Page Inspector - Web Page Analyser

Francisco Filipe and António Jorge Gouveia(✉)

School of Science and Technology, University of Trás-os-Montes and Alto Douro, Vila Real, Portugal
jgouveia@utad.pt

Abstract. Web Accessibility refers to apps, websites, and digital tools designed to be accessible by all users, including those with disabilities. The web is a source of public information, education, employment, government, commerce, health, entertainment, and many others, and every person, regardless of ability, has the right to access it. Imagine missing out on thousands of new customers because your competitor has an easier-to-use, WCAG-compliant website [1]. The design and development consist of an open-source web platform, *PageInspector*, which allows anyone to analyze a website. It was built using JavaServer and Apache Tomcat. The platform provides a set of reports, spelling, accessibility, cookies, technologies, images, subdomains, Google Lighthouse and hyperlinks. In addition to these reports, it allows the user to view their website in real time, desktop and mobile versions, as well as the source-code. All reports are available through a dedicated API, so you also can integrate with your platforms.

Keywords: web · accessibility · sustainability · compliant · systematic literature review · websites · web page design · Web content accessibility guidelines (WCAG)

1 Introduction

Have you checked to see if your institution is creating websites and apps that follow the Web Content Accessibility Guidelines (WCAG)? If not, millions of people won't be able to use them. Since September 2018, all British Higher Education institutions have been legally required to upgrade their websites, online courses, and network resources to comply with the current Web Accessibility standards [1]. According to [2], digital services must meet at least Level AA of the Web Content Accessibility Guidelines (WCAG 2.1) to meet government accessibility requirements. Considering that higher education institutions often have thousands of pages of information to consider, this is a massively daunting task. The design and development consist of an open-source web platform, *PageInspector*, which allows anyone to analyze a website. It was built using JavaServer and Apache Tomcat. The platform provides a set of reports, spelling, accessibility, cookies, technologies, images, subdomains, Google Lighthouse and hyperlinks. In addition to these reports, it allows the user to view their website in real time, desktop and mobile versions, as well as the source-code. All reports are available through a dedicated API, so you can integrate them with your platforms.

A. Cunha et al. (Eds.): MobiHealth 2022, LNICST 484, pp. 109–117, 2023.
https://doi.org/10.1007/978-3-031-32029-3_11

The main objective behind the PageInspector is to allow anyone to analyse their webpages for free. This platform provides a set of detailed reports to help anyone to make their websites more accessible and more appealing to search engines. The platform provides a set of reports, spelling, accessibility, cookies, technologies, images, subdomains, Google Lighthouse and hyperlinks. In addition to these reports, it allows the user to view their website in real time, desktop and mobile versions, as well as the source-code. All reports are available through a dedicated API, so you can integrate them with your platforms. This paper is divided in four sections. Section 2 explains the though process behind the concept and development, Sect. 3 will address results and at last, Sect. 4 will address conclusions and future work.

2 Development

This section aims to present an exemplary modelling, through requirements gathering, use cases and activity diagram.

2.1 Requirements Gathering

Requirements gathering is closely linked to the quality of the produced software [3]. This subsection is a list with the specification of everything that must be implemented. The requirements can be divided into two categories, functional and non-functional requirements. Functional are related to the services that the system should offer. Non-functional are related to specific system characteristics or restrictions.

Functional requirements:

1. The *PageInspector* system should allow viewing of the intended website in real time;
2. The *PageInspector* system should allow selecting the view mode (“desktop” or “mobile”);
3. The *PageInspector* system should allow consulting the website’s source code;
4. The *PageInspector* system should allow for a spelling analysis;
5. The *PageInspector* system should allow for an accessibility analysis;
6. The *PageInspector* system should allow for a cookies analysis;
7. The *PageInspector* system should allow for a technologies analysis;
8. The *PageInspector* system should allow an images analysis;
9. The *PageInspector* system should allow a subdomains analysis;
10. The *PageInspector* system should allow for a Google Lighthouse analysis;
11. The *PageInspector* system should allow a hyperlinks analysis;
12. The *PageInspector* system should allow viewing of spelling errors in the page context;
13. The *PageInspector* system should allow for the identification of the more common spelling errors;
14. The *PageInspector* system should allow for the identification of the less common spelling errors;
15. The *PageInspector* system should allow viewing the number of occurrences of each misspelling.

16. The *PageInspector* system must allow CRUD (Create, Read, Update and Delete) operations of the user's dictionary;
17. The *PageInspector* system should allow viewing in the context of the page the accessibility errors;
18. The *PageInspector* system should allow viewing the accessibility error in the context of the source code;
19. The *PageInspector* system should allow informing the guideline of the error of accessibility;
20. The *PageInspector* system should allow informing the name and domain associated with a cookie;
21. The *PageInspector* system should allow informing which category of technologies is most used.

Non-functional requirements:

1. The *PageInspector* system must provide access to reports through from an API;
2. The *PageInspector* system must give the possibility to use cache when generating a report;
3. The *PageInspector* system must use an SSL certificate to ensure client-server security.

2.2 Use Cases

One of the ways of identifying and documenting the requirements previously enumerated is the use-case modelling. So, actors and use cases were identified [4]. To try and identify the actors that interact with the system, the authors tried to answer the following questions:

- Who uses the system? What roles does it play in the interaction?
- What other systems interact with the system?
- Who provides information to the system?
- Are there external events that affect the system?
- Who provides the data? Who uses the information?

Since the application developed is a fully autonomous and maintenance-free web page inspection tool, the only actor identified was the ***User***.

Several use case diagrams have been developed to support all requirements. For space reasons, they are not presented in this paper.

2.3 Activity Diagram

The activity diagram presented in Fig. 1, refers to the base functioning of the platform, *PageInspector*. Activity diagrams allow you to model the flow of actions that are part of a larger activity. They are commonly used to specify use cases and class operations. It is very important that the diagram contains the minimum information necessary to communicate the message.

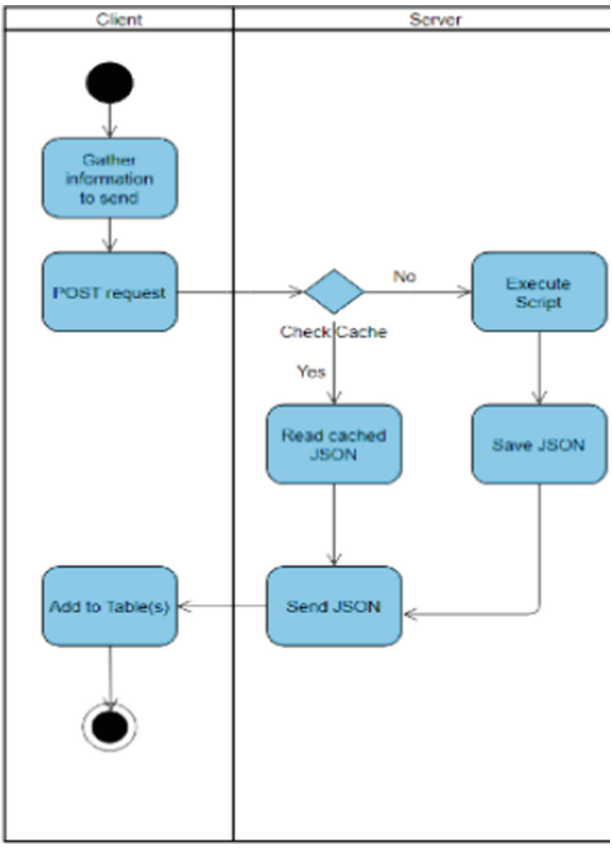

Fig. 1. Activity diagram

As a web page analysis platform, the following diagram represents the process to generate any report. The process starts by identifying the necessary information present on the website, for example, for the spelling report, the information will be the URL, the language, and the content. Then send that information to the server through a POST request. The requested report is checked to see if the requested report is cached, this check is made by the URL. If there is, then a JSON response is returned [5]. Otherwise, the dedicated script for each type of report is executed, the JSON is saved for future cache and finally the JSON is returned to the client. Finally, in the client, the JSON information is added to the desired table.

3 Results

This section aims to present the platform, as the API that supports it. Finally, an explanation of the platform's security.

3.1 Homepage

The platform's homepage allows the user to view the desktop version of the page to be analysed. The page can be loaded by entering the URL in the top bar, as well as selecting the page language and WCAG accessibility level. In the sidebar are the different views, "desktop", "mobile" and "code". In the desktop version, the page is loaded and displayed just as it would be on a desktop. The same goes for the mobile version. The view code allows the user to see the source code of the page. By default, the page is loaded in the desktop version. In the sidebar, the user can access the various reports. If the user wants to access the desired page directly, he can add the "URL" argument to the base URL. Users can also add the "lang" argument to indicate the language of the page to be analysed.

3.2 Spelling Report

The spelling report aims to make a spelling analysis of the page. Thus, allowing the user to view the spelling errors present on the page in order to correct them. The analysis is done through the open-source tool, LanguageTool [6, 7], developed in Java. This report, as we can see in Fig. 2, is available in the form of an API, receiving as arguments the URL, content and language of the page. The API returns a JSON response. If the user has not specified the language, then language detection is performed. If this argument cannot be found, a prediction is then made by the LanguageTool tool, finally, if both fail, the British English language is used by default. The user dictionary is managed through a cookie stored in the user's browser. In this way, it allows the user a more fluid and continuous experience.

```
let spellCheckJSON = await $.post("https://127.0.0.1/InspectorWS/LanguageTool", {
    content: spellTagsElem,
    langCode: langCode,
    url: siteUrl
}, function (result) {
    return result;
});
```

Fig. 2. API - Spelling Report

3.3 Accessibility Report

This report intends to present to the user the various accessibility errors of the page. This analysis allows you to verify whether the page complies with the Web Content Accessibility Guidelines (WCAG). This analysis is done through a library developed in NodeJS, HTML CodeSniffer [8]. This tool takes as arguments the URL of the page and the accessibility level, WCAG2A, WCAG2AA or WCAG2AAA [W3C 2022] – see Fig. 3. The API returns a JSON response. By default, the accessibility level is WCAG2AA.

```
let accessibilityJSON = await $.post("https://127.0.0.1/InspectorWS/Accessibility", {
    url: siteUrl, level: WCAGLevel
}, function (result) {
    return result;
});
```

Fig. 3. API - Accessibility Report

3.4 Cookies Report

The Cookies report seeks to show the user, as the name implies, which cookies are present on the analysed page. The analysis is done by a library developed by Google in NodeJS, Puppeteer [9]. This library allows controlling Chrome or Chromium through the DevTools protocol. The API takes the URL of the page as an argument and returns a JSON response, as we can see in Fig. 4.

```
let cookiesJSON = await $.post("https://127.0.0.1/InspectorWS/Cookies", {
    url: siteUrl,
}, function (result) {
    return result;
});
```

Fig. 4. API - Cookies Report

3.5 Technologies Report

This report, technologies report, aims to present the technologies used by the analysed page. This analysis is done by a library developed in NodeJS, Wappalyzer [10]. As we can see in Fig. 5, the API takes the URL of the page as an argument and returns a JSON response.

```
let wappalyzerJSON = await $.post("https://127.0.0.1/InspectorWS/Wappalyzer",
    url: siteUrl,
}, function (result) {
    return result;
});
```

Fig. 5. API – Technologies Report

3.6 Images Report

This report aims to present the images present on the analysed page to the user. The analysis is done by a library developed in Python, Pillow [11]. This library allows you

to identify and analyse images present on a given page. The API takes the URL of the page as an argument and returns a JSON response – see Fig. 6.

```
let wappalyzerJSON = await $.post("https://127.0.0.1/InspectorWS/Wappalyzer",
    url: siteUrl,
}, function (result) {
    return result;
});
```

Fig. 6. API - Images Report

3.7 Subdomains Report

This report aims to present to the user the subdomains related to the domain of the analysed page. The analysis is done by a tool developed for Linux, Windows and MacOs. This CRT tool [12] allows you to analyse the certificate transparency logs (CRT) [13] of a domain. As is shown in Fig. 7, the API takes the URL of the page as an argument and returns a JSON response.

```
let domainsJSON = await $.post("https://127.0.0.1/InspectorWS/DomainDiscovery",
    url: siteUrl,
}, function (result) {
    return result;
});
```

Fig. 7. API - Subdomains Report

3.8 Google Lighthouse Report

This report aims to present to the user the report made by Google Lighthouse [14] of the analysed page. The analysis is done by a library developed in NodeJS, Lighthouse. It has performance audits, accessibility, progressive web applications, SEO. Lighthouse receives a URL, and it performs a series of audits on the page and then generates a report on the page's performance. As is shown in Fig. 8, the API takes as an argument the URL and the device to be used as a test (desktop or mobile) and returns a JSON response.

3.9 Hyperlinks Report

This report intends to present the hyperlinks present on the analysed page to the user. The analysis is done by a library developed in NodeJS, url-status-code [15]. This library allows you to identify and analyse the hyperlinks present on a given page. As we can see in Fig. 9, the API takes the URL of the page as an argument and returns a JSON response.

```
let lighthouseJson = await $.post("https://127.0.0.1/InspectorWS/Lighthouse",
    url: siteUrl, device: device
}, function (result) {
    return result;
});
```

Fig. 8. API - Google Lighthouse Report

```
let linkJSON = await $.post("https://127.0.0.1/InspectorWS/Links",
    url: siteUrl,
}, function (result) {
    return result;
});
```

Fig. 9. API - Hyperlinks Report

3.10 Security

The user does not need to have authentication or authorization to enjoy all the features of the platform. However, there are several other areas of computer security that have been considered throughout the development of the platform. To access the hosting service, two-factor authentication is required, that is, a username/password combination as well as a six-digit verification code, sent to the mobile phone or to a mobile application. Server access via SSH (Secure Shell Protocol) is also ensured. Each user has their individual access account, and a username/password combination is required as well as a private key. The version of Linux, Ubuntu Server, is the most current and to combat computer attacks updates are being made regularly.

To avoid data loss in the event of a catastrophic failure, a backup policy was adopted. This policy is known as 3-2-1:

- 3 backups;
- In 2 types of storage;
- With at least 1 at a distant physical location.

Therefore, every night 3 full copies of the server are made, a local copy, another on an external device and another on a cloud service.

4 Conclusions and Future Work

In addition to the development of the web platform described above, this project also had as its general objective the active participation in projects of a company at an international level. In order to acquire experience in the field of Information Systems and Computer Engineering, understand how the professional environment of a company works and its framework in the world of work. After several months of work, these objectives were successfully met.

4.1 Future Work

Throughout the development of this project, there were several challenges. The first was the decision of which technologies to use. It was decided to use Java Server as the platform support framework. This framework, in addition to being able to respond to all requirements, is also used in several platforms developed by Little Forest, thus contributing to the decision of its use. However, this framework is too complex for the type of platform developed. Thus, it is planned, in future versions, to use a lighter web framework, for example, Python Flask, and be able to meet the same requirements.

References

1. Filipe, F., Pires, I., Gouveia, A.J.: Why web accessibility is important for your institution. In: Paper Presented at the CENTERIS - International Conference on ENTERprise Information Systems, Lisboa, Portugal (2022)
2. Making your service accessible: an introduction. https://www.gov.uk/service-manual/helping-people-to-use-your-service/making-your-service-accessible-an-introduction. Accessed 31 Mar 2022
3. Ramesh, M.R.R., Reddy, C.S.: Metrics for software requirements specification quality quantification. Comput. Electr. Eng. **96**, 107445 (2021)
4. Barros-Justo, J.L., Benitti, F.B.V., Tiwari, S.: The impact of use cases in real-world software development projects: a systematic mapping study. Comput. Stand. Interf. **66**, 103362 (2019)
5. Crockford, D.: JSON. https://www.json.org/json-en.html. Accessed 19 Nov 2021
6. LanguageTool: LanguageTool - Online Grammar, Style & Spell Checker. https://languagetool.org/. Accessed 23 Apr 2021
7. LanguageTool: LanguageTool: Style and Grammar Checker. https://github.com/languagetool-org/languagetool. Accessed 25 Nov 2021
8. Squiz: HTML_CodeSniffer. https://squizlabs.github.io/HTML_CodeSniffer/. Accessed 26 Dec 2021
9. Google "Puppeteer.": https://github.com/puppeteer/puppeteer. Accessed 01 Jan 2022
10. Wappalyzer: Find out what websites are built with - Wappalyzer. https://www.wappalyzer.com/. Accessed 02 Jan 2022
11. Lundh, F.: Pillow. https://pillow.readthedocs.io/en/stable/. Accessed 05 Jan 2022
12. Cumulus: crt. https://github.com/cemulus/crt. Accessed 18 Feb 2022
13. Google "Certificate Transparency: Certificate Transparency.": https://certificate.transparency.dev/. Accessed 18 Feb 2022
14. Google "Lighthouse.": https://github.com/GoogleChrome/lighthouse. Accessed 18 Feb 2022
15. Zrrrzzt "url-status-code.": https://github.com/zrrrzzt/url-status-code. Accessed 18 Feb 2022
16. digicert "WHAT IS na SSL CERTIFICATE": https://www.digicert.com/what-is-an-ssl-certificate. Accessed 24 Feb 2022