Accessibility Testing Tools and Methodologies

# Introduction

Web Accessibility is like any other any other functionality should be code unit tested and a part of test driven development. It ensures that individuals with disabilities will be able to use the system.

The two phases of testing accessibility:

1. Testing during Development
2. Testing post Development

In both the phases, accessibility evaluation requires both:

1. Automated Testing
2. Manual Testing

There a number of options and tools available for testing accessibility through Development to Delivery. The report consists of comparison of these tools and other available options and some recommendations based on the results and outcomes.

# Testing During Development

While development it is equally critical to keep a check of any possible code violations. While automated tools like grunt tasks are one of the quick steps in checking accessibility through the local front end development environment, parallel manual checks like component level tabbing and screen reading is something that needs to be verified to ensure that such issues are fixed in the primitive stages of development.

A detailed analysis of a few would enable us to understand the goals and how these tools report issues.

Goals:

1. **Check for HTML structure or element level violations in code** – Markup validation is an essential step during development. An invalid or inconsistent markup may fail to use standard device / browser features.
2. **Tabbing Order and Keyboard Navigation** – Individual components / segments fixed for tabbing focus and keyboard can optimize effort of fixing such issues at page level.
3. **WCAG Standard Based Validations / Checklist** – Standard Markup and content guidelines for accessibility need to be validated in the process of Development and post Development as well.

# Testing Tools and Techniques

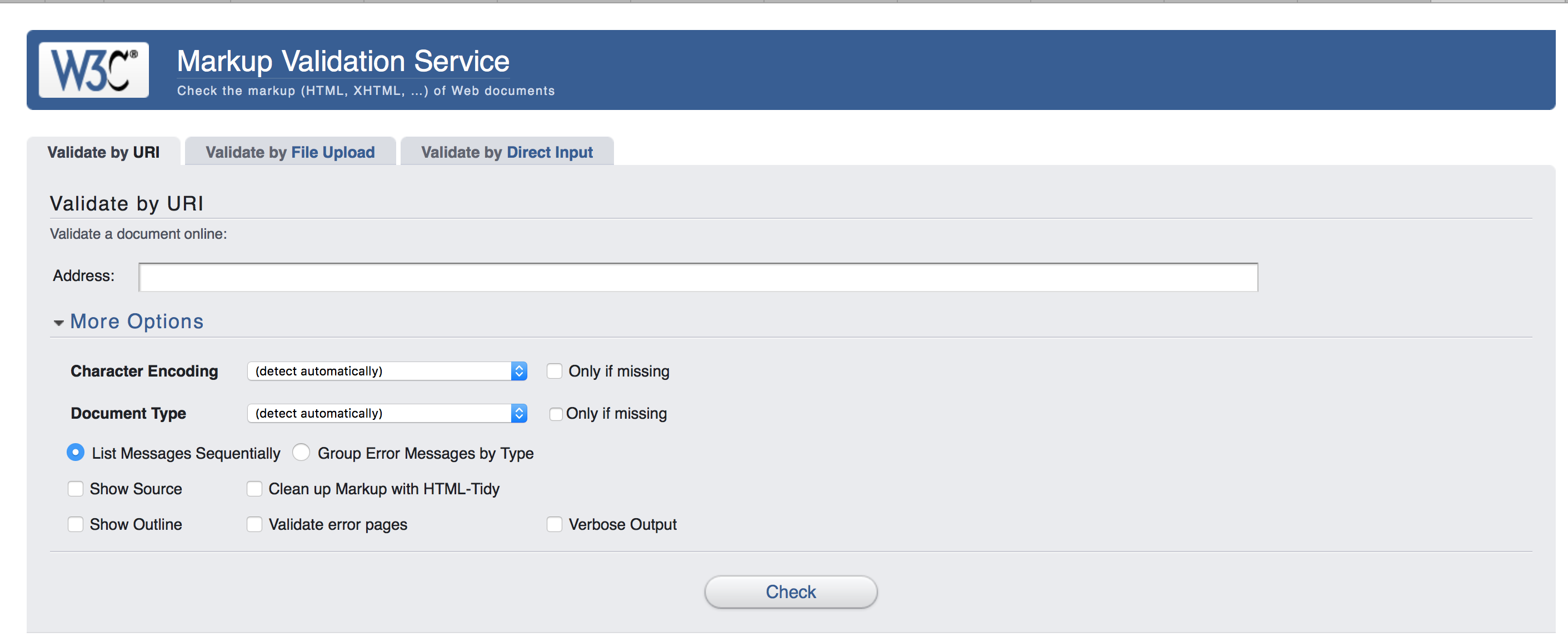
The following can be some generic parameters to choose an Automated Accessibility Testing Tool

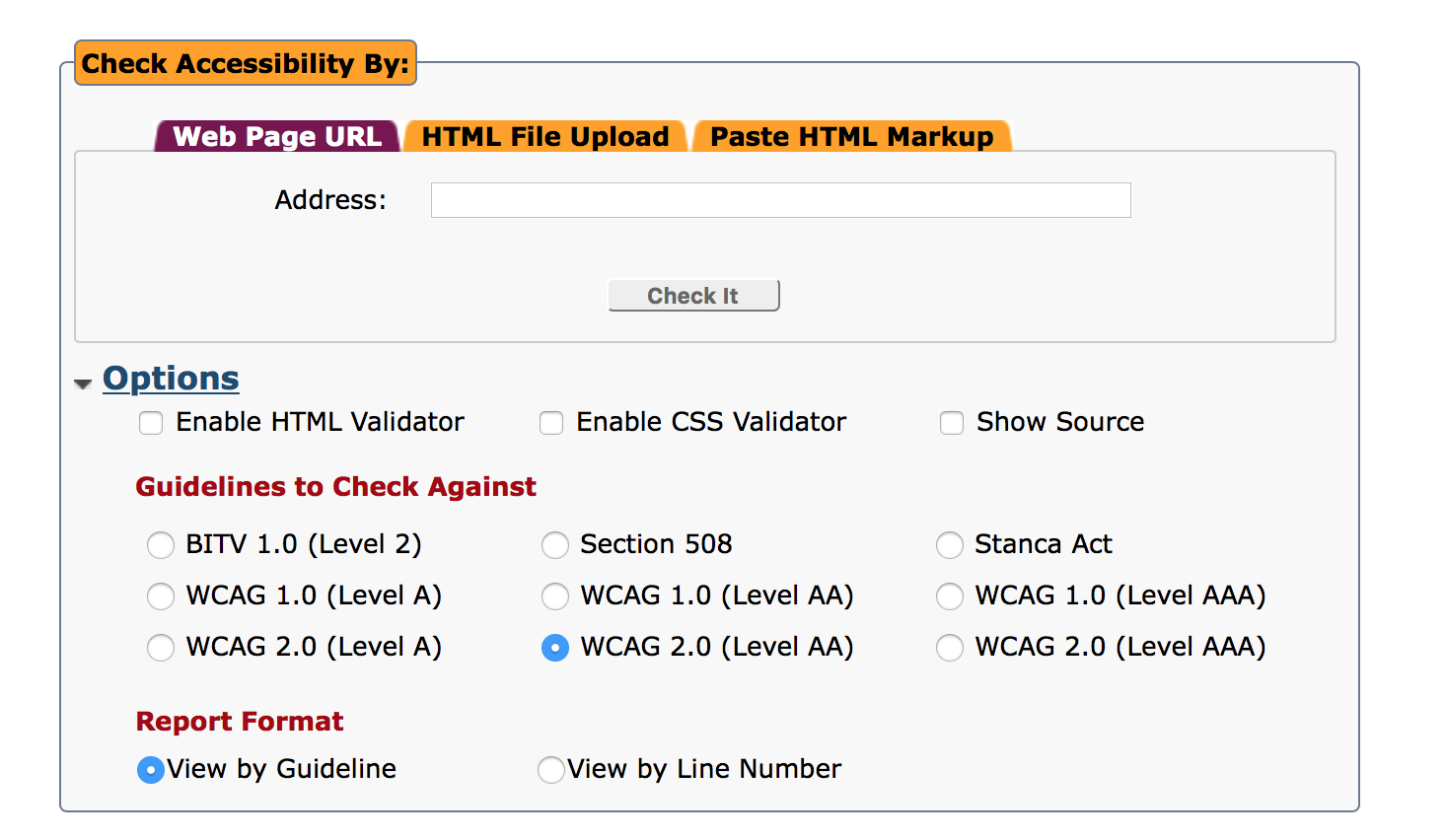
* User Friendly
* Quality and Reliability of Results
* Capability and Scope of DOM Testing
* Spider through URLs [applicable for Large Scale Testing]
* Continuous monitoring
* Uploaded files and/ or source code entered directly.
* Clear, easy to understand manual test guidance
* Configurable

Tools and Techniques:

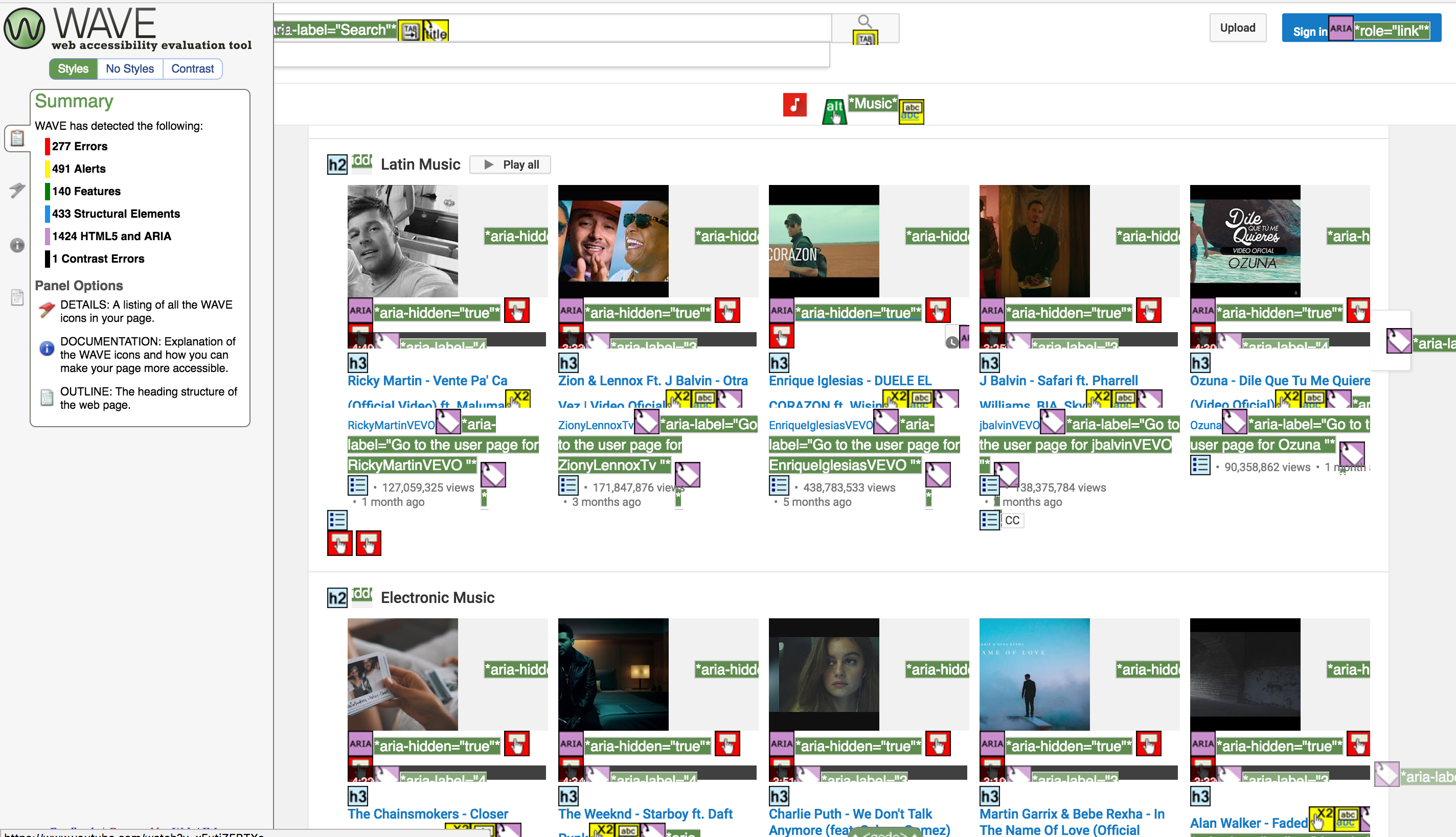
1. **HTML Validation:** While most would associate HTML Validation to full page markups, it is always good to test components / segments against standard templates for any markup violations. In cases your component contains forms or any interactive elements, it is very important to ensure that those are accessibility compliant.

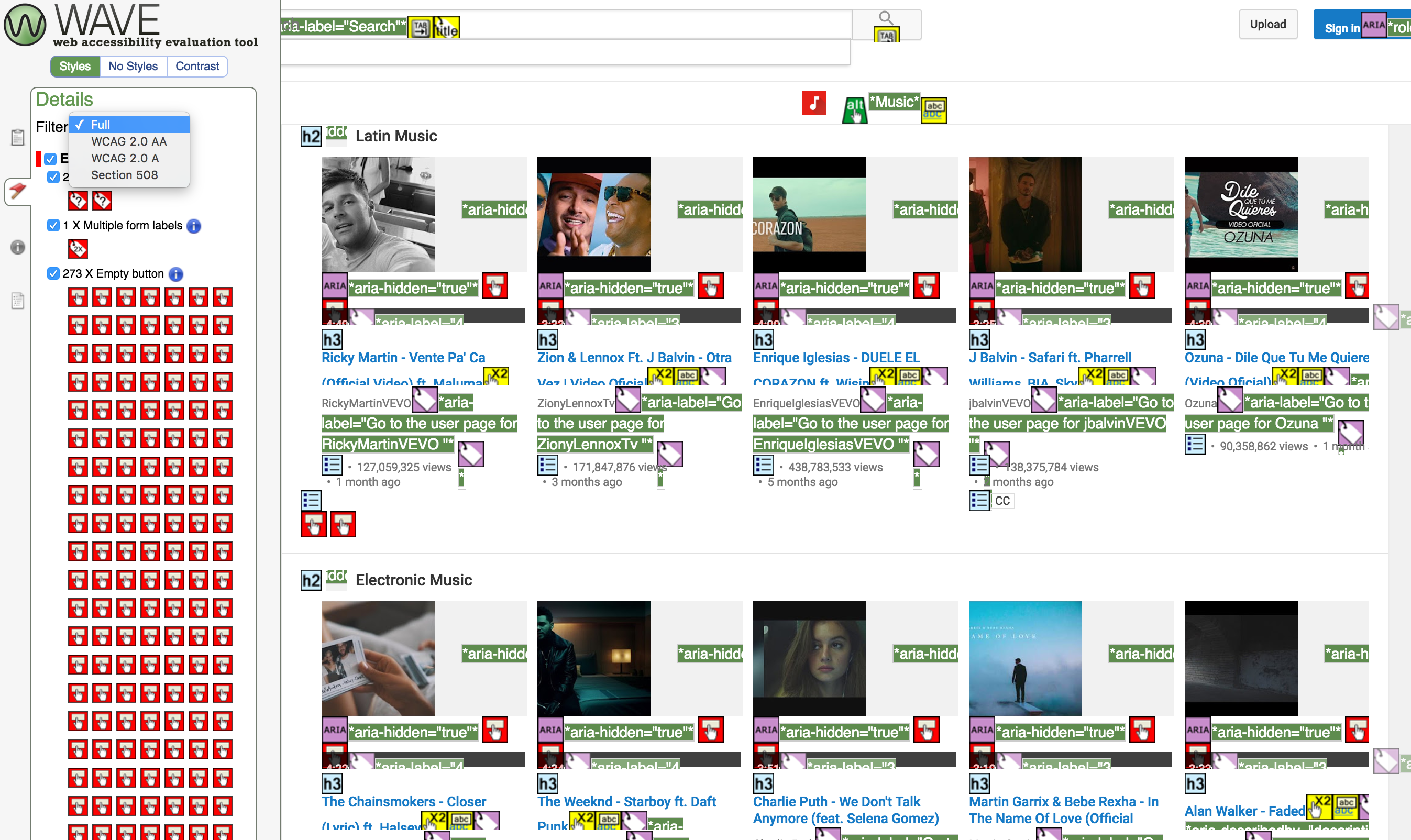
Some Standard tools for markup validation are:

1. <https://validator.w3.org> - The tool allows markup validation by URL, File input and direct entry of markup. 
2. <http://achecker.ca/checker/index.php> - AChecker is another tool that validates markup but also enables HTML pages for conformance with accessibility standards.



1. WAVE Toolbar: This is available as an online tool as well as a browser extension to validate accessibility code requirements on a page.





\*\* *Their often are arguments that Achecker and WAVE is a tool that should be used once the pages are ready and not necessary to be done on component level. But as a viewpoint, it is evident that it is suggested to use AChecker if the component or segment to be tested for accessibility contains form(s) or multiple interactive elements.*

**Other Automation Tools for Evaluating Accessibility:**

While Achecker and WAVE are standard and widely used tools, there are much more tools that are close to what these two tools offer. You can find a detailed summary of other tools compared with Achecker and WAVE on the following blog: <http://terrillthompson.com/blog/730>

1. **WCAG Standard Based Validations:** This test against WCAG standards during development can be monitored using grunt processes available. Some of the popular ones are:
   1. Pa11y
   2. grunt-accessibility
   3. grunt-wcag-accessibility
   4. a11y

All the above grunt tasks only differ in the way they present the issues reported. The reason for this is that internally all of the above internally use the [AccessSniff](https://github.com/yargalot/AccessSniff) and [HTML Codesniffer](http://github.com/squizlabs/HTML_CodeSniffer) services to grade your site's accessibility using different levels of the WCAG guidelines.

The rules against which validations are done in these grunt tasks using AccessSniff and HTML Codesniffer are:

<http://squizlabs.github.io/HTML_CodeSniffer/Standards/WCAG2/>

In case of “grunt-accessibility”, The following reporting format is provided. This is one of the popular grunt modules and provides a detailed CSV / HTML / JSON reporting for accessibility issues with a provided configuration.

**Key Notes**:

* Reports can be generated for levels **WCAG2A**, **WCAG2AA**, **WCAG2AAA**, and **Section508**
* Reporting formats supported are ".**csv**", ".**txt**" and ".**json**"
* Reported Attributes: *(as per CSV file generated)*
* Categories of reported Items: **Notices**, **Warnings** and **Errors.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Heading** | **Issue** | **Element** | **Line** | **Column** | **Description** |
|  |  |  |  |  |  |

# Manual Testing

**Keyboard Navigation and Tab Focus** – Component Level navigation via keyboard and tab focus should be tested to find out any issues that can be handled at component / segment level.

**Manual Testing:**

WCAG Guidelines for manual testing guidelines.

|  |  |
| --- | --- |
| **WCAG Guideline** | **Manual Test Description** |
| **Level A** | |
| 1.1.1 All non-text content that is presented to the user has a text alternative that serves the equivalent purpose. (Level A) | CSS background images should not be used for images that convey important information. This is because you can't put alternate text on a CSS background image. Right-click on an image to view source, and if there is no <img alt="" /> tag, then it is probably a CSS background image. |
| 1.3.1 Information, structure, and relationships conveyed through presentation can be programmatically determined or are available in text. (Level A) | Validate that changes in text presentation are not used to convey information without using appropriate markup *(e.g. text styled as a heading but not marked up with heading tags),* and that layout tables do not use any attributes associated with data tables *(e.g. IDs and headers for columns/rows, or E.g. Use of <th> tag, "summary" attribute, etc. for a LAYOUT table).* |
| 1.3.2 When the sequence in which content is presented affects its meaning, a correct reading sequence can be programmatically determined. (Level A) | Validate that an HTML layout table makes sense when linearized, i.e. when the table is read from top to bottom, left to right. |
| 1.3.3 Instructions provided for understanding and operating content do not rely solely on sensory characteristics of components such as shape, size, visual location, orientation, or sound. (Level A) | Validate that a graphical symbol alone is not used to convey information. For example, an empty shopping cart icon vs. a shopping cart icon with an item in it, with no text alternative to indicate the cart's status. |
| 2.1.1 All functionality of the content is operable through a keyboard interface without requiring specific timings for individual keystrokes, except where the underlying function requires input that depends on the path of the user's movement and not just the endpoints. (Level A) | Validate that all critical user flows are achievable through keyboard alone.  Validate that all remaining content is accessible through keyboard alone. |
| 2.1.2 If keyboard focus can be moved to a component of the page using a keyboard interface, then focus can be moved away from that component using only a keyboard interface, and, if it requires more than unmodified arrow or tab keys or other standard exit methods, the user is advised of the method for moving focus away. (Level A) | Validate that there are no keyboard traps (interactions you can get into with your keyboard, but can't get out of). These would be most likely to occur in overlays and other interactive elements on the page such as video players or image galleries. |
| 2.4.1 A mechanism is available to bypass blocks of content that are repeated on multiple Web pages. (Level A) | Validate that a link is provided to skip navigation and other page elements that are repeated across web pages OR that a page has proper heading structure.  Make sure it actually works to use the "skip navigation" link by tabbing to it, hitting "Enter" with your keyboard to follow the link. Ensure that the content area moves into focus, then tab again and ensure that the tab focus continues to be in the content area.  *Note: A developer can provide the skip navigation link as a hidden anchor link, but it's helpful for keyboard users not using a screen reader if the link becomes visible on focus, or is always visible.* |
| 2.4.3 If a Web page can be navigated sequentially and the navigation sequences affect meaning or operation, focusable components receive focus in an order that preserves meaning and operability. (Level A) | Validate that navigation order of links, form elements, etc. is logical and intuitive. Do this by tabbing through all elements. If the focus area does not move in a sensible way from in the reading order from top to bottom, there is an issue. |
| 2.4.4 The purpose of each link can be determined from the link text alone or from the link text together with its programmatically determined link context, except where the purpose of the link would be ambiguous to users in general. (Level A) | Validate that the purpose of each link (or form image button or image map hotspot) can be determined from the link text alone, OR from the link text and its context, by doing the following:  (1) Locate content needed to understand how link text describes the purpose of the link, and (2) Check whether the content is contained in the same sentence, paragraph, list item, or table cell, or in the preceding heading.  Validate that links (or form image buttons) with the same text labels that go to different locations are readily distinguishable (i.e. not in the same context in the markup).  *Note: If the design makes it difficult to include the link text IN CONTEXT in the markup, then the content should be updated.* |
| 3.2.1 When any component receives focus, it does not initiate a change of context. (Level A) | Validate that when any page element receives focus, it does not result in a substantial change to the page, the spawning of a pop-up window, an additional change of keyboard focus, or any other change that could confuse or disorient the user. *(E.g. Opening a new window when the page is loaded, or using a script to remove focus when focus is received).* |
| 3.2.2 Changing the setting of any user interface component does not automatically cause a change of context unless the user has been advised of the behavior before using the component. (Level A) | Validate that when a user inputs information or interacts with a control, it does not result in a substantial change to the page, the spawning of a pop-up window, an additional change of keyboard focus, or any other change that could confuse or disorient the user unless the user is informed of the change ahead of time. *(E.g. Automatic submission of a form and presentation of new content without a warning, or launching some action upon selection of a radio button, check box, or select list).* |
| 3.3.1 If an input error is automatically detected, the item that is in error is identified and the error is described to the user in text. (Level A) | Validate that required form elements or form elements that require a specific format, value, or length provide this information within the element's label. The information should be present between the <label> tags in the markup. Validate that form validation errors are presented in an efficient, intuitive, and accessible manner. *I.e., form errors should receive keyboard focus and be read out by the screen reader.* |
| **Level AA** | |
| 1.4.4 Except for captions and images of text, text can be resized without assistive technology up to 200 percent without loss of content or functionality. (Level AA) | Validate that text can be resized up to 200% and remain legible. Test this in Firefox: *View > Zoom > Zoom text only,* then ctrl+ on your keyboard to increase the font size on your screen. |
| 2.4.7 Any keyboard operable user interface has a mode of operation where the keyboard focus indicator is visible. (Level AA) | Validate that it is visually apparent which page element has the current keyboard focus (i.e., as you tab through the page, you can see where you are). *Note: This may need an associated design.* |
| 3.1.2 The human language of each passage or phrase in the content can be programmatically determined except for proper names, technical terms, words of indeterminate language, and words or phrases that have become part of the vernacular of the immediately surrounding text. (Level AA) | Validate that the language of page content that is in a different language is identified using the lang attribute (e.g.,  <blockquote lang="es">). |
| **Level AAA** |  |
| 2.2.4 Interruptions can be postponed or suppressed by the user, except interruptions involving an emergency. (Level AAA) | Validate that there are no meta redirects with time limits, that there are no meta refreshes with a time out.  Validate that interruptions (alerts, page updates, etc.) can be postponed or suppressed by the user. |
| 2.2.5 When an authenticated session expires, the user can continue the activity without loss of data after re-authenticating. (Level AAA) | Validate that if an authentication session expires, the user can re-authenticate and continue the activity without losing any data from the current page. |

# \*\* Source : W3C Manual Testing compiled by: <http://accessib.li/2016/02/12/using-tools-alone-to-test-for-accessibility-compliance-results-in-a-false-sense-of-security/>