Improving accessibility of CMS-based websites using automated methods

Balázs Csontos¹ · István Heckl¹

Published online: 30 November 2020
© The Author(s) 2020

Abstract

New methods of identifying and fixing accessibility issues on websites are presented in this article. The websites taken into consideration by the research are created with content management systems (like WordPress or Joomla!). Our main goal was to develop different methods to improve accessibility that may be used by various user groups (website creators, operators, content editors). Some of our methods are easy to use, some need more proficiency. The three methods we have developed (CSS/SCSS class override, MVC-based extension override, HTML output override) are described in detail. The use of an already existing method (Data entry checking) is also introduced, as well as some development options of this method. Each method is introduced in general, furthermore an example of their usage is also presented. Using the proposed methods, websites can fulfil the recommendations of the Web Content Accessibility Guidelines (WCAG) in order to make the content of the websites more accessible.

Keywords Web accessibility · Accessibility methods · WCAG · Content management systems · WordPress · Joomla!

1 Introduction

Nowadays, the Internet has become an integral part of people's everyday life. People use it for communication, getting information about worldwide news and events, moreover, they also use it to deal with their banking and other official affairs. It is important that people with disabilities are not excluded from the World Wide Web. This is why the World Wide Web Consortium (W3C) [1], the main international standardization organization on the Internet, has created the Web Content Accessibility Guidelines (WCAG) [2]. WCAG cover a wide range of recommendations to make web content accessible to people with disabilities, including blindness and low vision, deafness and hearing loss, learning disabilities, cognitive limitations, limited movement, speech disabilities, photosensitivity, and combinations of these.

Following WCAG will also result in web content becoming more usable to users in general. The guide has been evolving over the years and the current version in 2020 is WCAG 2.1 [3]. The WCAG standard defines three levels of accessibility. Level A is the basic, which has to be fulfilled by any accessible website. Level AA means a higher-level availability and accessibility. Typically, level AA is recommended, especially for public interest, state, and municipality websites. The highest level is Level AAA, which sets really high requirements to both website creators and operators. Level AAA is required of those who operate websites that are (or will be) visited by a significant number of disabled or handicapped people. However, W3C does not recommend Level AAA as a general policy for entire sites because it is extremely difficult to satisfy all Level AAA success criteria for some content.

After the introduction of WCAG, the W3C introduced the Authoring Tool Accessibility Guidelines (ATAG) [4], which provides guidelines for designing web content authoring tools that are both more accessible to authors with disabilities (Part A) and designed to enable, support, and promote the production of more accessible web content by all authors (Part B). In order to ensure that the process of using ATAG 2.0 and WCAG 2.0 together in the development of authoring tools is as simple as possible, ATAG 2.0 shares WCAG

✉ Balázs Csontos
csontos@dcs.uni-pannon.hu
István Heckl
heckl@dcs.uni-pannon.hu

¹ Department of Computer Science and Systems Technology, University of Pannonia, Egyetem u. 10, Veszprém 8200, Hungary

2.0's three level conformance model: Level A (lowest), AA (middle), AAA (highest).

Following the private sector, an increasing number of government services has become available on the web, therefore it is increasingly important to make the website content of public sector bodies more accessible. In order to facilitate this process, the European Union obliges its member states to make sure that, after 23 September 2021, all public websites and mobile applications are accessible [5]. In practice, this means that public websites must comply with [6] the level AA recommendations of WCAG 2.1.

Websites have been made up of a static HTML pages for a long time. Nowadays, web developers often use various toolkits and frameworks that are created using sophisticated development tools [7]. However, the amount of development techniques available is huge, furthermore the rapid changes in technology makes it difficult to follow developments [8]. While some developers are committed to incorporating accessibility guidelines in their solutions, some experiences [9] show that the intention and the actual implementation are two different things.

At present, more than 60% of websites on the Internet [10] are running on a content management system (CMS) [11] and this rate is steadily increasing. These systems are especially useful. They allow anyone to create websites and place various multimedia contents on them, without advanced programming knowledge. Their responsive design (e.g., using the Bootstrap framework) allows content on the website to fit properly on the display screen (e.g., widescreen monitor, tablet, mobile phone). In practice, this means that when the page is loaded, the framework checks the width of the display device used by the user, and then automatically selects and loads the associated page structure. Assigning different website structures to devices with different width allows users to properly access the information on the website. The responsive function of websites is useful in situations where a user with a more limited field of vision needs to increase the page's zoom level. Increasing the magnification may reorder the structure of the website, which is mapped to even smaller width screens, while magnifying multimedia content [12]. The combined use of a CMS and a responsive framework has a positive impact on accessibility [13].

Currently, the three most popular CMSs are WordPress [14], Joomla! [15], and Drupal [16]. These systems involve several risks from an accessibility point of view. Firstly, not all CMSs provide adequate support for the accessibility of websites. This is especially true for content input editors as well as for themes (templates) that display information on websites, see [17]. Secondly, the basic services of content management systems can be extended with the help of a number of extensions developed by third-party developers. A lot of times these developers do not focus on accessibility.

Thirdly, in many cases, the creation and editing of content is done by less trained users, consequently in the absence of proper IT training and experience they can use formatting that is not accessible. The main motivation of our research was to develop methods to improve the accessibility of websites created by CMS.

The remainder of this paper is structured as follows: Sect. 2 outlines an analysis of related works on the topic of accessibility. Section 3 describes how to check the accessibility of websites using an online tool. Section 4 presents four accessibility methods in general, and specific examples are also given. Finally, Sect. 5 concludes the paper.

2 Related works

So far, many research works have addressed the topic of web accessibility. However, in many cases the focus was only on detecting accessibility issues rather than correcting them. There are very few publications that accurately define the technology and tools, which can be utilized to address the accessibility problems encountered on websites in content management systems.

In some cases, the authors only conducted a web accessibility check, and no suggestions were made to fix the errors. Karaim and Inal [18] have examined 10 Libyan government websites for accessibility in accordance with the criteria of WCAG 2.0 [19], and one more detailed usability test was conducted using AChecker [20] and TAW [21] tools. The results showed that each website failed the accessibility test, and the website tested for usability had a significant number of problems. Verkijika et al. [22], examined 26 South African university websites based on accessibility considerations. The research revealed that the websites failed to pass an average of eight Level A WCAG 2.0 accessibility criteria. In addition, all of the websites contained malicious links, and four of them did not complete the Google Mobile-Friendly Test [23]. Ismailova and Inal [24] analysed the accessibility of 60 top Kyrgyz, Kazakh, Azerbaijani, and Turkish university websites. The survey revealed that most of the university websites did not pass the accessibility criteria of WCAG 2.0. Only two Kyrgyz and two Kazakh websites reached level A and only one Kyrgyz and two Kazakhs reached level AAA. Although the errors were grouped by level of adequacy and by country, no solutions were proposed to correct them.

In other cases, the research not only tested website accessibility but also proposed improvements. Ismail et al. [25] have performed accessibility tests of 59 Portuguese higher education websites, using various online evaluation tools. Based on the results, it was found that none of the educational websites are accessible. The authors categorized the errors and made suggestions for improvements. Csontos and

Heckl [26] have examined 25 Hungarian government websites based on usability, accessibility, and security aspects. The research revealed that none of the 25 websites fully complied with the WCAG standards. The authors systematized the results, and then made classifications of the errors based on different aspects. In the final phase of the research, they came up with proposals in order to improve the usability, accessibility, and security of these websites. Sik-Lanyi and Orban-Mihalyko [27] analysed 48 health-related websites in Eastern Europe and 51 in Western and Northern Europe based on accessibility criteria. Their research used AChecker and Nibbler [28] tools as well as user feedback questionnaires evaluated by a group of experts. The results showed that most of the websites did not fulfil accessibility guidelines. In order to improve accessibility, the authors formulated specific suggestions.

Previously, a number of automated accessibility repair methods was developed, however these were not CMS-specific. Di Lucca et al. [29] proposed heuristic techniques for automated reordering of website elements to reduce page load time based on structural analysis and summary. Bigham et al. [30] presented a repair tool that automatically creates and inserts alternative text into websites on the fly. Keyser et al. [31] introduced a proxy-based solution that automatically adds alt tags based on analysis of image contents. Jasselette et al. [32] developed a tool, which allows the repair of HTML source code of a page with user interaction. Their solution is not fully automatic, though the tool can repair every problem that was detected during the evaluation process. PHP web applications regularly generate invalid HTML codes. Modern browsers can fix HTML errors in the background, but faulty pages can occasionally cause browser crashes or create vulnerabilities. Samimi et al. [33] presented two repair tools, which automatically improve the generated HTML code in PHP web applications.

Only a very small part of the publications focuses on fixing accessibility problems in content management systems. López et al. [34] have investigated six CMS systems (Joomla! [15], Drupal [35], eZ Publish [36], OpenCMS [37], Plone [38] and Typo3 [39]) based on accessibility criteria. It was found that none of the CMS had fulfilled the level A of ATAG 1.0 [40]. One of the most important findings of the authors is that the "What You See Is What You Get" (WYSIWYG) editors [41] in content management systems do not support the creation of accessible content. On the other hand, CMS templates [42] which are responsible for displaying information, are not always accessible. They suggest replacing or modifying these extensions, but they do not provide a clear methodological approach to eliminate the problems. In a later study [43], López et al. have presented a method for identifying and correcting accessibility issues for websites in content management systems. The authors described the steps they developed in detail in the OpenCMS

and Typo3 environments, and then submitted suggestions for improvements. Rodríguez et al. [44] presented a methodology for assessing accessibility and usability for Open Course Ware (OCW) websites [45], as well as a framework to further enhance the accessibility and usability of these OCW websites. Calvo et al. [46] have examined one of the most popular Learning Management Systems (LMS) [47], Moodle [48], from the perspective of visually impaired users. The research has shown that users using screen reader software encountered a number of obstacles when using the system. For example, visually impaired users were unable to publish content without help. Based on the results, the authors made detailed suggestions on how to overcome these problems.

3 Web accessibility testing with automated tools

Nowadays, websites have become too big and complex for manual verification. As a result, a number of tools has become available that support the accessibility checking process in an automated way. The use of free and automated tools is very attractive as they are easy to use and provide fast results. Yet there are several limitations to these tools. Automated testing programs can identify items as false positive or false negative as they disregard the different disabilities or abilities of people with similar disabilities. These programs also fail to address usability or functionality problems, and they cannot handle accessibility issues that may only be identified by humans, such as keyboard access, meaningful alt texts. Automated software has at least 30% error rate. Therefore, automated tools can lead to the false impression that if the rating is good, then it is a completely accessible website [49]. Even though automated tools do not offer a complete solution, they are an essential part of any serious accessibility evaluation effort.

At present, there are number of online tools [50] for checking the accessibility of websites. One of the most versatile is the WAVE [51] online checker. Unlike other tools, WAVE not only indicates the problem at source code level, but also highlights it graphically on the original website. WAVE is developed by WebAIM [52], a non-profit organization. Due to the continuous development of the software, an increasing number of WCAG 2.1 [3] requirements can be checked with its help. For the most popular browsers, the tool is also available as a browser plugin. WAVE's limitations include being able to test only one page at a time and not the entire website and the results are not exportable. The tool cannot check all the issues in WCAG 2.1—no automated tool can.

There are two ways to use WAVE: website or plugin. In the first case, the address of the website has to be entered in the input box in the middle, and then, the scanning can be

started. In the second case, the browser-specific (Chrome, Firefox) WAVE plugin must be installed first. After installation, the gray WAVE button located on the right side of the browser can be used. After pressing the WAVE button, see Fig. 1, the detected problems are summarized on the left, while on the main panel, those website elements are graphically highlighted which do not fulfil the WCAG guidelines. By pressing the `<code>` button at the bottom, the checker tool displays the source code of the examined website. After clicking on an error, WAVE details the problem and then jumps to the source code to be corrected in the source code window. Based on the obtained information, the repair of the website can be started after selecting one of the proposed method from the following section.

4 Methods to repair accessibility problems

The WCAG guidelines and success criteria are categorized according to four principles that provide the foundation for web accessibility: perceivable, operable, understandable, and robust. If one of these principles is not fulfilled, the disadvantaged users will not be able to use the web. Under each of the principles are guidelines and success criteria that help to address these principles for people with disabilities.

In our previous study [26], we examined the accessibility of 25 Hungarian websites of public sector bodies. The five most common errors identified are shown in Table 1. Based on the survey, almost all websites contained empty links (does not describe the functionality and/or target of that link), more than half of the websites contained images that did not have alternative text, contained a form element without item description, or a button without explanation.

Usually, no one claims ownership of the problem. The European Union has created a directive on web accessibility

Fig. 1 Accessibility test with WAVE checker

Table 1 The five most common WCAG 2.0 errors

WCAG 2.0 problems	Problem description	Standards and guidelines	Number of websites
Empty link	A link contains no text	2.4.4 Link Purpose (In Context) (Level A)	21 (84%)
Linked image missing alternative text	An image without alternative text results in an empty link	1.1.1 Non-text Content (Level A) 2.4.4 Link Purpose (In Context) (Level A)	16 (64%)
Missing form label	A form control does not have a corresponding label	1.1.1 Non-text Content (Level A) 1.3.1 Info and Relationships (Level A) 2.4.6 Headings and Labels (Level AA) 3.3.2 Labels or Instructions (Level A)	15 (60%)
Empty button	A button is empty or has no value text	1.1.1 Non-text Content (Level A) 2.4.4 Link Purpose (In Context) (Level A)	14 (56%)
Missing alternative text	Image alternative text is not present	1.1.1 Non-text Content (Level A)	8 (32%)

[5] and obliged member states to implement it. Although member states have enacted it (as it is mandatory), there are no comprehensive plans to support and control the implementation. Website administrators usually think they are only responsible for the website operation. Even if the content creator has read the law, he or she does not have enough IT and accessibility knowledge to make the website accessible. As local governments in many cases are unable to provide resources for content editors, we believe that launching free nationwide training programs could reduce the problem.

Another problem is that CMS-based websites are very complex. In many cases, if a problem is detected, it is difficult to indicate exactly how it can be fixed. There are relatively few automated solutions to simplify problem correction. These issues have prompted us to develop new methods to improve website accessibility.

Examining a variety of technologies, we suggest four methods to tackle accessibility problems in content management system-based websites. Three of these methods are proposed by the authors and one is an already existing method. These methods are shown in Table 2.

Unlike in static HTML pages, in content management systems the structural elements of the website (base system,

extensions) and the uploaded content are separated. Consequently, the issues of page structure and content can be corrected differently. From this point of view the methods can be classified into three types:

- **Content repair:** A method for improving the formatting of multimedia content on the website editor interface;
- **Source code repair:** A method for improving the structural elements (source code) of a website;
- **Content and source code repair:** Website content and source code repair method at once.

Some of the proposed methods are easy to use, others require advanced knowledge. To help the decision between the methods the difficulty level of each method is given, which we have created based on our subjective experience:

- **Beginner:** We recommend using this method for novice users in accessibility. It requires basic IT skills, e.g., beginner HTML programming and CMS administrator knowledge;
- **Intermediate:** We recommend using this method for users who already know how to build websites and how

Table 2 Four methods to tackle accessibility problems in CMS-based websites

Name of method	Type	Difficulty level	Repair limitations	CMS limitations
Data entry checking	Content repair	Beginner	Only the problems defined in the plugin can be repaired	Where some data entry checking plugin is available, e.g., WordPress, Joomla!
CSS/SCSS class override	Content repair	Intermediate	Only style definition-based problems can be repaired	No limitations
HTML output override	Content and source code repair	Advanced	The maintenance of the rules can be time consuming	Where some HTML output override plugin is available, e.g., WordPress, Joomla!
MVC-based extension override	Source code repair	Developer	Only MVC-based extensions can be repaired	Only to be used in a CMS that supports the MVC design pattern, e.g., Joomla!, Drupal

to format content on websites. In addition to the requirements of the previous level, CSS/SCSS and HTML programming skills are needed here;

- **Advanced:** We recommend using this method for users who can confidently interpret the source code that builds websites. To use it, advanced HTML programming skills beyond the previous level are needed;
- **Developer:** We recommend using this method specifically for users who are familiar with web development. In addition to the requirements of the previous levels, developers need PHP and CMS programming skills.

The proposed development methods have different repair and CMS limits, which will be described in detail when the methods are presented. Concrete examples will be shown for each method.

4.1 Repair using data entry checking

4.1.1 Description of the method

The “Data entry checking” method is based on the principle that the user is warned during the creation of the content if this content uses formatting which does not properly comply with WCAG recommendations [3]. This method is not our own result, but for the sake of completeness, it is useful to present it here. In today’s content management systems, the actual content is usually entered using so-called WYSIWYG editors [41]. Like in the case of a desktop text editor, a WYSIWYG editor makes it easy to create multimedia content on websites. The information (text, image, video, etc.) is converted into HTML by the editor, so users do not need to know the HTML language. The default editor in WordPress is Gutenberg [53], while in Joomla! it is TinyMCE [54]. TinyMCE is also available as a WordPress plugin [55]. Fortunately, the aforementioned WYSIWYG editors are modular, so they can be extended by an accessibility checker plugin.

For example, *a11ychecker* developed by Instructure [56] is a TinyMCE plugin that enables automatic detection and easy repair of various WCAG accessibility issues. The Accessibility Checker [57] plugin developed by CKSource [58] for CKEditor [59] is also similar in functionality. The *a11ychecker* plugin is available under MIT free software license from the developer GitHub storage [60]. It is important to note that although the accessibility plugin is free, integration with the TinyMCE editor requires a TinyCloud subscription [61]. The integration for WordPress is detailed in [62]. After installation, pressing the “Check Accessibility” button on the right side of the editor starts checking the created content. Then, on the right, a pop-up window displays the detected problems, their descriptions, and the repair options. The *a11ychecker* has a number of

accessibility rules that have been implemented in JavaScript programming language. The default rules are the following:

- **Usage of paragraphs as headings:** This rule verifies that *h1–h6* tags are used instead of *p* tags;
- **Sequential headings:** This rule checks whether the headings are used sequentially (in ascending order), e.g., whether the header of *h2* is preceded by the header *h1*;
- **Adjacent links:** This rule verifies that adjacent links do not have the same *href* tag. For example, if there is an image link and a text link with the same *href*, they should be in the same element, not in two different elements;
- **Ordered list structure:** This rule checks to see if *ol* element is used for sorted lists. It is important not to use Arabic or Roman numeric paragraphs instead of *ol* elements containing *li* elements;
- **Unordered list structure:** This rule checks if *ul* element is used for unordered lists. It is important not to use * or— or paragraphs with similar characters instead of *ol* elements containing *li* elements;
- **Contrast ratio of the text:** This rule verifies that the contrast ratio of the text exceeds the specified values (5:1 for normal text and 3:1 for large text). Low contrast text is difficult to read, especially for visually impaired users;
- **Image ALT text:** This rule checks whether all images have *alt text* attribute;
- **Alt text filename:** This rule verifies that the *alt text* attribute of the image does not match the image file name;
- **Table caption:** This rule verifies that each table element has a caption that describes the data in the table;
- **Complex table summary:** This rule checks that all complex tables have a summary attribute explaining the assistive technologies how to navigate through the data inside of the table;
- **Table caption and summary:** This rule verifies that the caption of the table and the summary do not have the same text content;
- **Table markup:** This rule checks that each table contains both *td* and *th* elements;
- **Table headers:** This rule checks whether each table element contains at least one header—*th*—element;
- **Table heading scope:** This rule verifies that all tables contain an attribute that describes where the table header is located.

The data entry checking method is simple to use. After starting the plugin, it can detect the error when the WYSIWYG editor is used. It suggests a solution to the error and then jumps to the next error. Its disadvantage is that checking existing content is very time-consuming as pages have to be opened one by one. Another disadvantage is that it can only detect errors which have corresponding rules. However,

since the plugin is open source, it is possible to create new rules with JavaScript knowledge. We have also created a number of new rules, for example, one that verifies that the inserted link has the <alt> attribute (which describes the functionality and/or target of the link). If the link does not contain the attribute, then the system offers to fix the problem. It is definitely recommended to use the data entry checking method for new content.

4.1.2 Example of use

To illustrate the use of this method, an incorrectly formatted content, in this case an image without an *alt* tag, has been inserted into the editor. The original source code is shown in Fig. 2.

```
<p>

</p>
```

Fig. 2 An incomplete source code where an image without an alt tag has been inserted

When a llychecker is used, the plugin detects the faulty HTML code and then proposes a fix. In this case, an appropriate alt tag needs to be added to the image (field below the “add alt text for the image” text) to make it accessible. The process itself can be seen in Fig. 3.

After the *alt* tag is inserted, the fix will be visible in the source code. Figure 4 shows the corrected source code.

4.2 Repair with CSS/SCSS class override

4.2.1 Description of the method

In web programming, the HTML markup language defines the structure of a website and the CSS (and its extension the SCSS) style description programming language describes

```
<p>

</p>
```

Fig. 4 The improved source code where the inserted image already has an alt tag

Fig. 3 The use of the a11ychecker plugin

the formatting of a website. The essence of the second method is to automatically correct those CSS/SCSS codes which do not fulfil the requirements of WCAG [3]. The advantage of this solution is that if multiple websites use the same theme framework [63] (and same CSS styles), then the repair rule can be easily applied to all of these websites. This method can be implemented in the most popular content management systems.

In CMS, the layout of the user interface is defined by so-called themes (templates). A theme consists of a fixed or dynamic number of positions whose location is determined by the theme. In administration, different widgets (modules) can be assigned for these positions. Many frameworks use this approach. One such framework is the Gantry theme framework developed by Rockettheme [64]. It allows users to create websites with a dynamic and responsive layout. The use of the second accessibility method is illustrated by Gantry, but the method also works for other or the default CMS template framework.

The use of the Gantry framework offers the ability to create any number of top level containers on the website, in which it is possible to create any number of particles (which can be widgets, widget positions, page content or the framework's own building blocks) using "drag and drop". Unique

names can be given to the containers and a particle can be assigned to each position. The Page Content particle displays the actual web content. Figure 5 shows the layout of an example website.

The HTML structure of the example website can be seen in Fig. 6. The first line of the source code shows the Header (g-header) container, in which three widget positions (header1, header2, header3) were defined, to which a Custom HTML widget was finally assigned.

```
<header id="g-header"> == $0
  ::before
  <div class="g-container">
    <div class="g-grid">
      <div id="header1" class="g-block size-28">
        <div class="g-content">...</div>
      </div>
      <div id="header2" class="g-block size-44">...</div>
      <div id="header3" class="g-block size-28">...</div>
    </div>
  </div>
</header>
```

Fig. 6 HTML structure of a Gantry framework-based website

Fig. 5 Implementing the layout of a website in the Gantry framework

When the Gantry framework is used, the information displayed in the widget positions will appear within a g-content child class. Because all containers have a unique identifier, a formatting rule for the content displayed here can be written. For example, if all text information in the header1 widget of the header section is to appear in red, the CSS code in Fig. 7 should be used.

The accessibility method is based on defining formatting constraints for each container. These new constraints override improper user formatting, so the website becomes accessible. The override can happen because the CSS rules have an evaluation order, and the formatting constraint defined for the container is of higher priority than the user formatting. Using this method can improve many accessibility issues, including font size problems, line height issues, text align problems, content visually issues, contrast ratio issues, color problems, focus issues, grid and flat document structure problems, and so on [65].

4.2.2 Example of use

Let us assume a user entered text content through the editor that uses a justified formatting in WordPress. This is bad practice from an accessibility point of view, because the variable-width spaces appearing between words form so-called “rivers of white” [66], which can make reading difficult for people with dyslexia. The incorrectly formatted text in a Gantry framework-based template will appear within the g-content class, so an override rule for g-content or for one of its ancestors should be defined. The override will change the justified formatting without changing the source code entered by the user.

```
#g-header #header1 .g-content{
  color: #FF0000;
}
```

Fig. 7 Defining a formatting rule for text information appearing in the header1 widget of the g-header section

Figure 8 shows an example in which justify formatting is defined for the *p* element (style attribute). It is important to note that this kind of formatting occurs in most cases when the user formats the content with the help of the editor in the content manager. This is important because these formatting properties have the greatest precedence in the order of evaluation of the CSS standard. To override this type of formatting a special attribute is needed. The *!important* attribute can override any style definition regardless of precedence.

However, there may be cases where formatting with a smaller precedence should be overridden. In such cases, the *!important* attribute is not required. If more than one style definition of the same precedence is defined for a particular class, the last style definition will be valid. The Gantry architecture makes sure that the style definitions defined in the framework are always the last to be processed.

If content in a particular widget position needs to be overridden, a unique identifier or class that is the ancestor of the content in question must be searched for. In this case, header1 identifier has been selected. If an override rule that applies to content in all positions globally is needed, the override should be defined for the g-content class.

The next step is to create the custom.scss file in which the override rules are defined. The custom.scss file must be placed in the THEME_DIR/custom/scss folder. The next step is to add the override rule for the selected class in the custom.scss file, as shown in Fig. 9. It defines left alignment for the g-content in the header1 section.

Figure 10 shows the html code after reloading the website. It can be seen that although the given *p* element has a justified formatting originally, the crossed-out justified line on the right side of the figure, it is overridden using the accessibility rule defined above.

```
#header1 .g-content p{
  text-align: left !important;
}
```

Fig. 9 CSS repair rule to avoid justified text

Fig. 8 The HTML source code of the example text, which contains justified formatting

```
<div id="header1" class="g-block size-28">
  <div class="g-content">
    <div id="custom_html-2" class="widget_text widget widget_custom_html">
      <h3 class="widgettitle g-title">Lorem Ipsum</h3>
      <div class="textwidget custom-html-widget">
        <p style="text-align: justify;"> == $0
        "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec ultricies
        nisl congue blandit sit amet id ligula. Praesent et nulla eu augue tempus
        vestibulum sapien egestas. Curabitur ut lectus tortor."
      </p>
    </div>
  </div>
</div>
```

Fig. 10 The HTML source code of the repaired page

Adding new rules can further increase the accessibility of the website. If the same template framework is used on another website, then the rules can be applied on them as well. This method is more general than the data entry checking method. It is not necessary to check all content items one by one manually. Instead a repair rule is defined which is applied automatically to each content in the specified section.

4.3 Repair with MVC-based extension override

4.3.1 Description of the method

Popular content management systems use design patterns. One important pattern is the Model View Controller (MVC) pattern [67]. When using MVC, data management (model) and user interface (view) are separated so that they do not affect each other's operation. Because the CMS are modular, MVC is a great help to create new extensions (e.g., in a web store products can be displayed in multiple layouts). Problems arise when a developer creates a view for an extension (plugin) that does not fulfil WCAG requirements. The third accessibility repair method improves the faulty view files.

First the website should be checked with an online checker (e.g., WAVE) and the source code of the section with the error should be studied in order to identify which extension is using inadequate formatting for accessibility. The unique classes or identifier names in the source code of the page can help identify the extension. Even the template layout manager can help find the needed extension based on the location of the error. After the identification of the extension, the original file responsible for the rendering has to be copied to the appropriate folder. This so-called instantiation operation can be performed automatically in CMS administration. The extension in question has to be selected, after that the system can create the appropriate folder structure and copy the selected files. Following this operation, the newly created files must be modified to fix the accessibility issues marked by WAVE. After the necessary fixes are completed, the online checker should run again to check the improved interface. This process must be repeated until the page becomes error free. Using this method can improve many accessibility issues, for example replacing the function description of various web elements (buttons, forms, menus), correcting missing or unordered headings (*H1–H6*), replacing header (*<th>*) cells in tables, and so on.

4.3.2 Example of use

To illustrate the use of this method, Virtuemart [68] has been selected as one of the most popular webshop extensions for Joomla!. With the Virtuemart Products module, users can display the products in the webshop in a number of ways, but the module is not accessible. After verification, WAVE detected in the Products module (version 4.6.10) 4 errors and 1 warning, see Fig. 11 and Table 3.

Following the source code, it turned out that two sublayout files in the module (addtocartbar.php, customfield.php) contain formatting that does not fulfil WCAG requirements.

Fig. 11 The graphical result of the accessibility test of Virtuemart Product extension

Virtuemart does not consistently follow Joomla!'s MVC design pattern, so in this case, the files above cannot be automatically instantiated. The two files are copied manually to `TEMPLATE_NAME/html/com_virtuemart/sublayouts`. The necessary changes are implemented in the new files.

The methodology for fixing the problems is very similar, so only one detailed description is presented. Product quantity control buttons ($\pm$) are not provided with a field that describes their operation. Originally, the quantity selection buttons were implemented as shown in Fig. 12.

In this case, the repair was implemented by adding aria-label attribute to the buttons which defines the event that will occur when the button is pressed. The result of the improved source code is shown in Fig. 13.

For multilingual pages, this solution is not applicable because the value of the aria-label attribute will always be English. There is the option to define php variables for the values of the attributes. For each language, different text can be associated for these PHP variables in the Joomla! Languages component. The result of the extended source code supporting multilingualism can be seen in Fig. 14.

If a fix is created, it can be used on additional websites. Only the modified files have to be copied to the proper folder. A fix will work until the original view file of the plugin does not change significantly. In such cases, the repair process must be performed again. If an accessibility fix of any extension is developed, it is recommended that it should be sent to the developer of the extension in order to integrate it into the original source code.

4.4 Repair with HTML output override

4.4.1 Description of the method

The fourth accessibility improvement method is based on overriding HTML output, which can help improve both content and website structural issues. It is similar to the operation of the automatic spelling correction of a word processor. If the checking engine detects an accessibility problem, it is changed on the fly. In content management systems, when a user wants to access a page written in, for example, PHP, the server first processes the PHP instructions and sends the finished HTML output to the browser. By overriding this output, practically any accessibility issue can be fixed.

There are many HTML code override extensions available [69, 70] for content management systems. For Joomla! one of the most popular is the ReReplacer [71] developed by Peter van Westen (Regular Labs) [72], and the Real-Time Find and Replace [73] plugin developed by Marios Alexandrou for WordPress. Both of these extensions are free, but they also have a pro version where more sophisticated filter and replace rules can be applied.

Table 3 Error list of the Virtuemart Product extension's accessibility test

ID	WCAG 2.1 problems	Problem description	Standards and guidelines
1	Redundant link	The link to the product image and product name will point to the same link but in two separate <code><a></code> tags	2.4.4 Link Purpose (In Context) (Level A)
2	Missing form label	The item does not have a field that describes the operation of the product size selection. In this case, the size of the item to be purchased can be selected by pressing the drop-down button	1.1.1 Non-text Content (Level A) 1.3.1 Info and Relationships (Level A) 2.4.6 Headings and Labels (Level AA) 3.3.2 Labels or Instructions (Level A)
3	Missing form label	The item does not have a field that describes the operation of the product quantity field. In this case, the field indicates the amount of goods to be purchased	1.1.1 Non-text Content (Level A) 1.3.1 Info and Relationships (Level A) 2.4.6 Headings and Labels (Level AA) 3.3.2 Labels or Instructions (Level A)
4	Empty button	The item does not have a field that describes its operation (product + button). The amount of goods to buy can be increased by pressing the button	1.1.1 Non-text Content (Level A) 2.4.4 Link Purpose (In Context) (Level A)
5	Empty button	The item does not have a field that describes its operation (product — button). The amount of goods to buy can be decreased by pressing the button	1.1.1 Non-text Content (Level A) 2.4.4 Link Purpose (In Context) (Level A)

Fig. 12 The source code for the incompletely defined buttons in the Virtuemart Products module

```

107
108
109
110
111
112

```

```

<?php if ($product->orderable) { ?>
    <span class="quantity-controls js-recalculate">
        <input type="button" class="quantity-controls quantity-plus"/>
        <input type="button" class="quantity-controls quantity-minus"/>
    </span>
<?php }

```

Fig. 13 The improved source code for the buttons in the Virtuemart Products module

```

107
108
109
110
111
112

```

```

<?php if ($product->orderable) { ?>
    <span class="quantity-controls js-recalculate">
        <input type="button" aria-label="Increase product quantity"
        class="quantity-controls quantity-plus"/>
        <input type="button" aria-label="Decrease product quantity"
        class="quantity-controls quantity-minus"/>
    </span>
<?php }

```

Fig. 14 The improved source code for the buttons with language variables in the Virtuemart Products module

```

107
108
109
110
111
112

```

```

<?php if ($product->orderable) { ?>
    <span class="quantity-controls js-recalculate">
        <input type="button" aria-label="<?php echo vmText::_ (
        'COM_VIRTUEMART_CART_QUANTITY_PLUS') ?>" class=
        "quantity-controls quantity-plus"/>
        <input type="button" aria-label="<?php echo vmText::_ (
        'COM_VIRTUEMART_CART_QUANTITY_MINUS') ?>" class=
        "quantity-controls quantity-minus"/>
    </span>
<?php }

```

A typical example of using a replacer extension is when users publish e-mail addresses inadvertently, so they are included in the source code of the website without any protection. These addresses can be collected by malicious scripts and made available to spam lists. To prevent this, an override rule can be created that detects the e-mail addresses and transcribe them into a unique format, so it is much more difficult to collect them automatically.

The fourth method is based on the fact that if an accessibility issue is detected (e.g., with an online verification tool like WAVE), it can be fixed with a replacer extension without modifying the original content. Override rules have to be defined a priori. Each rule contains a pair, the defective and the appropriate source code. Regular expressions can be also used to make an override rule flexible. The replacer extension will be able to replace the faulty source code with

the corrected code according to the rules, thus making the website accessible. Using this method can improve almost all detected accessibility issues by modifying the source code of the website.

4.4.2 Example of use

The use of this method is illustrated by correcting an error found on the authors' University department website [74]. WAVE has detected that the search plugin in the top right corner of the website in Fig. 15 did not contain an item description, so users who use a reader software did not know what function the icon can perform. Here, it is important to note that the search plugin used on the website is not based on MVC design pattern, so it cannot be repaired using the third accessibility improvement method.

To fix this problem, a new override rule has been created in the ReReplacer extension. In ReReplacer each rule has a search part and a replace part. The wrong (original) HTML source code has been copied into the “Search” field, and

the repaired source code has been copied into the “Replace” field. Figure 16 shows, the incorrect source code was completed with an aria-label attribute in the fix. The rule runs every time a page is loaded. With the added attribute, users who use assistive software will now understand the function of the button.

A similar problem has also been fixed with the MVC-based extension override. However, this method could not be used in this case, because the applied search plugin did not use the MVC software design pattern. Fortunately, this is not an obstacle for the HTML output override method. The rules can be exported and can be used at other websites. It is important to mention that there may be fixes that are site-specific. A further disadvantage is that in some cases it is much more difficult to define new override rules (for example, using regular expressions) than to use the CSS/SCSS class override method. Also, it is important to know that override rules are language-specific, so for multilingual pages, the rules must be written for each language. It is easy in such cases to use language variables in the override rules instead of plain text to which the CMS can assign the specified value (in this case the text of the variable).

5 Conclusions

Four methods have been introduced here to make content management system-based websites more accessible. Beyond the step-by-step introduction of accessibility testing, detailed guidelines for using the proposed methods through

Fig. 15 The search function button on the department website

Search

Press F10 to toggle Full Screen editing.

```
1 <a href="#modal-search" data-uk-modal="{center:true}">
```

Replace

Press F10 to toggle Full Screen editing.

```
1 <a href="#modal-search" aria-label="Start the search function" data-uk-modal="{center:true}">
```

Fig. 16 Accessibility problem repair in ReReplacer extension

real examples have been prepared. The methods are organized so that each user can choose from them according to his or her IT skill. The “Data entry checking” method can be used when creating and modifying web content. Users are notified about problems and potential fixes are offered. This method can also be applied by users who are less familiar with IT.

A huge amount of web pages is already available and it is unlikely that it will be checked and corrected one by one. The CSS/SCSS class override method can be used to correct style definition problems on existing pages. The HTML output override method can be used to perform any accessibility fixes but requires advanced HTML and CSS/SCSS knowledge to define the rules. A rule always contains the faulty code and the corrected code. The MVC-based extension override method can be used to make accessibility enhancements for MVC-based extensions. This method is only recommended for users with web development experience. Using the methods described above allows websites to comply with the WCAG guidelines and helps to improve their accessibility.

Acknowledgement We acknowledge the financial support of Széchenyi 2020 under the EFOP-3.6.1-16-2016-00015 and the ÚNKP-20-3 new national excellence program of the ministry for innovation and technology from the source of the national research.

Funding Open access funding provided by University of Pannonia.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. W3C, World Wide Web Consortium (W3C). <https://www.w3.org> (2019). Accessed 04 Jul 2019
2. W3C, Web Content Accessibility Guidelines (WCAG) Overview. <https://www.w3.org/WAI/standards-guidelines/wcag/> (2019). Accessed 04 Jul 2019
3. W3C, Web Content Accessibility Guidelines (WCAG) 2.1. <https://www.w3.org/TR/WCAG21/> (2019). Accessed 19 Jun 2019
4. W3C, “Authoring Tool Accessibility Guidelines (ATAG) 2.0.” [Online]. Available: <https://www.w3.org/TR/ATAG20/> (2020). Accessed 02 Jul 2020
5. European Union, Directive (EU) 2016/2102 of the European parliament and of the council of 26 October 2016 on the accessibility of the websites and mobile applications of public sector bodies, Off. J. Eur. Union, (2016)
6. ETSI, EN 301 549 (V2.1.2): Accessibility requirements for ICT products and services. https://www.etsi.org/deliver/etsi_en/301500_301599/301549/02.01.02_60/en_301549v020102p.pdf (2018). Accessed 26 Nov 2018
7. Casteleyn, S., Garrig'os, I., Maz'on, J.N.: Ten years of rich internet applications. *ACM Trans. Web* **8**(3), 1–46 (2014)
8. Harper, S., Yesilada, Y.: *Web accessibility*. Springer, London (2008)
9. Petrie, H., Badani, A., Bhalla, A., Sex, lies and web accessibility: the use of accessibility logos and statements on e-commerce and financial websites, (2005)
10. W3Techs, Usage of content management systems. https://w3techs.com/technologies/overview/content_management (2019). Accessed 11 Dec 2019
11. Wikipedia, Content management system. https://wikipedia.org/wiki/Content_management_system (2019). Accessed 21 May 2019
12. Rathfux, T., Thöner, J., Kaendl, H., Popp, R.: Combining design-time generation of web-pages with responsive design for improving low-vision accessibility. *Proc. ACM SIGCHI Symp. Eng. Interact. Comput Syst EICS* **18**, 1–7 (2018)
13. Duarte, C., Matos, I., Vicente, J., Salvado, A., Duarte, C.M., Carriço, L.: Development technologies impact in web accessibility. *Proc. 13th Web All Conf. W4A* **16**, 1–4 (2016)
14. WordPress Foundation, WordPress. <https://wordpress.org> (2018). Accessed 12 Dec 2018
15. Open Source Matters Inc., Joomla!. <https://www.joomla.org> (2018). Accessed 12 Dec 2018
16. Buytaert, D.: Drupal. <https://www.drupal.org> (2020). Accessed 25 Jan 2020
17. Farrelly, G.: Practitioner barriers to diffusion and implementation of web accessibility. *Technol. Disabil.* **23**(4), 223–232 (2011)
18. Karaim, N.A., Inal, Y.: Usability and accessibility evaluation of Libyan government websites. *Univers. Access Inf. Soc.* **18**(1), 207–216 (2019)
19. W3C, Web Content Accessibility Guidelines (WCAG) 2.0. <https://www.w3.org/TR/WCAG20/> (2019). Accessed 19 Jun 2019
20. Greg, G., Li, C.Q.: AChecker: Web accessibility checker. <https://achecker.ca> (2019). Accessed 19 Jun 2019
21. CTIC Foundation Technology Centre, TAW: Web accessibility and W3C standardization services. <https://www.tawdis.net> (2019). Accessed 19 Jun 2019
22. Verkijika, S.F., De Wet, L.: Accessibility of South African university websites. *Univers. Access Inf. Soc.* **19**(1), 201–210 (2020)
23. Google, Mobile-Friendly Test - Google Search Console. <https://search.google.com/test/mobile-friendly> (2019). Accessed 19 Jun 2019
24. Ismailova, R., Inal, Y.: Accessibility evaluation of top university websites: a comparative study of Kyrgyzstan, Azerbaijan, Kazakhstan and Turkey. *Univers. Access Inf. Soc.* **17**(2), 437–445 (2018)
25. Ismail, A., Kuppusamy, K.S., Paiva, S.: Accessibility analysis of higher education institution websites of Portugal. *Univers. Access Inf. Soc.* **19**(3), 685–700 (2020)
26. Csontos, B., Heckl, I.: Accessibility, usability, and security evaluation of Hungarian government websites. *Univers. Access Inf. Soc.* (2020). <https://doi.org/10.1007/s10209-020-00716-9>
27. Sik-Lanyi, C., Orbán-Mihálykó, É.: Accessibility testing of European health-related websites. *Arab. J. Sci. Eng.* **44**(11), 9171–9190 (2019)
28. Silktide Ltd., Nibbler. <https://nibbler.silktide.com/> (2020). Accessed 06 Jul 2020
29. Di Lucca, G.A., Fasolino, A.R., Tramontana, P.: Web site accessibility: Identifying and fixing accessibility problems in client page code. In: *Seventh IEEE International Symposium on Web Site Evolution*, pp. 71–78 (2005)

30. Bigham, J.P., Kaminsky, R.S., Ladner, R.E., Danielsson, O.M., Hempton, G.L.: WebInSight: making web images accessible. *Proc. 8th Int. ACM SIGACCESS Conf. Comput. Access. Assets* **06**, 181 (2006)
31. Keyzers, D., Renn, M., Breuel, T.M.: Improving accessibility of html documents bygenerating image-tags in a proxy. *Proc. Int. ACM SIGACCESS Conf. Comput. Access. Assets* **07**, 249 (2007)
32. Jasselette, A., et al.: Automated repair tool for usability and accessibility of web sites. In: *Computer aided design of user interfaces*. pp. 261–272. Dordrecht, Springer (2007)
33. Samimi, H., Schafer, M., Artzi, S., Millstein, T., Tip, F., Hendren, L.: Automated repair of HTML generation errors in PHP applications using string constraint solving. In: *2012 34th International Conference on Software Engineering (ICSE)*, pp. 277–287. (2012)
34. López, J.M., Pascual, A., Masip, L., Granollers, T., Cardet, X.: Influence of web content management systems in web content accessibility. In: *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*. pp. 548–551. (2011)
35. Buytaert, D.: Drupal. <https://www.drupal.org> (2019). Accessed 14 Jun 2019
36. eZ Systems AS, eZ Publish. <https://ez.no> (2019). Accessed 14 Jun 2019
37. Alkacon Software GmbH, OpenCms. <https://wikipedia.org/wiki/OpenCms> (2019). Accessed 14 Jun 2019
38. Plone Foundation, Plone. <https://plone.org> (2020). Accessed 22 Jan 2020
39. TYPO3 Association, TYPO3. <https://typo3.org> (2019). Accessed 14 Jun 2019
40. W3C, Authoring Tool Accessibility Guidelines (ATAG) 1.0. <https://www.w3.org/TR/WAI-AUTOOLS/> (2019). Accessed 14 Jun 2019
41. Wikipedia, WYSIWYG. <https://wikipedia.org/wiki/WYSIWYG> (2019). Accessed 14 Jun 2019
42. Open Source Matters Inc., Template. <https://docs.joomla.org/Template> (2019). Accessed 14 Jun 2019
43. López, J.M., Pascual, A., Menduina, C., Granollers, T.: Methodology for identifying and solving accessibility related issues in web content management system environments. *Proc. Int. Cross Discip. Conf. Web Access. W4A* **12**, 1 (2012)
44. Rodríguez, G., Pérez, J., Cueva, S., Torres, R.: A framework for improving web accessibility and usability of open course ware sites. *Comput. Educ.* **109**, 197–215 (2017)
45. Wikipedia, OpenCourseWare. <https://wikipedia.org/wiki/OpenCourseWare> (2019). Accessed 14 Jun 2019
46. Calvo, R., Iglesias, A., Moreno, L.: Accessibility barriers for users of screen readers in the Moodle learning content management system. *Univers. Access Inf. Soc.* **13**(3), 315–327 (2014)
47. Wikipedia, Learning management system. https://wikipedia.org/wiki/Learning_management_system (2019). Accessed 14 Jun 2019
48. Dougiamas, M.: Moodle. <https://moodle.org/> (2019). Accessed 14 Jun 2019
49. Jaeger, P.T.: Assessing section 508 compliance on federal e-government web sites: a multi-method, user-centered evaluation of accessibility for persons with disabilities. *Gov. Inf. Q.* **23**(2), 169–190 (2006)
50. W3C, Web Accessibility Evaluation Tools List. <https://www.w3.org/WAI/ER/tools/> (2020). Accessed 22 Jan 2020
51. WebAIM, WAVE: web accessibility evaluation tool. <http://wave.webaim.org> (2019) Accessed 11 Dec 2019
52. Center for Persons with Disabilities, WebAIM. <https://webaim.org> (2018) Accessed 26 Nov 2018
53. Gutenberg Team, Gutenberg. <https://wordpress.org/plugins/gutenberg/> (2020). Accessed 20 Apr 2020
54. Ephox Corporation DBA Tiny Technologies Inc., TinyMCE. <https://www.tiny.cloud> (2019). Accessed 17 May 2019
55. A. Ozz, TinyMCE Advanced. <https://wordpress.org/plugins/tinymce-advanced/> (2020). Accessed 22 Jan 2020
56. Instructure Inc., Instructure. <https://www.instructure.com> (2019). Accessed 17 May 2019
57. CKSource sp. z o.o. sp.k., Accessibility Checker. <https://ckeditor.com/cke4/addon/a11ychecker> (2020). Accessed 24 Jan 2020
58. CKSource sp. z o.o. sp.k., CKSource. <https://cksource.com> (2020). Accessed 24 Jan 2020
59. CKSource sp. z o.o. sp.k., CKEditor. <https://ckeditor.com> (2020). Accessed 24 Jan 2020
60. Instructure Inc., TinyMCE Accessibility Checker Plugin (tinymce-a11y-checker). <https://github.com/instructure/tinymce-a11y-checker> (2019). Accessed 17 May 2019
61. Tiny Technologies Inc., TinyMCE subscriptions. <https://www.tiny.cloud/pricing/> (2020) Accessed 07 Jul 2020
62. Tiny Technologies Inc., How to use TinyMCE premium plugins in WordPress 5. <https://www.tiny.cloud/blog/how-to-use-tinymce-premium-plugins-in-wordpress-5/> (2020). Accessed 07 Jul 2020
63. Rockettheme LLC, Gantry Framework. <http://gantry.org> (2019). Accessed 21 May 2019
64. Rockettheme LLC, Rockettheme. <https://rockettheme.com> (2019). Accessed 21 May 2019
65. Matuzovic, M.: Writing CSS with Accessibility in Mind. <https://medium.com/@matuzo/writing-css-with-accessibility-in-mind-8514a0007939> (2020). Accessed 24 Jan 2020
66. Page, T.: Text justification—issues and techniques. <https://accessible-digital-documents.com/blog/justified-text/> (2019). Accessed 20 May 2019
67. Wikipedia, Model-view-controller. <https://wikipedia.org/wiki/Model-view-controller> (2019). Accessed 30 May 2019
68. Milbers, M.: Virtuemart. <https://virtuemart.net> (2019). Accessed 01 Jun 2019
69. Jones, C.: 5 Best Search and Replace WordPress Tools to Fix Content <https://www.cminds.com/5-best-search-replace-wordpress-tools-fix-content-website/> (2020). Accessed 31 Jan 2020
70. Joomla! Extensions Directory, Replace. <https://extensions.joomla.org/category/editing/replace/> (2020). Accessed 31 Jan 2020
71. van Westen, P.: ReReplacer. <https://www.regularlabs.com/extensions/rereplacer> (2019). Accessed 11 Dec 2019
72. van Westen, P.: Regular Labs. <https://www.regularlabs.com> (2019). Accessed 20 May 2019
73. M. Alexandrou, Real-Time Find and Replace. <https://wordpress.org/plugins/real-time-find-and-replace/> (2019). Accessed 11 Dec 2019
74. University of Pannonia, Department of Computer Science and Systems Technology. <https://dcs.uni-pannon.hu> (2019). Accessed 21 Jul 2019

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.