authors have the option to preview the time-based media in a user agent that is able to render the alternatives.

WCAG is not mentioned here, but this clearly refers to WCAG 1.1.2 Time-based Media.

3.5.3 Web Accessibility Initiative – Accessible Rich Internet Applications

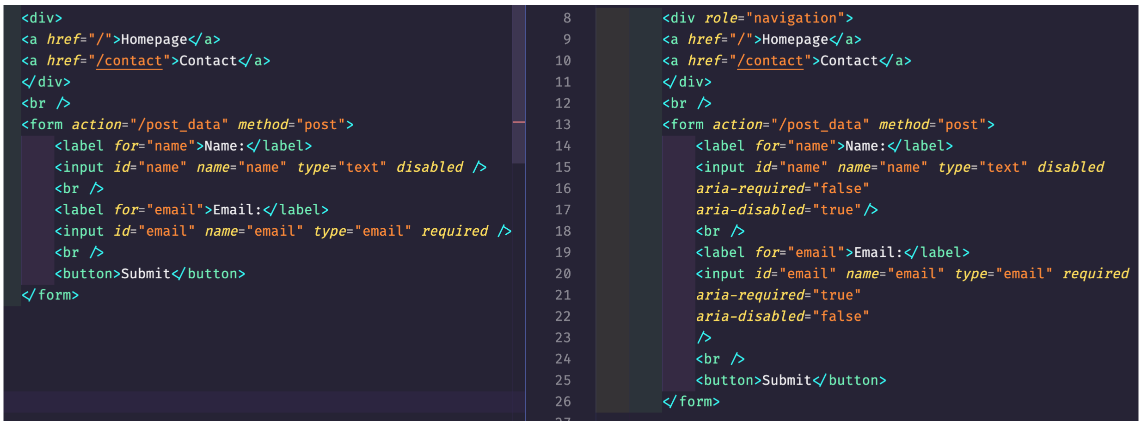
WAI-ARIA is a technical specification that defines additional HTML attributes. The goal is to provide additional semantics and improve accessibility[67]. As WCAG and ATAG aim to be technology-neutral, WAI-ARIA gives code-level guidance on how to improve web content accessibility. It is referenced frequently in Techniques for WCAG2.0 document.

WAI-ARIA is especially suited to help with dynamic content, and complex user interface controls made with Ajax, HTML, and JavaScript. An example would be a drag-and-drop list that can only be used with a mouse or time-based updates that change the content of the web page so that blind users or those using screen readers would not know about it. Before WAI-ARIA, those using keyboards and screen readers had no way of using complex web controls or knowing about dynamic changes in the web page, whether they were caused by the user's actions or by time- or event-based updates [68]. With WAI-ARIA, controls, live regions, and events can be mapped to accessibility APIs[66].

WAI-ARIA does not affect the web page outlook or normal functionality in any way, except by exposing information to the browser's accessibility APIs[67]. A common mistake is relying on ARIA before learning semantic HTML, which was discussed in Section 2.1.2. Semantic HTML should be preferred because it provides much of the functionality needed for accessibility APIs for free. Although WAI-ARIA gives more granularity, using its attributes in a wrong way can actually make the content less accessible. All interactive functionality would need to be built from the ground up. Furthermore, the way ARIA is interpreted is inconsistent between different browsers[69].

The inconsistencies grow when using screen readers. A test comparing ARIA behaviour between different screen reader and browser combination had an average reliability of 69%, with the best result being 90% on JAWS screen reader with Chrome browser. The detailed tables in the test show large differences in the behaviour [70].

W3C acknowledges the possible dangers. They provide a long document about WAI-ARIA authoring practices. It begins with a warning to the user: No ARIA is better than Bad ARIA[71]. This means that as ARIA markup can alter the way HTML elements function and are visible to assistive technologies, incorrect ARIA markup can have real consequences for assistive technology users.



```
<div>
<a href="/">Homepage</a>
<a href="/contact">Contact</a>
</div>
<br />
<form action="/post_data" method="post">
    <label for="name">Name:</label>
    <input id="name" name="name" type="text" disabled />
    <br />
    <label for="email">Email:</label>
    <input id="email" name="email" type="email" required />
    <br />
    <button>Submit</button>
</form>
```

```
<div role="navigation">
<a href="/">Homepage</a>
<a href="/contact">Contact</a>
</div>
<br />
<form action="/post_data" method="post">
    <label for="name">Name:</label>
    <input id="name" name="name" type="text" disabled
    aria-required="false"
    aria-disabled="true"/>
    <br />
    <label for="email">Email:</label>
    <input id="email" name="email" type="email" required
    aria-required="true"
    aria-disabled="false"
    />
    <br />
    <button>Submit</button>
</form>
```

Figure 3.2: The same HTML code describing a simple form, with and without WAI-ARIA attributes.

WAI-ARIA supplements HTML markup with three kinds of attributes:

Roles Describe the structure of a web page or the type of element. For example, *role="navigation"* describes a navigation element and *role="tab"* describes a single tab pane.

Properties Describe the state of elements or define live regions or drag-and-drop controls on the web page. An example would be an *aria-required* attribute for required form input.

States Define the current conditions of elements. Unlike properties, states can change throughout the lifecycle of the element. An example would be an *aria-disabled* attribute that describes to a screen reader whether a form input is disabled or not.

3.6 Implementing and evaluating web accessibility

Section 3.5 provided information on what common guidelines and code-level solutions exist for web accessibility. WAI has resources and tutorials on implementation, and recommendations on higher-level policies and statements a company can create on web accessibility. This section will also go through relevant research on web accessibility implementation.

3.6.1 Getting started

WAI provides tips on getting started for designers, developers, and writers, and resources for both initial and thorough check-ups of a web page's accessibility. WAI provides a methodology for auditing websites, Website Accessibility Conformance Evaluation Methodology (WCAG-EM), and a report generator for it. WCAG-EM can also be used for web applications and mobile sites[72].

WAI has a four-step guide for long-term planning and managing web accessibility that is suitable for both projects and organisation-level changes. The activities included are not required to be done in sequence[73]. The steps are as follows.

Initiate Develop understanding of accessibility and build organisational enthusiasm.

This step includes learning, exploring the current state of accessibility, setting objectives and building a business case, among others.

Plan Develop clear goals and an environment that supports accessibility.

The planning stage contains items like creating an accessibility policy, determining budget and resources, and reviewing the environment and identifying issues that need to be fixed. To track progress, a monitoring framework should also be set up.

Implement Ensure personnel are trained, tools are available, and accessibility is included throughout.

This step is about building skills, integrating goals into policies, evaluating, prioritising, and tracking and communicating progress. A policy framework for the whole organisation should be created.

Sustain Continue to review and report on content, processes, and resources.

As legislation and technologies change, they should be tracked and adapted to. User feedback should be incorporated, and the websites (or applications) monitored.

3.6.2 Implementation challenges

As there are guidelines and supporting material from WAI, what are the challenges when implementing web accessibility? If planned and designed in advance, the cost of accessibility is almost zero. Retrofitting accessibility is time-consuming and costly and could present extreme challenges [74].

Lazar (2015) argues that any developer can learn accessible design. With the right mindset and support from the executive level, it is not difficult to create an accessible product, nor does good design need to be sacrificed for that. He further mentions that accessibility standards like WCAG are the foundation that makes consensus possible for accessibility [68].

The study discussed in Section 3.3 does mention that some of the reasons for failed accessibility projects were related to not having support or for perceiving that web accessibility did not fit the design requirements.

A study made in 2016 by Abuaddous, Jali, and Basir went through web accessibility literature and grouped the challenges they discovered into three categories[75]. The first set of challenges was related to standards and guidelines, where they focus on WCAG. To summarise, the WCAG guidelines are a common challenge in themselves, because they can be hard to understand, ambiguous, and require some technical knowledge of accessibility. The material related to implementing WCAG is also extensive, as was also mentioned in Section 3.5.1.

The second category was about challenges during a website's design and development. These challenges were related to lack of accessibility awareness, motivation, and training. The study mentions that recommendations are widely available, but they are worlds apart from the way developers actually work, and thus the developers do not follow them. Something else than general guidelines are thus needed.

The final category was about challenges during evaluation, where the main challenges were related to problems with automated tools and user-based testing. Choosing tools can be slow, and they do not remove the need for manual inspection. Furthermore, their results can be inaccurate or hard to analyse. Automated tools will be discussed more in the next section. As for user-based testing, the challenges here are about cost, getting large enough samples sizes, preparing the tests and the need for an expert evaluator.

Based on their findings, they introduce recommendations and solutions for them.

- Providing applicable and usable guidelines
- Enforcing accessibility legislation
- Creating new web accessibility position
- Changing the web team mindset

When they mention providing applicable guidelines, they actually mean that WAI should provide guidelines that are categorised in a way that is suitable for the developer.

Regarding evaluation, Lazar considers it to be one of the most challenging steps for developers (and policymakers). There is no specific guidance for it in the standards or the policies, and the range of evaluation methods, technologies, users, and disabilities makes it impossible to use any single metric to gauge the accessibility of a product [76].

This is an issue, especially if the law enforces compliance with guidelines. It becomes an even more troubling issue once you realise that the parties monitoring compliance have issues with evaluation also. In the United States, compliance monitoring for Section 508 is required at the federal level, but in practice, it is not done very often. The way federal agencies do compliance monitoring is also varied - each agency (of over 100 analysed) either uses automatic tools of their choosing without result validation, have their own documentation on how to make content accessible, or simply trust the consultants and contractors who have said the content is accessible [77].

Lazar divides evaluation methods into three categories: user testing, expert reviews, and automated evaluation. Each has its own strengths and weaknesses, which is why they should be used in combination with each other [76].

On the topic of expert reviews, a study made in 2011 investigated the effect expertise plays in web accessibility evaluation methods. By comparing manual assessments made by experts and non-experts, they found out that expertise matters; that expert assessments produced more valid and reliable results, and that small number of experts was enough to discover all accessibility issues in the test material [78].

3.6.3 Automatic tools

Automatic evaluation tools can help identify potential issues and can be used through the design and development process, but not every aspect can be checked automatically. Humans must be involved in the process[79]. Their potential inaccuracy was also briefly mentioned in the previous section. A study comparing six free evaluation tools noticed that their results and reporting format varied wildly between them, which means that using only one tool is insufficient [80].

Another study also compared six evaluation tools with each other but additionally compared them to manual evaluation performed by experts. They found out that coverage at best was 50% against WCAG2.0 criteria, and that correctness in reported accessibility violations varied between 66 and 96 per cent. [81] It should be noted that at the time of writing the study is seven years old, and modern tools could present different results. Nevertheless, automatic tools are still not in the position to catch every accessibility violation.

WAI maintains a list of web accessibility evaluation tools that at the time of writing contains 142 entries[82]. The list has not been updated since 2016, so its relevance is in question.

The tools generally seem to either stand-alone or browser-add-ons, which means they look at the web page from an external point of view. Some tools can be incorporated into web development workflow and inspect the code on a more detailed level and at different phases of the build process, such as React specific react-axe.

3.6.4 Policies and statements

In addition to the organisational policy mentioned in Section 3.6.1, WAI recommends providing an accessibility statement. This is the same statement as mentioned in Section 3.4.2. According to WAI, accessibility statements are important since they show users that you care about accessibility, provide user information about accessibility, and demonstrate a commitment to accessibility and social responsibility.[83] The statement is primarily meant for the users and should contain at least a commitment, the accessibility standard used, and contact information about accessibility issues. WAI also provides a generator tool for writing the statement.

Chapter 4

Sanako Connect

As stated in Chapter 1, Sanako Connect is a web application for administering language learning lessons. It is marketed mostly to schools and universities, and its main users are teachers and students. Sanako has a long history of building language learning products, and over the years it has moved from physical language laboratories with furniture and cassette tapes or CDs, to PC-based language laboratories and finally towards browser-based applications.

This transformation has followed resources the customers - schools - have access to. Nowadays PC classrooms are becoming rarer, and it is more common to see a mixed device classroom with Chromebooks and iPads, owned by both the school and the students. The customers' need for something that works on multiple platforms was the primary motivation for developing Sanako Connect. The evolution of browser APIs like WebRTC and Web Audio has made it possible to develop a full-fledged language lab for the browser.

This chapter will go through the aspects of Sanako Connect that are required to understand the application and later to reflect on the accessibility requirements.

4.1 Use and features

Using Sanako Connect is quite different for teachers and students, much like the actual classroom experience is different for them. While the teacher should be able to control the whole classroom, dividing the class into groups and assigning and leading activities, the student's focus is only on the task they are given. In practice, the teacher and the student are using different applications, which also gives the freedom to develop them with some independence from another.

Every session has at least one teacher, although the teacher does not need to be online for the students to access the session. Since Sanako Connect can be used in different ways, there is no upper limit as to how many participants there can be in one session. In theory, one session could have hundreds of participants online at the same time or spread out during a longer time period. In this kind of an extreme scenario, most features would remain usable, but for example, broadcasting live audio from the teacher to everyone would no longer be possible.

4.1.1 Session administration

In Sanako Connect's terminology, a session is the umbrella term for one gathering of teacher and students. Each session will have its own participants, exercises, files, and groups.

Creating a session requires the teacher to log in with their email and password to their session dashboard, where they can administer their sessions. Every teacher belongs to an organisation which uses Sanako Connect on a trial basis or as a paid service. The organisation license dictates how many participants can be online at the same time.

Once the teacher has created a session, they receive the URL and QR code holding that URL that they can then distribute for their students. Optionally they can limit attendance by giving the session a password. Otherwise, the session can be attended by everyone who has the URL.

This means that there is no authentication and login for students. While the administration dashboard has a normal email and password authentication, the session application authenticates the device and not the user. If the teacher changes devices or browsers, they need to authenticate that device as the teacher's device through the session administration.

This approach has its benefits and drawbacks. The students do not need to create an account or install anything on their device or browser before attending. Sanako does not need to collect any personally identifiable information from them, except if they are using their real name for the session. The other side of the coin is that there is no continuity between sessions, and the teacher cannot get any statistics from students' progress, for example. If a student attends two sessions by the same teacher, from the application's point of view that student is just two different participants who are using the same browser.

When a student enters a session for the first time, they are asked for their name and optionally the password. A name must be unique so that teacher can distinguish students from each other. If the student has participated in that session before from the same session, only the password is asked.

If there is no password, a greeting dialog is still displayed for the user, and they need to click on the button to close it. This has a functional purpose. The Web Audio API requires starting it from a user interaction before it can be used. This is required so that web sites cannot play audio without user interaction[84]. Requiring the user to click a button whenever they enter a session ensures that the audio capabilities can be initialised without issues.

Once the student is in, they see any possible exercise or file the teacher has shared to the session beforehand and can use the session chat, if the teacher has enabled it. At its most limited, the student cannot do anything at first without the teacher providing them with the possibility.

4.1.2 Groups

When a student enters a session, they do not belong to any group. A group, in this case, means a selected grouping of students who can write to and speak with each other. Once the student is put into a group, they can see other group members.

The teacher can create groups and put students in them one by one. Alternatively, they can use the split feature, which assigns students randomly to evenly sized groups. The expectation is that during the session, the group configuration will change many times as the teacher assigns different tasks to the students.

4.1.3 Real-time audio

In language labs, real-time audio connections in different setups are an often used and important feature. The teacher needs to broadcast to the whole class, the students need to talk with each other in groups, the teacher needs to either listen or speak with a single student or a group, and so on. Recording is likewise important, with students recording themselves while reading a text or imitating live or recorded audio. In Sanako Connect, students can simultaneously record themselves and send and receive audio streams.

The teacher can speak to the whole class. This is a one-way broadcast, meaning that the teacher does not hear any of the students, and the teacher is the only one the students hear. At the time of writing this, there is development underway for the teacher to be able to stream their webcam or screen to the class.

Another important audio streaming feature is group audio. Once the teacher sets up a new group, that group is always in a silent mode first. This means the group members can only chat with each other, and there is no audio streaming between members. Once the teacher enables the audio, all group members can hear and talk with each other.

The teacher can join this group audio, either only as a listener or that so they can also speak with the group.

The third part of audio streaming is a private audio conversation between the teacher and an individual student. The teacher can select a student and either listen to them - used commonly to check what the student is doing at that moment - or speak with them. Group audio and private audio are mutually exclusive. When a student is in a group, th teacher cannot start a conversation with them.

4.1.4 Content authoring

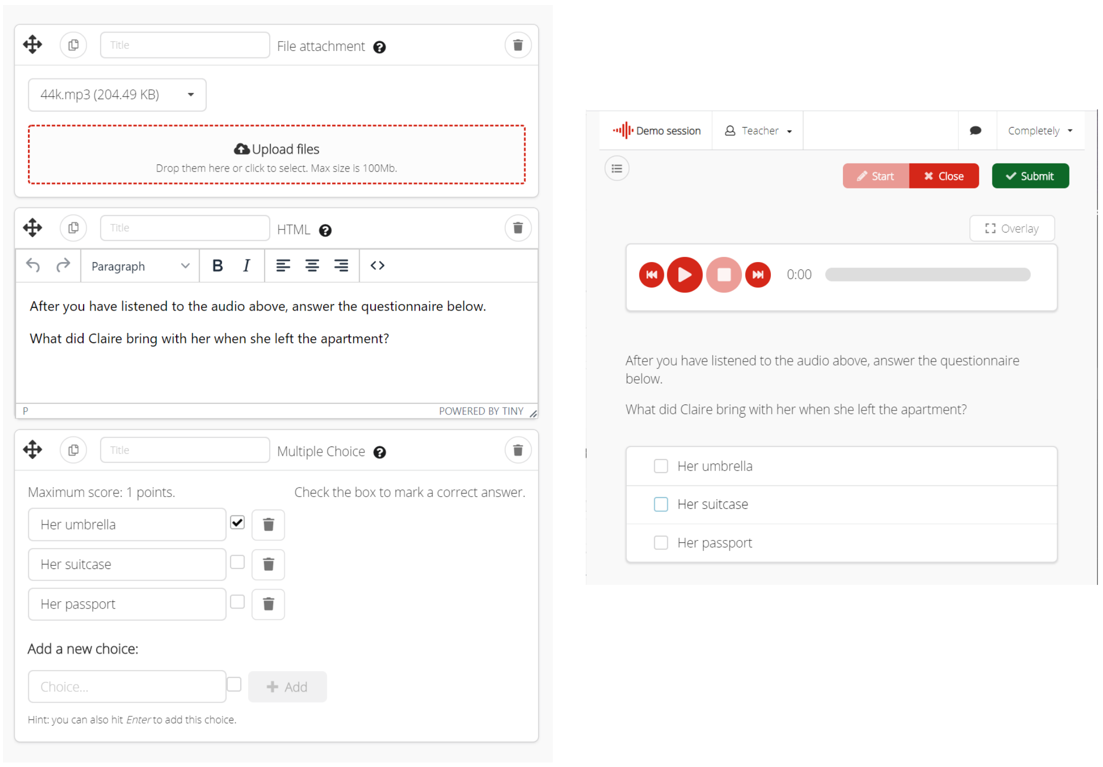
The exercises made by the teacher are another important feature. Sanako Connect provides an editor for the teachers to create content. The exercises are a mix of rich text, different types of files such as audio or images, and interactive tasks like recording audio, writing, or doing multiple-choice questionnaires. The student can perform the exercises either alone or in a group.

The structure of a single exercise is simple. Each exercise consists of a title and zero or more exercise blocks, which contain the actual functionality. Once the teacher has finished the exercise, they can share it with the session. When this is done, all students will see a notification about a new exercise, and the exercise appears to their exercise list.

When a teacher is creating an exercise, they are creating a template for the students to interact with. In Sanako Connect, the student part of the exercise is called a submission. The teacher also needs to be able to check everything the students have done, which means they can receive dozens of submissions for each exercise.

When the student receives the exercise, they need to start it before they can interact with it. This way, the teacher can monitor the progress of students. Once the student is ready, they can close the exercise should they want to continue later, or they can submit what they have done, and the teacher receives it instantly. Alternatively, the teacher can collect submissions from students who have not submitted themselves. Once the submission is submitted or collected, students cannot modify it anymore.

Some of the exercise blocks have automated grading. If the teacher has chosen this option, the correct answers and the student's score will be revealed immediately after they have submitted their work. For some exercise blocks, automated grading is not possible, which is why the teacher can write feedback for each submission and send it back to the student. The teacher can also re-open the submission for the student, or reset it, which means deleting everything the student has done for that exercise and giving them a blank slate.



After you have listened to the audio above, answer the questionnaire below.

What did Claire bring with her when she left the apartment?

Maximum score: 1 points.

Check the box to mark a correct answer.

- Her umbrella ✓
- Her suitcase
- Her passport

Add a new choice:

Choice... Add

Hint: you can also hit *Enter* to add this choice.

After you have listened to the audio above, answer the questionnaire below.

What did Claire bring with her when she left the apartment?

- Her umbrella
- Her suitcase
- Her passport

Figure 4.1: How the exercise blocks look in the teacher's editor and how they appear to the student.

The exercise blocks can be divided into two categories: static content, which is something the student can consume like text, images, or audio, and interactive tasks, which is something for the student to do.

Static content

Rich text A text editor that can be used to insert, edit and format text into the exercise. Formatting like bolding and italics, and semantic HTML elements like headers and paragraph can be used.

Files Any kind of file can be embedded into an exercise. If the browser supports the file type, Sanako Connect offers an audio or video player, image viewer, or a PDF viewer. If the file cannot be opened inside the browser, a download option is offered.

Embedded content Web pages or videos from selected streaming services can be embedded into an exercise. These will be displayed inside the iframe HTML element. The currently supported services are YouTube, Vimeo, and Daily Motion.

Interactive tasks

Recording tasks Recording is a crucial feature in language labs. Sanako Connect has two audio recorders for this. The Simple Recorder offers just a recording button and an optional time limit set by the teacher. The outcome is then uploaded to the file server. The Advanced Recorder is a two-track recorder modelled after recording features in Sanako's previous language lab products. The teacher can set a master track that the student will hear while recording their own voice. The student can then record one or more audio clips into the timeline. Optionally they can use the Voice insert feature, which allows the student to insert new audio so that any existing tracks for the starting point will be split in two. The teacher can then download the outcome either as a mono mp3 file or as stereo mp3 so that the master and student tracks are panned into left and right channels. This feature is commonly used when teaching interpreters.

File uploads Teacher can allow the student to upload one or more files. As with the teacher's file upload feature, the file will be handled inside the browser as possible. The teacher can distribute files not supported by the browser, such as Microsoft Word documents, let the students work on them on whatever application they need, and then upload the result back to the teacher.

Q&A Q&A allows the teacher to create a batch of questions. They can be open-ended, or the teacher can give the expected answer to a question. If there is an answer, the question will be automatically graded.

Multiple Choice Teacher can create one or more multiple-choice questions and mark which questions should be checked by the student as correct. Automatic grading is always used with this exercise block.

Gap fill The teacher can create a text using the same rich text editor as before, and mark which words or phrases should be displayed as inputs for the student. The student can then fill these gaps with their answers. Automatic grading is used with the same logic as in Q&A - if the gap has an expected answer set by the teacher, it will be graded.

4.1.5 Other features

In addition to the major features listed previously, Sanako Connect also has different messaging options. The session chat allows the whole class to chat with each other and the teacher. The teacher can decide to disable this. Each group also has its own chat, restricted to group members and the teacher. Finally, students can send direct messages to the teacher, and the teacher can send messages to each student. Students cannot send direct messages to each other.

Most commonly files are used by embedding them into an exercise. The teacher can also decide to share any uploaded file directly to the session. The shared files will be displayed in a list to the student, and at any moment, one file or exercise can be viewed. The files will behave the same as embedding them to an exercise. Depending on the browser support they will be offered for download or allowed to be used inside the browser.

4.2 Technical aspects

Sanako Connect uses several frameworks, libraries, and browser APIs to achieve its functionality. First, this subsection briefly goes through the back-end technology since it is needed for understanding how the application works. More focus is given on the front-end - the browser - since that is the area more relevant for web accessibility.

4.2.1 Back-end

As stated previously, teachers and students use Sanako Connect differently. When teachers share exercises and files with the students, split the students into groups, and enable different real-time audio configurations, technically, this means that they are continuously altering the state of the student application. This means that both the teacher and student applications need to reflect the changes real-time.

This is why the communications protocol between the back-end and its clients is WebSocket instead of the more common HTTP. Although there are some solutions for two-way communication with HTTP like HTTP Long Polling[1], with WebSockets it is easy to achieve full-duplex communication between the client and server. Essentially this enables pushing data for the clients in real-time, which in turn makes it possible to keep the client application in sync with the server state. Practically every modern browser supports WebSockets[2].

The back-end runs on Node.js, which is “a JavaScript runtime built on Chrome’s V8 JavaScript engine”[85]. The main framework that has been used is FeathersJS, an open-source project which describes itself as “a lightweight web-framework for creating real-time applications and REST APIs using JavaScript or TypeScript”[86]. FeathersJS itself is a thin layer built on top of Express, which is the most well known and used JavaScript

[1]The client sends constant requests to the server, and the server responds only when there is new data.

[2]The support in browsers currently in use is over 97%, according to https://caniuse.com/#feat=websockets

server framework[87][18].

FeathersJS was chosen since it makes it easy to distribute data in real-time to connected clients based on different rules. For example, a teacher should be able to know whenever a student returns a submission. However, the student should only be concerned with changes in their own submission, and a student should never receive another student's submission, even if this would only happen in the data layer. With the concept of channels in FeathersJS, it is easy to send data only for those clients that should have it. When a client connects and a user is authenticated, they can be added to different channels. Whenever there is an event that should be broadcasted to clients (such as a new database entry), the broadcast happens through the channels. The decision to add the event to a channel can be made based on the data itself and the client.

Even though real-time events are important, equally important is that everything is also stored in the database. This protects against data loss if the connection breaks up or if a student joins a class late - the current state can always be retrieved from the database.

4.2.2 Front-end

The main framework used in the front-end is React. The client application was initially set up through Create-React-App (CRA), Facebook's officially supported command-line interface to create single-page React applications. It sets up a complete development environment and hides the complexity and dependencies needed for a modern development workflow. The drawback in some situations is that CRA is opinionated. In these cases, it is possible to 'eject' the application and take care of everything yourself.

FeathersJS also provides a front-end library that helps with authentication and communication with the server. That library is used in Sanako Connect as well. Once the user/device has logged in, the client sets up the necessary event listeners. The events it receives depend on the channel setup on the server. When it receives an event, the listener triggers a function, which often updates the state of the React application.

The main user interface library used in Sanako Connect is Semantic UI React. It is the React version of the popular Semantic UI development framework, which provides HTML markup and CSS rules for a multitude of common user interface components. The React version provides React components, which can be included in a React project with a package manager like npm.

Browser requirements

Sanako Connect uses several browser APIs and standards to achieve its functionality, and some browsers either do not have the required features or their implementation is not mature yet. This is why Sanako has a few recommended browsers for using Connect.

Majority of the features works fine on any modern browser. But there are differences especially with audio streaming and recording.

For every other platform than iOS, Sanako recommends using Google Chrome. This is mostly related to real-time audio streaming. The standard for that, WebRTC, differs from browser to browser. The most reliable results are achieved when everyone uses the same browser.

For iOS, the recommendation is Safari. Like mentioned in Section 2.1.3, Apple restricts the functionality of other browsers than Safari on its mobile platforms. This means that Safari is the only iOS/iPadOs browser that can handle WebRTC, for example.

These recommendations do not seem to clash with any accessibility guidelines, but they do go against the idea of being able to use whatever browser the user prefers.

WebRTC

WebRTC stands for Web Real-Time Communication. It is an open-source project that governs standards and APIs for adding real-time communication capabilities to browsers and other devices. On a high level, the standard covers two technologies, media capture devices and peer-to-peer connectivity[88]. This means that it is used for capturing audio and/or video streams from the user's microphone, webcam, and screen, and for sending these streams to remote locations.

WebRTC has a history of over ten years, with different implementations in different browsers adopted on different schedules. WebRTC support first arrived in Chrome version 23 back in 2011 and for Firefox in 2013[89]. The latest W3C specification is in Candidate Recommendation stage and is dated 13 December 2019. Going through the previous versions of the document one can see that it has been in Candidate Recommendation stage since November 2nd, 2017/footnoteW3C specifications can mature from Working Draft, to Candidate Recommendation, to Proposed Recommendation, and finally to W3C recommendation..

Web Audio API

There are two ways to control audio on a web page: with the <audio>HTML element that was introduced with HTML5 (along with the <video>element for videos), or with Web Audio API. Using the <audio>element is more suitable for simple audio playback, whereas Web Audio API makes it possible to create more complex processing and routing of the audio signal. Web Audio API can be used to record the audio from the user's microphone, and it also can be used alongside WebRTC streams. For example, in Sanako Connect it is possible to record the student's audio stream from the microphone and the remote streams of other students in their group into the same file.

Another difference between the <audio>element and Web Audio API is that the <audio>element has a visual representation on the page, at least if the element is added as part of the DOM. As Web Audio API is a programming interface, it can be invisible for the user.

4.2.3 Mobile functionality

There are no native mobile applications produced or planned for Sanako Connect, which means that the same web application is used on computers and mobile devices.

Having only one client application instead of two or three (for iOS/iPadOs and Android) has obvious benefits for development, but it also limits the functionality as compared to native applications. As mentioned in Chapter 2.1.3, Apple restricts the functionality of other browsers than Safari on its mobile platforms. Sanako Connect is also designed to work as a Progressive Web App, which means it can be installed on Android and iOS platforms.

4.3 Development process

Sanako Connect is developed according to continuous delivery and agile methodology. The development is broken into sprints that take 1-2 weeks, the focus is on constant iterations, and the completed work is released into production as soon as possible.

Chapter 5

Choosing accessibility guidelines and development requirements

This chapter aims to find answers to the first two research questions asked in Chapter 1. The first question, what accessibility guidelines to follow, is handled in Section 5.1. That section's findings lead to Section 5.2, which goes through Sanako Connect's current state regarding accessibility. After that, I will move on to discuss how the guidelines can be turned into requirements.

The findings from this chapter serve as the starting point for the next one, which deals with the development process and implementation. All the research questions will then be discussed once more in Chapter 7.

5.1 What accessibility guidelines to follow

Applying different accessibility guidelines on software can be expensive, and in some cases with existing software, practically impossible. This is why Shirogane suggests a method to formulate and prioritise accessibility requirements early in the development of the software[90].

This is a luxury that cannot be applied to Sanako Connect, for two reasons. The first is that it is an already published software that is in active development. The second, as is discussed below, is about legal requirements.

Chapter 3 described different accessibility guidelines and regulations. The question is, as Sanako Connect is a globally sold education software, with public sector organisations as customers, what guidelines and regulations it should follow? The answer to this will affect the actual implementation work.

The same chapter mentions that according to the new EU directive, any service a school uses regularly must meet Web Content Accessibility Guidelines 2.1 Level AA criteria. In the United States' legislation, Section 508 is related to federal organisations, and ADA is a civil rights law which covers schools, among other things. A negotiation with one federal organisation is currently underway, and there is no sense to block that and other possible future sales by not conforming to Section 508. Even bigger market is the local schools, so making sure we fulfil ADA requirements is important.

Luckily, WCAG 2.1 Level AA is referenced in the latest version of Section 508 and should also be enough for ADA compliance. Since WCAG is also referenced in other countries' legislation, it would make sense to focus on its criteria. It is safe to assume that conformance with WCAG will be the most important thing.

Even though there is an authoring tool included in Sanako Connect that would benefit from compliance with Authoring Tools Accessibility Guidelines, Section 3.5.2 mentions that ATAG is largely about WCAG compliance and documentation on the authoring tool. This means that complying with WCAG criteria would help with ATAG also. Following ATAG would likely improve user experience and help the teachers to create accessible content, so it would be sensible to incorporate its criteria to Sanako Connect at some point to make it more user friendly.

For the present, all this would mean complying with 50 success criteria from WCAG 2.1 Levels A + AA. This would satisfy the necessary criteria in the EU and USA, even though some states might have their own legislation on the matter. This also means that it is not a matter of prioritising some accessibility requirements above others (except in the implementation schedule). All 50 must be fulfiled so that Sanako Connect can be said to be WCAG compliant.

As Sanako Connect is an education software, the WCAG criteria do have some exceptions for it. An example in Section 3.5.1 mentions Success Criterion 1.1.1, Non-text Content, according to which text alternatives must be provided for non-text content. But test and exercises that must be conducted using a particular sense, like hearing tests, are exceptions.

The exception should apply to Sanako Connect directly, as the exercises in Sanako Connect rely on recorded audio - with hearing as the particular sense - a lot.

There is also another criterion not mentioned earlier, 2.2.1, Timing adjustable, that might apply to Sanako Connect. It concerns time limits, and according to the success criterion, the user should be able to turn off, adjust, or extend the limit. The criterion mentions two exceptions: “The time limit is a required part of a real-time event (for example, an auction), and no alternative to the time limit is possible; or [. . .] The time limit is essential and extending it would invalidate the activity.

It can be argued that a teacher giving a test or conducting recording exercises with a time limit falls under these exceptions, at least if the event is real-time. Whether the time limit is essential is debatable. The term is explained in the guidelines as “if removed, would fundamentally change the information or functionality of the content, and information and functionality cannot be achieved in another way that would conform”.

Then again, as the time limits that can be set in Sanako Connect are all optional, the teacher would decide to set them. This would mean explaining this would be under ATAG compliance.

5.2 Current state of web accessibility in Sanako Connect

The outcome of the previous section was that Sanako Connect should fulfil the criteria in WCAG 2.1 Level A+AA, with the exceptions for test situations mentioned in some criteria. Now is a good place to examine how Sanako Connect currently fulfils them.

To get some metrics for improvement, the current situation needs to be assessed. As was discussed in Section 3.6, measuring the actual accessibility of a product is difficult. That is why compliance with WCAG is investigated instead.

Sanako Connect was examined against WCAG criteria with the help of *Understanding WCAG 2.1* document[61], which goes through examples of successes and failures for each criterion.

As mentioned in Section 3.6, one of the common challenges with web accessibility is that WCAG criteria are hard to understand. Having gone through all the Level AA criteria, the writer can concur with that argument. Even with the help of supplementary documents, some of the criteria were difficult to decipher. As was discussed in Section 3.5.1, the criteria are written to be technology-neutral. As a result, they can be somewhat abstract. There are plenty of explanatory sources for the criteria, but there is no guarantee that their interpretation is correct on the more challenging ones.

Since conformance with each criterion can be assessed as success or failure, it is possible to create simple statistics of the current situation. However, looking at numbers alone can be misleading. The workload and time required to fulfil any criterion depends largely on the application. For example, the application structure, the tools and 3rd party components that are in use will affect how easy or hard it is to implement changes.

Also, the criteria are not equal in importance - failing some will prevent using the product completely, while as failing others will have less drastic consequences for the user. WCAG itself states on four success criteria that failure to fulfil them could interfere with the use of the page in general [60].

On some cases deciding between success or failure is quite subjective. If text does not have enough contrast against its background (Criteria 1.4.3), it is self-evident to mark that criteria as fail. Then there are criteria like 2.4.6, according to which headings and labels should be clear and descriptive. Relatively simple, but still subjective.

Appendix A contains a list of criteria along with explanations of why they passed or failed. Out of the four criteria WCAG deems to be important, only one was passed. The passed criterion was 2.3.1 - the page does not contain anything that flashes more than three times in any one second period.

The failed criteria were as follows, paraphrased by the author:

- 1.4.2 - Audio Control - autoplaying audio longer than 3 seconds needs to be able to be stopped or paused, or to control the audio volume separately from the system volume (Level A)
- 2.1.2 - No Keyboard Trap - if a component can receive keyboard focus, the focus can also be moved away from the component with keyboard (Level A)
- 2.2.2 - Pause, Stop, Hide. Automatically moving, blinking, or scrolling content that lasts more than 5 seconds and is displayed with other content needs to be able to be paused, stopped, or hidden - except where it is essential for an activity. Automatically auto-updating information that is displayed with other content needs to be able to be paused, stopped, or hidden - except where it is essential for the activity. (Level A)

Figure 5.1 shows an example of a failed criteria, 2.1.1, according to which all functionality should be operable with a keyboard interface. Currently, the user cannot do anything on the main toolbar. The dropdown menus can be opened, but not interacted with. The ‘New group’ button on the left can be used with a keyboard - but there is no visible indicator when the keyboard focus is on that button. That is a failure of another criterion, 2.4.7.

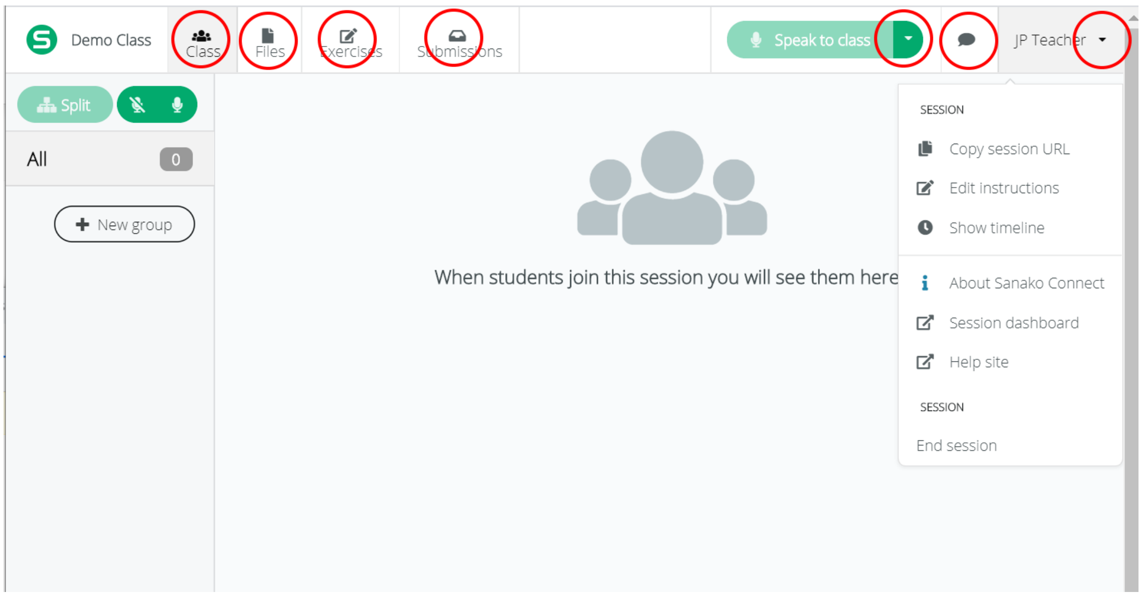


Figure 5.1: An example of failed keyboard navigation. The circled items are UI components that should be reachable by keyboard but currently are not.

This blocks the use of the application completely. It shows that the criteria that WCAG labels as important are not the only ones which could have application-wide effects.

In general, twenty-one criteria passed, and twenty-nine failed, making the current success rate 42%. It is good to keep in mind that achieving 100% accessibility for everything in a complex application could very well be impossible [91] or at least impossible to measure, as has been discussed. This makes it important to focus on best efforts and prioritise issues that benefit the users most.

5.3 From criteria to requirements

Despite their occasional abstractiveness, deciding whether any single criterion is a pass or fail on an existing product is fairly simple. It is much more complex the other way around - deciding whether a product or even just a single component passes all the 50 criteria. It is no wonder that in the challenges discussed in Section 3.6, the ambiguous and hard to understand nature of the guidelines comes up prominently, as well as the fact that there is a gap between the way recommendations are written and the developers' everyday work.

This is easy to sympathise with. Since the criteria are technology-neutral and sometimes hard to decipher, using the criteria directly in everyday design and development would not work. This section will look more closely at one of the research questions - how the accessibility guidelines could be transformed into more concrete requirements?

According to Usman et al., checklists are used to avoid human error and to improve safety and performance in many disciplines. In software engineering, they have been used for a long time in evaluating requirements, for example. They can also be used as "reminder systems" for remembering the necessary information[92].

Although safety is not an issue here, using a checklist to avoid errors and improve performance, and for evaluating if any component of the product fulfils the accessibility requirements sounds a simple enough approach for everyday development. The result here could indeed be a checklist.

There are multiple possibilities when bringing the criteria down to a more concrete level. First is technology-neutrality of the criteria - they can be applied on HTML, PDF, Flash, etc. However, Sanako Connect is an HTML web application, so the requirements can focus purely on HTML. The product does use React as its front-end framework, which abstracts the actual HTML. Should the requirements be more about React than HTML?

Rendered HTML will be the thing affecting accessibility most. And parts of the app may be developed using another framework or without a framework at all, so it would make sense to focus just on HTML. This would make the checklist more general.

It also means that WAI-ARIA can be utilised where needed since it is a technical specification that defines additional HTML attributes, as discussed in Section 3.5.3.

As discussed in Section 2.1.2, there are multiple ways how HTML elements can be constructed. Furthermore, sometimes using semantic HTML is preferred to WAI-ARIA, while in some cases WAI-ARIA can be used to augment the existing HTML. The requirements should then be written on a level that gives enough freedom for different approaches while making sure the accessibility does not suffer. The WCAG documentation has examples of successful HTML techniques for each criterion. These were used when constructing the requirements.

The full list can be found in Appendix B. Some criteria needed more than one requirement, and the same requirement could cover others. By chance, there are exactly 50 requirements like there were 50 WCAG criteria. However, they still need more contextualisation.

The original WCAG criteria are divided into the four principles - perceivable, operable, understandable, and robust - which is a logical division and one that is important to keep in mind. But one can see that some of the requirements are more related to each other than others. Some are directed more towards the application as a whole, some related to design, and some to just UI components, for example.

As the application consists of smaller units, maybe modularisation would be the answer here. Categorising the requirements would allow for the same requirement to be in multiple categories if necessary, but it would make multiple smaller checklists instead of one massive one. It could be used for developing new components or when refactoring existing ones.

The checklist can be found in Appendix C. It is not included in the thesis itself to keep the length in check, and the arguments on how it was constructed are more important. Also, the requirements list presents only one possibility of how to make the guidelines more concrete. Its usefulness will be discussed in Chapter 7.

It should be noted that a checklist like this is in no way a magic bullet for web accessibility. Even though fulfiling the WCAG criteria is necessary, the checklist is more a tool for ensuring that a certain baseline is achieved. The next chapter looks more closely on how the checklist could be used in real development, as the remaining research questions are discussed.

Chapter 6

Implementing web accessibility on Sanako Connect

The chosen guidelines and requirements that were developed based on them will be put to the test in this chapter, in which the remaining two research questions are discussed.

Web accessibility is something that should be thought of both on an organisational level and a project level. Even though this thesis has discussed web accessibility from many angles, here the scope is strictly on one project, Sanako Connect. And even as the web accessibility of that project will be important to stakeholders like sales and marketing and actions from this thesis will be relevant to them as well, the focus will be on software engineering. Finally, as will be discussed, the implementation work is something that very well could take several months, so another focal point is the development workflow and early phases of the implementation.

6.1 Development process background

The work done in previous sections enables us to find a way forward toward a more accessible product. As a reminder, the remaining research questions are about modifying the development process to ensure web accessibility is thought of and implementing the requirements to an existing application. This section will discuss potential changes to the development process. The changes, along with the previously defined requirements, will then be tried out in practice with the actual implementation in the following section.

Since the application is constantly changing - new features are added and current ones are changed, there is a big UI facelift coming soon, and so on - it is important that web accessibility would be an integral part of the development process. Current practices do not support or enforce web accessibility, which is why it is important to investigate changing them.

General

Currently, the development of new components and functionality is completely tied to Sanako Connect itself. This means that whatever is being developed, a new build of Sanako Connect is required to be deployed on a test server or run locally on the developer's computer. This is not ideal, for many reasons. It is cumbersome to share new components for commenting and to write automated tests for them. As the workflow would need to be adjusted in any case to accommodate a new designer, this was deemed to be a good opportunity to re-think our approach and also to incorporate accessibility into it.

Requirements were set for this new workflow:

- Components should be able to be developed and viewed in isolation, with no connection to Sanako Connect
- Components should be able to be viewed in different states
- Components should be interactive

This should yield at least the following benefits:

- Components can be shared for testing and commenting easily
- Different component states can be easily triggered without the need to execute possibly long user paths on Sanako Connect
- The solution could be used for both 3rd party components and ones developed by ourselves

It should be noted that nothing above is directly related to accessibility. Rather they are best practices for development and testing. However, the changes should also make it easier to focus on accessibility also:

- Handling the components in isolation will make also testing them in isolation easier - and this testing can also mean accessibility testing
- The components could be tested both automatically and manually
- Triggering different visual states is required to see if accessibility guidelines are followed, for example, if a button in different states still has enough contrast or if the keyboard focus is visible
- 3rd party components can be tested as well since their accessibility is our responsibility[93]

As described in Section 4.2.2, Sanako Connect's front-end is built with React. More specifically, the application was set up with the Create-React-App (CRA) command-line interface. CRA takes care of the build process, and there are limits to what you can configure.

To be even more precise, what is used in Sanako Connect is actually a configuration layer on top of Create-React-App. Create React App Configuration Override (CRACO) allows for more customisation than CRA. It was incorporated into the toolchain by necessity. The build process of the main UI library, Semantic UI React, only worked in

Node version 10 (the current active version is 12, soon to be 14). Since the back-end and front-end of Sanako Connect are in the same repository, it locked the whole project into Node 10. An alternative build process was introduced by the Semantic UI React project this year, and it required CRACO.

CRA's own documentation suggested two solutions for developing components in isolation: Storybook and Styleguidist[94]. The documentation provides Getting started guides on both. Storybook is described as a "development environment for React UI components" and Styleguidist as a combination of style guide and usage examples, with the possibility of developing components in isolation. Both tools are free and can be installed easily through NPM, so it was decided to try them both at the beginning of the implementation phase.

Automatic tooling

In tandem with finding a new way to develop components, it was investigated if any automatic tools could be included in the development process as well. As mentioned in Section 2.2.1, the build phase can be used for all kinds of tasks, including testing.

As described in the previous section, the front-end portion of Sanako Connect is made with React and set up with CRA. CRA's default build process includes a code linter, ESLint. It automatically checks the code against a pre-defined set of rules. This ruleset is minimal intentionally but can be extended[95]. For Sanako Connect, default ESLint rules are used, and a code formatter called Prettier is used for formatting the code. So ESLint is used for checking syntax, and Prettier on how the code looks.

For possible solutions, React's own documentation on accessibility proved to be a good resource. In addition to general guidelines, they provide instructions on accessibility tooling[96]. According to the documentation, an ESLint plugin called eslint-plugin-jsx-a11y is already included, with a subset of the rules enabled. The plugin project describes the plugin to be a "[s]tatic AST checker for accessibility rules on JSX elements"[97].

AST stands for Abstract Syntax Tree, and in this case, it means that it will go through the transpiled HTML markup that CRA outputs and checks it for accessibility violations.

React's documentation also mentions several browser accessibility tools. Out of these, the tools provided by Deque Systems seemed worthwhile for testing. Their tools include a browser extension called aXe (short for Accessibility Engine) and a module specifically for React, react-axe. The difference with these and a static tool like eslint-plugin-jsx-a11y is that these will test the actual output of the browser instead of the static markup.

Both eslint-plugin-jsx-a11 with a complete ruleset and the react-axe module were decided to be tested during the implementation phase. It is important to keep in mind that no automatic tool can catch all accessibility violations. Like mentioned before, some of the criteria are subjective, and can only be checked on by a human. Deque Systems themselves mention that automatic tooling such as theirs can catch only up to 50% of violations[93].

Manual testing

As has become evident at this point, automatic tools are not enough for accessibility testing and ensuring that the application actually works with alternative input modalities is more important than just mechanically fulfiling the WCAG criteria.

Although different input modalities, like keyboard, are mentioned in the requirements, it is worthwhile to mention them separately and make them a part of the general development workflow. Testing the application and thinking about the design of different UI states has been very mouse-centric. It is important to think about other ways of interacting with the application to make it more inclusive.

This is why keyboard navigation and screen reader navigation are going to be tried out during the implementation phase. Each component and the application as a whole needs to be able to be navigated with a keyboard. The same goes for screen readers.

The screen readers selected for testing were NVDA on Windows and VoiceOver on

Safari. According to a survey ran by a non-profit accessibility organisation WebAIM, the most popular screen readers are NVDA and JAWS, both used by a bit over 40% of respondents. The third most popular option with 12.6% was VoiceOver, the built-in screen reader in macOS[98].

JAWS and NVDA are both Windows applications. Historical trends in the survey show that JAWS is losing popularity, and NVDA is gaining it. JAWS is proprietary software with monthly and annual licenses, while NVDA is open-source and free. Even though every speech reader behaves differently, it was simpler to use just NVDA for this test on Windows. Together with VoiceOver, macOS and Windows are both covered, and the used screen readers represent a sizable chunk of real-world users. The survey mentions NVDA with Chrome being used by 18% of respondents (the most popular option was JAWS with Chrome, 21.4%). Since Chrome is the recommended browser for Sanako Connect, the test combinations were decided to be NVDA with Chrome on Windows and VoiceOver with Safari on macOS.

The survey also reveals other relevant data for development. For example, 86.3% of respondents use a screen reader on a mobile device or tablet, which means that the screen reader testing should be at some point carried out with mobile devices also.

This section covered the possible changes for the development workflow. The research question was not yet answered, as testing the changes and new tools will be discussed in the following section.

6.2 Changes to the development process

As planned, three items were tried during implementation: isolated component development, automatic tools, and keyboard and screen reader inclusion. Trying out the development tools happened in parallel with the implementation. For the sake of clarity, they are discussed here mostly separately.

Storybook versus Styleguidist

The requirements for development tools were discussed in Section 6.1. To recap, the goal is to find a tool for isolated development with the ability to view components in different states and to interact with them.

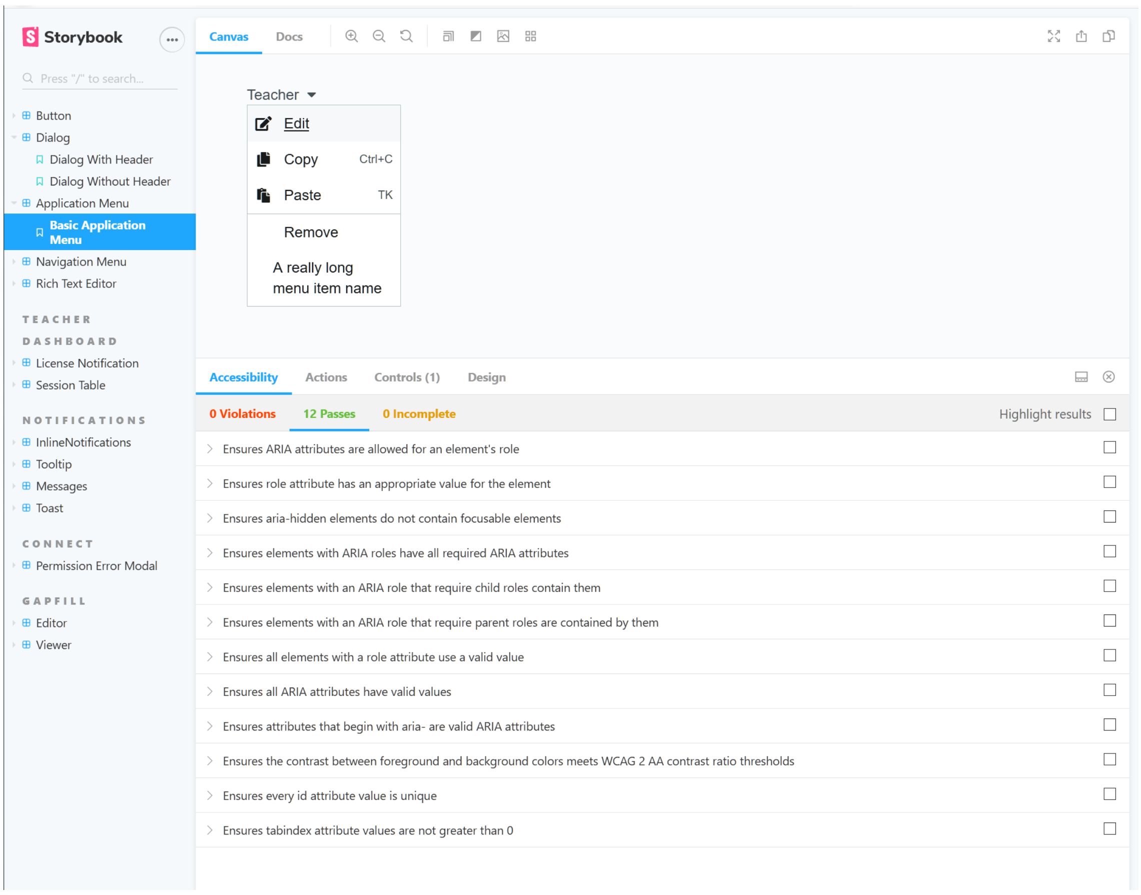
Both Storybook and Styleguidist are open-source tools, with their project management done in GitHub. At the time of writing, Styleguidist has 9200 “stars”, a GitHub metric that can be used as a proxy for their popularity. Storybook has over 53 000 stars, making it one of GitHub’s most popular projects.

Installing Storybook for the project was not straightforward. With CRA, it should work out-of-the-box, but remember that the project uses CRACO instead of CRA. This is an example of 3rd party tools clashing with each other - CRACO is used because Semantic UI React requires it, but using CRACO complicates using Storybook. Luckily a solution was finally found from writing custom configuration for CRACO.

With Storybook, you write “stories”, which can then be viewed with the Storybook browser application. Basically, the stories represent components in different states. The parameters each component is using can be modified in real-time. Storybook also has several add-ons. For Sanako’s development needs, the accessibility add-on and Figma add-on proved to be valuable. The accessibility add-on uses the previously mentioned aXe engine and displays the results in the Storybook interface. In effect, this provides the same functionality as the react-axe plugin, but only for an isolated component. The results might be different for a component that in the actual application would be, e.g. rendered on top of a different colour background, thus changing the contrast between foreground and background.

The Figma add-on allows embedding documents straight from Figma into Storybook. This makes it possible to compare the original design and the actual component directly.

Styleguidist works a bit differently. It generates documentation based on comments in the component source code and separate Markdown files. This means that the code is

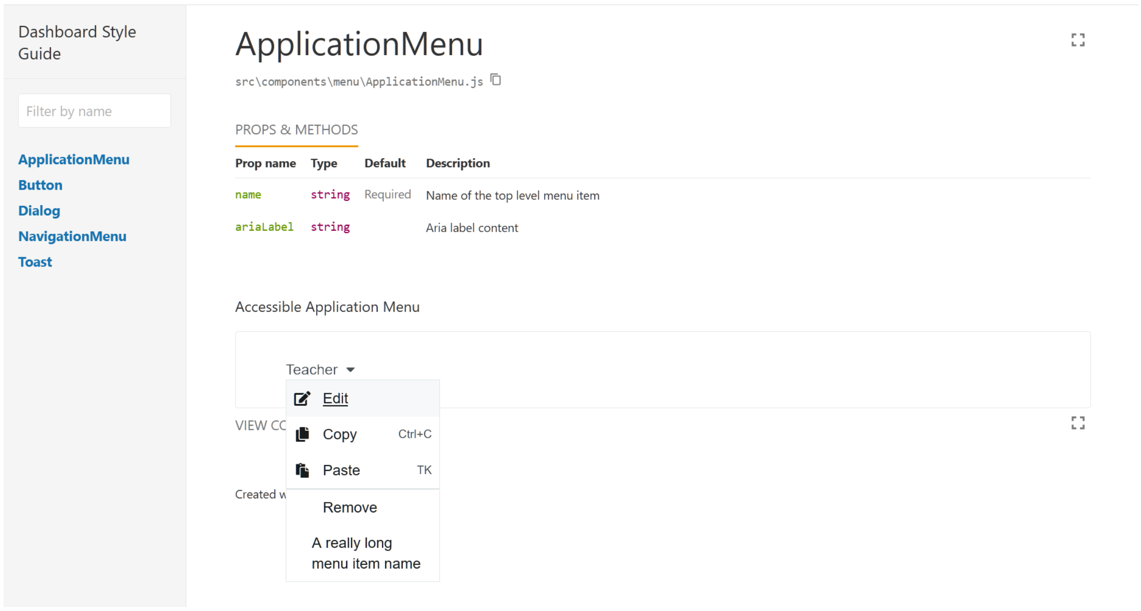


Ensures ARIA attributes are allowed for an element's role
Ensures role attribute has an appropriate value for the element
Ensures aria-hidden elements do not contain focusable elements
Ensures elements with ARIA roles have all required ARIA attributes
Ensures elements with an ARIA role that require child roles contain them
Ensures elements with an ARIA role that require parent roles are contained by them
Ensures all elements with a role attribute use a valid value
Ensures all ARIA attributes have valid values
Ensures attributes that begin with aria- are valid ARIA attributes
Ensures the contrast between foreground and background colors meets WCAG 2 AA contrast ratio thresholds
Ensures every id attribute value is unique
Ensures tabindex attribute values are not greater than 0

Figure 6.1: A typical Storybook view showing an application menu component. Results from the accessibility add-on can be seen on the lower right.

written inside Markdown code blocks. In practice, we noticed that the way Styleguidist works encourages to keep the documentation up to date.

On the surface, both solutions fill the requirements, and using them is easy. Any changes in the component re-renders the result automatically, making rapid testing possible. While trying the tools, it became clear that Storybook is aimed more towards development and Styleguidist towards documentation. Storybook also has other factors going for it, like the accessibility and Figma add-ons supporting it. Finally, the story format of Storybook is just normal JavaScript. Compared to this, the Markdown-based format felt clumsy and restricting. For these reasons, Storybook quickly became the favoured solution.



ApplicationMenu

`src\components\menu\ApplicationMenu.js`

PROPS & METHODS

Prop name	Type	Default	Description
name	string	Required	Name of the top level menu item
ariaLabel	string		Aria label content

Figure 6.2: The same application menu component viewed in Styleguidist.

The tools were used with both existing and new components. The difference between bringing in old components and developing new components highlights the benefits of isolated development.

The old components were imported with varying degrees of success. Some components were so tightly coupled with their parent components that extensive refactoring would have been needed for bringing them in Storybook.

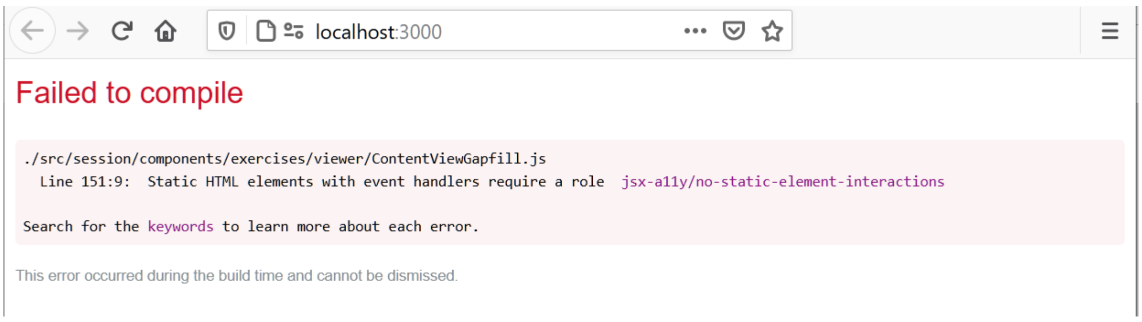
The new components were developed from scratch in isolation. This helped in thinking about the purpose of the component more clearly and to think about the abstraction it provided over the DOM. Different states, even complicated ones, were much easier to test than with the old development workflow.

New components were also tested with keyboard and a screen reader. This made it easy to ensure that they worked as they should and to think about all the different UI states that were needed for each component. For example, it was quickly understood that visually separating a hover state (where you move your mouse over a component) and a focus state (where you use the keyboard's Tab key to navigate and focus into an element one at a time) for a UI component is important and needs to be included into our design system for each component.

Automatic tools tryout

The automatic tools consisted of eslint-plugin-jsx-a11 with all the rules enabled and the react-axe module. Eslint-plugin-jsx-a11y ran as part of the command line build process, and react-axe results were displayed in the browser console.

Eslint-plugin-jsx-a11y was quickly removed from the normal build process. It essentially blocked compiling a new build until all the errors it found would be fixed, and it would display only one error at a time. Turning the errors into warnings that do not block the build would have been possible in the configuration, but it would have taken time, and the warnings would have still flooded the terminal. At this point, it did not make sense to go after every error.



Failed to compile

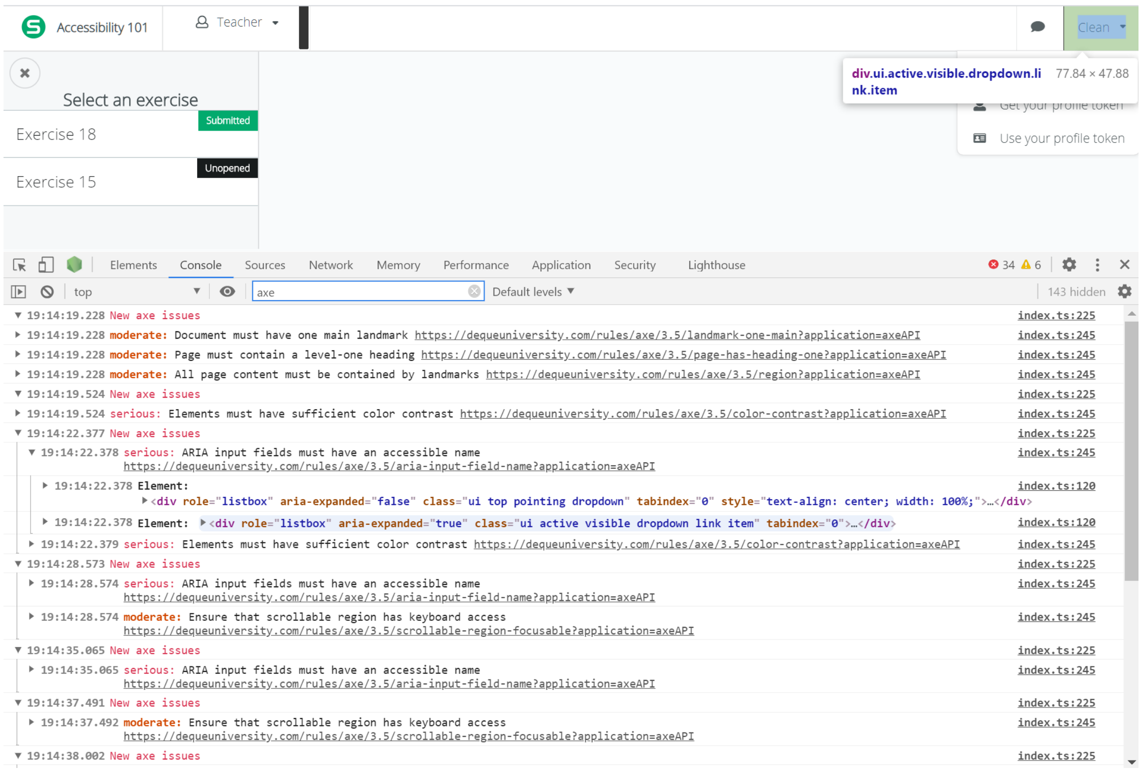
```
./src/session/components/exercises/viewer/ContentViewGapfill.js
  Line 151:9:  Static HTML elements with event handlers require a role  jsx-a11y/no-static-element-interactions

Search for the keywords to learn more about each error.
```

This error occurred during the build time and cannot be dismissed.

Figure 6.3: Eslint-plugin-jsx-a11y blocking the build.

React-axe, on the other hand, did not block anything, since it runs in the browser at the same time as the application. It did found several errors instantly, and it was easy to match the error to the correct component. The results were overwhelming since react-axe runs against the whole application. For this reason, react-axe was kept as a development tool, but it was moved behind a toggle so it would be run only when the developer wishes so. At this point, it made more sense to focus on particular areas of the application instead of trying to fix everything at once, and the accessibility add-on in Storybook gives roughly the same results.



```
▼ 19:14:19.228 New axe issues                                                                                        index.ts:225
▶ 19:14:19.228 moderate: Document must have one main landmark https://dequeuniversity.com/rules/axe/3.5/landmark-one-main?application=axeAPI    index.ts:245
▶ 19:14:19.228 moderate: Page must contain a level-one heading https://dequeuniversity.com/rules/axe/3.5/page-has-heading-one?application=axeAPI  index.ts:245
▶ 19:14:19.228 moderate: All page content must be contained by landmarks https://dequeuniversity.com/rules/axe/3.5/region?application=axeAPI   index.ts:245
▼ 19:14:19.524 New axe issues                                                                                        index.ts:225
▶ 19:14:19.524 serious: Elements must have sufficient color contrast https://dequeuniversity.com/rules/axe/3.5/color-contrast?application=axeAPI  index.ts:245
▼ 19:14:22.377 New axe issues                                                                                        index.ts:225
  ▼ 19:14:22.378 serious: ARIA input fields must have an accessible name                                             index.ts:245
          https://dequeuniversity.com/rules/axe/3.5/aria-input-field-name?application=axeAPI
    ▶ 19:14:22.378 Element:                                                                                          index.ts:120
          ▶<div role="listbox" aria-expanded="false" class="ui top pointing dropdown" tabindex="0" style="text-align: center; width: 100%;">…</div>
    ▶ 19:14:22.378 Element: ▶<div role="listbox" aria-expanded="true" class="ui active visible dropdown link item" tabindex="0">…</div>  index.ts:120
  ▶ 19:14:22.379 serious: Elements must have sufficient color contrast https://dequeuniversity.com/rules/axe/3.5/color-contrast?application=axeAPI  index.ts:245
▼ 19:14:28.573 New axe issues                                                                                        index.ts:225
  ▶ 19:14:28.574 serious: ARIA input fields must have an accessible name                                             index.ts:245
          https://dequeuniversity.com/rules/axe/3.5/aria-input-field-name?application=axeAPI
  ▶ 19:14:28.574 moderate: Ensure that scrollable region has keyboard access                                         index.ts:245
          https://dequeuniversity.com/rules/axe/3.5/scrollable-region-focusable?application=axeAPI
▼ 19:14:35.065 New axe issues                                                                                        index.ts:225
  ▶ 19:14:35.065 serious: ARIA input fields must have an accessible name                                             index.ts:245
          https://dequeuniversity.com/rules/axe/3.5/aria-input-field-name?application=axeAPI
▼ 19:14:37.491 New axe issues                                                                                        index.ts:225
  ▶ 19:14:37.492 moderate: Ensure that scrollable region has keyboard access                                         index.ts:245
          https://dequeuniversity.com/rules/axe/3.5/scrollable-region-focusable?application=axeAPI
▼ 19:14:38.002 New axe issues                                                                                        index.ts:225
```

Figure 6.4: Results of react-axe on the student application. Highlighted is one component that triggers a react-axe issue.

Keyboard and screen reader navigation

This part of development tool testing was the most simple and also the most revealing. Keyboard and screen readers were used as part of isolated component development, and they were also tested with the whole application. As mentioned, the used screen readers were NVDA on Windows and Chrome, and VoiceOver on macOS and Safari.

Trying to use Sanako Connect with only a keyboard quickly revealed its shortcomings. The student application was impossible to use.

Here are some issues that were discovered and that are related to several accessibility requirements. They will be discussed more in Section 6.3.2.

- Header menu items did not work
- Dropdown menus did not work
- Chat tab panel did not work (this could only be tried after opening the chat side panel with a mouse)
- Exercise menu did not work
- When entering a session as an existing user, there was no way to activate the welcome screen's Continue button, keeping the user stuck here
- When a modal window appeared, keyboard focus was somewhere hidden behind it, and you could focus on elements outside the modal.

Since screen reader users typically use some other input modality than a mouse, like a keyboard, the results were not better when a screen reader was included. The user was as equally stuck. When using the application with a screen reader and a mouse, new issues were revealed. There were several places where ARIA roles, labels, or correct semantic HTML elements were missing, so a user who cannot rely on visual aids would be completely lost.

Using a keyboard and screen reader with isolated component development yielded good results since any shortcomings of new components could be instantly fixed.

6.3 Modifying an existing application

The previous section went through the changes in the development process and tools. This section begins with a short description of what will be prioritised during the implementation.

It should be noted once more that the actual implementation work could take several months. That is why the focus here is on the last of the research questions - what are the challenges and solutions when implementing the requirements to an existing application. The initial phases of the implementation will hopefully be comprehensive enough to answer the question.

6.3.1 Prioritisation

The ultimate goal is to have the whole application be accessible - and WCAG compliant - but not everything can be fixed instantly. WAI recommends prioritising high-impact and low-effort repairs. With high-impact repairs, they refer to WCAG Level A issues, issues which are critical to complete processes, and issues that appear on multiple or frequently-used web pages. With low-effort repairs, they refer to issues that are cheap or fast to fix or require less testing[99].

Because the majority of Sanako Connect users are students, it would make sense to prioritise the part of the application used by students. The student part of the application also has less functionality than the teacher application, making it faster to repair and simpler to try out new approaches. Then again, part of its accessibility depends on the content made by teachers with the exercise editor. The editor would need to output accessible content (and the teachers would need to know how to make it) to make the student application accessible.

Eventually, the following prioritisation was decided:

- The automatic tools and keyboard + screen reader navigation will be tested first and see what could be fixed based on their findings (possible low-effort repairs)
- Primary focus should be on the student application's basic layout, which should also give some high-impact and fast results. The expectation is that the fixes done here can be utilised in the other sub-applications also.

6.3.2 Implementation

As discussed in the previous section, the student application was prioritised for implementation. The exercises were largely left out since they are teacher-made content and will also require changes to the editor on the teacher application.

As Sanako Connect is already in production, there are practical limits and considerations what can be done to it and when. There are bug fixes, new features, and customer expectations that need to be met. There are hundreds of existing components in the UI, some 3rd party, some in-house. Adding a completely new element in the mix - web accessibility - is not without its challenges, as will be discussed next.

Starting point

To set the baseline, a new session with two exercises was created in Sanako Connect. Even though exercises themselves were not tested, having two exercises allowed testing the exercise menu. This new session was then tested with automatic tools, a keyboard and a screen reader, and against the checklist created in Section 5.3. As was already mentioned in the previous section, the initial results were not good.

The automatic tools - or rather react-axe, since eslint-plugin-jsx-a11y was quickly removed - were not as big a help as initially assumed, even though they did reveal accessibility issues. They did not catch nearly everything, but they served as a reminder for simple issues, like the contrast of the text.

But using the application with a keyboard and a screen reader helped to notice and actually understand the issues. As for the checklist, it was easier to focus on the application level requirements at first, and not focus on individual components. In the beginning stages of the implementation we are more after quick fixes and the low hanging fruit as recommended by WAI. This is why the focus was more on the most important features and issues which are critical to complete processes.

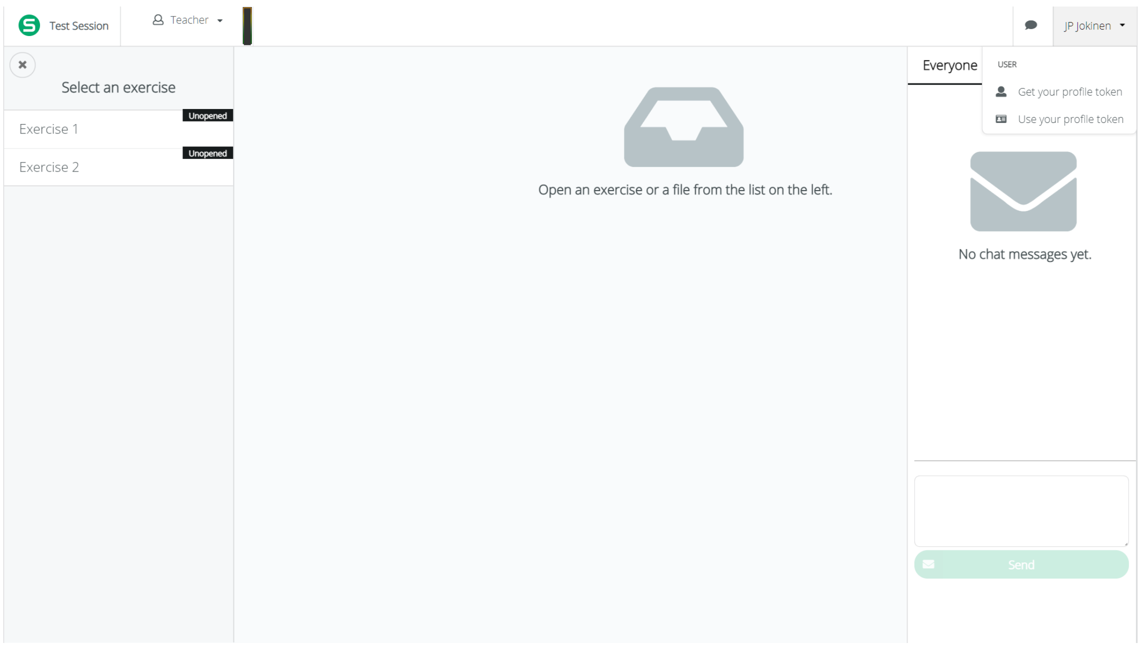


Figure 6.5: The student app before implementation project, with the menu and chat sidebar open.

Looking at the checklist in Appendix C, many of the application-level items are related to navigation:

R9: Use ARIA landmarks to mark the application structure

R10: Ensure that content is in a meaningful sequence in the source code (tables, for example)

R22: Ensure that all content can be navigated to and controlled with a keyboard

R27: Allow the user to skip repeated blocks (header) to reach the main content OR use semantic section markup

This is why a set of critical user goals were created and then made sure that the goals could be reached with a keyboard and a screen reader. The goals were as follows:

- The user should be able to select and open an exercise

- The user should be able to write a chat message to both session and group chat
- The user should be able to retrieve an access token
- The user should be able to send a direct message to the teacher

At first, there was not much the user could do without a mouse. All the menus were inaccessible, and the user was stuck at the welcome screen. The common factor was that all the inaccessible components were from the Semantic UI React library. A little research revealed that the library has many accessibility issues.

As discussed in Section 2.2, building interfaces with React means using modular React components which abstract the underlying HTML. With some Semantic UI React components, it also meant that necessary changes to the HTML elements could not be practically done. As the library is open-source, the source code is available, but changing it would have meant that either changes to it would be overwritten with the next library update, or that the library's GitHub repository would need to be forked for our own use, which is cumbersome.

In Sanako Connect, the Menu, Dropdown Menu, and Tab components were the ones causing the biggest issues. It became clear that the components were not developed with accessibility in mind.

No matter if the component was from Semantic UI React or our own, the issues were related to missing semantic markup or ARIA landmarks and roles which would make the application navigable with a keyboard and a screen reader. With some Semantic UI React components, it was possible to add the relevant markup. With others, like the Dropdown Menu, it would have required much work to make the component accessible.

Actions taken

The components that could easily be fixed were fixed. In some cases, it required only adding one new HTML attribute, tabindex, to the element, which made it focusable by keyboard. Additional ARIA roles and attributes would make the element usable with a screen reader so that the reader will allow the user to navigate directly between different sections of the web app and read aloud semantic information about it.

The worst component with regards to accessibility was Semantic UI React's Dropdown Menu, which is used in many places of the application. An application menu on a web page has detailed attributes and functionality for keyboard navigation it needs to fulfil, as described by in a tutorial by WAI[100]. Modifying the Dropdown Menu would have been tricky and taken much time, which is why a new application menu component was created according to the WAI tutorial. The checklist for Single components and UI components proved to help make sure every aspect of accessibility is covered.

The menu was one of the new components where Storybook was used for isolated component development. Its benefits became clear, as it was easy to create several stories with different content, and then interact with menus in those stories with a keyboard and a screen reader. Furthermore, the accessibility add-on here was helpful, since all the passes and fails were about one component only. The functionality could be completed while the designer worked on the design (and honoured the relevant accessibility requirements), and finally, the design was incorporated into the component.

Although the functionality needed to be created from scratch and developing the component took some time, it was deemed worth it because there were no limitations like with 3rd party components, and the component could be created completely according to our needs. And once the component was ready, it was easy to replace it gradually everywhere, not just in the student application.

New 3rd party components

Creating every component by ourselves is something that cannot be practically done among other development tasks. Apart from the application menu component, a navigation menu component was also created, because its functionality is so specific that no suitable readymade solutions were found.

For other common components, accessible 3rd party solutions were looked for. React has an active user community with hundreds of open-source components. Since Semantic UI React initially sped up development but ultimately caused issues, possible solutions were vetted for accessibility.

With that in mind, we settled on a React component library focused on accessibility, Reakit. The library is described as a lower lever component library for building accessible high-level UI libraries while strictly following WAI-ARIA 1.1 practices.

Reakit had detailed documentation not only of the components themselves but also on how accessibility was approached. An added benefit was that the components come without any styling, which was the desired case for us. Reakit also provided examples of how to abstract each component.

One example that illustrates both a tricky accessibility issue and the danger of focusing into one component at a time is Success Criteria 2.4.3, Focus Order:

“If a Web page can be navigated sequentially and the navigation sequences affect meaning or operation, focusable components receive focus in an order that preserves meaning and operability.”

What this means that if a user opens a modal window, the keyboard focus should shift inside the modal window and be available for only interactive components inside it. Once the modal window is closed, the focus should go back to the element that triggered the modal, if any.

In the student application, it is possible to open modal windows from the main user menu. To fulfil Success Criteria 2.4.3, the menu should stay open, the focus should go inside the new modal, and when it is closed, the focus should return to the menu which has stayed open.

After the menu components were finished, making the modal windows accessible was the next step. For this, Reakit’s Modal component was used, because it was deemed suitable for our purposes and would save development time. However, it did not work correctly with the previously made Menu component. In the end, it was faster to do a new abstraction on top of Reakit's Menu component, since it was already known that it would work with the Reakit Modal.

Doing the first Menu component was not a complete waste of time, however, because it deepened the understanding of what is actually required from an accessible component.

Other challenges

The previous section demonstrated a challenge that is related more to software development in general. With regards to accessibility, this implementation phase showed that the solutions are not always straightforward. Requirement 9 tells to use ARIA landmarks to mark the application structure. This is related to Success Criteria 1.3.1: Info and Relationships: "Information, structure, and relationships conveyed through presentation can be programmatically determined or are available in text".

With ARIA landmarks one issue was that it was a subjective decision which sections were marked with them. This crosses over to UX design - what would benefit the user most.

As the image above shows, giving more context to the user with ARIA landmarks is probably better.

In the end, a lot was accomplished during the initial implementation period. Trying to fulfil every requirement at once was not practical. This is why the focus was on those requirements that made the biggest impact, like ensuring that all content can be navigated with a keyboard.

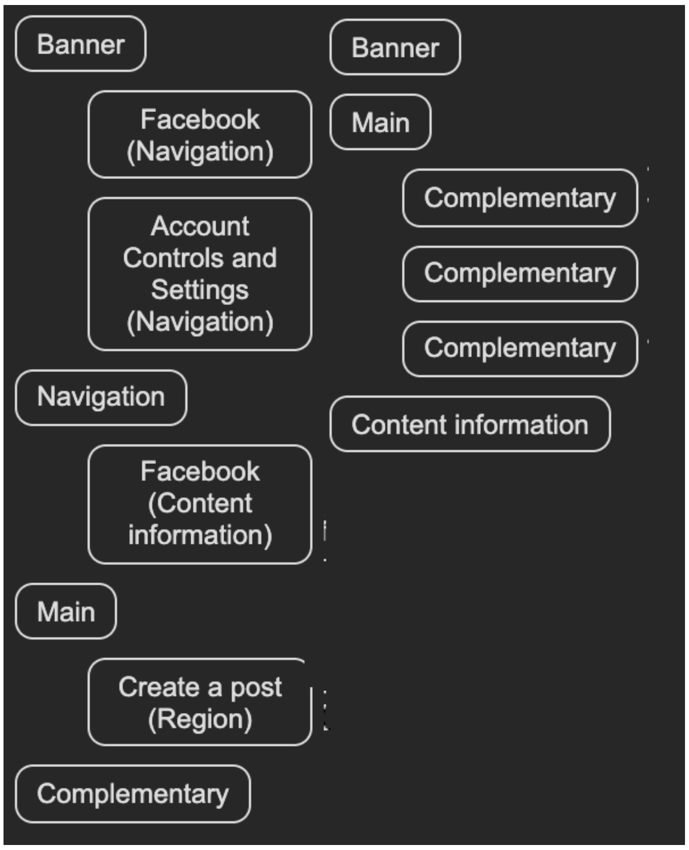


Figure 6.6: Example of ARIA landmarks from two web sites, as illustrated by a Chrome add-on ‘Landmarks’. On the left is Facebook’s landmarks, on the right is Helsingin Sanomat.

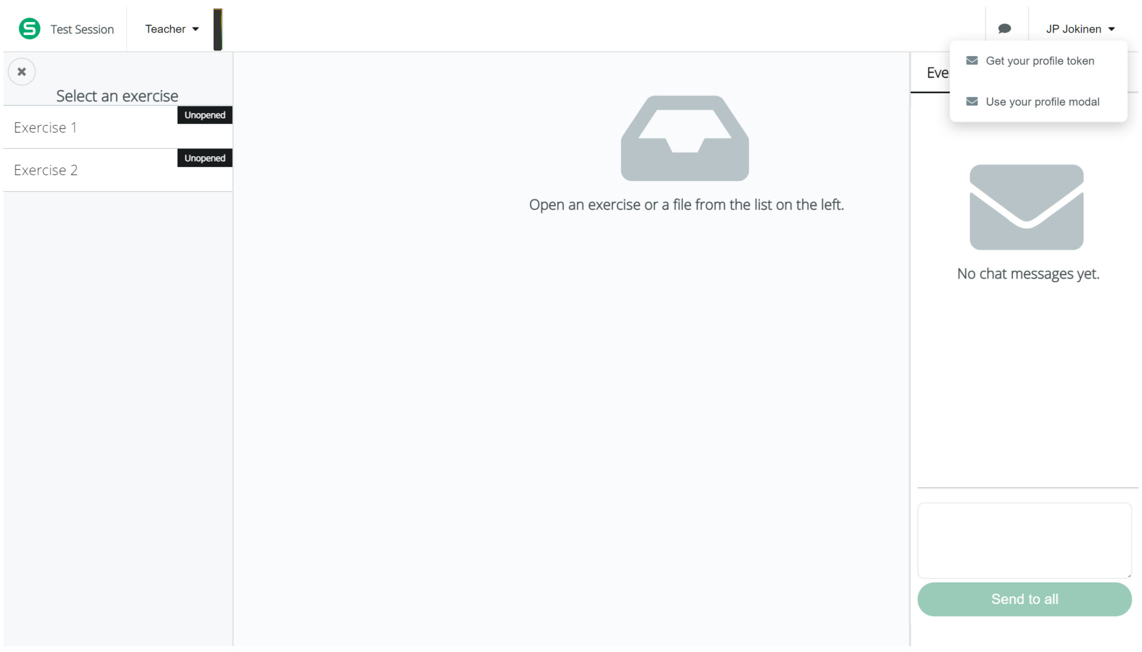


Figure 6.7: The student app after implementation project, with the new dropdown menu. As one can see, visually the app has not changed much. The green colour was changed to a darker one so that buttons would have the necessary contrast.

Chapter 7

Discussion

Although some of the research questions have been touched upon already, the practical part of this thesis has given new insight for all the questions. That is why this chapter discusses all of them and tries to validate their answers.

7.1 What accessibility guidelines a globally sold educational software like Sanako Connect should follow?

As discussed in Section 5.1, the legal requirements are straightforward. WCAG Level AA compliance is required in the United States (through Section 508 and ADA) and the European Union. Sanako Connect being an educational software only gives minimal leeway with few of the criteria.

WCAG can be considered to be the de facto accessibility standard. It is important to keep in mind that different countries can still have different requirements, and in the case of the United States, there is accessibility legislation at the state level. Those were outside the scope of this work. Also largely absent were the voluntary WCAG Level AAA Criteria and ATAG, although aiming to fulfil at least some of their criteria would be beneficial for accessibility.

It is also important to remember the point that has been repeated time and time again throughout this work: web accessibility should not be just about checking items off the list. Instead, the focus should be real users and their needs. The legal requirements do set the required baseline, but the real question should always be how web accessibility could be included and improved.

A related example was discussed in Section 3.5.1. The RTC Accessibility User Requirements are still only in a draft stage. This does not mean that the accessibility of real-time audio and video streaming should not be thought of. It just means that there are no official guidelines for that yet.

The general viewpoint that will be discussed with each of the research questions is that there is little reason not to include web accessibility as an integral part of practically every new web application or web site project. Thinking about it afterwards is always the hard road, and the benefits go beyond just fulfiling the possible legal needs.

The approach to web accessibility here has been centred on legal requirements and WCAG. It would be interesting to see research that approaches web accessibility from another angle - maybe from the viewpoint of inclusive design or user experience, setting the legal basis aside. This is because web accessibility is also very much a matter of design and UX, even though the implementation level challenges are more in the domain of software engineering.

7.2 How can the accessibility guidelines be transformed into more concrete requirements?

Section 5.3 showed one possible way to turn the WCAG criteria into development requirements. The requirements were focused on HTML and based on WCAG's own suggestions where possible. Although the requirements were geared towards one software product, there is not much that is specific to Sanako Connect.

Chapter 3 discussed the challenges related to the guidelines. Their technology-neutral and hard-to-understand nature means that they need more concreteness to be useful in every-day development work.

While going through the application and developing new components against the requirements, it was noticed that depending on the requirement, one still needed to be familiar with the criteria behind it. This happened even with simple requirements, like the minimum text contrast. This is because there are always edge cases and complex use cases, where the reasoning behind the criteria needs to be understood. Furthermore, all that cannot be conveyed in the requirements, and they need not to, since WCAG has plenty of material.

This means that certain expertise of HTML and its related areas like WAI-ARIA is very much relevant, even with all the abstractions and frameworks created on top of HTML. One could say that the abstractions and frameworks can also cause issues, as was discussed in Chapter 2. This ties in with the "expertise matters" discussion from Section 3.6.2. Although the studies there were concerned about the difficulty of evaluation, it fits in the other end of the development pipeline also.

In fact, to allow for a personal anecdote, much of the requirements about valid HTML, semantic elements, etc., can already be found in web design books from the 1990s, when the writer of this thesis started with web development. This enforces the point made in the previous section of including web accessibility into every new project. Much of web accessibility is just about good web development practices. The benefits of following good development guidelines would not be limited to web accessibility.

Although Sanako Connect is a web app, and there are no native iOS or Android apps, this work largely left out mobile devices and touch screens. Although the WCAG criteria and the requirements do take small screens into account, mobile still needs special attention, especially in a complex application like Sanako Connect. That is something that maybe falls more under implementation and design than the requirements themselves.

As the WCAG criteria were developed before the birth of web applications, they are more focused on web sites than web apps. Further research could be found on comparing the relevance of the requirements on web sites and web apps. For example, Criteria 2.4.5, Multiple Ways, which states that there should be more than one way to reach a web page within a wet of pages is something that is much more beneficial for web sites.

The importance of each criterion was discussed in Section 5.2. Further research could point out if the criteria could be weighted in some way. Although WCAG compliance requires fulfiling all of the criteria, limited development resources would benefit from focusing on the most beneficial criteria first. The research by Shirogani suggests that end-users' characteristics could be used to prioritise accessibility requirements. Although there is some truth in that, and they speak of specialised software (versus general use software), it can also be quite risky to make any assumptions about the potential users of the software. Furthermore, it assumes that there is no benefit in fulfiling the criteria for other than users with specific disabilities, which is not the case, as discussed in Section 3.2.2.

7.3 How to modify the development process so that accessibility will be thought of in future development and revisions of the software?

The practical part of this thesis started with focus on improving the development process. Section 6.1 described what was looked for in the new development process. Section 6.2 tested the new process and tooling.

The results tell that there are many suitable tools available for a development process that supports web accessibility. The automatic accessibility tools that were tried out during the testing did not give great results, as was expected based on studies discussed in Section 3.6. But manual testing with a keyboard and a screen reader proved to be essential.

Not only were the development changes beneficial for web accessibility, but they also had other benefits. Isolated component development made it easier to developt and test the components. It also forced to think about the composition and responsibility of individual components, enforcing good software development practices.

Automated and end-to-end testing frameworks like Cypress and Microsoft's Playwright were outside the scope of this work. Since those kinds of frameworks can be used to control actual browser instances and thus the real rendering output, it is probable that custom accessibility testing solutions could be built on top of them. For example, keyboard navigation could be tested automatically and made sure that all user actions are possible with only a keyboard.

Not only is this a possible next step with Sanako Connect's development, but a valid future research question is also what could be achieved with this kind of automated accessibility testing. The difference to the automated tools discussed in this thesis is that the tests and user flows would be created for a specific project, and the tool itself would not provide them. This would allow for precise tests and evaluation.

7.4 What are the challenges and solutions when implementing the requirements to an existing application?

The previous research questions all led to this one. To implement web accessibility, you have to know what are the requirements. To know the requirements, you need to know what guidelines to follow. The second practical part of this thesis was about implementing the requirements to an existing application, Sanako Connect. That was described in Section 6.3.

One important part of the implementation was deciding on the prioritisation. It takes time to implement all the requirements, and you have to start somewhere. Luckily with Sanako Connect, it was possible to find a clearly defined part of the application where to focus on first. After that, the implementation depended more on the current application - what requirements are not met, what is the current implementation and so on. Since there were over 50 requirements, and the work is still in progress, this thesis can not cover everything.

In many ways, we were in an ideal position to start the web accessibility implementation project. There was the necessity from legislation and pressure from clients, which meant that management buy-in was easy. As was discussed in Section 3.6, many of the challenges are not related to actual software engineering work.

In the same section, it is stated that creating an accessible product is not difficult, and for the most part, the experiences from the implementation phase validate that statement. There are some criteria which are subjective or hard to understand, and using WAI-ARIA can be tricky, as discussed in Section 3.5.3. As was discussed in the previous section, gaining expertise in HTML and related areas is really the solution for overcoming those kinds of issues. This means that the act of researching thesis has been part of the solution.

For the most part, the challenges came from previously made decisions. Most issues were related to 3rd party components. The new development process described in Section 6.2 discovered a failure to adhere to good software development practices in component development.

Although the particular challenges discovered were specific to Sanako Connect, the reasons behind them are universal. Using 3rd party components is common, and not vetting the components properly can lead to problems later. Not just with web accessibility, but for example, with security. And poor development practices will come back to haunt you, even if initially they would speed up development.

If web accessibility for one reason or another becomes a necessity, implementation on an existing application is by no means impossible. However, it will still take up valuable development time.

Luckily it is possible to improve the situation quite quickly, as was shown in Section 6.3, where requirements related to keyboard navigation were dealt with. Nevertheless, fulfiling every single requirement is another issue altogether. Since front-end development is quite modular these days, it is possible to do the changes piece by piece with some planning.

Taking web accessibility into account could have big implications for development work and processes. But the impact is not limited to development. At the very least, it affects designers and UX research.

For these reasons, web accessibility should be considered an integral part of every new web project. The arguments for not doing so should be heavy. As was mentioned in Section 3.6, retrofitting costs more and takes more time than planning web accessibility from the start. The implementation project that was started for Sanako Connect is a prime example of that. Even the possible development speed gains could be negated later if circumstances change. The key takeaway to remember is that thinking about web accessibility will benefit all the users.

A more specific recommendation concerns 3rd party components. Their use is easy and common, but they should be vetted carefully before starting to use them on any project. This concerns not only web accessibility but, e.g. the security of the 3rd party components as well. When a small front-end component could have hundreds of dependencies, and there could be tens of these components on any project, the amount of external code imported into the project is huge.

One very concrete outcome is that the decision was made during implementation to move away from Semantic UI React components gradually. Accessibility alone was not the deciding factor. The Semantic UI React project is very much alive, but its parent project, Semantic UI, is in practice abandoned. It is a popular project with over 40 000 stars on GitHub. But it has only one maintainer, who last released an update in October

2018, which at the time of writing was almost two years ago. And as discussed, using Semantic UI React has locked us into using CRACO also.

The change does not happen overnight, especially since the Semantic UI React components are used all over the application. In hindsight, it would have made sense to create abstractions over Semantic UI React components and use them instead. Then, the internal implementations of those components could be changed freely. If the library had been initially vetted for accessibility, its issues would have become clear instantly, and it would not have been taken into use in the first place.

Chapter 8

Conclusions

By going through the history of web development and web accessibility and by modifying the product and its development process, this thesis found answers to the research questions.

Legislative demands are clear on the necessity of using Web Content Accessibility Guidelines 2.1 Level AA as the foundation for web accessibility, even though following other guidelines like Authoring Tool Accessibility Guidelines should be considered to improve the usability of the product. Transforming the guidelines into more precise requirements for the web is possible, although familiarity with the criteria from the original guidelines is still needed, as is expertise on HTML.

The practical part of this thesis modified the development process, showing the benefits of isolated component development and inclusion of keyboard and screen reader. The implementation project discovered issues with 3rd party components and previous development practices, which hopefully will be amended with the new development process. Vetting of 3rd party components for accessibility, among other things, is strongly suggested.

Finally, retrofitting web accessibility to an existing application always takes up time and resources, even in ideal circumstances. Web accessibility has many benefits beyond making the application usable to diverse users and planning for it in advance is the easier option. There hardly are valid arguments against it.

The work done for this thesis has already had an impact on the development and the actual product. Although the circumstances are different for each existing application, the findings can still be generalized, and they back up the sources discussed in this work.

The topic of this thesis was vast. It offers a multitude of possibilities for further research on more focused topics, like those mentioned in the previous chapter.

I hope that after reading this work, the very least you do is try out surfing the web for a few moments without a mouse (if you are not doing so already). The results may surprise you.

References

[1] Shawn Henry. Web content accessibility guidelines (wcag) overview, 2018. Last accessed 25 April 2020, URL: `https://www.w3.org/WAI/standards-guidelines/wcag/`.

[2] CERN. A short history of the web. Last accessed 17 April 2020, URL: `https://home.cern/science/computing/birth-web/short-history-web`.

[3] W3C. Help and faq. Last accessed 17 April 2020, URL: `https://www.w3.org/Help/`.

[4] Kadhim Shubber. First ever web page put back online by cern, 2013. Last accessed 17 April 2020, URL: `https://www.wired.co.uk/article/first-web-page`.

[5] W3C. Html5, 2014. Last accessed 17 April 2020, URL: `https://www.w3.org/TR/2014/REC-html5-20141028/introduction.html`.

[6] W3C. World wide web consortium (w3c) brings a new language to the web as webassembly becomes a w3c recommendation, 2019. Last accessed 13 April 2020, URL: `https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en`.

[7] MDN Web Docs. Html, 2019. Last accessed 17 April 2020, URL: https://developer.mozilla.org/en-US/docs/Glossary/HTML.

[8] Bert Bos. A brief history of css until 2016, 2016. Last accessed 17 April 2020, URL: https://www.w3.org/Style/CSS20/history.html.

[9] Paul Krill. Javascript creator ponders past, future, 2008. Last accessed 17 April 2020, URL: https://www.infoworld.com/article/2653798/javascript-creator-ponders-past--future.html.

[10] GitHub. The state of the octoverse, 2019. Last accessed 17 April 2020, URL: https://octoverse.github.com/.

[11] Jonathan Robie. What is the document object model?, 1998. Last accessed 28 April 2020, URL: https://www.w3.org/TR/WD-DOM/introduction.html.

[12] MDN Web Docs. Ajax, 2019. Last accessed 17 April 2020, URL: https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX.

[13] StatCounter. Global stats, 2020. Last accessed 17 April 2020, URL: https://gs.statcounter.com/.

[14] Chris Hoffman. Why third-party browsers will always be inferior to safari on iphone and ipad, 2014. Last accessed 29 March 2020, URL: https://www.howtogeek.com/184283/why-third-party-browsers-will-always-be-inferior-to-safari-on-

[15] Paul Brown. State of the union: npm, 2017. Last accessed 26 April 2020, URL: https://www.linux.com/news/state-union-npm/.

[16] Adam Barked. The super-brief history of javascript frameworks for those somewhat interested, 2018. Last accessed

28 April 2020, URL: https://dev.to/_adam_barker/the-super-brief-history-of-javascript-frameworks-for-those-som

[17] The State of Javascript. The state of javascript 2019: Front end frameworks, 2020. Last accessed 5 May 2020, https://2019.stateofjs.com/front-end-frameworks/.

[18] Stack Overflow. Stack overflow developer survey 2019, 2020. Last accessed 29 March 2020, https://insights.stackoverflow.com/survey/2019#technology-_-web-frameworks.

[19] React. Introducing jsx. Last accessed 5 May 2020, URL: https://reactjs.org/docs/introducing-jsx.html.

[20] Oleg Isonen. What actually is css-in-js?, 2019. Last accessed 5 May 2020, URL: https://medium.com/dailyjs/what-is-actually-css-in-js-f2f529a2757.

[21] WebAim. The webaim million, 2020. Last accessed 14 November, URL: https://webaim.org/projects/million/.

[22] J Clement. Mobile internet usage worldwide - statistics & facts, 2019. Last accessed 5 May 2020, URL: https://www.statista.com/topics/779/mobile-internet/7.

[23] MDN Web Docs. Progressive enhancement, 2019. Last accessed 5 May 2020, URL: https://developer.mozilla.org/en-US/docs/Glossary/progressive_enhancement.

[24] Google Developers Web. Twitter lite pwa significantly increases engagement and reduces data usage, 2017. Last accessed 5 May 2020, URL: https://developers.google.com/web/showcase/2017/twitter.

[25] United Nations. Article 9 – accessibility, 2007. Last accessed 5 May 2020, URL: https://www.un.org/development/desa/disabilities/convention-on-the-rights-of-persons-with-disabilities/article-9-accessibility.html.

[26] World Health Organization. Assistive technology, 2018. Last accessed 2 May 2020, URL: https://www.who.int/en/news-room/fact-sheets/detail/assistive-technology.

[27] HIE Help Center. Assistive technology. Last accessed 2 May 2020, URL: https://hiehelpcenter.org/treatment/assistive-adaptive-technologies/.

[28] Rebecca J Rose. Why are glasses perceived differently than hearing aids? *The Atlantic*, Dec 2013.

[29] Centre for Excellence in Universal Design. What is universal design. Last accessed 2 May 2020, URL: http://universaldesign.ie/What-is-Universal-Design/.

[30] Cynthia Curry. Universal design: Accessibility for all learners. 2003.

[31] Frank Bowe and Neal Little. Computer accessibility: A study. *Rehabilitation Literature*, 45(9-10):289 – 291, 1984.

[32] Aluehallintovirasto. Tietoa saavutettavuudesta. Last accessed 23 May 2020, URL: https://www.saavutettavuusvaatimukset.fi/tietoa-saavutettavuudesta/.

[33] Daniel Dardailler. Wai early days, 2009. Last accessed 21 May 2020, URL: https://www.w3.org/WAI/history.

[34] Jay Hoffman. Putting web accessibility first, 2017. Last accessed 21 May 2020, URL: `https://thehistoryoftheweb.com/putting-web-accessibility-first/`.

[35] Kari Selovuo. Saavutettavuusopas. 2019.

[36] Introduction to web accessibility, 2019. Last accessed 21 May 2020, URL: `https://www.w3.org/WAI/fundamentals/accessibility-intro/`.

[37] Diverse abilities and barriers, 2019. Last accessed 21 May 2020, URL: `https://www.w3.org/WAI/people-use-web/abilities-barriers/`.

[38] Shadi Abou-Zahra. Tools and techniques, 2017. Last accessed 24 May 2020, URL: `https://www.w3.org/WAI/people-use-web/tools-techniques/`.

[39] Darcy Ann Umphred, Rolando T. Lazaro, Margaret Roller, and Gordon Burton. Neurological rehabilitation. *Elsevier Health Sciences*, page 383, 2013.

[40] Eurostat. Internet use by individuals, 2020. Last accessed 22 May 2020, URL: `https://ec.europa.eu/eurostat/databrowser/view/tin00028/default/`.

[41] Kristin Fuglerud and Till Halbach. An evaluation of web-based voting usability and accessibility. *Universal Access in the Information Society*, 11, 11 2011.

[42] Sharron Rush. The business case for digital accessibility, 2018. Last accessed 23 May 2020, URL: `https://www.w3.org/WAI/business-case/`.

[43] Marie-luise Leitner, Christine Strauss, and Christian Stummer. Web accessibility implementation in private sector organizations: motivations and business impact. *Universal Access in the Information Society*, 15(2):249–260, 06 2016. Copyright

- Springer-Verlag Berlin Heidelberg 2016; Document feature - ; Last updated - 2016-08-03.

[44] World Health Organization. Why is the convention on the rights of persons with disabilities important?, 2013. Last accessed 22 May 2020, URL: `https://www.who.int/news-room/q-a-detail/why-is-the-convention-on-the-rights-of-persons-with-disabiliti`

[45] United Nations. Frequently asked questions regarding the convention on the rights of persons with disabilities. Last accessed 22 May 2020, URL: `https://www.un.org/development/desa/disabilities/convention-on-the-rights-of-persons-with-disabilities/frequently-asked-questions-regarding-the-convention-on-the-rig` `html#sqc5`.

[46] Mary Jo Mueller, Robert Jolly, and Eggert. Web accessibility laws & policies, 2018. Last accessed 22 May 2020, URL: `https://www.w3.org/WAI/policies/`.

[47] Garenne Bigby. United states web accessibility laws, 2019. Last accessed 22 May 2020, URL: `https://dynomapper.com/blog/27-accessibility-testing/569-united-states-accessibility-laws`.

[48] Kathryn R. Green and Steven Tolman. Equitable means accessible. pages 125–147, 2019.

[49] Terese L. Arenth. Ada web site accessibility claims on the rise: Practical strategies for defense. *Journal of Internet Law*, 23(4):1–14, 10 2019. Name - Winn-Dixie Stores Inc; Congress; Copyright - Copyright Aspen Publishers, Inc. Oct 2019; Last updated - 2019-11-20; SubjectsTermNotLitGenreText - United States–US.

[50] Minh Vu, Kristina Launey, and Susan Ryan. 2019 was another record-breaking year for federal ada title iii lawsuits, 2020. Last accessed 22 May 2020, URL: https://www.adatitleiii.com/2020/02/2019-was-another-record-breaking-year-for-federal-ada-title-ii

[51] Seyfarth Shaw. Ada title iii litigation: A 2019 review and hot trends for 2020, 2020. Last accessed 22 May 2020, URL: https://www.adatitleiii.com/2020/01/ada-title-iii-litigation-a-2019-review-and-hot-trends-for-2020

[52] Equidox. State accessibility legislation, 2020. Last accessed 22 May 2020, URL: https://equidox.co/accessibility-resources/state-accessibility-legislation/.

[53] Level Access. State and local laws, 2013. Last accessed 22 May 2020, URL: https://www.levelaccess.com/accessibility-regulations/state-local-laws/.

[54] European Union. Directive (eu) 2016/2102 of the european parliament and of the council of 26 october 2016 on the accessibility of the websites and mobile applications of public sector bodies, 2016. Last accessed 22 May 2020, URL: https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016L2102&from=EN.

[55] European Union. Directive (eu) 20162102 of the european parliament and of the council of 26 october 2016 on the accessibility of the websites and mobile applications of public sector bodies, 2016. Last accessed 22 May 2020, URL: https://directive2102.eu/.

[56] Eduskunta. Laki digitaalisten palvelujen tarjoamisesta, 2019. Last accessed 22 May 2020, URL: `https://www.finlex.fi/fi/laki/alkup/2019/20190306`.

[57] Aluehallintovirasto. Siirtymäajat. Last accessed 22 May 2020, URL: `https://www.saavutettavuusvaatimukset.fi/lait-ja-standardit/siirtymaajat/`.

[58] Aluehallintovirasto. Mitä palveluja ja sisältöjä laki koskee? Last accessed 22 May 2020, URL: `https://www.saavutettavuusvaatimukset.fi/lait-ja-standardit/mita-palveluja-ja-sisaltoja-laki-koskee/`.

[59] Joanmarie Diggs, McCarron Shane, Michael Cooper, Richard Schwerdtfeger, and James Craig. Accessible rich internet applications (wai-aria) 1.1, 2017. Last accessed 25 April 2020, URL: `https://www.w3.org/TR/wai-aria/`.

[60] Andrew Kirkpatrick, Joshue O'Connor, Alastair Campbell, and Michael Cooper. Web content accessibility guidelines (wcag) 2.1, 2018. Last accessed 25 April 2020, URL: `https://www.w3.org/TR/WCAG21/`.

[61] Alastair Campbell, Michael Cooper, and Andrew Kirkpatrick. Understanding wcag 2.1, 2020. Last accessed 25 April 2020, URL: `https://www.w3.org/WAI/WCAG21/Understanding/`.

[62] Aluehallintovirasto. Tietoa wcag-ohjeistuksesta. Last accessed 26 May 2020, URL: `https://www.saavutettavuusvaatimukset.fi/lait-ja-standardit/tietoa-wcag-kriteereista/`.

[63] WAI. Understanding success criterion 1.1.1: Non-text content. Last accessed 5 June 2020, URL: `https://www.w3.org/WAI/WCAG21/Understanding/non-text-content.html`.

[64] WAI. Understanding success criterion 1.2.1: Audio-only and video-only (prerecorded). Last accessed 8 June 2020, URL: `https://www.w3.org/WAI/WCAG21/Understanding/audio-only-and-video-only-prerecorded`.

[65] Joshua O'Connor. Rtc accessibility user requirements – call for review, 2020. Last accessed 7 June 2020, URL: `https://www.w3.org/blog/2020/03/rtc-accessibility-user-requirements-call-for-review/`.

[66] Shawn Henry. Authoring tool accessibility guidelines (atag) overview, 2015. Last accessed 25 April 2020, URL: `https://www.w3.org/WAI/standards-guidelines/atag/`.

[67] MDN Web Docs. Wai-aria basics, 2020. Last accessed 25 April 2020, URL: `https://developer.mozilla.org/en-US/docs/Learn/Accessibility/WAI-ARIA_basics`.

[68] Jonathan Lazar, Daniel Goldstein, and Anne Taylor. Chapter 4 - technical standards for accessibility. In Jonathan Lazar, Daniel Goldstein, and Anne Taylor, editors, *Ensuring Digital Accessibility Through Process and Policy*, pages 59 – 75. Morgan Kaufmann, Boston, 2015.

[69] Lindsey Kovacz. *The Bootcampers Guide to Web Accessibility*. self-published, 2020.

[70] PowerMapper. Wai-aria screen reader compatibility, 2020. Last accessed 14 November 2020, URL: `https://www.powermapper.com/tests/screen-readers/aria/`.

[71] Matt King, JaEun Jemma Ku, James Nurthen, Zoë Bijl, and Michael Cooper. Wai-aria authoring practices 1.1, 2019. Last accessed 8th October 2020, URL: `https://www.w3.org/TR/wai-aria-practices-1.1/`.

[72] Shawn Henry and Shadi Abou-Zahra. Wcag-em overview: Website accessibility conformance evaluation methodology, 2020. Last accessed 7 June 2020, URL: `https://www.w3.org/WAI/test-evaluate/conformance/wcag-em/`.

[73] Kevin White, Shadi Abou-Zahra, and Shawn Henry. Planning and managing web accessibility, 2016. Last accessed 7 June 2020, URL: `https://www.w3.org/WAI/planning-and-managing/`.

[74] Brian Wentz, Paul T Jaeger, and Jonathan Lazar. Retrofitting accessibility: The legal inequality of after-the-fact online access for persons with disabilities in the united states. *First Monday*, 16(11), Nov. 2011.

[75] Hayfa.Y. Abuaddous, Mohd Zalisham Jali, and Nurlida Basir. Web accessibility challenges. *International Journal of Advanced Computer Science and Applications*, 7(10), 2016.

[76] Jonathan Lazar, Daniel Goldstein, and Anne Taylor. Chapter 8 - evaluation methods and measurement. In Jonathan Lazar, Daniel Goldstein, and Anne Taylor, editors, *Ensuring Digital Accessibility Through Process and Policy*, pages 139 – 159. Morgan Kaufmann, Boston, 2015.

[77] Jonathan Lazar, Daniel Goldstein, and Anne Taylor. Chapter 9 - compliance monitoring policies and procurement. In Jonathan Lazar, Daniel Goldstein, and Anne Taylor, editors, *Ensuring Digital Accessibility Through Process and Policy*, pages 161 – 182. Morgan Kaufmann, Boston, 2015.

[78] Giorgio Brajnik, Yeliz Yesilada, and Simon Harper. The expertise effect on web accessibility evaluation methods. *Human-computer interaction*, 26(3):246–283, 2011.

[79] Shadi Abou-Zahra, Nicolas Steenhout, and Laura Keen. Selecting web accessibility evaluation tools, 2017. Last accessed 8 June 2020, URL: `https://www.w3.org/WAI/test-evaluate/tools/selecting/`.

[80] Marian PADURE and Costin PRIBEANU. Comparing six free accessibility evaluation tools. *Informatica economica*, 24(1/2020):15–25, 2020.

[81] Markel Vigo, Justin Brown, and Vivienne Conway. Benchmarking web accessibility evaluation tools: Measuring the harm of sole reliance on automated tests. 05 2013.

[82] WAI. Web accessibility evaluation tools list, 2016. Last accessed 7 June 2020, URL: `https://www.w3.org/WAI/ER/tools/`.

[83] Shadi Abou-Zahra, Eric Velleman, Sanne Eendebak, Roel Antonisse, and Leon Baauw. Developing an accessibility statement, 2018. Last accessed 7 June 2020, URL: `https://www.w3.org/WAI/planning/statements/`.

[84] MDN Web Docs. Web audio api best practices, 2019. Last accessed 24 April 2020, URL: `https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API/Best_practices`.

[85] Node.js. Node.js, 2020. Last accessed 29 March 2020, URL: `https://nodejs.org/en/`.

[86] Feathers.js. Feathers — a framework for real-time applications and rest apis, 2020. Last accessed 29 March 2020, URL: `https://feathersjs.com/`.

[87] The State of Javascript. The state of javascript 2019: Back end frameworks, 2020. Last accessed 29 March 2020, `https://2019.stateofjs.com/back-end/`.

[88] WebRTC. Getting started with webrtc, 2020. Last accessed 29 March 2020, https://webrtc.org/getting-started/overview.

[89] Carl Blume. History of the webrtc api [infographic], 2018. Last accessed 29 March 2020, URL: https://www.callstats.io/blog/2018/05/11/history-of-webrtc-infographic.

[90] Shirogane. Support method to elicit accessibility requirements. *Communications in Computer and Information Science*, 432:210–223, 2014.

[91] MDN Web Docs. Handling common accessibility problems, 2020. Last accessed 16 August 2020, URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/Accessibility.

[92] Muhammad Usman, Kai Petersen, Jürgen Börstler, and Pedro Santos Neto. Developing and using checklists to improve software effort estimation: A multi-case study. *Journal of Systems and Software*, 146:286 – 309, 2018.

[93] Mark Steadman. Debunking the myth: Accessibility and react, 2020. Last accessed 13 September 2020, URL: https://www.deque.com/blog/debunking-the-myth-accessibility-and-react/.

[94] Ian Sutherland. Developing components in isolation, 2019. Last accessed 13 September 2020, URL: https://create-react-app.dev/docs/developing-components-in-isolation.

[95] Brody McKee. Setting up your editor, 2020. Last accessed 13 September 2020, URL: https://create-react-app.dev/docs/setting-up-your-editor.

[96] React. Accessibility. Last accessed 13 September 2020, URL: https://reactjs.org/docs/accessibility.html.

[97] eslint-plugin-jsx a11y. eslint-plugin-jsx-a11y, 2020. Last accessed 13 September 2020, URL: `https://github.com/jsx-eslint/eslint-plugin-jsx-a11y`.

[98] WebAIM. Screen reader user survey #8 results, 2019. Last accessed 3th October 2020, URL: `https://webaim.org/projects/screenreadersurvey8/`.

[99] Shadi Abou-Zahra. Web accessibility first aid: Approaches for interim repairs, 2019. Last accessed 14 September 2020, URL: `https://www.w3.org/WAI/planning-and-managing/`.

[100] Eric Eggert and Shadi Abou-Zahra. Application menus, 2019. Last accessed 7th October 2020, URL: `https://www.w3.org/WAI/tutorials/menus/application-menus/`.

Appendix A

Sanako Connect and WCAG 2.1 Level AA

Criteria		
1.1.1 Provide text equivalents (A)	FAIL	- All images in the product have alt text (pass) - Alt text can be added to images in the exercise editor (pass) - Alt text cannot be added to uploaded images directly - Not all UI controls have text equivalents - Editor does not support adding external caption files to audio/video - Audio/video player does not support displaying captions - Audio/video player does not support multiple tracks for audio description - Adding a text alternative for audio/video is not possible in the editor, although nothing prevents adding the alternative separately as a text block

Table A.1 continued from previous page

Criteria		
1.2.1 Provide alternatives for pre-recorded audio-only and video-only content (A)	FAIL	Adding a text alternative for audio/video is not possible in the editor, although nothing prevents adding the alternative separately as a text block
1.2.2 Provide captions for web-based video (A)	FAIL	- Editor does not support adding external caption files to audio/video - Audio/video player does not support captions
1.2.3 Provide text transcript or audio description for web-based video (A)	FAIL	- Adding a text alternative for audio/video is not possible in the editor, although nothing prevents adding the alternative separately as a text block
1.2.4 Provide captions for live audio (AA)	**PASS**	Not applicable to Sanako Connect, video calls are excluded
1.2.5 Provide audio descriptions for prerecorded video (AA)	FAIL	- Audio/video player does not support multiple tracks for audio description
1.3.1 Info and relationships (A)	FAIL	- ARIA landmarks are not used to mark the app structure and regions - Labels are missing on some input elements, like in the exercise editor
1.3.2 Meaningful content sequence (A)	**PASS**	- The content is in logical order in the source code
1.3.3 Sensory characteristics (A)	**PASS**	- Instructions do not rely only on sensory cues
1.3.4 Orientation (AA)	FAIL	- Fails on small devices on both landscape and portrait
1.3.5 Identify Input Purpose (AA)	FAIL	- autocomplete fields are not used in inputs
1.4.1 Use of color (A)	**PASS**	- Color is not used as the only way to distinguish elements
1.4.2 Audio controls (A) x	FAIL	- Audio player does not have separate volume control
1.4.3 Minimum contrast (AA)	**PASS**	- Text content has a contrast of 3:1 or bigger
1.4.4 Resize text (AA)	FAIL	- Browser zoom can be used, but some components do not scale properly

Table A.1 continued from previous page

Criteria		
1.4.5 Images of text (AA)	**PASS**	- Images of text are not used, expect where allowed, like in logotypes
1.4.10 Reflow (AA)	FAIL	- App fails way before 320x256 and content is either hidden or behind scroll bars
1.4.11 Non-Text Contrast(AA)	FAIL	- Some UI components do not have the necessary contrast ratio
1.4.12 Text Spacing (AA)	FAIL	- Text will be hidden on some components when overriding app styles
1.4.13 Content on Hover or Focus (AA)	FAIL	- Use of tooltips is not consistent and does not satisfy the criteria
2.1.1 Keyboard (A)	FAIL	- Main menu items cannot be navigated with keyboard- Drop-down menus cannot be navigated with keyboard
2.1.2 No keyboard trap (A) x	FAIL	- Some modal dialogs will trap the focus
2.1.4 Character Key Shortcuts (A)	**PASS**	- There is only one keyboard shortcut, which is active only on focus
2.2.1 Timing is adjustable (A)	**PASS**	- App does not have timing, plus test situations are excluded
2.2.2 Pausing, stopping, hiding (A) x	FAIL	- Loader animations can last longer than 5 seconds and cannot be paused
2.3.1 Three flashes, or below threshold (A) x	**PASS**	- There is no flashing content
2.4.1 Bypass blocks (A)	FAIL	- Repeated blocks cannot be bypassed with links or semantic structure markup
2.4.2 Include page title (A)	**PASS**	- Page title is included
2.4.3 Logical focus order (A)	FAIL	- Modal dialogs do not prevent focusing on elements hidden by the dialog - Modal dialogs do not restore focus to trigger element when closed
2.4.4 Link purpose (in context) (A)	**PASS**	- Link texts are descriptive
2.4.5 Multiple navigation mechanisms (AA)	FAIL	- No alternative method of navigation

Table A.1 continued from previous page

Criteria		
2.4.6 Headings and labels (AA)	**PASS**	- Headings and labels are descriptive
2.4.7 Visible focus for focusable elements (AA)	FAIL	- Focus is not visible on all components
2.5.1 Pointer Gestures (A)	**PASS**	- Sliders can be used with keyboard / single-point gesture
2.5.2 Pointer Cancellation (A)	FAIL	- Down events are used in some components when tracking keypresses or clicks - Drag and drop events cannot be cancelled
2.5.3 Label in Name (A)	FAIL	- Use of labels and names is not consisted
2.5.4 Motion Actuation (A)	**PASS**	- device or user motion functionality is not used
3.1.1 Language of Page (A)	**PASS**	- Language of the page is marked with lang attribute
3.1.2 Language of Parts (AA)	FAIL	- Connect itself does not use other languages except the default one - The exercise editor does not support a way to mark other languages correctly
3.2.1 On Focus (A)	**PASS**	- Focusing on any element does not trigger a change of context
3.2.2 On Input (A)	**PASS**	- Changing settings does not trigger a change of context anywhere
3.2.3 Consistent Navigation (AA)	**PASS**	- Navigation is consistent inside the app
3.2.4 Consistent Identification (AA)	**PASS**	- Functionality or label/name/icon of UI elements is not consistent
3.3.1 Error Identification (A)	FAIL	- Errored inputs are not identified clearly consistently, and error descriptions can be vague
3.3.2 Labels or Instructions (A)	FAIL	- Not all UI components have labels or instructions
3.3.3 Error Suggestion (AA)	FAIL	- Correct/valid choices are not suggested on errors consistently
3.3.4 Error Prevention (Legal, Financial, Data) (AA)	**PASS**	- There is a confirmation when student submits their answers - There is no user data that can be modified / deleted

Table A.1 continued from previous page

Criteria		
4.1.1 Parsing (A)	**PASS**	- HTML is valid (according to React's error linter)
4.1.2 Name, Role, Value (A)	FAIL	- Use of names and roles is not consistent on all UI Components - Settings values programmatically is not possible everywhere
4.1.3 Status Messages (AA)	FAIL	- Use of status messages is not consistent - some gain focus, some do not - Status messages cannot be identified programmatically
PASS	21	
FAIL	29	

Appendix B

Requirements based on WCAG 2.1

Related criteria	ID	Description
1.1.1, 1.2.1,1.2.3	R1	Video players should support text alternatives
1.1.1, 1.2.1	R2	Audio players should support text alternatives
1.1.1	R3	All images should have alt text (decorative images an empty one)
1.2.2	R4	Audio players should support captions
1.2.5	R5	Video players should support multiple audio tracks
1.3.1	R6	Use semantic HTML elements whenever possible, like headers and labels
1.3.1	R7	Use ARIA regions wherever suitable
1.3.1	R8	Use ARIA landmarks in the application structure
1.3.2	R9	Ensure that content is in a meaningful sequence in the source code (tables, for example)
1.3.3	R10	Ensure that instructions do not rely solely on sensory cues (location, color)
1.3.4	R11	Ensure that content works on landscape and portrait orientation

Table B.1 continued from previous page

Related criteria	ID	Description
1.3.5	R12	Use autofill / autocomplete fields on input fields where possible (refer to WAI's list of 57 autocomplete fields)
1.4.1	R13	Do not rely solely on color when providing information
1.4.2	R14	Audio/video player should have volume control
1.4.3	R15	Ensure that text or images of text have a contrast ratio of at least 4.5:1 (decorations etc excluded)
1.4.4	R16	Ensure that text can be scaled up to 200%
1.4.10	R17	Ensure that single components and larger compositions are displayed without scrolling down to width320xheight256
1.4.11	R18	Ensure that UI controls and states, and graphics, have a contrast of at least 3:1
1.4.12	R19	Ensure that content is visible with the following settings: Line height 1.5 * font size, spacing after paragraph 2* font size, letter spacing 0.12 * font size, word spacing * 0.16 * font size
1.4.13	R20	Ensure that content that appers with focus or hover can be dismissed easily and is persistent (tooltips, submenus etc)
2.1.1	R21	Ensure that all content can be navigated to and controlled with a keyboard
2.1.2	R22	No keyboard trap - ensure that focus traps can be moved away from (modal windows)
2.1.4	R23	Keyboard shortcuts should be able to be turned off, remapped, or are active only on focus
2.2.2.	R24	Moving, blinking, flashing content should be able to be paused, stopped, hidden (upload loader icon!)

Table B.1 continued from previous page

Related criteria	ID	Description
2.3.1	R25	Ensure that there is no content that flashes more that three times a second
2.4.1	R26	Allow the user to skip repeated blocks (header) to reach main content, OR use semantic section markup
2.4.2	R27	Ensure that the web application has a page title
2.4.3	R28	Ensure that focus order remains sequential (modal window closing restores focus)
2.4.4	R29	Ensure that purpose of all links can be deciphered from the link text itself
2.4.5	R30	Ensure that all pages of the application can be reached in multiple ways (sitemap, for example)
2.4.6	R31	Ensure that all existing headings and labels are descriptive
2.4.7	R32	Ensure that keyboard focus is visible on all components
2.5.1	R33	Multipoint or path gestures can be performed with single point gestures
2.5.2.	R34	Ensure that drag and drop actions can be cancelled
2.5.2	R35	Ensure that all keyboard and mouse events are triggered on the up event (not down)
2.5.3	R36	Ensure that text in labels matched the programmatic name
3.1.1	R37	Ensure that there is a correct lang attribute on the html element
3.1.2	R38	Ensure that content in other languages is marked correctly
3.1.2	R39	Add editor options to mark the language of text, parts of text, textareas for the student
3.2.1	R40	Ensure that no component that receives focus will change the context (new window, dialog, etc)

Table B.1 continued from previous page

Related criteria	ID	Description
3.2.2	R41	Ensure that changing any setting does not change the context
3.2.3	R42	Ensure that navigation is consistent on all pages
3.2.4	R43	Ensure that components with the same functionality also have the same label/accessible name/text alternative
3.3.1	R44	Display detailed explanation and identify the input item in question (also programmatically) when there is an input error
3.3.2	R45	Provide labels or instructions for every input
3.3.3	R46	On input errors, suggest correct/possible input choices if possible
3.3.4	R47	Modifying/deleting user data or submitting test responses should be reversible or confirmed
4.1.1	R48	Ensure that HTML is valid / according to spec (opening/-closing tags on every element, proper nesting, no duplicate ids)
4.1.2	R49	Name/role of UI components can be programmatically determined, and state/property/value programmatically set
4.1.3	R50	Ensure that status messages can be programmatically determined

Appendix C

Categorized checklist of requirements

Note that the numbering of the requirements is not sequential there, since it is the categorized list and not the original from Appendix B.

Application

Some of the requirements are such that they make sense only when looking at the application as a whole. This category is meant for requirements that either need to be implemented only once for the application, and for requirements that refer to functionality exceeding singular web components. Because of this there is some overlap with the Single Components category.

R9 - Use ARIA landmarks to mark the application structure

R10 - Ensure that content is in a meaningful sequence in the source code (tables, for example)

R12 - Ensure that content works on landscape and portrait orientation

R17 - Ensure that text in every component / element can be scaled up to 200

R22 - Ensure that all content can be navigated to and controlled with a keyboard

R27 - Allow the user to skip repeated blocks (header) to reach main content, OR use semantic section markup

R28 - Ensure that the web application has a page title

R31 - Ensure that all pages of the application can be reached in multiple ways (sitemap, for example)

R49 - Ensure that HTML is valid / according to spec (opening/closing tags on every element, proper nesting, no duplicate ids)

Single Components

As described in Section 2.2, modern web development allows for modularity . Application interfaces are be composed of components, which in turn can be composed of other components. This allows for development on the component level.

7 Use semantic HTML elements: headers, labels

8 Use ARIA regions

10 Ensure that content is in a meaningful sequence in the source code (tables, for example)

12 Ensure that content works on landscape and portrait orientation

14 Do not rely solely on color when providing information

16 Ensure that text or images of text have a contrast ratio of at least 4.5:1 (decorations etc excluded)

17 Ensure that text in every component / element can be scaled up to 200

18 Ensure that single components and larger compositions are displayed without scrolling on 320x256 viewport size

20 Ensure that content is visible with the following settings: Line height (line spacing) to at least 1.5 times the font size; Spacing following paragraphs to at least 2 times the font size; Letter spacing (tracking) to at least 0.12 times the font size; Word spacing to at least 0.16 times the font size.

21 Ensure that content that appers with focus or hover can be dismissed easily and is persistent (tooltips, submenus etc)

22 Ensure that all content can be navigated to and controlled with a keyboard

23 No keyboard trap - ensure that focus traps can be moved away from (modal windows)

24 Keyboard shortcuts should be able to be turned off, remapped, or are active only on focus

25 Moving, blinking, flashing content should be able to be paused, stopped, hidden

26 Ensure that there is no content that flashes more that three times a second

29 Ensure that focus order remains sequential (modal window closing restores focus)

33 Ensure that keyboard focus is visible on all components

39 Ensure that content in other languages is marked correctly

41 Ensure that no component that receives focus will change the context (new window, dialog, etc)

49 Ensure that HTML is valid / according to spec (opening/closing tags on every element, proper nesting, no duplicate ids)

51 Ensure that status messages can be programmatically determined

UI Components

In this categorization, UI Component can be understood to be a subcategory of a Component. All the requirements of a Component apply here too, but because they are interactive, UI Components have more requirements.

13 Use autofill / autocomplete fields on input fields where possible (refer to WAI's list of 57 autocomplete fields)

19 Ensure that UI controls and states, and graphics, have a contrast of at least 3:1

35 Ensure that drag and drop actions can be cancelled

36 Ensure that all keyboard and mouse events are triggered on the up event (not down)

37 Ensure that text in labels matched the programmatic name

42 Ensure that changing any setting does not change the context

44 Ensure that components with the same functionality also have the same label/accessible name/text alternative

45 Display detailed explanation and identify the input item in question (also programmatically) when there is an input error

46 Provide labels or instructions for every input

47 On input errors, suggest correct/possible input choices if possible

48 Modifying/deleting user data or submitting test responses should be reversible or confirmed

49 Ensure that HTML is valid / according to spec (opening/closing tags on every element, proper nesting, no duplicate ids)

50 Name/role of UI components can be programmatically determined, and state/property/value programmatically set

51 Ensure that status messages can be programmatically determined

Audio/video Components

This category can be thought of to be a subcategory of UI Component. Any audio or video component needs to be interacted with.

1 Text alternatives for all video content should be supported

2 Ttext alternatives for all audio content should be supported

5 Audio players should support displaying captions from multiple alternatives

6 Video content should support multiple audio tracks the user can choose from

15 Every audio/video player should have volume control independent from system overall volume

Content

This category contains all requirements related to content.

11 Ensure that instructions do not rely solely on sensory cues (location, color)

30 Ensure that purpose of all links can be deciphered from the link text itself

32 Ensure that all headings and labels are descriptive