Journal of Information Systems Engineering and Management
2021, 6(2), em0138
e-ISSN: 2468-4376
https://www.jisem-journal.com/
Research Article
MODESTUM
OPEN ACCESS

The Creation Process of the Acceasy: A Framework for Creating Front-ends that are Accessible for Visually Impaired People in the Web

Renan Soares Germano [1]*, Maria Amelia Eliseo [1]

[1] Mackenzie Presbyterian University, São Paulo, BRAZIL
***Corresponding Author:** renan.rsg@hotmail.com

Citation: Germano, R. S. and Eliseo, M. A. (2021). The Creation Process of the Acceasy: A Framework for Creating Front-ends that are Accessible for Visually Impaired People in the Web. *Journal of Information Systems Engineering and Management, 6*(2), em0138. https://doi.org/10.29333/jisem/9662

ARTICLE INFO

Published: 23 Jan. 2021

ABSTRACT

This article presents the process of creating a framework for the front-end development of web applications with accessibility for the visually impaired and responsivity. As a result, Acceasy was created. Its operation, some examples and its contributions to web application developers and end users are presented. The framework created proved to be able to generate web pages with responsivity support, semantic codes and use of accessibility markings automatically, thus facilitating the work of developers. The pages created with the tool also show improvements in accessibility, giving support to screen readers and keyboard navigation. The framework is still in development. Some validations still need to be made and at the end, future works are listed.

Keywords: accessibility, responsiveness, web applications

INTRODUCTION

According to the World Report on Disability, a document created by the World Health Organization in 2011, there were about 1 billion people with some kind of disability in the world. This is equivalent to about 15% of the global population (World Health Organization & World Bank, 2011). This value also continues to grow, due to the population aging, chronic health issues and the improvement of the measurement techniques. This proportion also was kept in 2018 (World Health Organization, 2020a).

Estimates from the World Health Organization also show that the amount of people with visual impairment are increasing along the time. This value passed from 148 thousand in the first survey, in 1995 (Thylefors et al., 1995), to 2.2 million in 2019 ((World Heath Organization, 2020b). The chart in **Figure 1** expresses this growing.

From the shown data the accessibility importance is clear. The United Nations define accessibility as an indispensable social and economic transformation mechanism. It comes to ensure the human rights for the people with disabilities and also to enable these people to participate in every social activity, being active citizens. Accessibility also can be seen as a characteristic of things that were adapted so the people with disabilities can use them (United Nations, 2013a).

Another factor that contributes to the high importance of the accessibility are the global initiatives. Among them, the Sustainable and Development Goals for 2030, that aims a better World and mention that the disability must not be a barrier to achieve these goals (United Nations, 2013b). And also, the Convention on the Rights of Persons with Disabilities, that aims to be a social transformation instrument and also contributes to change the way people with disabilities are seen (United Nations, 2006).

The great presence of the technology in our lives also is a very known factor nowadays. A report made by Kemp (2018) brings some relevant data about the topic. According to the report the number of internet users overpass 4 billion; the number of active social networks users overpass 3 billion; and there are a bit more than 5 billion people that use only mobile devices. When relating this data with the one presented in the last paragraphs it's inevitable to think about the importance of web accessibility. This way, it is very important to give support to the accessibility needs in the web pages, enabling the people with disabilities to perform the same tasks as the people without disabilities. This is extended for any other technology involving hardware and/or software.

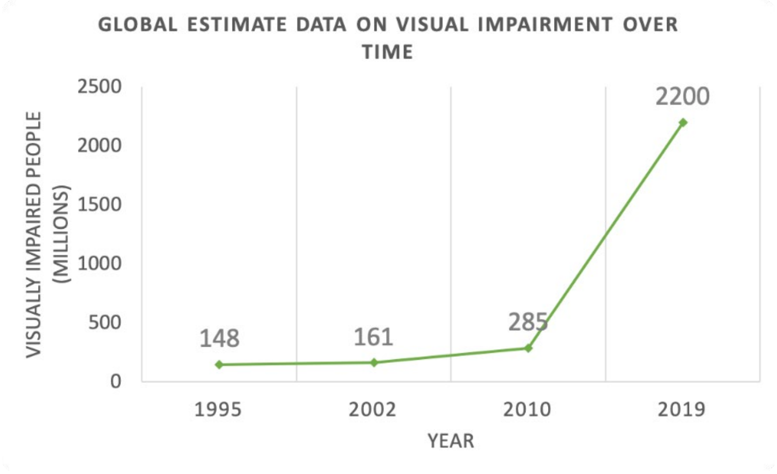


Figure 1. Chart showing the estimated number of people with some type of visual impairment in the world over the years, according to WHO data (Thylefors et al., 1995; Resnikoff et al., 2004; Word Heath Organization 2012, 2020b)

Universal Design is a possible approach that can be used when creating new things. Because of its inherent concepts it is strongly connected with accessibility. This approach, created in 1997, aims to develop products (and things in general) that can be used by the greatest number of people that is possible. One of the things that need to be considerate to achieve this is the people's different abilities (Centre for Excellence in Universal Design, 2020).

With the great number of users accessing the web through mobile devices it also becomes important to address the usability and accessibility for this specific case. For that, the responsivity concept emerges. The responsivity allows the web pages to adjust their layouts according to the device screen size that is being used to navigate. This way, the idea is to have a specific layout to each possible size - the one that provides the best usability without having to redo the entire page for each size (Carver, 2014).

From everything that has been said it is possible to affirm that the web developers have a great responsibility when creating the web pages. To implement all these requirements also demand a lot of knowledge of different things. Because of this, it's common to opt in using a third-party framework. Most of them deliver the responsivity and ready design implementations. In contrast, most of them don't consider the accessibility necessities, or demand specific knowledge for its implementation.

The objective of the present work was to show a development of a web front-end framework to allow the developers to create responsible and accessible for visually impaired people web applications. As the result, the Acceasy was elaborated, and the initial version of it is presented here.

Following, the article organization. Introduction: this section, presenting a general vision of the scenario, the motivation and objective of the project. Theoretical framework: definition of some concepts and related works. Methodology: the step by step of the project creation. Acceasy: the created framework, how to use it and some examples. Discussions and results: discussion of the results achieved. Conclusion and future works: the final conclusions and a list of plans for Acceasy.

THEORETICAL FRAMEWORK

Concepts

The world report on disability (World Health Organization & World Bank, 2011) says that the disability concept has changed along the time, passing from a medical model, focused on the individuals and their body and health conditions, to a social model, where people are considered disabled according to the environmental and social barriers that exist in the place where they live. Despite it, the report says that the disability needs to take in consideration both contexts, the heath and the social.

Changing the perspective of the way disability is seen to a more social model is very important to identify and remove the barriers that exist in this sphere. These barriers are in lots of components in society: the governments, the public and private companies, public politics, in the way the population understand the disability in a general way and many others. The major initiative to change this scenario is through education about the topic. Disability also is considered a human rights subject, because people with disabilities must have granted the same rights and opportunities as any other person without disabilities (World Health Organization & World Bank, 2011).

From the disability concept emerges the accessibility term, that comes to define the process of adaptation of things and society, so the people with disabilities may have granted the above rights. The Convention on the Persons with Disabilities, in the article 9, lists actions that can be taken by the States Parties to address this challenge. It also says that beyond taking actions, to take measures is important - basic levels of accessibility need to be established and the measures must be used to achieve, monitor, and keep these levels. Another important action is to capacitate the citizens to help the persons with disability in using the services and products provided to the society. All these actions also are directed to the technologies' adequation (United Nations, 2006). And, as mentioned in the introduction, it also is possible to say that things that were adapted, or originally created in a way that allows people with disabilities to use them, are accessible (United Nations, 2013a).

Ronald Mace, with his research group, created in 1997, at North Carolina State University, the Universal Design concept. This idea proposes the development of environments, products and communications - and later would be extended to all areas of knowledge, including that of computer systems - that can be used by the greatest number of people possible (Centre for Excellence in Universal Design, 2020).

It consists of seven principles: (1) equitable use - the design must be useful and pleasant for people with different skills; (2) flexibility in use - the design must accommodate the preferences of different users with different skills; (3) simple and intuitive use - the design must be easily understood, regardless of the level of instruction or any other factor related to the user; (4) perceptible information - the design must communicate the necessary information effectively, regardless of the environment conditions or the user's sensory abilities; (5) error tolerance - the design must minimize the risks and consequences of accidental actions; (6) little physical effort - the design must be able to be used efficiently and comfortably with the minimum effort exerted by the user; and (7) size and space for approach and use - the design must have enough size and space to be able to use it properly, regardless of the user's characteristics (Centre for Excellence in Universal Design, 2020).

To create a web page involves three main technologies: Hypertext Markup Language (HTML), Cascading Style Sheet (CSS) and JavaScript (JS). Each of these has a specific responsibility. The HTML is used to struct the web page content (Faulkner et al., 2017). The CSS is used to implement the page design, including the layout, colors, fonts and related things (Atkins, 2019). And the JS is used to perform actions and make the web page more interactive and dynamic (Ecma International, 2020).

Using HTML, the web developers can struct the contents using tags. Tags are the small piece used to structure a page. There are many tags with different objectives. Each tag contains a start and an end indication (or opening and closing indications). For example, the tag ***<p>…</p>*** is used to represent a paragraph. The tags also can receive different attribute values (for example an ***id*** to identify a specific tag). To be rendered in a web browser a HTML document is transformed into a Document Object Model (DOM), that is a tree in-memory that represents the document. This DOM tree can be manipulated with JavaScript. The DOM HTML tree is important for the web accessibility, because from it is derived the Accessibility tree, considered a simplified version of the DOM HTML tree. With this second tree the people with disabilities that use assistive technologies are able to navigate in a website. (Faulkner et al., 2017; Scheuhammer et al., 2014).

Carver (2014) defines in his work, "The responsive web", a responsive site as the one that can be accessed from the same URL on devices with different screen sizes, and which, according to the screen size, adjusts the page layout. Most of the work necessary to create a responsive website is done using CSS, and more specifically the Media Queries and Breakpoints, both CSS resources. Media Queries allow the developer to import and apply different styles to a page accordingly with some criteria, including the screen size. Breakpoints are the points where the layout should change, the definitions of the sizes and which layout and design apply to each case (Carver, 2014).

When associating accessibility and technology, one of the most common items is the Assistive Technology (AT). The ATs are technologies, hardware and software, that help people with disabilities to perform some daily activity, or even to use another technology, facilitating their lives and giving more independence to them. It also is considered a necessary component to promote the social inclusion of people with disabilities (World Health Organization, 2020c). In the web development, a very commonly used assistive technology is the screen reader, used by the visually impaired people (Zahra & Brewer, 2017). In a general way, the role of assistive technologies in the web is to present the website contents in a better form, that fits the users' needs. These tools make it by using the HTML trees (Scheuhammer et al., 2014).

However, to ensure an adequate and good experience to the users with accessibility needs in the web, the pages content must be well implemented. Here, the Universal Design concept can be a helpful guide. Another simple way of addressing this need is to use the HTML semantic tags, present from the fifth version of the language (see **Figure 2**). With the HTML5 semantic tags, the accessibility trees of the pages are richer in semantic contents and contributes to a better accessibility support (MDN, 2020).

Thinking about establish a pattern to the web development, as well as define a set of rules to implement accessibility in the pages, the World Wide Web Consortium (W3C), the responsible organization for creating patterns for the internet, with the objective of achieving the greatest potential of this global network, created a set of accessibility guidelines. These are the Web Content Accessibility Guidelines (WCAG). The next section explains a bit about them.

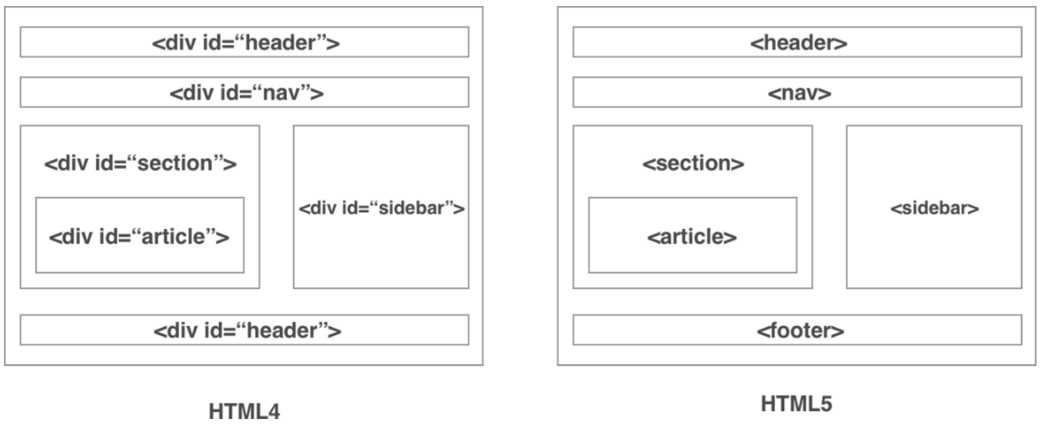


Figure 2. Example of a basic HTML structure of a simple layout in two versions. At left, the version using non semantic tags. At right, the version using semantic tags

Accessibility Guidelines

With the WCAG, the web pages can be classified according to the accessibility level of them: **A**, **AA** and **AAA** (the first being the classification of the least accessible sites, and the last, the classification of the most accessible sites, respectively) (Henry, 2009). The version 1.0 has 14 guidelines, each with a specific list of items. Each item has a priority (1, 2 or 3), and the sites are classified as follows: **A**, if all priority 1 items are implemented; **AA**, if all priority items 1 and 2 are implemented; and **AAA**, if all priority items 1, 2 and 3 are implemented (Chisholm et al., 2001).

The WCAG 2.0 contains 12 guidelines separated in 4 accessibility principles: noticeable, operable, understandable and robust. In this second version, each guideline has a set of success criteria, levels **A**, **AA** and **AAA**, which are used in the same way as the previous version to classify the pages (Caldwell et al., 2008). Also, there is the WCAG 2.1 that works in the same way as 2.0 but has new success criteria (Kirkpatrick et al., 2018).

In addition to WCAG, the Web Accessibility Initiative - Accessible Rich Internet Applications (WAI-ARIA, for short) is also an important standard created by the W3C. It is a technical specification that provides a framework that makes it possible to improve the accessibility of web pages from markings in the HTML's tags. These markings define different characteristics of the interface components. They can define role, state, and properties. Thus, it is possible to inform ATs what is on the screen, how the user should interact with it, what is the state of each component, and notify if there is any state update (Diggs et al., 2017).

Many countries have laws that define accessibility requirements for the web - these laws are very common for governments and public services sites. A very known law, and one of the firsts, used as base to create other guidelines, is the Section 508 (U.S. Government, 2018), a 1998 American law. It is possible to check a list of countries and their web accessibility laws in (Mueller et al., 2018).

Related Works

The ARIA-ACCESS framework created by Mahmud (2016) helps to improve the accessibility of the pages created with it. The initial goal of the author was to create a framework to help in the creation of accessible web games, but later it was changed to the creation of web applications in general. The ARIA-ACCESS functioning is similar to the most popular front-end frameworks, like Bootstrap (n.d.) and Foundation (n.d.). This is, by marking the HTML tags with specific values for the ***class*** property, from which the framework can manipulate the tags (in the DOM tree) and perform changes on them. In the case of the Mahmud framework, the alterations consist in adding WAI-ARIA markings in the tags. The visualization of the interface components was kept the default - the ARIA-ACCESS doesn't have any personalized CSS. It also doesn't have implementation for responsivity support.

The Turretcss framework is another relevant tool to this project. It is a CSS framework to implement accessible web interfaces. This framework presents a simple and minimalist design implementation, uses the HTML5 semantic tags and have support for responsivity. It doesn't generate automatic HTML code and also doesn't add WAI-ARIA markings automatically, so most of the accessibility implementation continues in the developers' hands (Turretcss, 2020).

The PUXL library, also a CSS based tool, is another project that can help in the creation of accessible web pages. It uses the HTML5 semantic tags, promises a site that is automatically level AA in the WCAG 2.1 classifications, and contains a feature that checks the code and prevents basic accessibility mistakes, showing alerts and hints for how to fix each issue. It supports responsivity implementation (PUXL, 2020).

A list with some other accessibility helpful resources also can be found in (Peri, 2020). Some of them use only one of the main web development technologies (HTML, CSS, JS) or a mix of them. They also can be a set of util codes created by some company to facilitate the web developers work, or even a separated package of another tool to improve the accessibility.

METHODOLOGY

Initially, some tests were made to check the accessibility support of different technologies. Four technologies were used: HTML5, the ARIA-ACCESS framework, the Turretcss framework and the Foundation framework. With exception of HTML5, the frameworks were chosen by reading their documentations and checking if they mentioned native accessibility improvements in the sites created with them. Two different tests were made, and the objective was to find the technology that could build the most accessible web pages.

The tests consisted in creating different versions of the same web page (one version for each technology) and performing some keyboard navigation using different browsers. The first test was made with the implementation of a navigation menu. The second test was made with the implementation of a form with different inputs. They were executed in the computer and browsers with the following specifications:

- MacBook Pro with macOS High Sierra version 10.13.6.
- Safari web browser, version 12.0.
- Chrome web browser, version 69.0.3497.100.
- Firefox web browser, version 62.0.2.

The page for the navigation menu test has three navigation items at the top and three sections with a title and a text content. Each section title has a different id. Each navigation item links to a title of one section. Navigating with the keyboard, the following tests were made: (1) check if it is possible to give focus in the entire menu; (2) check if it is possible to navigate in the menu items in some way; (3) check if it is possible to navigate in the menu items using the arrow keys; (4) check if it is possible to click in a

Table 1. Results for the navigation menu test. The check means that it was possible to execute the task for the indicated technology and browser. Each check computed one unit in the total sum

	Chrome						**Safari**						**Firefox**						**Total**
	1	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**	**5**	**6**	**1**	**2**	**3**	**4**	**5**	**6**	
HTML5		✓		✓	✓														3
ARIA-ACESS	✓	✓		✓	✓	✓	✓					✓	✓					✓	9
Turretcss		✓		✓	✓														3
Foundation	✓	✓		✓	✓														4
Total			11						2						2				

Table 2. Form test results. The check means that it was possible to execute the task for the indicated technology and browser. Each check computed one unit in the total sum

		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**		**Total**
Chrome	HTML5			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	11	46
	ARIA-ACESS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	13	
	Turretcss			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	11	
	Foundation			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	11	
Safari	HTML5			✓	✓			✓		✓	✓		✓		✓	7	30
	ARIA-ACESS	✓	✓	✓	✓			✓		✓	✓		✓		✓	9	
	Turretcss			✓	✓			✓		✓	✓		✓		✓	7	
	Foundation			✓	✓			✓		✓	✓		✓		✓	7	
Firefox	HTML5			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	11	46
	ARIA-ACESS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	13	
	Turretcss			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	11	
	Foundation			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	11	

menu item using the space key; (5) check if it is possible to click in a menu item using the enter key; (6) check if it is possible to give focus in the paragraphs of each section. **Table 1** shows the result for this test.

The page for the form test has a title and a form with two field sets. The first one contains 8 inputs: first name - an input of type text; last name - an input of type text; birth date - an input of type date; id - an input of type text; photo - an input of type file; country - a select with three options; sex - two radio buttons. The last one contains three inputs: e-mail - an input of type email; password - an input of type password; confirm password - an input of type password. The form also has a submit button, and its action attribute value points to another HTML page containing a single text saying that the test is over. Every input is required and has unique values for the id and name properties.

Navigating only with the keyboard, the following tests were made to check if it is possible: (1) to give focus in the page title; (2) to give focus in the field sets; (3) to give focus in the text inputs; (4) to give focus in the date input; (5) to give focus in the data input's sub items (year, month, day); (6) to give focus in the file input; (7) to give focus in the select; (8) to give focus in the radio buttons; (9) to give focus in the e-mail input; (10) to give focus in the password inputs; (11) to give focus in the submit button; (12) to activate the form action by pressing the enter key; (13) to navigate through the inputs using the arrow keys; (14) to fill in the entire form using only the keyboard. **Table 2** shows the result for this test.

In the first test the results were better in the Chrome web browser and there was a draw between Safari and Firefox. In the second case there was a draw between the Chrome and Firefox web browsers, in which was possible to perform most of the actions successfully. In most of the cases, the page that was built using the ARIA-ACCESS technology had better results. It happened due to the technology adds WAI-ARIA markings to the tags. This way, this test contributed by showing the importance of the WAI-ARIA markings and that different web browsers have different keyboard navigation support levels. All the test files can be found at https://github.com/rsg73626/acceasy/tree/master/dev/accessibility-tests.

Then, a study of the web accessibility guidelines was made, in order to understand their functioning, how to implement them and how they could be put in a generic framework. A study about the WAI-ARIA also was made with the same objective. After that, a list of requirements was created. The Acceasy requirements were organized in three different groups, accessibility, technology and development utility.

The accessibility requirements are: (1) enable the user to navigate in the pages using the keyboard; (2) use the WAI-ARIA markings to improve the accessibility when it is possible; (3) create components that are in accordance to the W3C guidelines; (4) prevent basic mistakes that can compromise the accessibility. The technology requirements are: (5) use the semantic tags from the HTML5; (6) don't use nested tags to implement layouts; (7) use the variables from CSS to implement a flexible design; (8) use only CSS to implement the layouts; (9) use the CSS to implement the responsivity support. And the utility requirements are: (10) allow the web developers to create pages that are automatically responsive and accessible to visually impaired people; (11) allow the web developers to create codes that can be reused in different pages.

Beyond these requirements it was thought that the framework also could have ready components. A ready component is an interface element that contains some points that the web developers can personalize. Some of the possible components are navigation menu, alerts, cards, lists, tables, color themes, built-in layouts and skip links. This list is not exhaustive and along the time lots of them can be implemented in the Acceasy.

After this, it was possible to start working in the Acceasy. It was necessary to define how the framework would work and what would be its functionalities. Thinking about to facilitate the web developers work, it was decided to create a tool that works mainly with JS. The idea is that the developers won't need to write HTML code, and the details of accessibility and responsivity will be held by the JS framework mechanism. Some of the functionalities are to provide an easy way for HTML tags creation, provide an easy way to define a basic layout, provide a way to prevent mistakes that can compromise the accessibility. The next section presents the framework in more details.

ACCEASY

The Acceasy is a JavaScript framework to create front-ends accessible for the visually impaired people in web applications. With this tool, the creation of the pages is done using the framework's provided functions. Each HTML tag has a corresponding function in the framework (Germano et al., 2020). The framework also provides a ready navigation menu component with responsive and accessible implementation, helpful functions to stylize the page, functionality to position inputs inside a form without using nested tags and without compromise the accessibility and a generic function to create tags that still do not have a specific function in the framework. All the implementation, documentation, a template, and some examples can be found at https://github.com/rsg73626/acceasy.

JSON Object to HTML Tag

The Acceasy uses JSON objects to create the HTML tags. Every JSON object that represents a tag contains two basic properties: type, that must be an integer value between 0 and 26; and content, that can have different values, according to the tag that will be created. For example, a valid JSON to create a simple paragraph tag would be: ***{type: 0, content: "This is a paragraph."}***. The value type zero represents the ***<p>*** tag in the framework. The zero value also could be replaced by the Acceasy helper constant ***_p_*** (each HTML tag type has a similar constant helper in the framework).

Another example of a JSON that would be recognized by the framework is: ***{type: _a_, content: {text: "This is a link", link: "#"}}***. This object would be transformed in a link tag (***<a>***). The helper constant is being used to indicate the tag type, and the content property is another object with the link ***text*** and ***href*** attribute value for the tag. The output of this object would be: ***<a href = "#">This is a link</a>***.

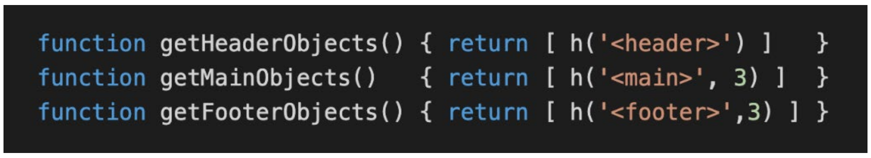
But the framework users don't need to be aware of all these details. Instead, they can use the functions that represent the tags. With these functions it isn't necessary to type all the characters to create the JSON objects directly. Each function has the name of the corresponding HTML tag, so the type value is not necessary anymore. Now, it is just necessary to pass the values required to build the tag through the functions' arguments.

Now, to create a paragraph it is possible to write the following code: ***p("This is a paragraph.")***. It also is possible to pass a list of ***strings*** to the ***p*** function: ***p(["Text 1 ", "Text 2 ", "Text 3 "])***; in this case the texts will be joined into a single ***string*** and the output will be ***<p>Text 1 Text 2 Text 3 </p>***. To build a link, the following code is used: ***a("This is a link", "#")***, that will have the same HTML output presented in the example using the JSON object. Every supported HTML tag has a similar function with their specific arguments. Behind the scene, what each of these functions do is to create the JSON objects and return them. More details about each function can be found in the documentation.

Default Page Layout

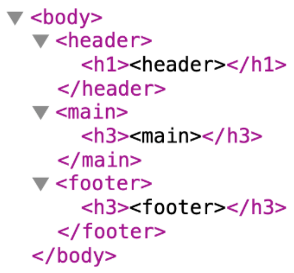
A very common and simple layout for a web page is the one that contains a header, a main content and a footer. Because of that, the Acceasy framework contains three functions that need to be implemented by the developer. Each function must return a list of JSON objects that will be transformed in HTML tags and will be put into the specific page section.

The ***getHeaderObjects*** function will transform the returned JSON objects in HTML tags and put them into a header tag (***<header>***). The ***getMainObjects*** function will transform the returned JSON objects in HTML tags and put them into a main tag (***<main>***). The ***getFooterObjects*** function will transform the returned JSON objects in HTML tags and put them into a footer tag (***<footer>***). If any of them are not implemented the page renderization won't break. The header, main and footer tags will be placed inside the document's body tag (***<body>***). **Figures 3**, **4** and **5** show a simple example of the default layout.



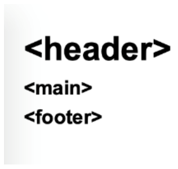
```
function getHeaderObjects() { return [ h('<header>') ]   }
function getMainObjects()   { return [ h('<main>', 3) ]  }
function getFooterObjects() { return [ h('<footer>',3) ] }
```

Figure 3. Code implementation for the default layout. The *getHeaderObjects*, *getMainObjects* and *getFooterObjects* are implemented. Each one returns a header tag containing the tag name for the corresponding section



```
▼ <body>
  ▼ <header>
      <h1><header></h1>
    </header>
  ▼ <main>
      <h3><main></h3>
    </main>
  ▼ <footer>
      <h3><footer></h3>
    </footer>
  </body>
```

Figure 4. HTML code generated for the default layout functions inplementaiton



<header>

<main>

<footer>

Figure 5. Visualization of the HTML code generated by the default layout functions implementation

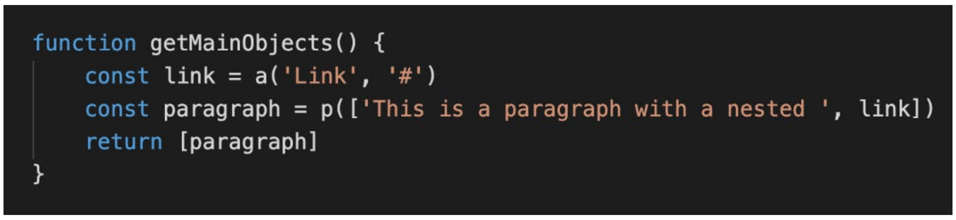
How to Use it

To use the Acceasy, clone the repository and create a new HTML file. In this new page, import the JQuery (a JS library that simplifies the DOM objects manipulation) library and the framework files: ***js/framework.js*** and ***css/framework.css***. Then, create a new JS file and also import it. Use this new JS file to implement the default layout functions.

All these steps are already done in the ***template*** folder of the Acceasy repository. The separated JS file in the template folder also contains the implementation of two other functions. The ***didStartSetup*** function is called by the Acceasy before the calling of the default layout functions. And the ***didEndSetup*** function is called after the calling of the default layout functions. The framework's users can implement these functions to perform some required routine in these specific moments (nothing happens if they are not implemented).

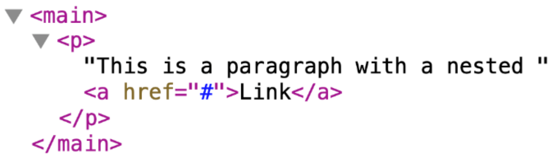
Examples

Currently, the Acceasy has 26 functions to create the JSON objects that are transformed in HTML tags by the framework. The documentation explains each of them in detail. This section presents some examples. The first one is the creation of a paragraph containing a nested link. See the example in **Figures 6**, **7** and **8**.



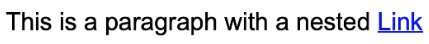
```
function getMainObjects() {
    const link = a('Link', '#')
    const paragraph = p(['This is a paragraph with a nested ', link])
    return [paragraph]
}
```

Figure 6. Implementation of the *getMainObjects* function to create a paragraph with a nested link



```
▼ <main>
  ▼ <p>
      "This is a paragraph with a nested "
      <a href="#">Link</a>
    </p>
  </main>
```

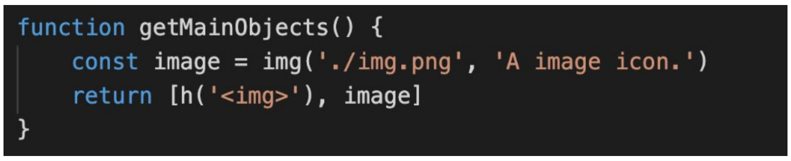
Figure 7. HTML code generated for the implementation of the *getMainObjects* function returning a paragraph with a nested link



This is a paragraph with a nested Link

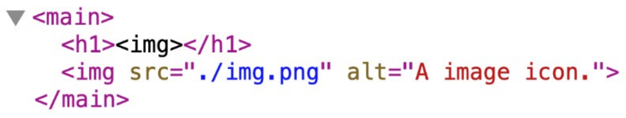
Figure 8. Visualization of the HTML code generated by the *getMainObjects* function returning a paragraph with a nested link

When creating images, a basic implementation to support accessibility is to provide an alternative text to it, so screen reader users are able to know the image content. In the Acceasy, the developer can use the ***img*** function to create an image. The first parameter is the image path and the second is the alternative text. If an alternative text is not passed the image tag is not created. See the example in **Figures 9**, **10** and **11**.



```
function getMainObjects() {
    const image = img('./img.png', 'A image icon.')
    return [h('<img>'), image]
}
```

Figure 9. Implementation of the *getMainObjects* function to create an image



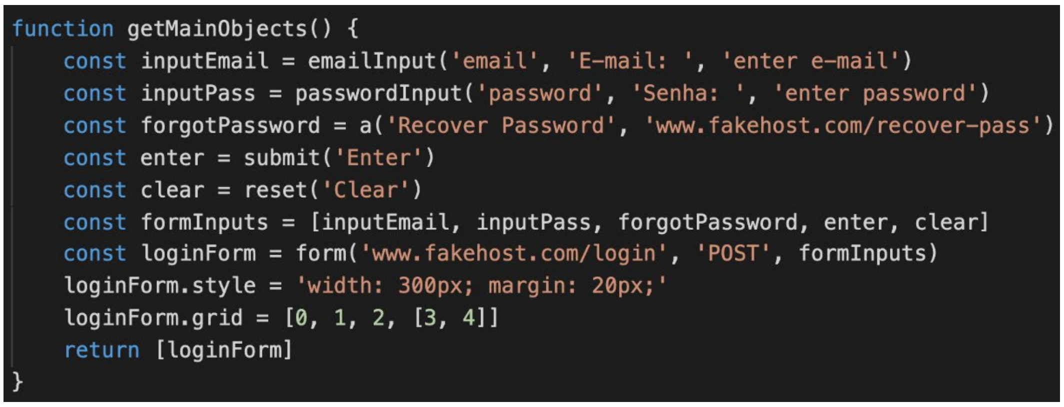
```
▼ <main>
    <h1><img></h1>
    <img src="./img.png" alt="A image icon.">
  </main>
```

Figure 10. HTML code generated for the implementation of the *getMainObjects* function returning an image. The second attribute from the *img* function is used as the value of the *alt* property for the *<img>* tag

Figure 11. Visualization of the HTML code generated by the *getMainObjects* function returning an image

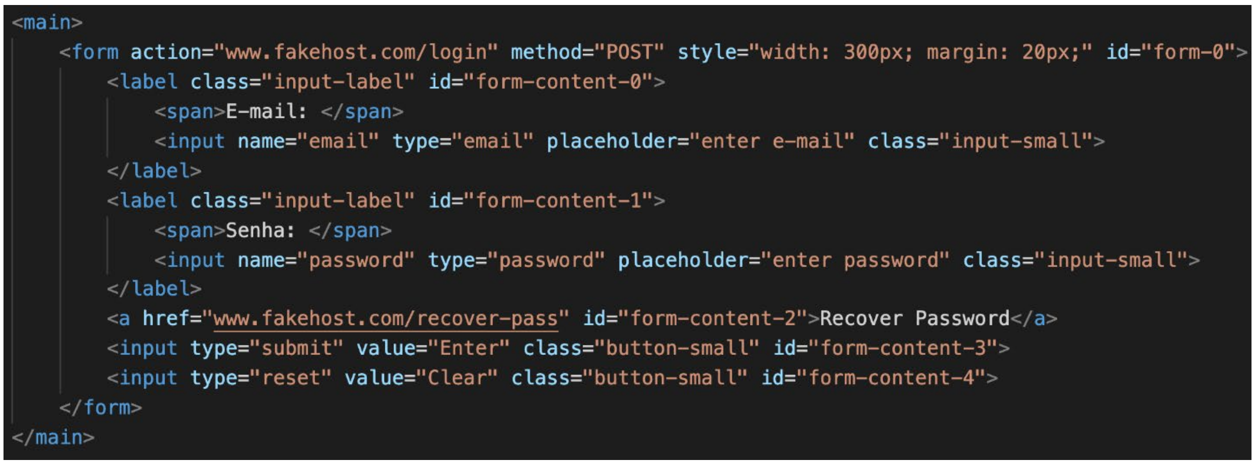
To build a form, the Acceasy provides a generic function to create different types of inputs. The ***input*** function receives the following arguments: ***type***, must be a *string* with a valid input type value (the framework has the helper constants ***_text***, ***_number***, ***_tel***, ***_email***, ***_date***, ***_submit***, ***_reset***, ***_password*** and ***_file*** that can be used); ***name***, must be a *string* with a value for the input name attribute; ***label***, must be an *string* to be used as the label of the input; ***placeholder***, must be a *string* with a text hint for the input; and ***required***, must be a Boolean value indicating if the input is required or not

To simplify the input creation, the framework also provides some helper functions, that doesn't require the first argument (the type), because the function name already says which input will be created. They are ***textInput***, ***numberInput***, ***telInput***, ***emailInput***, ***dateInput***, ***passwordInput***, ***fileInput***, ***submit*** and ***reset***. The ***submit*** and ***reset*** functions only receive a text argument, that is used as the button name. The ***fileInput*** function receives one more argument in the last position, the ***accept***, must be a *string* to limit the accepted types of files. **Figures 12**, **13** and **14** show the implementation of a login form.



```
function getMainObjects() {
    const inputEmail = emailInput('email', 'E-mail: ', 'enter e-mail')
    const inputPass = passwordInput('password', 'Senha: ', 'enter password')
    const forgotPassword = a('Recover Password', 'www.fakehost.com/recover-pass')
    const enter = submit('Enter')
    const clear = reset('Clear')
    const formInputs = [inputEmail, inputPass, forgotPassword, enter, clear]
    const loginForm = form('www.fakehost.com/login', 'POST', formInputs)
    loginForm.style = 'width: 300px; margin: 20px;'
    loginForm.grid = [0, 1, 2, [3, 4]]
    return [loginForm]
}
```

Figure 12. Implementation of the *getMainObjects* function to create a login form



```
<main>
    <form action="www.fakehost.com/login" method="POST" style="width: 300px; margin: 20px;" id="form-0">
        <label class="input-label" id="form-content-0">
            <span>E-mail: </span>
            <input name="email" type="email" placeholder="enter e-mail" class="input-small">
        </label>
        <label class="input-label" id="form-content-1">
            <span>Senha: </span>
            <input name="password" type="password" placeholder="enter password" class="input-small">
        </label>
        <a href="www.fakehost.com/recover-pass" id="form-content-2">Recover Password</a>
        <input type="submit" value="Enter" class="button-small" id="form-content-3">
        <input type="reset" value="Clear" class="button-small" id="form-content-4">
    </form>
</main>
```

Figure 13. HTML code generated for the implementation of the *getMainObjects* function returning a login form. The *id* property value is generated automatically for each form input, so this the Acceasy can position them using CSS

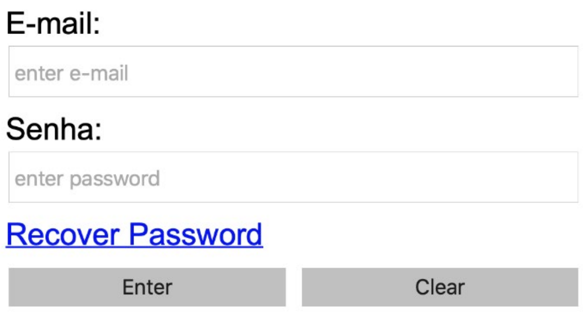
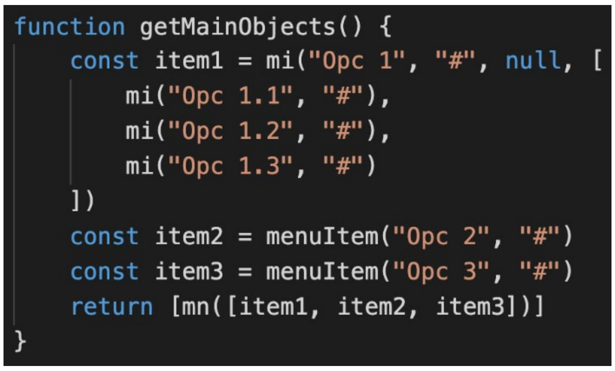


Figure 14. Visualization of the HTML code generated by the *getMainObjects* function returning a login form

Creating a label tag for each form input also improves the site accessibility. **Figure 13** also shows the grid property configuration for the form object. It is used to position the elements inside the form. This property must receive a matrix, where each line represents a new line in the layout, and each line value represents the position of the corresponding form element. If a line contains only one element the square brackets can be omitted. Each form element is represented by an integer number, from 0 to the number of elements in the form minus 1. The order of the components in the grid property must be ascending (from left to right and up to bottom), because the order of the tags in the HTML must be the same shown in the screen, avoiding accessibility issues. The Acceasy doesn't add more HTML tags to implement the position of the elements in the form.

In the current version, the Acceasy also provides a ready navigation menu component. This element supports navigation with the keyboard, using the tab and arrow keys. It also adjusts its functioning to provide a better user experience when accessing the page in a device with a screen less or equal 800 pixels. **Figures 15**, **16**, **17** and **18** show an example. The function ***getMenuColors*** can be implemented to personalize the component colors (check details of it in the documentation).



```
function getMainObjects() {
    const item1 = mi("Opc 1", "#", null, [
        mi("Opc 1.1", "#"),
        mi("Opc 1.2", "#"),
        mi("Opc 1.3", "#")
    ])
    const item2 = menuItem("Opc 2", "#")
    const item3 = menuItem("Opc 3", "#")
    return [mn([item1, item2, item3])]
}
```

Figure 15. Implementation of the *getMainObjects* function to create a navigation menu

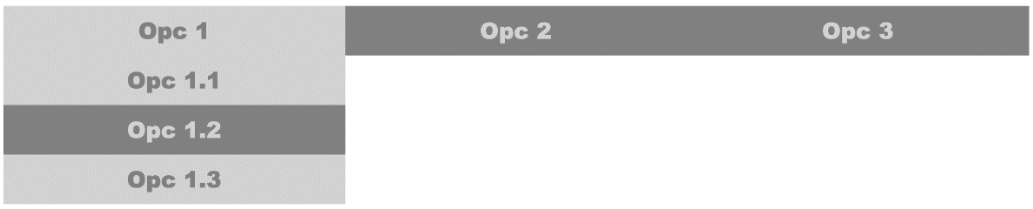


Figure 16. Visualization of the HTML code generated by the *getMainObjects* function returning a navigation menu

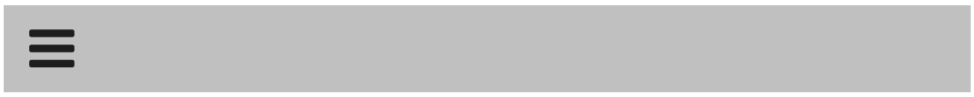


Figure 17. Visualization of the navigation menu with a screen width less or equal 800 pixels. The menu is transformed in a button with the options closed. It is possible to click in the button to show the options

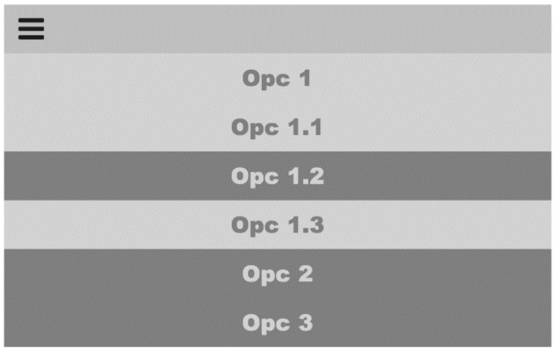
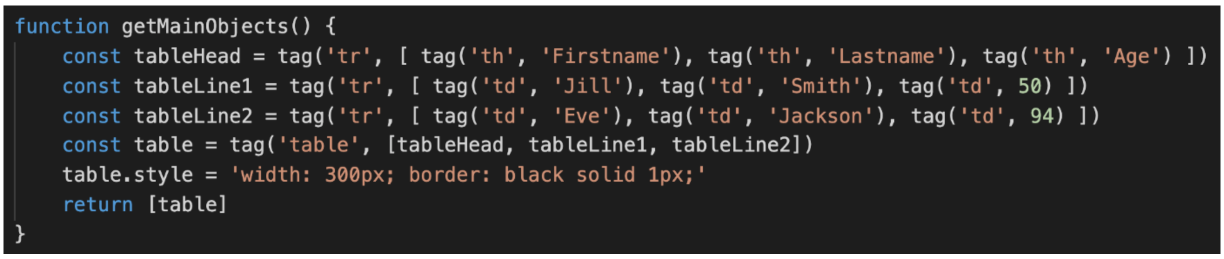


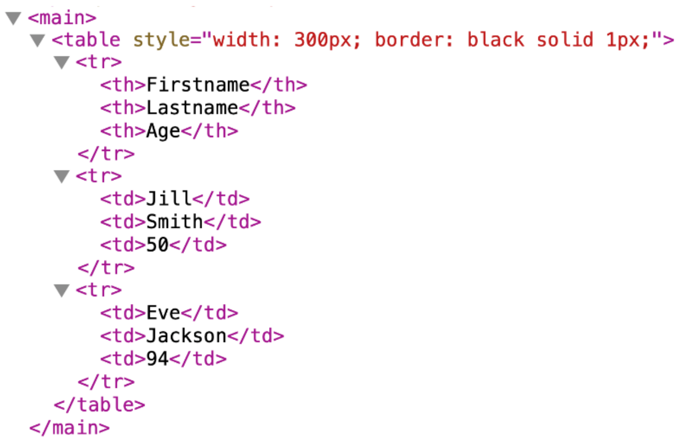
Figure 18. Visualization of the navigation menu with a screen width less or equal 800 pixels and the options opened

Due to the fact that the Acceasy still do not have a function to every HTML tag, the ***tag*** function was created. It allows the developers to create any HTML tag. The first argument is a *string* with the tag name and the second argument is the HTML tag content - it can be a text, a number, another object that represents another tag or a list of values of different types. **Figures 19**, **20** and **21** show the implementation of a table using the ***tag*** function.



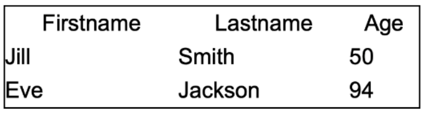
```
function getMainObjects() {
    const tableHead = tag('tr', [ tag('th', 'Firstname'), tag('th', 'Lastname'), tag('th', 'Age') ])
    const tableLine1 = tag('tr', [ tag('td', 'Jill'), tag('td', 'Smith'), tag('td', 50) ])
    const tableLine2 = tag('tr', [ tag('td', 'Eve'), tag('td', 'Jackson'), tag('td', 94) ])
    const table = tag('table', [tableHead, tableLine1, tableLine2])
    table.style = 'width: 300px; border: black solid 1px;'
    return [table]
}
```

Figure 19. Implementation of the *getMainObjects* function to create a table using the *tag* function



```
▼ <main>
  ▼ <table style="width: 300px; border: black solid 1px;">
    ▼ <tr>
        <th>Firstname</th>
        <th>Lastname</th>
        <th>Age</th>
      </tr>
    ▼ <tr>
        <td>Jill</td>
        <td>Smith</td>
        <td>50</td>
      </tr>
    ▼ <tr>
        <td>Eve</td>
        <td>Jackson</td>
        <td>94</td>
      </tr>
    </table>
  </main>
```

Figure 20. HTML code generated for the implementation of the *getMainObjects* function returning a table built using the *tag* function



Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94

Figure 21. Visualization of the HTML code generated by the *getMainObjects* function returning a table built using the *tag* function

In order to facilitate the creation of styles, the Acceasy provides some helper functions to create global CSS styles and variables. To do that, the functions ***styleVariable***, ***addGlobalStyleVariable***, ***addGlobalStyleVariables***, ***style***, ***addGlobalStyle*** and ***addGlobalStyles*** can be used. All the global styles are put into a style tag (***<style>***) in the document header tag (***<header>***). More details can be found in the documentation.

DISCUSSIONS AND RESULTS

The Acceasy is the main result of this work. It is a framework for the front-end development of web applications with support for users of screen readers, responsivity, ready-made components and design standards, based on WCAG guidelines and the use of markings and standards of WAI-ARIA. Currently it was possible to develop its main structure, with functions to facilitate the creation of most HTML tags.

As a ready-made component there is the navigation menu, with responsivity and accessibility, allowing its use on devices with small screens, as well as by users of screen readers and who navigate through the keyboard. The grid functionality is also a fundamental component to facilitate the life of the developers, while contributing to the responsivity and accessibility of the forms.

Since the characteristics of the web pages created with Acceasy benefit a wide group of users (those who need accessibility for navigation with keyboard and screen readers, those who do not, and those who access through devices with different screen sizes), and that such benefits can also reach other groups, it can be said that the framework meets the concepts of Universal Design. In addition, it is possible to state that Acceasy improves quality of life and contributes to the social inclusion of people with visual impairments, as it provides them with greater independence so they can navigate in the web.

CONCLUSIONS AND FUTURE WORKS

This article presented the entire process to create a new web front-end framework. The Acceasy framework comes to address the need of a tool to create websites with responsivity and accessibility for visually impaired users. These characteristics are important in the current scenario because of the great number of people with disabilities and also the great number of mobile devices. The tool also comes to simplify the developers' works at the same time that contributes to the social inclusion of people with visual impairments. For last, the tool also covers the users without disability by promoting a simplified design.

Despite having lots of features to improve the accessibility of web pages, and also to facilitate the web developers' work, still there are lots of work to do. Following, a list of future works: test and validation of the accessibility aspect with final users of the pages created with the Acceasy; test and validation of the tool with web developers; implement native functions for the other HTML tags; implement more options of personalization for the tags; implement more ready components like the menu; implement other layout formats; and implement ready and personalized themes.

REFERENCES

Atkins, T., Etemad, E. J. and Rivoal, F. (2019). CSS Snapshot 2018. *WWW Consortium (W3C)*. Available at: https://www.w3.org/TR/css-2018/

Caldwell, B., Cooper, M., Reid, L. G., Vanderheiden, G., Chisholm, W., Slatin, J. and White, J. (2008). Web content accessibility guidelines (WCAG) 2.0. *WWW Consortium (W3C)*. Available at: https://www.w3.org/TR/WCAG20/

Carver, M. (2014). The responsive web. New York: Manning Publications.

Centre for Excellence in Universal Design. (2020). What is universal design. Retrieved from http://universaldesign.ie/What-is-Universal-Design

Chisholm, W., Vanderheiden, G. and Jacobs, I. (2001). Web content accessibility guidelines 1.0. *Interactions, 8*(4), 35-54. https://doi.org/10.1145/379537.379550

Diggs, J., McCarron, S., Cooper, M., Schwerdtfeger, R. and Craig, J. (2017). Accessible Rich Internet Applications (WAI-ARIA) 1.1. *WWW Consortium (W3C)*. Available at: https://www.w3.org/TR/wai-aria/

Ecma International. (2020). Language Specification. Retrieved form https://tc39.es/ecma262/

Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., Moon, S., Navara, E. D., O'Connor, T. and Berjon, R. (2017). HTML 5.2. *WWW Consortium (W3C)*. Available at: https://www.w3.org/TR/html52/

Germano, R. S., & Eliseo, M. A. (2020, June). Acceasy-A front-end framework prototype for developing responsive web applications with accessibility for visually impaired people. In *2020 15th Iberian Conference on Information Systems and Technologies (CISTI)* (pp. 1-6). IEEE. https://doi.org/10.23919/CISTI49556.2020.9141063

Henry, S. L. (2009). How WCAG 2.0 differs from WCAG 1.0. *Web Accessibility Initiative (WAI)*. Available at: https://www.w3.org/WAI/WCAG20/from10/diff.php

Kemp, S. (2018). Digital in 2018: world's internet users pass the 4 billion mark. Retrieved from https://wearesocial.com/blog/2018/01/global-digital-report-2018

Kirkpatrick, A., Connor, J. O., Campbell, A., Cooper, M., Caldwell, B., Reid, L. G., Vanderheiden, G., Chisholm, W., Slatin, J. and White, J. (2018). Web Content Accessibility Guidelines (WCAG) 2.1. *WWW Consortium (W3C)*. Available at: https://www.w3.org/TR/WCAG21/

Mahmud, P. B. (2016). Um framework para apoiar o desenvolvimento de aplicações on-line acessíveis. São Carlos - SP: Universidade Federal De São Carlos. Available at: https://repositorio.ufscar.br/bitstream/handle/ufscar/8581/DissPBM.pdf?sequence=1&isAllowed=y

MDN. (2020). HTML: A good basis for accessibility. Retrieved from https://developer.mozilla.org/en-US/docs/Learn/Accessibility/HTML

Mueller, M. J., Jolly, R., Eggert, E., Brewer, J., Henry, S. L. and Sutton, J. (2018). Web Accessibility Laws & Policies. *WWW Consortium (W3C)*. Available at: https://www.w3.org/WAI/policies/

Peri, R. S. (2020). Accessible UI Component Libraries Roundup. Retrieved from https://www.digitala11y.com/accessible-ui-component-libraries-roundup/

PUXL framework. (2020). Build the Accessible Web with PUXL Framework. Retrieved from https://puxl.io

Resnikoff, S., Pascolini, D., Etya'Ale, D., Kocur, I., Pararajasegaram, R., Pokharel, G. P., & Mariotti, S. P. (2004). Global data on visual impairment in the year 2002. *Bulletin of the world health organization, 82*, 844-851. Available at: https://apps.who.int/iris/bitstream/handle/10665/263950/PMC2486591.pdf?sequence=1&isAllowed=y

Scheuhammer, J., Cooper, M., Snow-Weaver, A. and Leventhal, A. (2014). WAI-ARIA 1.0 User Agent Implementation Guide. A user agent developer's guide to understanding and implementing Accessible Rich Internet Applications. *WWW Consortium (W3C)*. Available at: https://www.w3.org/TR/wai-aria-implementation/

Thylefors, B., Negrel, A. D., Pararajasegaram, R., & Dadzie, K. Y. (1995). Global data on blindness. *Bulletin of the world health organization, 73*(1), 115. Available at: https://apps.who.int/iris/bitstream/handle/10665/263950/PMC2486591.pdf?sequence=1&isAllowed=y

Turretcss. (2020). Turretcss - a responsive front-end framework for accessible and semantic websites. Retrieved from https://turretcss.com/

U.S. Government. (2018). Section508.gov About US. Retrieved from: https://www.section508.gov/about-us

United Nations. (2006). Convention on the Rights of Persons with Disabilities (CRPD). New York: United Nations Headquarters. Retrieved from https://www.un.org/development/desa/disabilities/convention-on-the-rights-of-persons-with-disabilities.html

United Nations. (2013a). Accessibility and Development – Mainstreaming disability in the post-2015 development agenda. New York: United Nations Headquarters. Retrieved from https://www.un.org/disabilities/documents/accessibility_and_development.pdf

United Nations. (2013b). Transforming our world: the 2030 agenda for sustainable development. New York: United Nations Headquarters. Retrieved from https://sustainabledevelopment.un.org/content/documents/21252030%20Agenda%20for%20Sustainable%20Development%20web.pdf

World Health Organization. (2012). *Global data on visual impairments 2010*. Geneva, Switzerland: World Health Organization. Available at: https://www.who.int/blindness/GLOBALDATAFINALforweb.pdf

World Health Organization. (2020a). Disability and Health. Retrieved from https://www.who.int/en/news-room/fact-sheets/detail/disability-and-health

World Health Organization. (2020b). World report on vision. Geneva: World Health Organization. Retrieved from https://www.who.int/publications/i/item/world-report-on-vision

World Health Organization. (2020c). Global Cooperation on Assistive Technology (GATE). Retrieved from https://www.who.int/disabilities/technology/gate/en/

World Health Organization., & World Bank. (2011). World report on disability. Geneva, Switzerland: World Health Organization. Retrieved from https://www.who.int/publications/i/item/world-report-on-disability

Zahra, S. A. and Brewer, J. (2017). Tools and Techniques. In How People with Disabilities Use the Web. *WWW Consortium (W3C)*. Available at: https://www.w3.org/WAI/people-use-web/tools-techniques/