

Puzzel as a Accessibility and Universal Design compliant solution (WCAG)

Through our focus on web page structure, keyboard navigation, keyboard shortcuts, WAI-ARIA code conventions, and colour contrast, Puzzel has created an inclusive and accessible user experience and has successfully met the WCAG AA 2.1 compliance criteria.

By addressing the WCAG AA 2.1 guidelines, we have demonstrated our commitment to providing equal opportunities and accessibility to all users. These efforts contribute to a more inclusive digital environment for Puzzel users of all abilities.

Puzzel's commitment to accessibility fosters equal opportunities, empowers users, and contributes to a more inclusive digital environment.

Introduction

This page provides a detailed overview of how the Puzzel Agent application has achieved compliance with the Web Content Accessibility Guidelines (WCAG) 2.1 AA standard.

The Agent application has undergone significant enhancements to ensure accessibility for all. In this page, you will find the measures taken by Puzzel in key areas, including web page structure, keyboard navigation, keyboard shortcuts, WAI-ARIA code conventions for screen readers like JAWS, and colour contrast.

Web Accessibility, WCAG and Why They Matter

Web accessibility is of paramount importance as it ensures that individuals with disabilities have equal access to digital content and services. The Internet has become an integral part of our daily lives, providing essential information, services, and opportunities. However, without accessibility, people with disabilities face barriers that limit their ability to fully participate and engage online.

WCAG compliance is instrumental in assisting users with accessibility needs by ensuring that digital content and services are designed and developed inclusively. By adhering to WCAG guidelines, websites and applications strive to provide equal access to all. WCAG compliance improves the overall user experience by making digital content perceivable, operable, understandable, and robust.

Web Page Structure

Our application utilises proper HTML markup to establish a logical hierarchy of content using headings (h1-h6). This hierarchical structure enables users relying on screen readers or other assistive technologies to navigate through the application more efficiently and understand the hierarchy of information on each page.

Semantic markup has been implemented throughout our application. Form elements, buttons, and other interactive components have been appropriately labelled to provide meaningful descriptions, helping users with disabilities understand the purpose and functionality of various elements within the application.

Keyboard Navigation

Keyboard navigation is crucial for users who cannot utilise a mouse or other point and select devices. Puzzel's agent application ensures comprehensive keyboard accessibility. All interactive elements, including buttons, links, and form controls, can be accessed and activated using the keyboard alone. The application maintains a logical tab order, allowing users to navigate through elements in a predictable and meaningful sequence.

Focus indicators have been thoughtfully implemented to highlight the currently focused element, assisting users with visual impairments in understanding their position within the application. The focus management strategy ensures that keyboard users can effectively navigate and interact with the various components of the application.

Keyboard Shortcuts

To enhance efficiency and accessibility, we have integrated keyboard shortcuts for the agent application. These shortcuts

provide users with quick access to frequently used features or functions, reducing the reliance on mouse interactions.

A comprehensive list of keyboard shortcuts is available within the application's documentation and help section. Puzzel has designed these shortcuts to align with WCAG AA 2.1 compliance guidelines, avoiding conflicts with browser or operating system shortcuts and ensuring an optimised user experience.

WAI-ARIA

WAI-ARIA enables web developers to enhance the accessibility of dynamic and interactive web content, particularly for users relying on screen readers like JAWS.

The agent application incorporates appropriate ARIA roles, properties, and states to provide additional context and information to assistive technologies. This ensures that users with disabilities using screen readers can effectively navigate, interact, and understand the various components of the application.

Colour Contrast

We understand the significance of colour contrast for individuals with visual impairments or colour vision deficiencies. The agent application has been designed to comply with WCAG AA 2.1 colour contrast guidelines.

The application ensures a sufficient contrast ratio between foreground and background elements, including text, icons, and other visual indicators. By prioritising adequate colour contrast, we have ensured that all users, irrespective of their visual capabilities, can perceive and understand the content within the application.

Consistent Navigation and Design Patterns

Our application maintains consistent navigation and design patterns throughout the user interface. This consistency ensures that users can predict and understand how to interact with various elements and functionality across different areas.

By maintaining a uniform layout, labelling conventions, and interaction behaviours, Puzzel provides a more intuitive and accessible experience for all users.

Form Accessibility

Forms within the application have been designed with accessibility in mind. Form fields include descriptive labels or placeholders that provide clear instructions or context for users.

Additionally, appropriate error messages are displayed and associated with specific form fields, allowing users to understand and resolve any input errors. The form validation is performed both client-side and server-side, providing feedback to users in an accessible manner.

Examples of WCAG AA 2.1 Compliance in the Puzzel Agent Application

Semantic HTML

- Use appropriate HTML elements (buttons, anchors etc..)
- h1 should appear first, then h2, h3 etc.
- Do not use <h> tags to just emphasise content. Instead use or CSS.
- Using <a> without href makes it just a div. If you absolutely have to, add role="button" (so it behaves like a button) and

tabindex="0" (make it focusable).

- Do not use custom forms. Use standard form control elements.

ARIA attributes

[aria-][action]

- aria-label - labels current element with custom text. Used a lot for icons inside links.
- aria-expanded - for menus, toggle using JS.
- aria-labelledby - f.e. for image captions.
- role (still aria attr) - change semantics for element that by default does something else (image button, etc..). Always, when setting a role="button" (We also allow navigation using Tab on disabled buttons) for an element, also add tabindex="0" or tabindex="-1", which enables the element to be focused only programmatically (not specific to buttons) so it's focusable using tab.

Designing colour accessible web pages

1 in 200 women and 1 in 12 men are colour blind.

- Most of them see colours, but have trouble distinguishing between them.
- Do not rely solely on colours to relay meaning. Put labels for colours, when user has to choose a colour. For input validation - use obvious text notification (big font, icon) as well on errors.
- Text contrast - minimum ratio to be checked using a tool.
- Hover effects - use not only colour, but a border as well, or bolden title.
- Start designing mockups in grayscale, and then add colour.
- Useful tools: WebAIM Contrast Checker, Grayscale chrome plugin, "I want to see like the colour blind" Chrome plugin, Color contrast checker.

Outline property

- Never disable focus outline.
- Use custom outline: > 1px with good contrast, because different browsers do it differently.
`*:focus { outline: 4px solid xxxxx; }`
- Add the same / similar effect for hover and focus for all elements.

Tabindex

- Tabindex is usually automatically added by the browser to interactive elements.
- Adding manually tabindex="0" makes element interactive. Important to note is that the "0" does not make it first element to focus.
- Screen readers will read all text on the page. No need for it to be focused to be read.
- Display: none removes element from tabindex. Same for hidden attribute.
- To manually remove element from tabindex, set tabindex="-1". It remains focusable by JS.
- Don't use tabindex values other than 0 and -1.

- Use `tabindex="-1"` is to hide modal or popup window (or display: none).

Bypass blocks

- Used to skip large groups of interactive elements (menus, filters, etc.) to avoid tabbing like mad through them to reach main content.
- `<div class="skip">Skip to main content</div>`
- `.skip a { position: absolute; left: 10000px; top: auto; width: 1px; height: 1px; overflow: hidden; font-size: 2em; }`
- `.skip a:focus { position: static; width: auto; height: auto; }`
- Do not use ids in CSS.

Accessible video and audio

- Providing captions and transcripts to videos and podcasts (or any type of audio you might have) not only shows compassion and makes the day of a disabled person better, but also enables search engines to index the contents of your media. Search bots can't play videos or podcasts.
- Some visitors might be unable to listen to your podcast at the moment (for example if they are on a train or in the library and have forgotten their headphones) but they could read the transcript. Other people prefer learning and consuming information by reading rather than listening.
- OzPlayer is a good example of accessible media player.

Accessible navigation

- Should have good enough contrast.
- No uppercase font in navigation (screen readers may read it as abbreviation)
- For navigation - use `<nav>` and `<a>`
- If there are multiple navigations - add `aria-label` attributes describing their function.
- Letter spacing should be fair so letters are not squished together.
- No point of underlining titles, since it's a common sense they are to perform an action.
- Current page should be highlighted in the navigation menu.
- Instead of dealing with "active" class for the above, use `aria-current="page"` on `<a>`, then style with CSS.
- `#primary-nav ul li a[aria-current="page"] { font-weight: bold; border etc... }`
- No sense of adding title attribute to navigation title - it's read twice by screen readers and is not really readable. Use tooltip instead.
- Outline does not move content, border does.

Accessible mobile hamburger navigation

- Button should be inside `<nav>` with `aria-expanded="true"`
- Only hide regular navigation and leave hamburger menu if user has JS enabled.

Accessible iframes

- Everytime an iframe is added, a title attribute has to be used so the user knows what is in the iframe.
- Maps - add as text the full address and if possible - detailed driving directions (not everybody uses GPS, printing directions may be easier for older people), on separate page if needed. Near landmarks description would be great too.

Accessible emojis

- Wrap emojis in a span with role="img" aria-label="emoji description".
- Don't use private emojis, instead use global unicode emojis.

Accessible sliders and carousels

- Get rid of small arrows and change them with buttons. Get rid of dots.
- Use sliders only if really necessary.
- Do not make slider autoplay or at least provide pause button and enough timeout.

Accessible contact forms

- Always put labels. Do not rely on placeholders (color contrast is not great).
- Every form must have a heading, but <h>, not <p> or other made to look like a heading.
- Inputs must have label (inside or outside <input> is ok).
- Easier is to put <input> in <label> to avoid "for" and "name" attributes. <label>Cool label<input></label>.
- For clarifications, put a after the <input> still in the <label>.
- If input is required, add required attribute.
- If possible, add a red * after the required label name wrapped in a span, but hide them from screen readers using aria-hidden="true".
- If for some reason you're unable to use "required" attribute, put "(required)" at the end of label name. You can also use aria-required="true" attribute to do the trick.
- Submit buttons - instead of a title "Submit" or "Send" describe a bit better what action it performs.
- Visual order should follow tab order. Do not put links between them.
- Outline should be well visible.

Accessible icons

- Most accessible are svg icons.
- Should have good contrast ratio.
- Always put text next to the icon (except probably for well known social media).
- Hide decorative icons from screen readers using aria-hidden="true".

- Icons in <a> - add aria-label="what link action does" and for the icon put aria-hidden="true". Same for font icons.

Accessible links

- Should have good contrast ratio.
- Never disable link underline and outline.
- Never open a link in new window or provide warning if you do: text or icon or aria-label attribute describing that link opens in new window. Other approach is to add hidden div somewhere in the page with an id f.e. "new_window" and text "opens in new window". Then use aria-labelledby attr on all links opening in new window and target this div.
- Do not underline plain text that is not a link.
- Do not use link names "click here" or similar, but provide a good description. If a blind person is scanning through the active elements they won't have the context.
- Logo images - alt attribute should assume image does not exist and describe where the link leads (Homepage, etc...).
- Even for sighted users, image links are not visible - leave the link underline.

Finish and test for site for accessibility

- Navigate using tab button.
- Use chrome dev tools lighthouse tool, but don't overly rely on it.
- Turn off your monitor and only use a screen reader.
- Test all pages.