Barrierelos

Release 0.0.1

Michael Hofmann, Pascal Lehmann

Jan 12, 2024

TABLE OF CONTENTS

CHAPTER

ONE

ABSTRACT

In December 2023, a revision of regulations was announced by the Swiss Federal Council, also mandating private companies to enhance the accessibility of their digital services. Therefore, a far greater number of actors will now have to adhere to these rules. Despite existing regulations, accessibility issues persist, particularly in government services at the cantonal and municipal levels.

The project aims to address this gap by recording and scoring accessibility issues on websites, categorizing them by region and other factors, and making the data publicly available. The project also aids website maintainers in resolving these issues by providing a list of issues found in an easy-to-use format.

This thesis presents a software system, incorporating crowdsourcing, scanning for accessibility issues using the axe-core engine, and a scoring algorithm named Barrierelos-Score. The system comprises a React-based web application frontend, a Spring Boot-based backend with a RESTful web API, and a scanner implemented in TypeScript running on Node.js. Containerization and temporal decoupling ensure scalability, while a modularized architecture facilitates easy implementation of new requirements.

The project embraces an agile workflow, with automated testing and deployment, enabling continuous integration and continuous deployment. The documentation is made available online through GitLab Pages. The result is a comprehensive approach to enhance web accessibility awareness and facilitate issue resolution in Switzerland and Liechtenstein.

CHAPTER

TWO

MANAGEMENT SUMMARY

In response to the revised regulations announced by the Swiss Federal Council in December 2023, private companies will also be mandated to improve the accessibility of their digital services. Therefore, a far greater number of actors will now have to adhere to these rules. Despite existing regulations, accessibility challenges persist, particularly in government services at the cantonal and municipal levels.

To address this gap, our project focuses on identifying and scoring accessibility issues on websites, categorizing them by region, and making the data publicly available. The initiative also supports website maintainers by offering a user-friendly list of identified issues for resolution.

The thesis introduces a software system that utilizes crowdsourcing, the axe-core engine for scanning accessibility issues, and a scoring algorithm called Barrierelos-Score. The system comprises a React-based web application frontend, a Spring Boot-based backend with a RESTful web API, and a TypeScript-based scanner running on Node.js. A modularized architecture allows for the seamless implementation of new requirements.

The project follows an agile workflow with automated testing and deployment, enabling continuous integration and continuous deployment. Documentation is hosted online through GitLab Pages. The result is a comprehensive approach to raise awareness about web accessibility and streamline issue resolution in Switzerland and Liechtenstein.

CHAPTER

THREE

INTRODUCTION

This project was created as a bachelor thesis at the Eastern Switzerland University of Applied Sciences (OST) in the subject area of computer science, or more precisely in the field of software engineering. The official project title is “Crowdsourced E-Accessibility Dashboard” and was posted by Prof. Dr. Markus Stolze.

3.1 The State of Accessibility in Switzerland

According to the Swiss Federal Statistical Office (BFS), almost 18% of the population have disabilities as defined under the Disability Discrimination Act (BehiG). The Federal Council of Switzerland announced in December 2023 that they are working on a revision of these regulations also obligating private companies to make their digital services accessible. Similar legislative changes have already been adopted in the EU and will come into effect in 2025 in Germany for example. Therefore, a far greater number of actors will now have to adhere to these rules. Despite existing regulations, there are clear deficits with the accessibility of digital services in Switzerland, especially with government services at the cantonal and municipal levels. However, the extent of these deficits cannot currently be determined.

3.2 Goal of the Project

For this reason, our project aims to: record accessibility issues on websites, score these websites based on their accessibility and classify them by region and other categories. The collected data is made available to the public to raise awareness of web accessibility and help website maintainers resolve such issues.

3.3 Task

Our task for this thesis included the following main points: - Creating a web dashboard, that would provide an overview of the state of web accessibility in Switzerland. - Collecting websites and categorizing through a crowdsourcing approach, enabling users to add websites to the system and categorize them. - Automatically assessing the accessibility of websites, with a focus on websites on the three government levels in Switzerland: federal, cantonal and municipal. - Allowing moderation of the crowdsourced data, to ensure the quality of the data.

CHAPTER

FOUR

REQUIREMENTS

4.1 User Stories

As part of the requirement analysis we identified a number of user stories which we've grouped into epics. Each user story is described with a sentence in the form of "*As a <user type>, I want <goal>, so that <benefit>.*". The user stories were used to analyze the requirements from the view of a user and to discuss them with the supervisor.

Since the user stories act as functional requirements, they are updated as the project progresses.

4.1.1 User Types

A user types are fictional characters that represent the different user types that might use our application. A person can belong to multiple user types. They are used to better understand the needs of the users and to design the application to meet their needs.

We identified the following user types:

User Type	Description
User	A *user* represents a generic person using the system. This user type is used when the user type is not important. The term *registered user* is used to refer to a user who has an account on the system.
Viewer	A *viewer* represents a person who wants to learn more about the current state of accessibility of websites. They're not required to have account on the system.
Contributor	A *contributor* represents a person who wants to contribute by: adding new websites and webpages to the system, maintaining *categories*, tagging websites, initiating automated analyses or reporting unwanted behavior or content.
Moderator	A *moderator* represents a person who is responsible for the quality of the data in the system and removing inappropriate content.
Administrator	An *administrator* represents a person who is responsible for the operation of the system.

4.1.2 Epic: Basic Features

This epic contains the basic features to enable all other user stories.

ID	Description
ST-1	As a user, I want to be able to sign up for an account, so that I can use the service as a registered user.
ST-2	As a user, I want to be able to use an already existing account from another platform, so that I can start using the service.
ST-47	As a user, I want to be able to create a dedicated account on the platform, so that I don't have to use an account from another platform.
ST-43	As a user, I want to be able to delete my account on the system, so that it will be marked as such.
ST-48	As a user, I want to be able to see my profile, so that I can see what data is stored in it.
ST-49	As a user, I want to be able to change my profile, so that I can update my contact information.
ST-50	As a user, I want to be able to change my password, when using a dedicated account on the platform, so that I can change my password on a regular basis.
ST-3	As a user, I want to be able to sign in, so that I can use restricted features.
ST-4	As a user, I want to be able to sign out, so that my account is protected when I'm not using the service.
ST-6	As a user, I want the application to be responsive, so that I can use it on my mobile phone.
ST-7	As a user, I want the application to be accessible, so that I can use it despite my disability.
ST-8	As a user, I want the application to be self-explanatory, so that I can use it without any prior knowledge.
ST-9	As a user, I want the application to load quickly, so that I can use it without having to wait.
ST-41	As a user, I want the application to be in my native language, so that I can understand it better.

4.1.3 Epic: Automated Testing

This epic contains the user stories to enable automated testing of websites.

ID	Description
ST-5	As a contributor, I want to add a website, so that I can analyze its a11y.
ST-10	As a contributor, I want to initiate an automated analysis of a website, so that I can check how accessible it is with respect to WCAG guidelines.
ST-11	As a viewer, I want to inspect the results of an analysis in a report, so that I can see how the website performed and what issues were found.
ST-12	As a viewer, I want the report to include a score, so that I can compare the accessibility of different websites.
ST-13	As a contributor, I want to be able to see the results after a website has been re-analyzed, so that I can see if the website's score has changed.
ST-14	As a contributor, I want the system to handle large numbers of automated analyses, so that I can analyze many websites.
ST-66	As a contributor, I want to know the status of a scan, so that I know whether a scan succeeded or failed.

4.1.4 Epic: Statistics and Changes Over Time

This epic contains the user stories to add statistics about a single website or a group of websites and to be able to track changes over time.

ID	Description
ST-15	As a viewer, I want the system to track how a website's score has changed over time, so that it is possible to assess whether the website's score is improving or getting worse.
ST-16	As a viewer, I want the system to track how a *category*'s average score has changed over time, so that it is possible to assess if the group's score is improving or getting worse.
ST-17	As a viewer, I want to see a ranking of websites by the score, so that I can see which websites are most accessible.
ST-18	As a viewer, I want to see a ranking of regions by average score, so that I can see which regions handle accessibility best.
ST-57	As a viewer, I want to be able to search for a website, so that I can find the website im looking for without having to go through the entire list.
ST-58	As a viewer, I want to be able to see when a website was added, so I can see how long the website has been tracked.
ST-59	As a viewer, I want to be able to see when a website was last scanned, so that I know how relevant the scan still is.
ST-60	As a viewer, I want the scoring system to be transparent, so that I know roughly how the score is calculated.

4.1.5 Epic: Moderation

This epic contains the user stories to moderate the system.

ID	Description
ST-30	As an administrator, I want to appoint moderators, so that they can moderate the system.
ST-42	As an administrator, I want to dismiss moderators, so that they can no longer moderate the system.
ST-33	As a registered user, I want to report websites, so that I can report incorrect, misleading or inappropriate content.
ST-51	As a registered user, I want to report the webpages of a website, so that I can report incorrect, misleading or inappropriate content.
ST-52	As a registered user, I want to report other users, so that I can report incorrect, misleading or inappropriate content.
ST-34	As a moderator, I want to be able to review websites that have been reported, so that I can verify the claims and remove the website if necessary.
ST-53	As a moderator, I want to be able to review webpages that have been reported, so that I can verify the claims and remove the website if necessary.
ST-54	As a moderator, I want to be able to review users that have been reported, so that I can verify the claims and remove the website if necessary.
ST-55	As a registered user, I want to be able to communicate to the moderators when reporting, so that I can explain to them why I reported the website, webpage or user.
ST-56	As a registered user, I want to be able to communicate with the moderators when I was reported, so that I have the chance to justify myself to the moderator and/or the user who reported me.

4.1.6 Epic: Maintenance

This epic contains the user stories to operate and maintain the system.

ID	Description
ST-35	As an administrator, I want to maintain the system without any help from the original developers, so that I can keep the costs of the project low.
ST-36	As an administrator, I don't want to have to pay license fees, so that I can keep the costs of the project low.
ST-37	As an administrator, I want to keep resource consumption on my infrastructure to a limit, so that I can keep the costs of the project low.
ST-38	As an administrator, I want to be able to easily deploy the system, so that I can move it to a different infrastructure if necessary.
ST-39	As an administrator, I want the source code to have automated tests, so that I can add or modify features without breaking existing functionality.

4.1.7 Epic: Implement Additional Features

This epic contains the user stories for additional features that are not part of the project scope.

ID	Description
ST-40	As a viewer, I want to see where exactly on the website a11y issues were found, so that I can fix them more easily.
ST-61	As a viewer, I want to know what those issues mean, so that I can fix them more easily.
ST-62	As a viewer, I want to know the weight of webpages on the scoring, so that I know which page is most relevant to fix the issues there.
ST-63	As a user, I want to know about the privacy policy of the platform, so that I know how my data is handled.
ST-64	As a user, I want to know who is behind this platform, so that I can contact them if necessary, and to know what the legal basis is.
ST-65	As a user, I want to know what legal basis applies to this platform, so that I know the place of jurisdiction.

4.2 Non-functional Requirements

The following table lists the non-functional requirements (NFR) for this project. They were formulated with the SMART criteria in mind, meaning that the NFRs are:

- **S**pecific
- **M**easurable
- **A**greed upon
- **R**ealistic
- **T**ime-bound

In accordance with FURPS, the NFRs are categorized into the following categories:

- **F**unctionality (not relevant for NFRs)
- **U**sability

- **R**eliability
- **P**erformance
- **S**upportability

4.2.1 Usability

ID	Description	User Stories
NR-1	The application must be responsive. Meaning that the application must be usable both on mobile devices and on desktop computers. This will be verified through a usability test.	US-6
NR-2	The application must be usable by disabled people, especially people with visual impairments. Meaning that the application must be accessible and usable with a screen reader. This will be verified through an a11y test by a visually impaired person with a screen reader.	US-7
NR-3	The application must be usable without any prior knowledge. Meaning that the application must be self-explanatory. This will be verified through a usability test with a person who has no prior knowledge of the application.	US-8

4.2.2 Reliability

ID	Description	User Stories
NR-4	The application should be containerized and run in a container orchestration system. Meaning that the application should be able to recover from a failure of a single instance.	US-38
NR-5	The software should not get overloaded by a high number of automated a11y analysis. Instead the software should queue the analysis requests and process them one by one.	US-14

4.2.3 Performance

ID	Description	User Stories
NR-6	The application must be able to run on a server with 2 vCPUs, 4 GB of RAM and 30 GB of storage. This are the specifications of the server we were provided for the project.	US-37
NR-7	The page load time of all webpages must be below 2 seconds 95% of the time, when accessed from Switzerland.	US-9

4.2.4 Supportability

ID	Description	User Stories
NR-8	The application must be maintainable without consulting the original developers. Meaning that the development setup und deployment process must be documented. Additionally, the code and all produced artifacts must comply with the quality standards as per the quality_assurance section.	US-35
NR-9	The application may not cause any running costs beyond those for the server the application is running on. Meaning that the application may not use any paid services.	US-36
NR-10	The source code must have 60% test coverage through automated unit tests.	US-39
NR-11	The application must support i18n. Meaning that the application could be translated into another language just by adding another language file with translations.	US-41

CHAPTER

FIVE

DESIGN

5.1 Scoring

The primary task of this thesis was to create a web dashboard, that would provide an overview of the state of web accessibility in Switzerland. In order to capture the whole picture, we would need to be able to assess large numbers of websites. Performing a manual accessibility assessment takes a considerable amount of time and the testing protocol has to be adjusted based on the functionality present on the website. Additionally, both manual and automated accessibility assessments are relatively short-lived. Websites often change many times a year or even get completely refurbished after some years. So the assessment has to be repeated regularly to stay up to date. This fact, combined with the large number of websites to assess meant it would not be feasible to employ manual testing to capture the state of accessibility in Switzerland. As such, we were tasked to create a system that could automatically determine the accessibility of websites. Additionally, it should be able to aggregate the results, so that different cantons could be compared to each other for example.

5.1.1 Automated accessibility assessments

During our research for this thesis, we looked at many existing projects that assess the accessibility of websites. Many of these employed a hybrid between manual and automated testing, some were based entirely on manual tests and only a few were entirely relying on automated tests. One such example is the well-known *The WebAIM Million* project, which we will discuss in more detail later in this chapter. Through our research, as well as talking to experts in the field of web accessibility, we discovered that it is largely impossible to exactly determine the accessibility of a website through automated testing. Some issues can currently only be detected by humans, especially those concerning interactive elements. As such, our system would not be able to provide an exact assessment of the accessibility of a website but serve approximation or indicator of the accessibility of a website. Since we want to capture the state of accessibility in Switzerland, we concluded that we need a quantification of the accessibility of a website. This would allow us to compare the accessibility of different websites and aggregate the results of different categories and groups.

5.1.2 Existing solutions

Since we are not experts in the field of web accessibility and we had limited experience from lectures and exercises from courses at OST, we wanted to get familiar with existing solutions and approaches to automatically assess the accessibility of websites.

There are a lot of companies offering automated accessibility testing services. We looked at many examples, like Accesible Web, acessiBe or AccessibilityChecker.org. These services are usually offered as a Software as a Service (SaaS) and start with a free tier, with additional features available through paid subscription to the service. Unfortunately, none of these services offer any information on how they perform their tests, what criteria they consider or how they calculate their metrics.

There are some more sophisticated solutions, like the Wave web accessibility evaluation tool from WebAIM. This tool is available as a browser extension for Chrome and Firefox and also, as a web API and a standalone engine. We evaluated the API as a candidate for use in our system. More information on this can be found in the *Automated a11y analysis tool* section. This tool is also used by *The WebAIM Million* project. They do however not provide any scoring and only offers a list of issues found on the website and doesn't report on the severity of the issues.

We also looked at LightHouse, which is a tool developed by Google and directly integrated into the developer tools of the Chrome browser. Due to this reason, it is widely used and does also provide a scoring system. It is based on the axe-core accessibility engine by Deque Systems. Lighthouse's scoring mechanism is based on the Axe engine's impact assessment, which rates issues based on their severity. It does however not take into account how often an issue occurs on a website.

The WebAIM Million is possibly the most well-known, large-scale automated accessibility assessment project. They publish a yearly report on the accessibility of the top 1 million most visited websites on the internet. They use the Wave engine to perform their tests but don't provide a score but a ranking of the websites. They summarize how their ranking works but don't provide exact details. We evaluated their scoring system based on 41 websites that were tested in the Swiss Accessibility Study 2020 by Access for All. We also compared the ranking to the issues found by the Wave engine. However we found some inconsistencies, in that websites that ranked high in the WebAIM Million but had many issues reported by the Wave engine. We read through their whole publication and could not find any mention of how they filter issues or how they calculate their ranking. Another issue with their approach is that they only test the homepage of a website. While the homepage is certainly important, they are missing out on many issues that might be present on other pages of the website.

5.1.3 Conclusion

Based on our findings, there is no existing solution that would provide a scoring system that would be suitable for our use case. Many of the existing solutions are not transparent in how they perform their tests and how they calculate their scores or rankings. This appears to be an issue with many automated accessibility assessment tools.[1]

Therefore, we decided to derive our own scoring system. Also implementing an accessibility engine was out of scope for this thesis, so we decided to instead use one of the more sophisticated existing engines, that would scan the websites for accessibility issues. We would then use the issues found by the engine and process the results. Since we had no good references to compare our scoring system with, we wanted to rely on an established solution as much as possible. The axe-core engine is developed by Deque Systems, which is a company specializing in providing accessibility services. It's integrated into many existing solutions, like the Lighthouse tool from Google. Additionally, it uses a large number of rules to detect issues and categorizes them based on their severity. Their rules are largely based on the Web Content Accessibility Guidelines (WCAG) which is also the basis of most regulations concerning web accessibility in Switzerland and Europe in general. This means that the issues found are relevant to our use case and can also help organizations to improve the accessibility of their services according to the new regulations described in the *Introduction*. By processing the results, we can also determine how often an issue occurs on a website and use this information to weigh the issues. For this reason, we decided to use the axe-core engine as the basis for our scoring system.

[1] Marco Manca, Vanessa Palumbo, Fabio Paternò, and Carmen Santoro. 2023. The Transparency of Automatic Web Accessibility Evaluation Tools: Design Criteria, State of the Art, and User Perception. ACM Trans. Access. Comput. 16, 1, Article 3 (March 2023), 36 pages. https://doi.org/10.1145/3556979

5.1.4 Inspiration

For our scoring to be meaningful, we wanted to base it on existing research. This is why we took inspiration from the studies conducted by Access for All. They are a Swiss foundation that advocates for technology access and use by people with disabilities. They also act as an independent certification authority for the accessibility of websites and apps. They release a study every 3-5 years, where they assess the accessibility of a sample of relevant Swiss websites and apps. They are mainly focused on manual testing but have sometimes augmented their assessments with automated tests. Their Swiss Accessibility Study 2016[2] describes how they quantify the accessibility of websites, which can be found in chapter 7.1.1.3.1.

They describe four aspects they considered:

1. How often is a test criterion violated?
2. To which degree is it being violated?
3. Is it only a minor issue or does it make an entire process inaccessible?
4. Which kinds of disabilities are affected to what extent?

We are able to address aspect 1 since axe-core does not only provide a list of issues, but also a list of tested elements that passed the tests. We can therefore determine how many elements passed the tests and how many failed.

Aspect 2 is not directly addressed by the axe-core engine. The output for an issue also includes metadata about the occurrence, but this is not standardized and not always present. Considering a contrast issue, for example, we could determine the contrast ratio of the text and the background color. So we would be able to tell if the contrast is only slightly off or if it is completely unreadable. However since this information is not always present, we would implicitly weight certain issues lower than others, thus skewing our results. Additionally, we would have to intervene with specific rules used by axe-core, which means that a future update of the engine could break our implementation. We therefore decided to not consider this aspect in our scoring system.

Aspect 3 is well addressed by the axe-core engine. The engine categorizes issues based on their severity. It uses four categories: minor, moderate, serious and critical. We associate an impact value with each category, which we use to weigh the issues. We tested several values based on 41 websites that were tested in the Swiss Accessibility Study 2020 by Access for All and compared the results to the score in the study. We finally settled on the following values:

- minor: 2
- moderate: 4
- serious: 8
- critical: 16

This means that to compensate for a serious issue, for example, 8 elements would have to pass the same rule. These values are based on a limited sample of websites. Since we're relying on crowdsourcing the websites to test, the data collection will only really start once our thesis is completed and the system is deployed. So we integrated these values into our scoring system in such a manner that they can be easily adjusted in the future.

Aspect 4 is unfortunately not addressed by the axe-core engine. The engine does not provide any information on the type of disability that is affected by an issue.

[2] Anton Bolfing, Bernhard Heinser, Gianfranco Giudice, Petra Ritter. 2016. Schweizer Accessibility-Studie 2016. 152 pages. https://access-for-all.ch/wp-content/uploads/2022/11/SchweizerAccessibilityStudie2016.pdf

5.1.5 Methodology

The biggest challenge in creating our scoring system was that there are few existing solutions to compare it to. This means that we are not able to empirically determine how well our scoring system works. To remedy this as much as possible, we based our approach on existing research like the studies conducted by Access for All and we used the established axe-core engine as the basis for our scoring system. This section further describes the aspects that we take into account to make our scoring system as meaningful as possible.

As mentioned in the *Introduction*, we employ a crowdsourcing approach to collect the websites to test. We allow users to record multiple webpages for a website. We emphasize to users, that they should record pages that are relevant for the website and avoid adding many pages with the same structure. This gives us the advantage that we can test multiple pages of a website and that these webpages are not just a random sample, but are chosen based on their relevance.

In this chapter so far we have only talked about the axe-core output in a simplified manner for better understanding. Now we'll go into more detail on how we process the output of the axe-core engine to determine the score of a website. Axe-core doesn't have a concept of a website with multiple webpages. It only takes a single URL as input and then tests the webpage at that URL. So we execute the axe-core engine for each webpage that was recorded for a website.

The output of a single webpage is four lists of test results: passed, incomplete, inapplicable and violated. Each of these lists contains several rules that were tested on the website. Each rule contains a list of tested elements on the website. Since one element can violate a rule, but another element can pass the same rule, the same rule can be present in multiple lists. So we have to process the lists and link the rules together. We only consider results that are present in the passed and violated lists. Since those contain the rules with the elements that passed or violated respectively.

This is because the *incomplete-list* contains those rules for which axe-core could not determine if they are violated or not. This is due to the fact that axe-core was designed not to produce any false positives. So if it is not sure if a rule is violated, it will put it in the *incomplete-list*. We decided to ignore these potential issues to prevent us from scoring a website worse than it is. This also means that we might miss some issues, but we consider this an acceptable trade-off. Or to put it jokingly, in the words of William Blackstone: "It is better that ten guilty [websites] escape than that one innocent suffer".[3] The *inapplicable-list* contains those rules for which axe-core could not find any elements to test on the website.

So to extract the number of times an issue occurs on a webpage, we have to count the number of tested elements in the *violated-list*. The size of the different webpages can also be very different. To take this into account, we weigh the webpages based on the number of elements that axe-core tested on the webpage. We can determine this by linking up the in the *passed-* and *violated-lists* and then summing up the number of elements in each rule. Additionally, we also take into account the axe-core impact severity of the issues, as was described in the previous section.

5.1.6 Barrierelos-Score

The result of this scoring system is a score between 0 and 100. We call this score the *Barrierelos-Score*. We first intended to call it the *Accessibility-Score*, but we decided against it since we want our score to be interpreted as an exact measure of the accessibility of a website. We want to emphasize that our score is only an approximation of the accessibility of a website. As such an individual score should not be interpreted as an exact measure. But it is a useful indicator and can also be used to capture the state of accessibility in Switzerland. Also allowing us for example to compare the aggregated results of different cantons.

The condensed technical documentation on the implementation of our scoring system can be found in the *Barrierelos-Score Calculation* section.

[3] Blackstone, Sir. Commentaries on the Laws of England in Four Books, vol. 2. J. B. Lippincott. 1753. https://oll.libertyfund.org/title/sharswood-commentaries-on-the-laws-of-england-in-four-books-vol-2

5.2 User Interface

We created an extensive wireframe that was used to create a visual representation of the user interface. The wireframe includes all possible functionality that we were able to identify. Not all of it was implemented in the end, due to the time constraints of the project.

Figma was used to design the wireframe. Figma is a collaborative web application for interface design, therefore the wireframe is only available online under the following link: Wireframe on Figma

5.3 Database

5.3.1 Logical

Logical structure of the data with entities and their properties and relationships with each other:

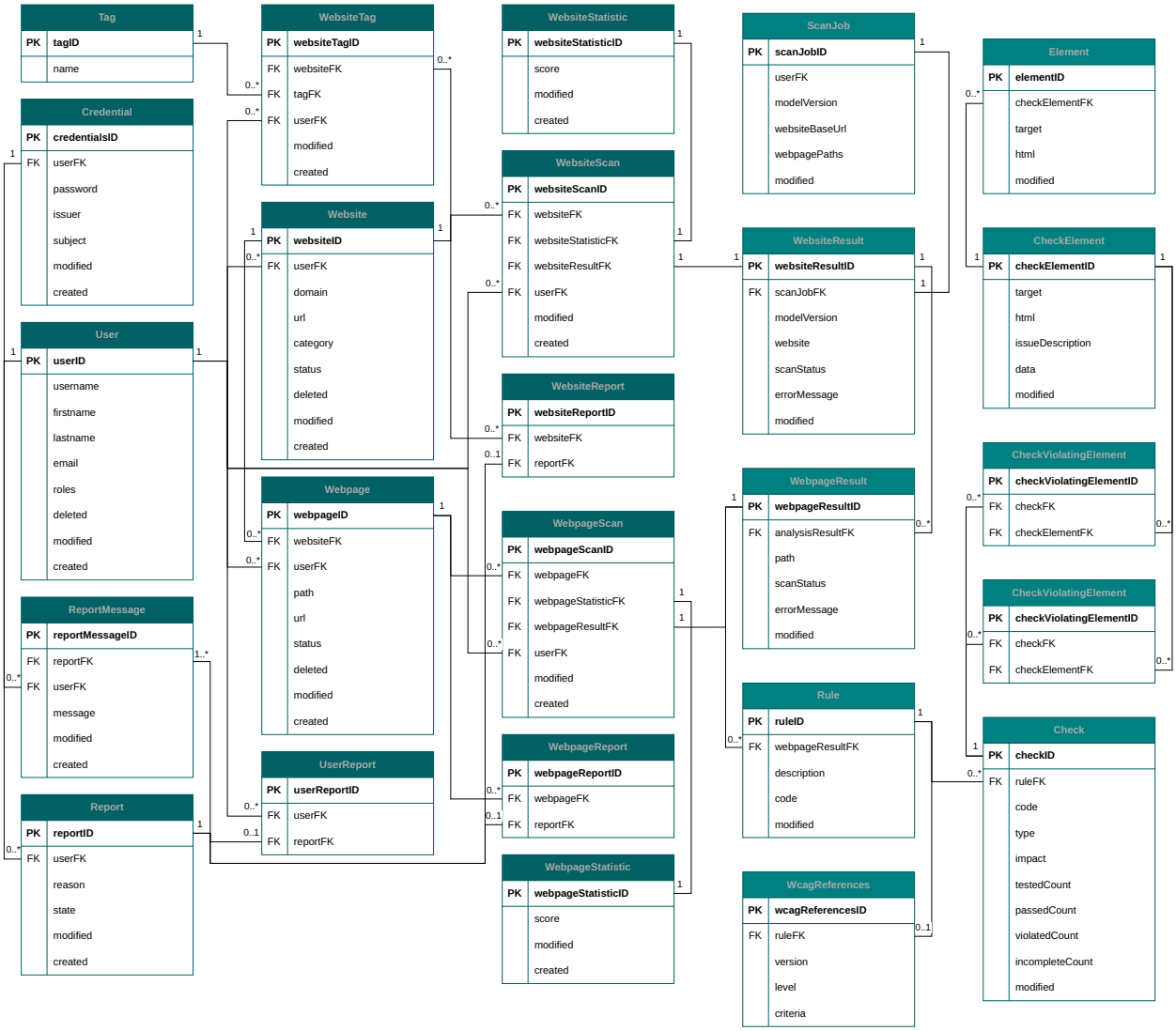

Fig. 1: Database Logical

5.3.2 Physical

Physical structure of the data with entities, their relationships with each other, their properties, and descriptions of what type these properties are:

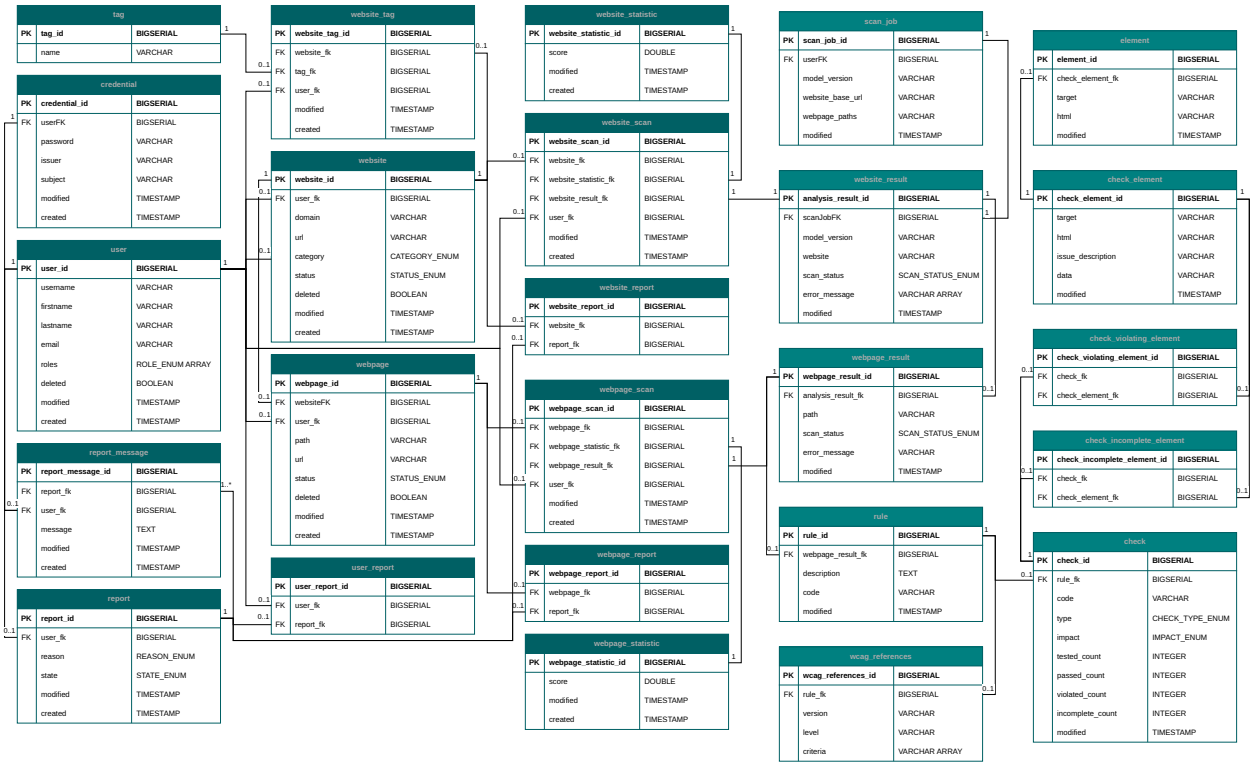

Fig. 2: Database Physical

CHAPTER

SIX

ARCHITECTURE

This section focuses on the architecture and concepts used to achieve the goals of this project. Besides addressing the core concepts on a high level, it also goes into detail on the implementation of the core functionality.

What is not included in this section is the reasoning of why certain decisions were made. This is instead covered in the *decision* section.

6.1 C4 Model

The overall architecture of the system is documented using the C4 model. The C4 model was created as a way to help software development teams describe and communicate software architecture, by creating maps of the code, at various levels of detail (like zooming in and out of an area of interest).

The C4 model is an "abstraction-first" approach to diagramming software architecture. The software system is made up of one or more **containers** (applications, data stores), each of which contains one or more **components** (library, module), which in turn are implemented by one or more **code** elements (classes, interfaces, objects, functions). And **people** (actors, roles, personas) who may use the software system.

- **Person**: Represents one of the human users of the software system. People correspond to the roles of the system.
- **Container**: Represents an application or a data store. Containers are separately runnable/deployable units that need to be running in order for the overall software system to work.
- **Component**: Represents a grouping of related functionality encapsulated behind a well-defined interface. Components are not separately deployable units.

6.1.1 Context

The starting point of the architecture documentation. It allows to step back and see the big picture. It is a zoomed out view showing a big picture of the system landscape. The focus is on people and software systems rather than technologies, protocols and other low-level details.

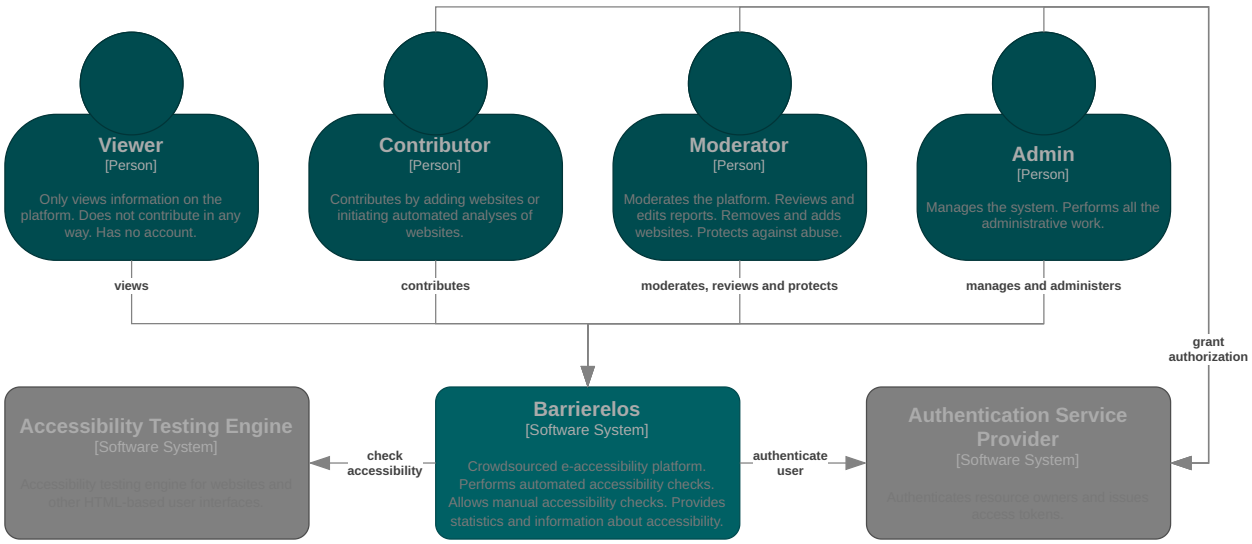

Fig. 3: C4 Context Diagram

6.1.2 Container

How the system fits in to the overall IT environment. It's a zoom-in to the system boundary. Shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another.

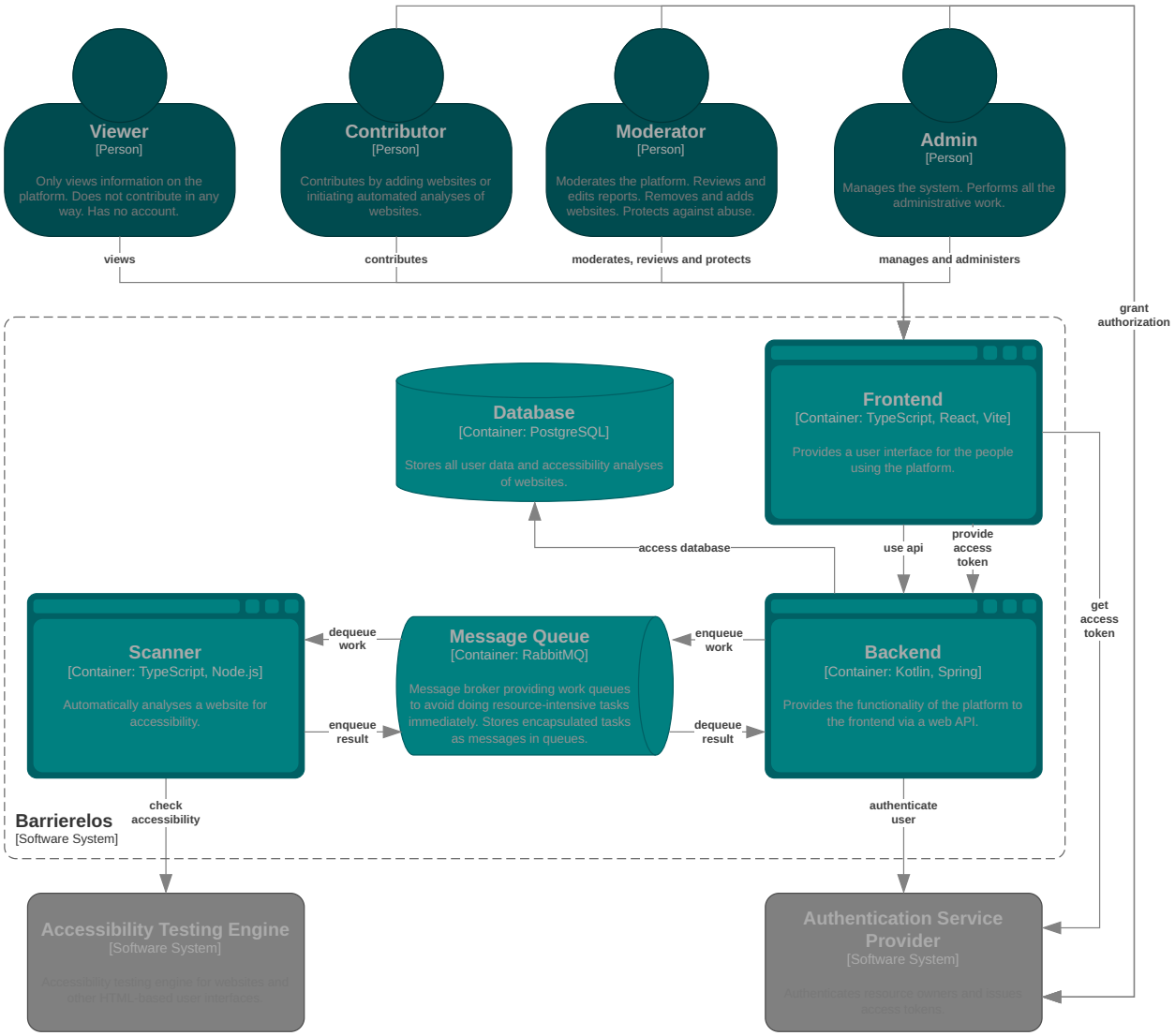

Fig. 4: C4 Container Diagram

6.2 Pipeline

The documentation for the pipeline can be found in the `README.md` file in the deployment.

CHAPTER

SEVEN

DECISIONS

This section contains the most important decisions made during the course of the project.

7.1 Automated a11y analysis tool

In order to perform automated a11y analysis of websites we rely on a third party tool to do the heavy lifting. Developing our own tool would be a huge undertaking, that would warrant its own project. So we decided to use an existing tool and integrate it with our system. For this purpose we chose to evaluate three different tools:

- Lighthouse
- WAVE API
- Axe-core

7.1.1 Lighthouse

Lighthouse is an open-source, automated tool for improving the quality of web pages by Google. We chose to evaluate it because it's a popular tool and we were already familiar with it from courses at OST. Unlike the other two tools we evaluated, Lighthouse is not a dedicated accessibility tool, but rather a general purpose tool for improving the quality of web pages. But it's also possible to only run the accessibility audit. It is available as a browser extension, a Node module, and a CLI tool which are released under the Apache 2.0 license. Since it is available as a standalone tool there is no additional cost associated with using it.

Lighthouse's a11y engine is based on the open-source Axe-core. The resulting report can be retrieved as JSON or HTML and contains a list of accessibility issues, their severity, and a description of how to fix them. It even contains screenshots of the areas of the page that are affected by the issue.

7.1.2 WAVE API

WAVE® is a suite of evaluation tools that helps authors make their web content more accessible to individuals with disabilities by WebAIM. We chose to evaluate it because that's the engine that was used by the WebAIM Million project. They release an annual report on the accessibility of the top 1 million websites so they have a lot of experience with analyzing websites efficiently. The engine is available as a browser extension, an API or a standalone testing engine. The API uses a pricing model that is based on a per-page fee, that varies depending on the level of detail of the report.

- For 0.04 USD per page you get a report that contains only the number of violations per issue category.
- For 0.08 USD per page you get a report that contains everything from above as well as the number of violations per guideline.

- For 0.12 USD per page you get a report that contains everything from above as well as a list of all violating elements in the page per guideline.

So to analyze a whole website containing dozens or even hundreds of pages would run up a huge bill quite quickly. The standalone testing engine can only be licensed annually and the license covering our use case would cost 12'000 USD a year. The ruleset used by the engine is explicitly based on WCAG and each rule contains a link to the corresponding WCAG guideline. The resulting report can be retrieved as JSON or XML.

7.1.3 Axe-core

Axe-core is an open-source accessibility testing engine by Deque Systems. We chose to evaluate it because it is quite popular and it's the engine that is used by Lighthouse. So we figured we could cut out the middleman and use it directly. It is available as a JavaScript library which is released under the Mozilla Public License 2.0. It is intended to be integrated into a testing framework (e.g. Mocha) or a library to control a headless browser (like Puppeteer). Since it is available as a standalone tool there is no additional cost associated with using it.

The engine contains multiple different rule sets that are mostly based on WCAG. There are not quite as many WCAG rules as in the WAVE API, but it also contains rules that are not based on WCAG. Additionally, it's possible to implement custom rules. The resulting report can be retrieved as JSON and contains a list of accessibility issues, their severity, and a description of how to fix them. It's also possible to retrieve a list of all violating elements in the page per guideline.

7.1.4 Conclusion

All three tools are capable of covering our use case and they are licensed in a way that would allow us to use them. But there are some differences between them that we have to consider. Since Lighthouse a11y audit simply uses Axe-core under the hood, there would have had to be a very compelling reason to use it instead of Axe-core. Besides the ability to generate screenshots of the affected areas of the page, we didn't find any reason to use Lighthouse. And said feature could also be implemented using Axe-core in combination with a tool like Puppeteer. Additionally, the violations in the report are not linked to the corresponding WCAG guidelines. So we decided to not use Lighthouse and were therefore left with the WAVE API and Axe-core.

The WAVE API is a very powerful tool, that quality-wise returned the best results. But since OST would like to use the application after the project is finished, we wanted to avoid any recurring costs. And since the WAVE API is licensed on a per-page basis, it would have been very expensive and we would have had to strictly limit the number of pages that can be analyzed. Which would have reduced the usefulness of the application quite considerably. And since axe-core can cover our use case and is free to use, we decided to use it for our project. The option to implement custom rules is another advantage of axe-core, since it allows us to implement rules that are specific to our needs.

7.2 Barrierelos-Score Calculation

This section will explain how the Barrierelos-Score is calculated.

7.2.1 Webpage Scoring

Each automated analysis is run over multiple webpages of a website. So the first step is to calculate the score for a single webpage. Since the size of the webpages and the number of contained elements can vary, we need to account for that. Hence, we also calculate a weight for each webpage.

The results of Axe-core are grouped according to their outcome into the following lists:

- passes: These results indicate what elements passed the rules
- violations: These results indicate what elements failed the rules
- inapplicable: These results indicate which rules did not run because no matching content was found on the page. For example, with no video, those rules won't run.
- incomplete: These results were aborted and require further testing. This can happen either because of technical restrictions to what the rule can test or because a JavaScript error occurred.

For our calculations, we're only interested in the passes and violations because these are the results of the rules that were applicable to the webpage in question. Each rule in these two lists also contains a count of how many elements were tested for that rule.

The weight is calculated by summing up the number of tested elements for each rule in the *passes-* and *violations-list*.

$$\text{webpage\_weight} = \sum_{i=1}^{n}(\text{passes\_tested\_elements}_i) + \sum_{i=1}^{m}(\text{violations\_tested\_elements}_i)$$

Now to calculate the score, Axe-core provides another useful value for each rule: the impact. The impact describes the severity of a rule violation. So each rule has an impact belonging to one of four categories, which we've each mapped to a numeral value for our calculations:

- minor: 1
- moderate: 2
- serious: 4
- critical: 8

To calculate the score for a webpage, we sum up the impact values of all rules in the *passes-list*. Then we divide that by the sum of all impact values in the *passes-* and *violations-list*. Finally, we multiply that by 100 to get a score between 0 and 100.

$$\text{webpage\_score} = \frac{\sum_{i=1}^{n}(\text{passes\_impact\_value}_i)}{\sum_{i=1}^{n}(\text{passes\_impact\_value}_i) + \sum_{i=1}^{m}(\text{violations\_impact\_value}_i)} \cdot 100$$

So each webpage is assigned a score and a weight.

7.2.2 Website Scoring

Now that we have the score and weight for each webpage, we can calculate the score for the whole website. This is done by calculating the weighted arithmetic mean of the webpage scores:

$$\text{website\_score} = \frac{\sum_{i=1}^{n}(\text{webpage\_score}_i \cdot \text{webpage\_weight}_i)}{\sum_{i=1}^{n}\text{webpage\_weight}_i}$$

7.2.3 Implementation

The processing of the axe-core results as well as the calculation of the Barrierelos-Score is implemented in the backend. However, the calculations are executed in the database, using an SQL Query. The reason for this is that the axe-core results contain a list of all tested elements. For large websites, i.e. websites with a lot of associated webpages, this list can become very large. If we were to process the results in the backend, in the form of Kotlin objects, this process would take a very long time and would require a lot of memory. Many concurrent scans could slow down the system significantly. Additionally, when rescans of existing webpages are performed, we don't have to load the data into the backend first to do the calculations.

7.3 Website Categories

When adding a new website, the user has to indicate which category the website belongs to. This gives us the ability to enable statistics on a per category basis. For example, we can compare how the different cantons are doing in terms of accessibility. It also allows users to search for websites by filtering on the category.

We distinguish between government and private websites and further divide them into specific categories. All categories are mutually exclusive, so a website can only belong to one category.

7.3.1 Government Websites

We define a government website as a website that is fully owned and controlled by a government entity, i.e. the federal government, a canton or a commune. This excludes websites controlled by other entities, even if those entities are partially or fully owned by the government, e.g. public universities. Note that despite these categories being hierarchical, they are still mutually exclusive. So a communal website can't also be a cantonal website because we categorize the websites based on the entity that owns the website. There might however be statistics, that for example show the aggregated data of all communal websites of a canton.

Category	Description
Federal Government (DE: Bund)	Websites of the federal government.
Canton (DE: Kanton)	Websites of one of the 26 Swiss cantons. This category consists of 26 subcategories, one for each canton.
Commune (DE: Gemeinde)	Websites of the communes. This category consists of many subcategories, one for each commune.

7.3.2 Private Websites

We define a private website as a website that is not in direct control of a government entity. The controlling entity might however be partially or fully owned by the government. Unlike the categories in the *Government Websites* section, the categories of private websites don't have a hierarchical structure and don't consist of subcategories. Hence they can easily be extended in the future.

These categories are based on those used in the Accessibility-Studie 2016. They are intended to capture websites that either receive public funding or have a public mandate. News portals that are subsidized for example, are required to provide their services in a way that is accessible to all people.

Note that some websites could be assigned more than one category. But to keep things simple, we decided to only allow one category per website. So if a website could be assigned to multiple categories, we choose the one that we think is the most appropriate. For example, *SRG SSR* is both a *news portal* and a *federally affiliated company*. But categorizing it as as a *news portal* is more specific and more fitting because it's main purpose is to provide news.

Category	Description
Federally Affiliated Company (DE: Bundesnaher Betrieb)	Websites of companies that are partially or fully owned by the federal government. Examples of such companies are *SBB*, *Swisscom* or *Swiss Postal Service*.
University	Websites of universities. Examples of such websites are *ETH Zürich*, *Universität Zürich* or *Ostschweizer Fachhochschule (OST)*.
News Portal	Websites that provide news. Examples of such websites are *SRG SSR*, *NZZ* or *Blick*.
Online Shop	Websites that sell products online. Examples of such websites are *Digitec*, *Interdiscount* or *Migros*.
Others	Websites that don't fit into any of the other categories. Examples of such websites are those of private Swiss companies.

CHAPTER

EIGHT

IMPLEMENTATION

8.1 Code Metrics

Besides using automated tools to check the code quality, we also used SonarQube to manually check the code quality and to get code metrics. We performed this manual check every time we reached a code milestone and refactored the code accordingly. In this section, we will showcase some quality indicators and code metrics of the code base at the end of the project. We decided in agreement with the examiner that we would only analyze the code metrics down to the file-level.

8.1.1 Backend Code Metrics

Overview

Table 1: Overview

Metric	Value
Number of classes (implementation)	203
Number of classes (tests)	323
LOC (implementation)	3'507
LOC (tests)	8'824
Test coverage	71%
Number of test cases	870

Classes per Package (CPP)

Here we will look at the number of classes of the 10 packages with the highest number. All the packages below are listed relative to the backend root package `ch.barrierelos.backend` to make package name easier to read.

Table 2: The 10 packages with the most CPP

Package	LOC
entity	17
model	16
service	15
converter	15
controller	14
exception	14
entity.scanner	10
model.scanner	8
message	8
message.scanner	8

According to an article from DZone about Rule 30 from the book Refactoring in Large Software Projects, the number of CPP should not exceed 30. Admittedly this project would not be considered a large software project as described in the book, but our biggest packages only slightly exceed the midpoint of this rule. So at this point, it would not make sense to further split up the packages in our opinion. This decision might have to be revisited when we continue working on this project in the future.

Lines of Code (LOC)

Here we will look at the LOC of the 10 files with the highest values. All the files below are listed relative to the backend root source directory `src/main/kotlin/ch/barrierelos/backend/` to make file names easier to read.

Table 3: The 10 files with the most LOC

File	LOC
service/WebsiteService.kt	276
configuration/SecurityConfiguration.kt	164
repository/Repository.kt	135
security/Security.kt	134
service/WebpageService.kt	134
service/UserService.kt	123
service/StatisticService.kt	94
service/CredentialService.kt	85
converter/scanner/RuleConverter.kt	75
converter/WebsiteConverter.kt	74

All files other than `service/WebsiteService.kt` are in a range that is completely acceptable. The `service/WebsiteService.kt` file contains 17 methods, which is a bit too much, though 6 of them are short validation methods. It contains the logic to perform CRUD operations on websites and a lot of validation logic for this purpose. The biggest method has 21 LOC, which is reasonable. Now, 276 LOC is bit too large but still below the threshold of all test tools listed in this Stack Exchange answer. Because of the more detailed metrics above and because the logic in this file belongs together, we decided to not split up this file.

Cyclomatic Complexity (CC)

Here we will look at the CC of the 10 files with the highest values. All the files below are listed relative to the backend root source directory `src/main/kotlin/ch/barrierelos/backend/` to make file names easier to read.

Table 4: The 10 files with the highest cyclomatic complexity

File	CC
service/WebsiteService.kt	51
security/Security.kt	45
repository/Repository.kt	34
service/WebpageService.kt	29
service/UserService.kt	21
service/CredentialService.kt	20
service/ReportService.kt	12
security/AuthenticationDetails.kt	10
service/TagService.kt	9
util/Common.kt	8

The first four files with the highest values stand out, since they have the highest CC by far. Let's look at them in more detail:

The `service/WebsiteService.kt` file with a CC of 51, consists of 276 LOC and 17 methods, though 6 of them are just short validation methods. It contains the logic to perform CRUD operations on websites and a lot of validation logic for this purpose. Even if we divide 51 by 11, ignoring the small validation methods, we get an acceptable average CC of 4.6 per method and this value is of course a little inflated because of the methods we ignored. Since the logic in this file belongs together, we wanted to retain its cohesion and decided to not split up this file.

The `security/Security.kt` file with a CC of 45, consists of 134 LOC and 27 methods. It contains the logic that the services use to perform fine-grained access control. Now 27 methods sounds like a lot, however all but two of them contain only a single LOC with a conditional to check for some type of access. This also explains the high CC value and means that it would not make sense to splitt up this file just for a better CC score, since it easily understandable as it is.

The `repository/Repository.kt` file with a CC of 34, consists of 135 LOC and 19 methods. It is the base interface for all our Spring Repositories and includes pagination, sorting and filtering functionality. 12 of these are only marker method definitions which make Spring generate the required code. The 7 remaining methods implement the pagination, sorting and filtering functionality based on the code generated by Spring. They contain many conditionals to differentiate between different parameters and use the appropriate Spring classes in response. This is necessary for performance reasons so that Spring / JPA is able to generate efficient SQL queries. The 7 methods have an average CC of 6.4, which is acceptable. The code is easily understandable and mostly consists of long if-else-if chains with single LOC bodies. For this reason, we decided to not split up this file.

The `service/WebpageService.kt` file with a CC of 29, consists of 134 LOC and 11 methods, though 5 of them are just short validation methods. It contains the logic to perform CRUD operations on webpages and a lot of validation logic for this purpose. Even if we divide 29 by 6, ignoring the small validation methods, we get an acceptable average CC of 4.8 per method and this value is of course a little inflated because of the methods we ignored. Since the logic in this file belongs together, we wanted to retain its cohesion and decided to not split up this file.

8.1.2 Frontend Code Metrics

Overview

Table 5: Overview

Metric	Value
LOC	4'723

Note: Since the frontend is using React JSX / TSX the LOC also include the HTML markup.

Lines of Code (LOC)

Here we will look at the LOC of the 10 files with the highest values.

Table 6: The 10 files with the most LOC

File	LOC
src/pages/ProfilePage.tsx	449
src/pages/SignupPage.tsx	236
src/pages/PrivacyPolicyPage.tsx	200
src/pages/website/OverviewTab.tsx	186
src/components/ReportComponent.tsx	183
src/pages/reports/ReportsPage.tsx	174
src/App.tsx	166
src/components/GoogleLoginComponent.tsx	140
src/pages/LoginPage.tsx	133
src/components/nav_bar/NavBar.tsx	129

All files other than `src/pages/ProfilePage.tsx` are in a range that is completely acceptable. The `src/pages/ProfilePage.tsx` file contains 9 functions and 10 overloads for those functions. It contains the profile page and the validation logic for the user details and the password change. 215 of the 449 LOC are just HTML markup. Leaving as at 234 LOC for the actual logic, which is acceptable. Since the validation logic is unique to this component, we decided to leave it in this file, even though it is quite large.

Cyclomatic Complexity (CC)

Here we will look at the CC of the 10 files with the highest values.

Table 7: The 10 files with the highest cyclomatic complexity

File	CC
src/pages/ProfilePage.tsx	67
src/components/ReportComponent.tsx	47
src/pages/reports/ReportsPage.tsx	42
src/components/nav_bar/NavBar.tsx	32
src/services/AuthenticationService.ts	29
src/pages/SignupPage.tsx	28
src/pages/website/OverviewTab.tsx	27
src/components/GoogleLoginComponent.tsx	24
src/util/formatter.tsx	20
src/pages/website/IssuesTab.tsx	19

The first three files with the highest values stand out, since they have the highest CC by far. Let's look at them in more detail:

The `src/pages/ProfilePage.tsx` file with a CC of 67, consists of 449 LOC and 9 functions and 10 overloads for those functions. It contains the profile page and the validation logic for the user details and the password change. 215 of the 449 LOC are just HTML markup. Leaving as at 234 LOC for the actual logic. The reason the CC is quite high is because of the validation logic and the 10 overloads for the functions. The logic itself is not very complex and it is only needed for this component, so we decided to leave it as is for the time being, but if more features were to be added, this would become a candidate for refactoring.

The `src/components/ReportComponent.tsx` file with a CC of 47, consists of 183 LOC. It contains the logic to report a website, webpage or a user. The reason the CC is so high is that it contains the logic for all three types of reports and conditions. Keeping this logic together allows us to prevent some duplication of code. When reading through the code it is still easily understandable and the LOC are not too high either. Thus, we decided to leave it as.

The `src/pages/reports/ReportsPage.tsx` file with a CC of 42, consists of 174 LOC and 10 functions. It contains the reports page that displays the website, webpage and user reports a user is involved in as well as the logic to retrieve this data from the backend. When we divide the CC of 42 by the 10 functions, we get an average CC of 4.2 per function, which is acceptable. Reading through the code, it is easily understandable and therefore, we decided to leave it as is.

8.1.3 Scanner Code Metrics

Overview

Table 8: Overview

Metric	Value
LOC (implementation)	474
LOC (tests)	395
Test coverage	88%
Number of test cases	26

Lines of Code (LOC)

Here we will look at the LOC of the 10 files with the highest values.

Table 9: The 10 files with the most LOC

File	LOC
src/formatter.ts	188
src/__tests__/formatter.test.ts	142
src/scanner.ts	86
src/model.ts	82
src/__tests__/scanner.test.ts	53
src/__mocks__/axeResults.ts	46
src/logger.ts	44
src/rabbitmq.ts	42
src/__tests__/rabbitmq.test.ts	34
src/__tests__/util.test.ts	33

These are all completely acceptable values so no action was taken.

Cyclomatic Complexity (CC)

Here we will look at the CC of the 10 files with the highest values.

Table 10: The 10 files with the highest cyclomatic complexity

File	CC
src/formatter.ts	41
src/__tests__/formatter.test.ts	25
src/scanner.ts	13
src/logger.ts	6
src/__tests__/scanner.test.ts	6
src/__tests__/rabbitmq.test.ts	5
src/__tests__/util.test.ts	5
src/util.ts	5
src/rabbitmq.ts	3
src/__mocks__/axeResults.ts	2

The first three files with the highest values stand out, since they have the highest CC by far. Let's look at them in more detail:

- The `src/formatter.ts` file with a CC of 41, consists of 188 LOC and 10 functions. It contains the logic to transform the axe-core output into our internal representation. It has an average CC of 4.1 per function and is thus not very problematic. Also, splitting it up would reduce the cohesion so we decided against it.
- The `src/formatter.ts` file with a CC of 25, consists of 142 LOC and 19 functions. It contains the tests for the `src/formatter.ts` file. It has an average CC of 1.3 per function making it non-problematic.
- The `src/scanner.ts` file with a CC of 13, consists of 53 LOC and 3 functions. It contains the logic to interact with axe-core and puppeteer. It has an average CC of 4.3 per function. The reason for this is that it includes error-recovery and retry logic which needs conditionals. We decided against splitting it up since it would make the code too fragmented and keeping it as-is makes it easier to follow the logic flow.

8.2 Dependencies

8.2.1 3rd Party Runtime Libraries

This section lists all the 3rd party runtime libraries that are used by the Barrierelos application. This excludes development libraries.

Scanner

All runtime libraries that are used by the scanner are npm packages. They are recorded in the `package.json` file at the root of the scanner repository.

Table 11: Scanner Runtime Libraries

Name	License
@axe-core/puppeteer	MPL-2.0
@cloudamqp/amqp-client	Apache-2.0
axe-core	MPL-2.0
dotenv	BSD-2-Clause
puppeteer	Apache-2.0
winston	MIT

Frontend

All runtime libraries that are used by the frontend are npm packages. They are recorded in the `package.json` file at the root of the frontend repository.

Table 12: Scanner Runtime Libraries

Name	License
@emotion/react	MIT
@emotion/styled	MIT
@fontsource/roboto	Apache-2.0
@mui/icons-material	MIT
@mui/lab	MIT
@mui/material	MIT
@mui/x-data-grid	MIT
@react-oauth/google	MIT
i18next	MIT
i18next-browser-languagedetector	MIT
i18next-http-backend	MIT
i18next-resources-for-ts	MIT
jwt-decode	MIT
react	MIT
react-dom	MIT
react-helmet-async	Apache-2.0
react-i18next	MIT
react-router-dom	MIT
swr	MIT

Backend

All runtime libraries that are used by the backend are Maven artifacts managed by Gradle. They are recorded in the `build.gradle.kts` file at the root of the backend repository.

Table 13: Scanner Runtime Libraries

Name	License
commons-validator:commons-validator	Apache 2.0
com.fasterxml.jackson.module:jackson-module-kotlin	Apache 2.0
me.paulschwarz:spring-dotenv	MIT
org.flywaydb:flyway-core	Apache 2.0
org.jetbrains.kotlin:kotlin-reflect	Apache 2.0
org.jetbrains.kotlinx:kotlinx-datetime	Apache 2.0
org.jetbrains.kotlinx:kotlinx-serialization-json	Apache 2.0
org.springdoc:springdoc-openapi-starter-webmvc-ui	Apache 2.0
org.springframework.amqp:spring-rabbit	Apache 2.0
org.springframework.boot:spring-boot-starter-amqp	Apache 2.0
org.springframework.boot:spring-boot-starter-data-jpa	Apache 2.0
org.springframework.boot:spring-boot-starter-security	Apache 2.0
org.springframework.boot:spring-boot-starter-web	Apache 2.0
org.springframework.security:spring-security-oauth2-jose	Apache 2.0
org.springframework.security:spring-security-oauth2-resource-server	Apache 2.0
org.yaml:snakeyaml	Apache 2.0

8.2.2 Tools for Continued Operation

This section lists all tools that are needed to continue the operation and maintenance of the Barrierelos application.

More detailed information about the usage of these tools can be found in the READMEs of the corresponding repositories.

Table 14: Tools for Continued Operation

Name	Description
GitLab	All code, scripts and documentation is stored in the corresponding GitLab repository.
GitLab CI/CD	The GitLab CI/CD pipeline is used to build and test the application. It is also used to build and publish the Docker images to the Gitlab container registry. The deployment to the server is also handled by the CI/CD pipeline.
Docker	All application components are containerized with Docker.
JetBrains IDEs	The JetBrains IDEs were used to develop the application. We used PyCharm for the documentation, IntelliJ for the backend and WebStorm for the scanner and frontend. We highly recommend continuing to use these IDEs for development, since our workflow is very well integrated with them and there are many predefined running configurations.
Make	The Make utility is used for various scripts and commands.

8.3 Repositories

The Barrierelos project consists of the three primary components which each have their own Git repository. The documentation and the deployment configuration are also stored in their own repositories.

- scanner: The Typescript / Node.js application that runs the Axe-core a11y scan jobs submitted by the backend via RabbitMQ and provides a REST API for the frontend.
- backend: The Kotlin / Spring backend API used by the frontend, which also submits Axe-core a11y scan jobs to the scanner via RabbitMQ.
- frontend: The Typescript / React frontend which uses client-side-rendering (CSR) and retries its data from the backend API.
- deployment: The deployment configuration and instructions to run specific components locally in Docker.
- documentation: The ReStructuredText / Sphinx documentation for the Barrierelos project which is published to GitLab Pages.

For more details about the individual components, look at the `README.md` files in the respective repositories.

The Barrierelos project also includes the following additional repositories:

- tools/scripts: Contains some helper scripts to add example websites via the backend API and to scrape information we needed while developing the project.
- tools/bruno: Contains the Bruno configuration. Bruno is an open-source API tool similar to Postman.

CHAPTER

NINE

TESTING

9.1 Requirements Testing

All functional and non-functional requirements were tested in detail. The results of these tests is recorded in the following test protocols. This includes a textual test feedback and whether the test failed or passed.

9.1.1 Test Functional Requirements

Each functional requirement was tested individually and represents a test case. Each test case refers to the corresponding user story in which the functional requirement was defined.

Basic Features

Functional Requirement	Feedback	Result
ST-1	Sign up for new users is implemented. After an account is created, the services for registered users become available.	passed
ST-2	Users can use their existing Google account to login to the system, using the OAuth standard.	passed
ST-47	Users can create a new account on the platform with username and password.	passed
ST-43	Users can delete their account in the profile page.	passed
ST-48	Users can access their profile after login and view all their profile data.	passed
ST-49	Users can update their contact information in the profile page.	passed
ST-50	Users can change their password in the profile page.	passed
ST-3	Users can login to their account with username and password, or with their Google account via OAuth.	passed
ST-4	Users can logout using the logout button. After logging out, restricted features become unavailable.	passed
ST-6	The website is responsive and usable on both desktop computers and mobile devices.	passed
ST-7	Manual tests have ensured that the entire website is keyboard navigable and used appropriate labels and ARIA properties whenever necessary. Also, we used the MUI library, that already covers many accessibility features.	passed
ST-8	The entire website is designed to be self-explanatory using simple language, no prior knowledge is necessary. There is also an FAQ with the most frequent questions users may have.	passed
ST-9	The website is a single-page application with client-side rendering. In this test, all the network requests took 232 ms to load and 233 ms to load the DOM content. After the initial loading, further page loads were much quicker. Loading Websites for example took 46 ms. Which is sufficiently fast to fulfill this requirement.	passed
ST-41	The website supports both German and English. Users can switch between those two languages. As agreed with the supervisor, these are the languages to be supported.	partially passed

Automated Testing

Functional Requirement	Feedback	Result
ST-5	After signing up, users can add a website. Once the website is added, an automated scan is started. As soon as that scan is completed, the results become available for all viewers and contributors to see.	passed
ST-10	Users can initiate a scan of a website by adding that website. Once the website is added, an automated scan is started. As soon as that scan is completed, the results become available for all viewers and contributors to see. These results include accessibility issues with respect to WCAG guideline.	passed
ST-11	For each added website there is a report with all the results of the scan, including what issues where found and how the website performed otherwise.	passed
ST-12	Each report includes a Barrierelos-Score to compare the accessibility of different websites.	passed
ST-13	When a website is re-scanned, the results become available. The scan infos also provide information about when the last scan was performed.	passed
ST-14	Temporal decoupling is used for the resource-intensive scanning via a message broker. This way the system can handle a large number of requests simultaneously.	passed
ST-66	The status of a scan is displayed for each website. The scan infos also provide information about the scan status of each webpage.	passed

Statistics and Changes Over Time

Functional Requirement	Feedback	Result
ST-15	The system keeps previous scan results enabling it to track how a website's score has changed over time. As agreed with the supervisor, a statistic to visualize this change over time has not been implemented.	partially passed
ST-16	The system keeps previous scan results enabling it to track how a *category*'s average score has changed over time. As agreed with the supervisor, a statistic to visualize this change over time has not been implemented.	partially passed
ST-17	All the websites can be sorted by their Barrierelos-Score, resulting in a ranking of the websites, so that the users can see which websites are most accessible.	passed
ST-18	Websites can be grouped by region, an average Barrierelos-Score is provided for each group. The groups are already sorted by their Barrierelos-Score, resulting in a ranking of the regions, so that the users can see which regions handle accessibility best.	passed
ST-57	The website provides the ability to search for websites by domain name and then open the details page for that website.	passed
ST-58	The website details for each website provide information about when the website was added.	passed
ST-59	The scan infos for each website provide information about when the last scan was performed.	passed
ST-60	The Barrierelos-Score and how it is calculated is explained in the FAQ.	passed

Moderation

Functional Requirement	Feedback	Result
ST-30	There is a moderator role on the system. Moderators have more rights than contributors that allow them to moderate the system.	passed
ST-42	Moderators can be marked as deleted, blocking them from the system.	passed
ST-33	Users can report websites, give a reason (incorrect, misleading or inappropriate) and provide a written explanation.	passed
ST-51	Users can report webpages, give a reason (incorrect, misleading or inappropriate) and provide a written explanation.	passed
ST-52	Users can report other users, give a reason (incorrect, misleading or inappropriate) and provide a written explanation.	passed
ST-34	Moderators can review websites and remove them if necessary. But this is only possible via the Web API, no graphical user interface has been implemented for this task.	partially passed
ST-53	Moderators can review webpages and remove them if necessary. But this is only possible via the Web API, no graphical user interface has been implemented for this task.	partially passed
ST-54	Moderators can review users and remove them if necessary. But this is only possible via the Web API, no graphical user interface has been implemented for this task.	partially passed
ST-55	Users can provide a written explanation with every report.	passed
ST-56	Users can exchange text messages within a report, so that reporter, reported and moderators can communicate with each other prior to any action by the moderator.	passed

Maintenance

Functional Requirement	Feedback	Result
ST-35	All the code is available on GitLab. The entire deployment pipeline is automated vai GitLab CI/CD. And all the projects include extensive documentation in their `README.md` files.	passed
ST-36	All the software used in this project is free software. No licences have been purchased, no non-free licences are required. More information about this can be found in the *Dependencies* section.	passed
ST-37	All the software runs in docker containers, as such they can be scaled individually. The server we were provided by OST for development also only had 1 core and 2 GB of RAM and the system ran sufficiently fast.	passed
ST-38	All the software runs in docker containers. Images are created for the frontend, backend, scanner, database, RabbitMQ and the documentation so that they can easily be moved to a different infrastructure.	passed
ST-39	Large suites of automated unit and integration tests were written to improve quality and maintainability of the code. These tests run everytime code gets pushed to GitLab. Only for the frontend, did we employ manual testing after each sprint. More information about the test coverage can be found in the *Code Metrics* section.	partially passed

Implement Additional Features

Functional Requirement	Feedback	Result
ST-40	Issues are presented per webpage.	passed
ST-61	For each issue, a link is provided to a detailed explanation what this issue entails.	passed
ST-62	A weight is shown for each webpage next to the scoring.	passed
ST-63	A privacy policy is provided for the users.	passed
ST-64	Contact information is provided as part of the legal notice.	passed
ST-65	The place of jurisdiction, legal form, exclusion of liability and copyright information is provided in the legal notice.	passed

9.1.2 Test Non-functional Requirements

Usability

Non-Functional Requirement	Feedback	Result
NR-1	The website is responsive and usable on both desktop computers and mobile devices.	passed
NR-2	Manual tests have shown that the entire website is keyboard navigable and uses labels and ARIA properties whenever necessary. Also, we used the MUI library, that already covers many accessibility issues.	passed
NR-3	The entire website is designed to be self-explanatory using simple language, no prior knowledge is necessary. There is also an FAQ the most frequent questions users may have.	passed

Reliability

Non-Functional Requirement	Feedback	Result
NR-4	All the software runs in docker containers. The frontend, backend, scanner, database and RabbitMQ automatically restart on failure.	passed
NR-5	Temporal decoupling is used for the resource-intensive scanning via a message broker. This way the system can handle a large number of requests simultaneously.	passed

Performance

Non-Functional Requirement	Feedback	Result
NR-6	The entire system is able to run on a such a system.	passed
NR-7	The website is a single-page application with client-side rendering. In this test, all the Network requests took 232 ms to load and 233 ms to load the DOM content. After the initial loading, further page loads were much quicker. Loading Websites for example took 46 ms. Which is sufficiently fast to fulfill this requirement.	passed

Supportability

Non-Functional Requirement	Feedback	Result
NR-8	All the code is available on GitLab. The entire deployment pipeline is automated vai GitLab CI/CD. And all the projects include extensive documentation in their `README.md` files.	passed
NR-9	All the software used in this project is free software. No licences have been purchased, no non-free licences are required.	passed
NR-10	The targeted test coverage of 60% has surpassed by the scanner and the backend. More information about the test coverage can be found in the *Code Metrics* section. In the frontend manual testing was used at the end of every sprint.	partially passed
NR-11	The website supports both German and English. Users can switch between those two languages. As agreed with the supervisor, these are the languages to be supported.	partially passed

9.2 Usability Testing

To test the usability of the application, we have created a set of tasks that the participant should be able to complete. For this purpose, we have created a *Test Script* for us to perform the test in a consistent manner.

Additionally, we have created a *Task Description* for the participant to guide him through the tasks independently so that we can observe the participant's actions and reactions and so we're not giving away any hints or influencing the participant's actions. We purposely did use the terms used on the website in the task description as to not give implicit hints. One participant was asked to navigate the website on his smartphone, so that we could also test the responsiveness of the website.

The results of the tests are documented in the *Test Report* this includes details about the participants, the findings from the tests, the feedback from the participants and the changes we implemented based on the findings and the feedback.

9.2.1 Test Script

Introduction

Tell the participant the following information:

- **Purpose of the test**: This test aims to evaluate the usability of the "Barrierelos" system, in particular how a user that has never used the system before can operate the Website and find the information he is looking for.
- **Data recording**: During the test, usability issues are recorded manually. There is no audio or video recording. The observed problems are presented in an anonymized and aggregated form in the bachelor thesis report.

Pre-Test Questionnaire

Ask the participant for the following details about him:

- **Age**
- **Experience with web accessibility:** Describe your experience with web accessibility Have you already checked or designed websites for accessibility?
- **Technical affinity:** How would you describe your technical affinity or more specifically your experience with using websites?

Role description for the tasks

Explain to the participant the he will perform the task in the role described below, since this is one of the main target groups of the Barrierelos website and it allowed us to create a more realistic scenario.

- **Role:** You are the webmaster of the Exlibris website and you live in Rümlang.
- **Background:** A colleague has informed you that the Exlibris website has achieved a less than favorable rating on the Barrierelos website. Your task is now find more about this rating and the website in general.

Tasks

At this point, tell the participant that you have prepared a list of tasks for him to perform. Explain that he should read the tasks out load and to also think out loud while performing them. Mention that he can ask questions at any time but that would like to ask him to try to solve the tasks on his own first, since we are interested whether a user who has never used the Barrierelos website before can operate it without any help.

Ask if he has any questions before starting the tasks, then hand him the *Task Description*.

Post-Test Questionnaire

1. **Satisfaction with the website:** How satisfied are you with the user-friendliness of the Barrierelos website?
2. **Comprehensibility of the information:** How easy was it for you to understand the information on the website?
3. **Suggestions for improvement:** Do you have any suggestions on how the usability of the Barrierelos website could be improved?
4. **Experience with the tasks:** How did you find the implementation of the tasks in terms of comprehensibility and difficulty?

Conclusion of the test

- Thank the participant for taking part in the usability test.
- Explain that the data collected will be used to improve the usability of the Barrierelos website.

9.2.2 Task Description

1. The language of the website will be set according to your browser settings. How could you change to another language?
2. Search for the Exlibris website and take a look at the evaluation.
   - What is the Barrierelos-Score of the website?
   - In which area are there the most problems?
   - Which webpage has the greatest influence on the rating?
   - Where can you find a description of specific problems that were found on that webpage.
   - How do you find out exactly what the Barrierelos-Score measures?
3. You want to get an overview of state of accessibility in Switzerland. Look for a list of all evaluated websites.
   - How many websites have been evaluated?
   - Which website has the worst score?
   - Which canton has the best score?
4. You're interested in how the website of the municipality of Rümlang, where you live, is rated. Try and find out. The website's URL is https://www.ruemlang.ch.
   - Has the website already been evaluated?
   - How could you get the website evaluation even if it hasn't been evaluated previously?
   - Why are there no evaluation results for this website (yet)?
5. How can you contact the Barrierelos team?
6. How can you find out how the data that you enter on the website is used?

9.2.3 Test Report

This is the test report for the usability test that was conducted with two participants on the 6th of January 2024.

Participant Details

We refer to all participants as he / him since we feel that the gender of the participants is not relevant for the test and this way don't have to disclose that information unnecessarily.

Participant 1 is 26 years old and stated that he has no prior experience with accessibility. He has however heard about the topic and is interested in learning more about it. Concerning his technical affinity, he stated that he works in a semi-technical role and is very familiar with computers and the usage of websites.

Participant 2 is 51 years old and stated that he has no prior experience with accessibility. The the topic was only known to him from news articles about SBB making train stations more accessible. Concerning his technical affinity, he stated that he works in a non-technical role and is not very familiar with computers and the usage of websites.

Findings

Both participants were able to understand the purpose of the usability test and how to go about it. They were both able to change the language of the website and to navigate to the evaluation page of the Exlibris website, via the search function on the homepage without any issues. This participant was asked to perform the test on his smartphone, which was an iPhone 14, so that we could also test the responsiveness of the website.

Participant 1 was able to answer all the questions in task 2 correctly. He was confused however, why the webpages have different weights assigned to them. He was able to tell which area there are the most issues in, but asked himself what WCAG principles are. When looking for how the Barrierefreie Website is evaluated, he looking for the information on the evaluation page for quite a while, before he figured out that he needed to go to the FAQ page for this. This participant performed the test on a Laptop with a 15.6" screen.

Participant 2 found the Barrierelos-Score of the Exlibris website, but wasn't sure of it, because the word "Barrierelos-Score" was not present where the score was displayed. He was able to answer the other questions except for how the Barrierelos-Score is calculated. After giving him a hint that he should look at the FAQ page, he was able to find the information. He stated however, that he found the explanation quite complicated.

On task 3, both participants were able to answer the questions correctly. Participant 2 didn't notice that the list could be ordered and clicked through all the pages to get to the website with the lowest score.

On task 4, both participants figured out that website hadn't been evaluated yet.

Participant 1 was recognized that he needed to add a new website to get its evaluation. We was confused at first when he was taken directly to the login page, but when he clicked on the button to add a website again and was taken to the login page again, he realized that he probably needed to log in / sign up first. He opted to Login with Google. The process of adding the website went smoothly for him. He also notices the chip with the status "Initial Scan Pending" and therefore knew that the evaluation was still in progress. When he later compared the two websites, he was switching back and forth between the language settings and noticed that the entries in the navigation bar were not translated.

Participant 2 didn't make the connection, that he needed to add a new website to get its evaluation. When the search didn't turn anything up and he couldn't locate the website in the list, we gave him a hint that he needed to add the website first. He also didn't know why button to add a website took him to the login page. After some attempts, we gave him the hint that he needed to log in / sign up first. He opted to create an account with email and password instead of logging in with Google, since he had no Google Account. The sign up process went smoothly for him. He was able to add the website and also noticed the chip with the status "Initial Scan Pending".

Both participants were able to find the legal notice (impressum) with the information on how to contact the website owner in task 5. Participant 2 noted that to his knowledge, the term "impressum" is German and not English.

And on task 6, participant 1 was able to find the privacy policy, while participant 2 only stumbled upon it by accident. He noted the term "Privacy Policy" meant nothing to him and he was looking for "data… something".

Feedback

1. **Satisfaction with the website:** How satisfied are you with the user-friendliness of the Barrierelos website?
    - Participant 1 was satisfied with the overall usability and mentioned that he thought it looked quite nice and is relatively easy to navigate because there are not that many options.
    - Participant 2 stated that he was a little overwhelmed by the amount of information on the website and that there were too many terms he didn't understand.
2. **Comprehensibility of the information:** How easy was it for you to understand the information on the website?
    - Both participants stated that information was visually presented in a way that was easy to understand, but that some explanations of terms and concepts were missing.
3. **Suggestions for improvement:** Do you have any suggestions on how the usability of the Barrierelos website could be improved?

    - Participant 1 suggested adding explanations about WCAG principles and webpage weights. He also suggested adding some sort of hint that you have to log in / sign up before being able to add a website.
    - Participant 2 also suggested adding explanations for all uncommon terms and concepts used on the website. He also mentioned it would be nice to be able to add websites without having to log in / sign up first.

4. **Experience with the tasks:** How did you find the implementation of the tasks in terms of comprehensibility and difficulty?

- Participant 1 stated that the tasks were easy to understand and that he didn't have any problems with them.
- Participant 2 stated that he would have liked the tasks to be more specific, but other than that he deemed them okay.

Resulting Changes

- We extended the introductory text on the homepage to better explain the purpose of the website.
- We fixed the translation of the navigation bar entries.
- We added a tooltip to the Barrierelos-Score on the evaluation page, to label it.
- We added a link to the relevant FAQ page next to the Barrierelos-Score.
- We added a link to the relevant FAQ page under the Issues by WCAG principles.
- We added a question mark tooltip to the webpage weight column explaining what the weight means and how it is derived.
- We split up the explanation of the Barrierelos-Score into two separate questions, one explaining what the score means in simpler terms and one explaining how the score is calculated which goes into more detail.
- We added a hint to the add website button, that you need to log in / sign up first.
- We added a link to add a website to the empty search results text.
- We changed the term "impressum" to "legal notice" in the footer for the English version of the website.

CHAPTER

TEN

CONCLUSION

The result of this thesis is a software system to automatically determine the accessibility of websites and a web dashboard to highlight these results. Crowdsourcing is employed to collect and categorize websites. The system was developed from scratch but uses the axe-core accessibility engine to search for accessibility issues. The work on this thesis included software engineering, requirements engineering, software architecture, database modeling, wireframing, UX design and the development of a scoring algorithm. The score quantifies the accessibility of websites (from 0 to 100) and was given the name Barrierelos-Score. It is based on the number of accessibility issues found on the website and the severity of these issues.

The frontend is a web application providing an overview of the state of web accessibility in Switzerland and Liechtenstein, that also allows collaboration and moderation. It is a React single- page application with client-side rendering written in TypeScript. The backend implements the business logic that commissions website scans, assesses the results and calculates the score. It is a Spring Boot application written in Kotlin that provides a RESTful web API with OAuth 2.0 offered for authorization. A PostgreSQL database and JPA is used to manage persistence. The scanner receives jobs from the backend via a RabbitMQ message broker, to temporally decouple the resource-intensive scanning tasks. It searches for accessibility issues and returns the findings to the backend via RabbitMQ. The scanner is implemented in TypeScript and runs on Node.js.

Thanks to the containerization and temporal decoupling of resource intensive tasks, the system is easily scalable. The highly modularized architecture is designed so that new requirements can be implemented with little effort. Large suites of automated unit and integration tests were written to improve quality and maintainability of the code. An agile workflow was fully embraced, with an automated pipeline for building, testing and deployment with GitLab CI/CD, and Flyway for automated database migration, enabling both continuous integration and continuous deployment. Writing the documentation with RST and Sphinx allowed making it available online as a web-documentation through GitLab Pages.

CHAPTER

ELEVEN

GLOSSARY

a11y
: Common abbreviation for accessibility as in, “a”, then 11 characters, and then “y”. Pronounced “a-eleven-y”.

website
: A website is a collection of web pages and related content that is identified by a common domain name.

webpage
: A webpage is a single page of a website.

category
: A category to differentiate different types of websites, e.g. all websites from a particular canton. For more details, have a look at the *Website Categories* section.

RI-n
: The ID of a **RI**sk, where *n* denotes the sequence number.

ST-n
: The ID of a User **St**ory, where *n* denotes the sequence number.

FR-n
: The ID of a **F**unctional **R**equirement, where *n* denotes the sequence number.

NR-n
: The ID of a **N**on-functional **R**equirement, where *n* denotes the sequence number.

LOC
: The **L**ines **O**f **C**ode metric, which is a measure of the size of a computer program, i.e. the number of lines of text in the program’s source code. Excluding empty lines and comments.

CC
: The **C**yclomatic **C**omplexity metric, which is a quantitative measure of the number of linearly independent paths through a program’s source code.

CPP
: The **C**lasses **p**er **P**ackage metric, which is a measure of the number of classes a package contains. Classes in a subpackage are not counted.

INDEX