Approach and Design Choices

Project Goal

The goal of this project is to build a browser-based automated agent that tests whether a given web page meets the requirements of **WCAG 2.2 Success Criterion 1.4.4 (Resize Text)**.

Overall Architecture

The workflow for each target page is:

1. Open the page with Playwright in a headless browser.
2. Take a baseline screenshot before resizing.
3. Resize text to 200% programmatically.
4. Take an after screenshot to capture the actual layout changes.
5. Send both screenshots to a Vision-Language Model (Gemini) with a carefully engineered prompt.
6. Parse the structured JSON returned by the model.
7. Generate a detailed Markdown report, including summary, screenshots, issue details, and raw output.

Key Design Choices

🙋 How to Simulate User Text Resizing ？

Here's how it works step by step:

1️⃣ Launch a Clean Browser Context

It spin up a Chromium browser in headless mode and create a fresh context and page. This makes sure there's no leftover cache, cookies, or plugins that might interfere.

```
async with async_playwright() as p:
    browser = await p.chromium.launch(headless=True)
    context = await browser.new_context()
    page = await context.new_page()
```

2️⃣ Navigate to the Target Page

Then load the URL with `wait_until="networkidle"` to ensure all resources are fully loaded and the page is stable.

3️⃣ Disable Animations

To get consistent screenshots for visual analysis, it inject some CSS to stop all transitions and animations. This helps the vision model focus on layout issues without flickering elements.

```
await page.add_style_tag(content="""
  * {
    transition: none !important;
    animation: none !important;
  }
""")
```

4️⃣ Take a Baseline Screenshot

Before changing anything, save a screenshot of the original page. This gives a "before" reference.

5️⃣ Resize Text to 200%

It simulate the user action by setting `document.body.style.zoom` to `200%`. It also dispatch a `resize` event so the page's responsive logic can react properly. A short wait ensures the layout has time to adjust.

```
await page.evaluate("""
    document.body.style.zoom = '200%';
    window.dispatchEvent(new Event('resize'));
""")
await page.wait_for_timeout(3000)
```

👉 **Why `zoom` instead of just `font-size`?**

Using `zoom` is closer to how many users resize text in their browser settings. It also affects elements that use relative units like `rem` and `em` more realistically.

6️⃣ Take a Resized Screenshot

It grab another screenshot after the zoom. These two images ("before" and "after") will be fed into the vision-language model to detect any issues like overlapping text, hidden content, or horizontal scrolling.

7️⃣ Clean Up

Finally, we close the browser and save the output for reporting.

🧐How to Use a Vision-Language Model for Visual Accessibility Checks？

Model ：Gemini 2.5 Pro

（

- It accepts images + instructions together
- It generates structured JSON output
- It's easy to call via API with our API key setup

)

How the Prompt Was Designed ？

A big part of making this work is giving the model very clear instructions.

Here's what I focus on in our prompt design:

✅ **Concrete Success Criteria**

The prompt mirrors the exact WCAG 2.2 SC 1.4.4 rules. It spell out:

- What *must* be flagged (loss of content, horizontal scroll, overlap)
- What *must not* be flagged (normal line wrapping, vertical scrolling, responsive menu changes, dismissible pop-ups, etc.)

```

**Flag ONLY when ALL these are true:**
- A visible change clearly hides, clips, or overlaps important content in a way that prevents reading or using it.
- A visible horizontal scroll is required to read lines or view content.
- Elements overlap so that important information becomes unreadable or unclickable.
---

**Acceptable changes that should NOT be flagged:**
- Normal line wrapping, text flow changes, or increased vertical scrolling.
- Navigation menus that collapse, merge, or become a hamburger menu if all options remain available.
- **Consent banners, cookie pop-ups, or modal dialogs** that temporarily cover part of the page if **they can be closed or dismissed**.
- Decorative elements like logos or page titles that shrink, move, or simplify if they do not affect main content or functionality.
- Content that extends above or below the visible viewport is normal if it can be reached by vertical scrolling.
---
👉 **If any situation fits both Flag and Acceptable, treat it as acceptable and do NOT flag it.**
```

✅ **Human-Centric Thinking**

The prompt explicitly tell the model to “think like a real user.” If a change doesn’t actually block someone from reading or using the page, don’t flag it. This reduces false positives.

Instead of acting like a rigid pixel-comparison tool, the model is explicitly told to **“Think like a real user.”** If a visible difference doesn’t truly make the page harder to read, understand, or use — then it shouldn’t be flagged. This mindset prevents over-reporting, which is a major issue in automated accessibility testing.

Example guardrails:

```
**Think like a real user:**
- Ask: Does this visible change really make you can't read, understand, or use the page?
- If not, do not flag it.
- If unsure, do not flag.
- If content just flows out of view vertically, assume it can be reached by scrolling.
```

✅ Clear Output Format

Clearly instruct the model to output **only a JSON object**. Each issue must include:

- `type` (standardized for easier reporting)
- `description` (short and clear)
- `suggestion` (a practical fix a dev can actually use)

✅ Screenshots as the single source of truth

A subtle but crucial principle: the model is told to only evaluate what is visible in the images, never to guess about hidden or off-screen content. This keeps the test realistic and aligns with how real users experience responsive layouts — by scrolling, not by assuming something vanished.

👻 How I Designed the Accessibility Report ？

Once the vision-language model finishes analyzing the before-and-after screenshots, the agent generates a **clear, structured Markdown report** that developers and QA teams can actually use.

Key Goals

Keep It Human-Readable

The report is plain Markdown so it's easy to read in any text editor or Git repo.

Summarize Clearly, Detail When Needed

At the top, we show:

- The tested URL
- The WCAG criterion (1.4.4 Resize Text)
- The total number of issues found
- A breakdown by issue type (`ContentLoss`, `HorizontalScroll`, `OverlappingElements`)

Then we list each issue with:

- **Type** (standardized for easy tracking)
- **Description** (short, plain language)
- **Suggestion** (practical fix devs can act on)

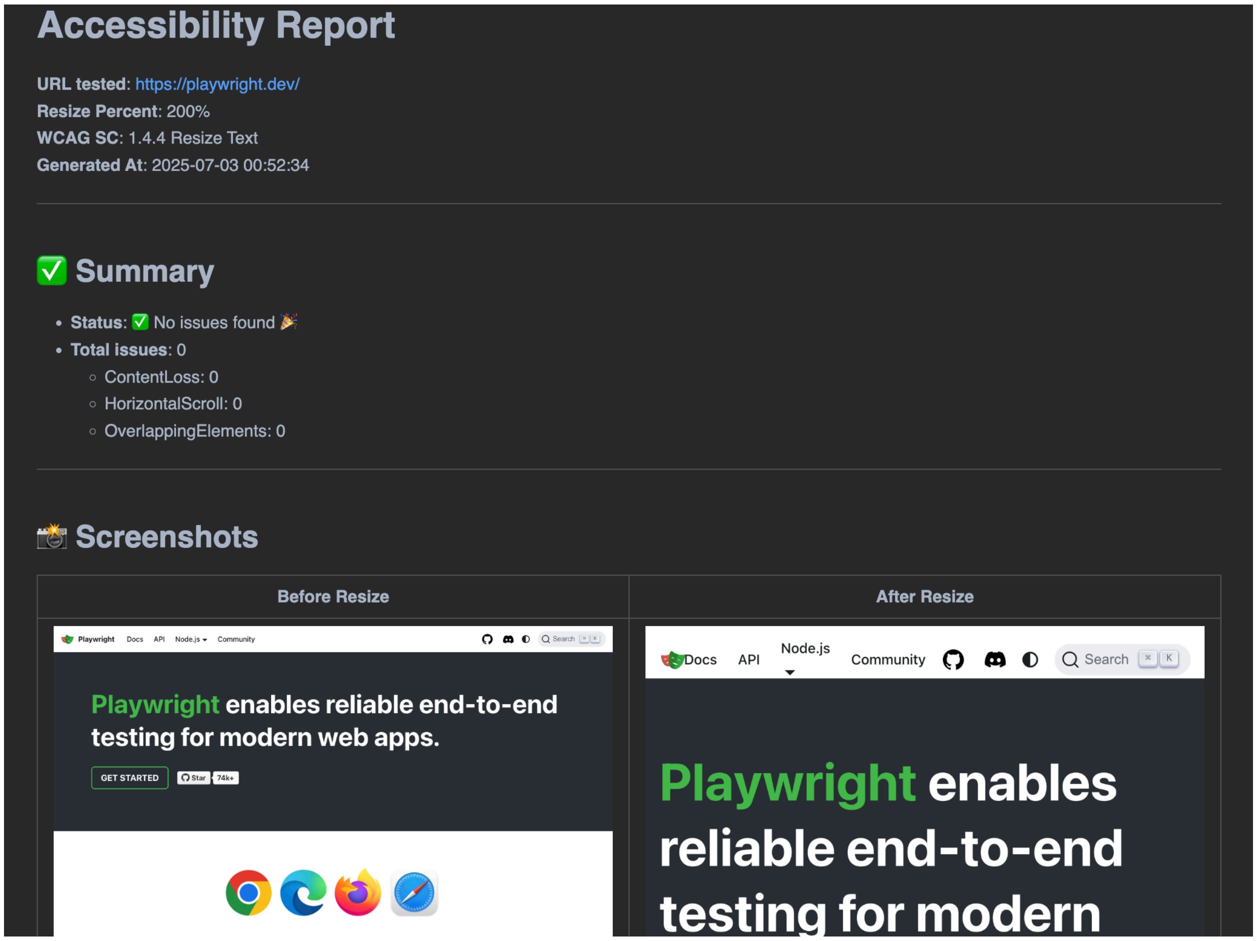
If no issues are found, the report makes that very clear

🗂️ Raw Model Output for Transparency

At the end, we embed the raw JSON output inside a collapsible `<details>` block. This serves two purposes:

- If someone wants to verify or debug the model's output, they can.
- The structured data can be re-parsed later for CI pipelines or trend tracking.

📸 Examples



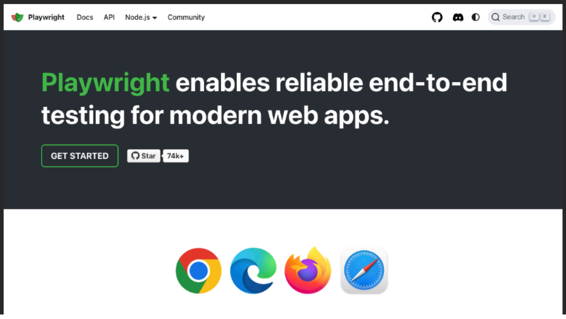
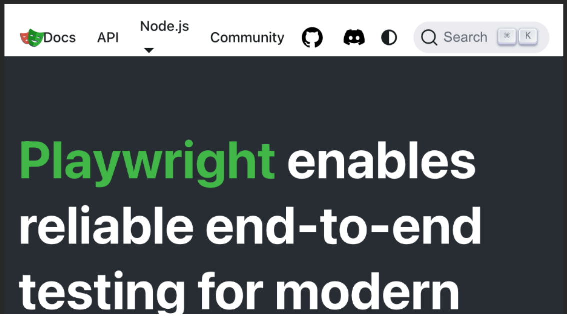
Accessibility Report

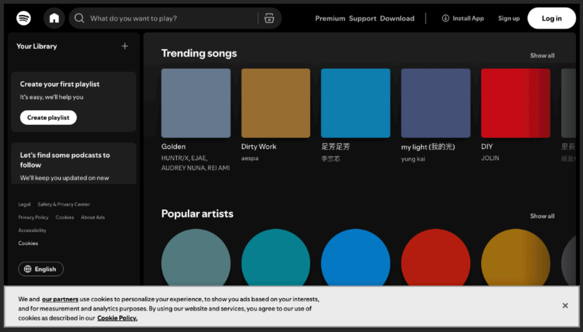
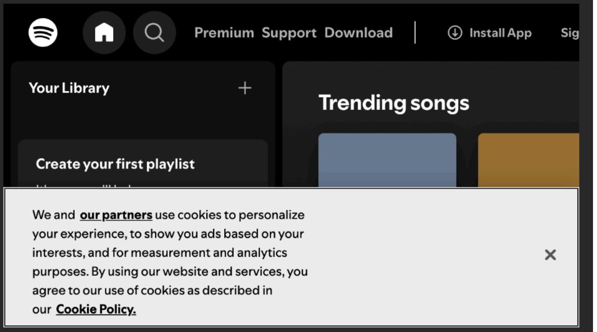
URL tested: https://playwright.dev/
Resize Percent: 200%
WCAG SC: 1.4.4 Resize Text
Generated At: 2025-07-03 00:52:34

✅ Summary

- **Status**: ✅ No issues found 🎉
- **Total issues**: 0
  - ContentLoss: 0
  - HorizontalScroll: 0
  - OverlappingElements: 0

📸 Screenshots

Before Resize	After Resize
 Playwright Docs API Node.js Community Search Playwright enables reliable end-to-end testing for modern web apps. GET STARTED Star 74k+	 Docs API Node.js Community Search K Playwright enables reliable end-to-end testing for modern

Before Resize	After Resize
	

We and our partners use cookies to personalize your experience, to show you ads based on your interests, and for measurement and analytics purposes. By using our website and services, you agree to our use of cookies as described in our Cookie Policy.

🗂️ Issues Details

1️⃣ Type: OverlappingElements

Description:
In the top header, the navigation links 'Premium', 'Support', and 'Download' overlap each other, and the 'Sign up' and 'Log in' buttons are pushed out of view, becoming inaccessible.

Suggestion:
Implement a responsive header that allows items to wrap or collapses them into a hamburger-style menu at larger text sizes to ensure all navigation options remain visible and usable.

2️⃣ Type: ContentLoss

Description:
In the 'Create your first playlist' section on the left, the descriptive text is cut off horizontally because its container has a fixed height and does not expand with the content.