Nick Pisarczyk | SI 339 CPD4

GitHub Pages: npisar.github.io/SI339-CPD4/ | Github Repo: github.com/npisar/SI339-CPD4/tree/main

Google Slides Slide Deck: docs.google.com/presentation/d/1zhehziH9MUnIoSbxJw-M8G-gLoeV-hv6SzIGPA7pGrc/edit?usp=sharing

Large Screen Optimization and Responsiveness

After designing my CPD3 site with mobile in mind first, I've included media queries to account for small tablet, laptop / large tablet, and desktop screen sizes. I never use max-width anywhere in my CSS. Here's a snapshot of my CSS showing how I use responsive design to resize elements on the page as the window size changes.

```css
/* desktop styles */
@media screen and (min-width: 1200px) {
    body {
        font-size: 22px;
    }

    table, details, .profile-header {
        width: 80%;
    }

    table {
        margin-top: 10px;
        margin-bottom: 20px;
    }

    .profile-header {
        grid-template-columns: 350px 1fr;
    }

    .profile-picture {
        width: 350px;
        height: 350px;
    }

    .profile-details h1 {
        font-size: 2.5em;
    }

    .profile-details h2 {
        margin-top: 15px;
        font-size: 2em;
    }

    .profile-details p {
        margin-top: 20px;
        margin-bottom: 15px;
        font-size: 1.2em;
    }


    .skyline-logo {
        width: 300px;
    }
}
```

```css
/* mobile styles */
body {
   font-size: 14px;
}

table {
   width: 100%;
   margin-top: 10px;
   margin-bottom: 20px;
   justify-self: center;
}

th, td, li {
   padding: 10px;
   font-size: 0.9em;
}

details {
   width: 90%;
   justify-self: center;
}

.profile-header {
   justify-self: center;
   align-items: center;
}

.profile-picture {
   width: 150px;
   height: 150px;
}

.profile-details h2 {
   margin-top: 5px;
   font-size: 1.2em;
   color: var(--primary-color);
}

.profile-details p {
   margin-top: 5px;
   font-size: 0.8em;
   color: var(--primary-color);
}

.athletic-link-button {
   font-size: 0.8em;
}

.skyline-logo {
   justify-self: center;
   width: 100px;
}

.data-chart {
   justify-self: center;
   height: auto;
}
```

CSS Quality

Like on my CPD3, I heavily commented on my CSS and sectioned out different areas of my styling with large comment blocks. This helps readability for others reading my CSS, and working off of this structure made making changes for the CPD4 very efficient. I still don't have any repetitive code, even with all the media queries, making my CSS very readable. When I styled an element in more than one instance, it was so I could organize the styling better in a way that made sense to myself and to other readers.

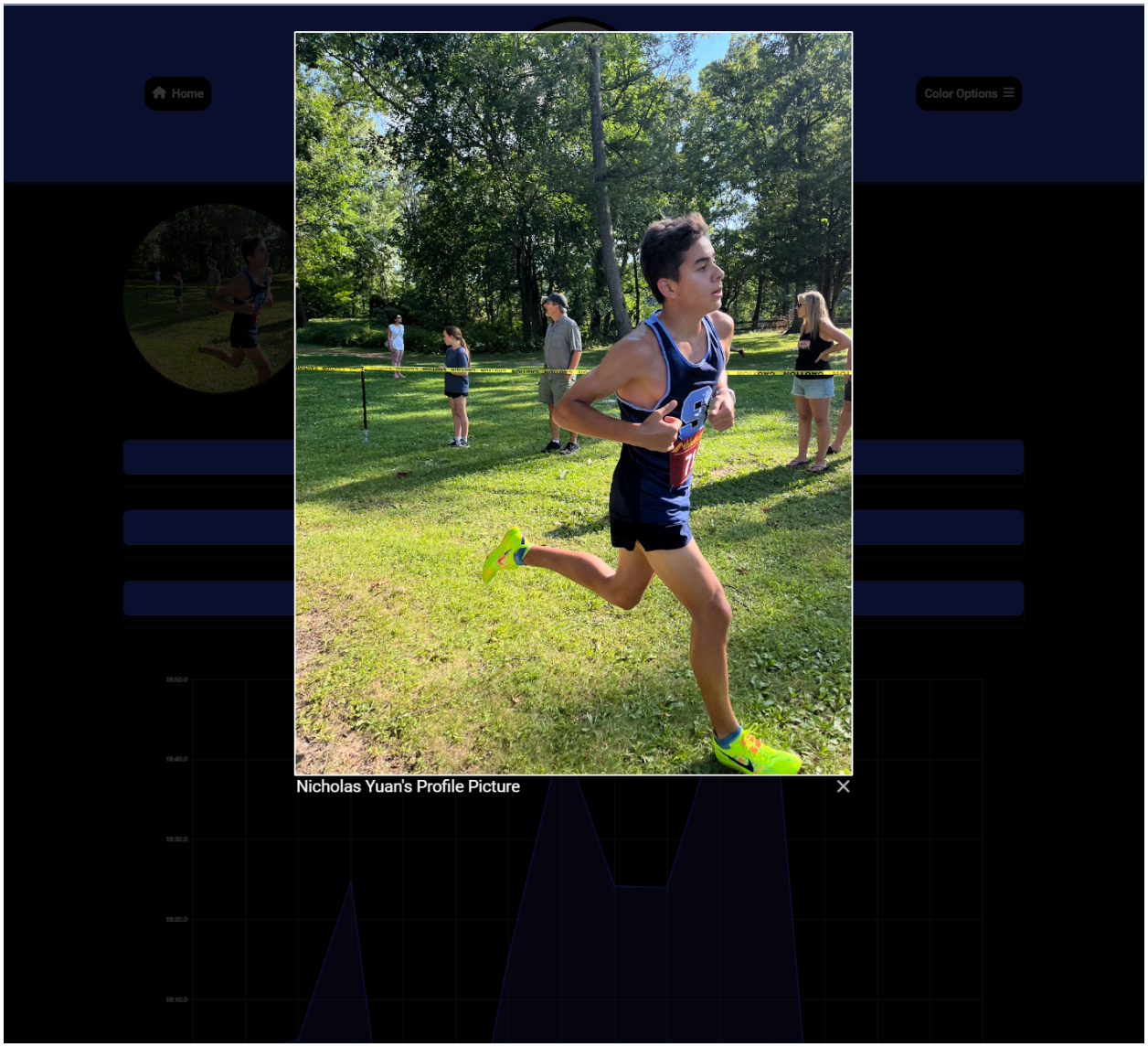
For example:

```
/* base styles */
...

header {
   position: sticky;
   top: 0;
   background-color: var(--secondary-color);
   box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
   padding: 15px;
   display: flex;
   justify-content: space-between;
   align-items: center;
}

/* grid setup */
header {
   display: grid;
   grid-template-columns: repeat(3, 1fr);
   grid-template-rows: 1fr;
}
```

Incorporates JavaScript

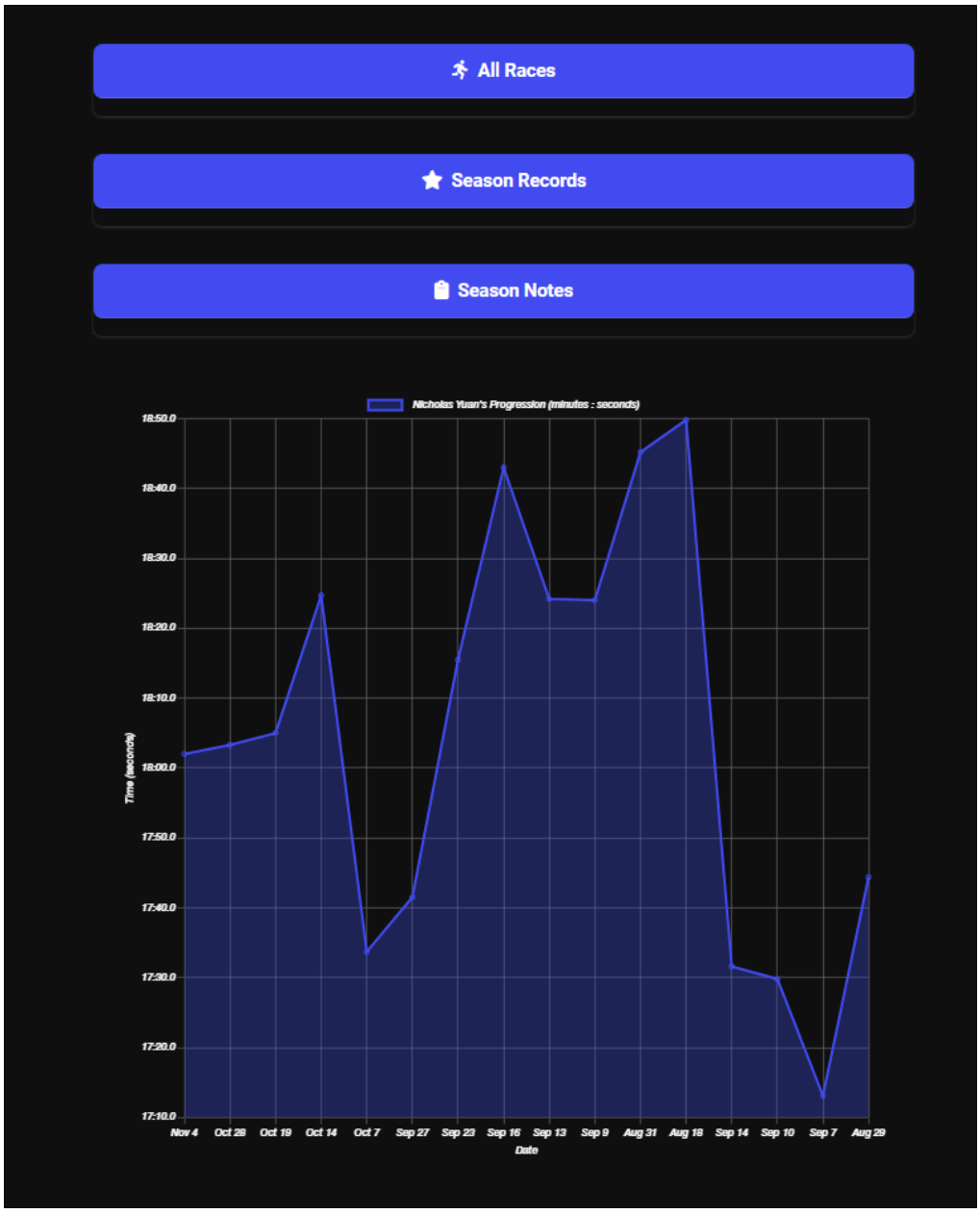
I included JavaScript on all of my athlete pages that really enhanced the functionality and UX of the site, as well as using it in with the color mode switching to make that experience seamless. Lastly, I used JS to make the header automatically shrink as you scroll.

First, I used lightbox to allow users to enlarge the athlete image. It even displays the athlete's name in the caption underneath with the appropriate alt text!



Additionally, I used Chart.js to include a progression graph showing every one of each athlete's race times and their date. This chart works in all of my color modes and

automatically resizes to fit the window, which works by using another JS function that detects when the window is resized, then redraws the graph. I style the width of the graph by accessing the canvas element (where the graph is drawn in) in the CSS media queries.



There's a good amount of JavaScript involved with the color modes. First, I define a color scheme toggle function which is able to detect what color schemes are active when the corresponding button is pressed. If you press the dark mode button to switch into dark mode, the button will automatically update to say "Light Mode" with the corresponding

sun icon, and vice versa for going back into light mode. I had to do a similar function for my high contrast mode, but it also accesses the user's previous color scheme. This way, when turning on and off high contrast mode, it will remember what scheme you were just using. Additionally, there's JS that automatically applies the dark or light mode class depending on the user's color scheme preference.

```
<script>

    function colorSchemeToggle() {
        const body = document.body;
        const colorSchemeToggleButton = document.querySelector(".color-scheme-toggle");

        // If high contrast mode is active, switch to the opposite scheme directly
        if (body.classList.contains("high-contrast-mode")) {
            body.classList.remove("high-contrast-mode");

            // Switch to the opposite scheme of the current remembered scheme
            if (previousColorScheme === 'dark-mode') {
                body.classList.remove("dark-mode");
                body.classList.add("light-mode");
                colorSchemeToggleButton.innerHTML = 'Dark Mode<i class="fa-solid fa-moon"></i>';
                previousColorScheme = 'light-mode';
            } else {
                body.classList.remove("light-mode");
                body.classList.add("dark-mode");
                colorSchemeToggleButton.innerHTML = 'Light Mode<i class="fa-solid fa-sun"></i>';
                previousColorScheme = 'dark-mode';
            }
        } else {
            // Toggle between dark mode and light mode
            if (body.classList.contains("dark-mode")) {
                body.classList.remove("dark-mode");
                body.classList.add("light-mode");
                colorSchemeToggleButton.innerHTML = 'Dark Mode<i class="fa-solid fa-moon"></i>';
            } else {
                body.classList.remove("light-mode");
                body.classList.add("dark-mode");
                colorSchemeToggleButton.innerHTML = 'Light Mode<i class="fa-solid fa-sun"></i>';
            }

            // Update previous color scheme to reflect current state
            previousColorScheme = body.classList.contains("dark-mode") ? 'dark-mode' : 'light-mode';
        }
    }

    // Toggle high contrast mode on and off
    function highContrastModeToggle() {
        const body = document.body;

        // If high contrast mode is already active, switch back to previous color scheme
        if (body.classList.contains("high-contrast-mode")) {
            body.classList.remove("high-contrast-mode");
            body.classList.add(previousColorScheme);
        } else {
            // Save current color scheme before switching to high contrast mode
            previousColorScheme = body.classList.contains("dark-mode") ? 'dark-mode' : 'light-mode';

            // Clear all classes and apply high contrast mode
            body.className = '';
            body.classList.add("high-contrast-mode");
        }
    }

    // Automatically set color scheme based on user preference
    document.addEventListener("DOMContentLoaded", function() {
        const prefersDarkScheme = window.matchMedia("(prefers-color-scheme: dark)");

        // Initial mode setup based on preference
        if (prefersDarkScheme.matches) {
            document.body.classList.add("dark-mode");
            document.querySelector(".color-scheme-toggle").innerHTML = 'Light Mode<i class="fa-solid fa-sun"></i>';
        } else {
            document.body.classList.add("light-mode");
            document.querySelector(".color-scheme-toggle").innerHTML = 'Dark Mode<i class="fa-solid fa-moon"></i>';
        }

        // Listen for system color scheme changes
        prefersDarkScheme.addEventListener("change", () => {
            if (!document.body.classList.contains("high-contrast-mode")) {
                colorSchemeToggle();
            }
        });
    });
</script>
```

I also used JS to automatically set the scale of the header to shrink. I had to make sure I only adjusted the height of the header so that it stayed touching the edge of the viewport. Then, I only scaled the width of the child elements so that it would even out and so that the content in the header would keep its original proportions.

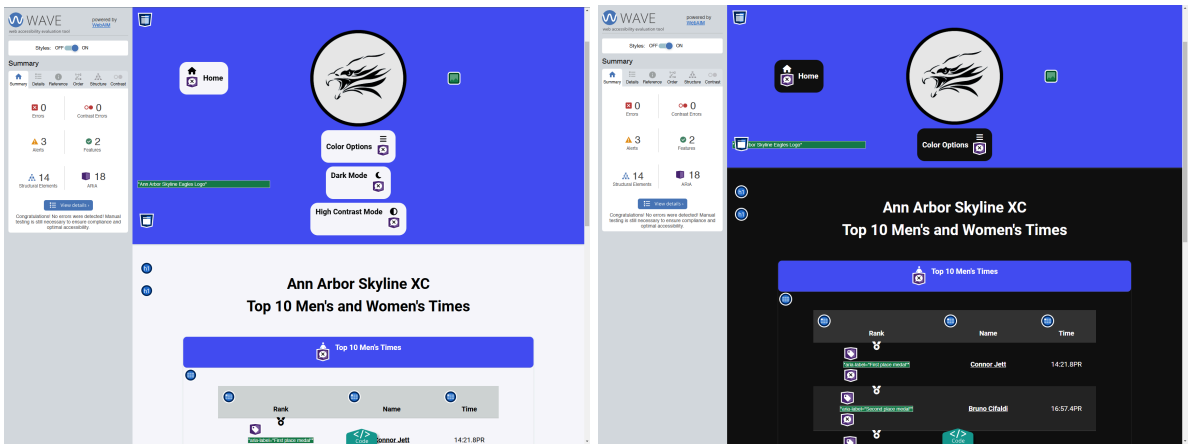
```
<script>
    window.addEventListener('scroll', function() {
        const header = document.querySelector('header');
        const contentElements = header.children;  // Get all children of the header
        const maxScroll = 200; // Adjust for scroll sensitivity
        const scale = Math.max(1 - window.scrollY / maxScroll, 0.50);

        // Apply a scale only to the height of the header itself
        header.style.transform = `scaleY(${scale})`;
        header.style.transformOrigin = 'top'; // Shrinks from the top

        // Scale each child element's height proportionally to fit within the
header
        for (let element of contentElements) {
            element.style.transform = `scaleX(${scale})`;
            element.style.transformOrigin = 'top';
        }
    });
</script>
```

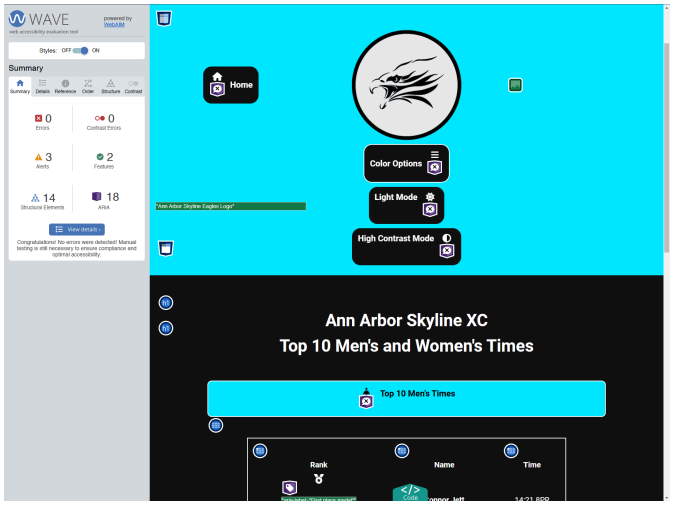
While one of the pieces of JS that was suggested for our sites was a way to add a default profile image if one wasn't found, this was something I already accounted for in my Python script.

Passes WAVE / aXe Validation

I tested both the homepage (index.html) and an athlete page (athlete-nicholas-yuan.html) with all three color schemes on both WAVE and aXe. Both returned 0 errors in all instances.

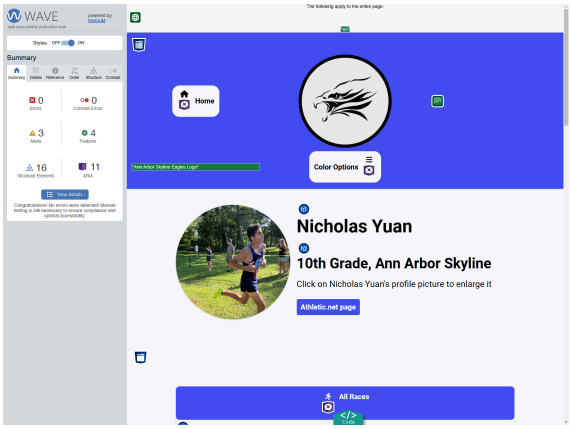
WAVE - Homepage:

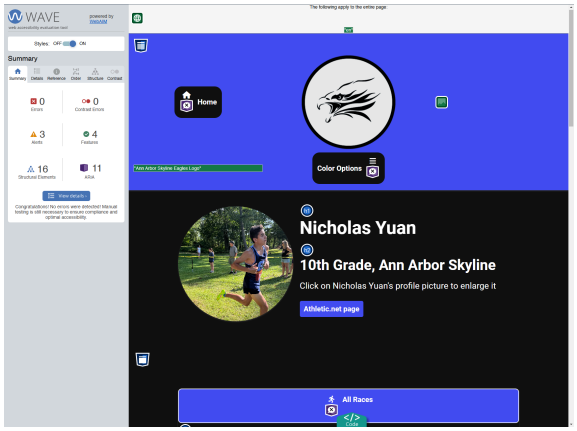


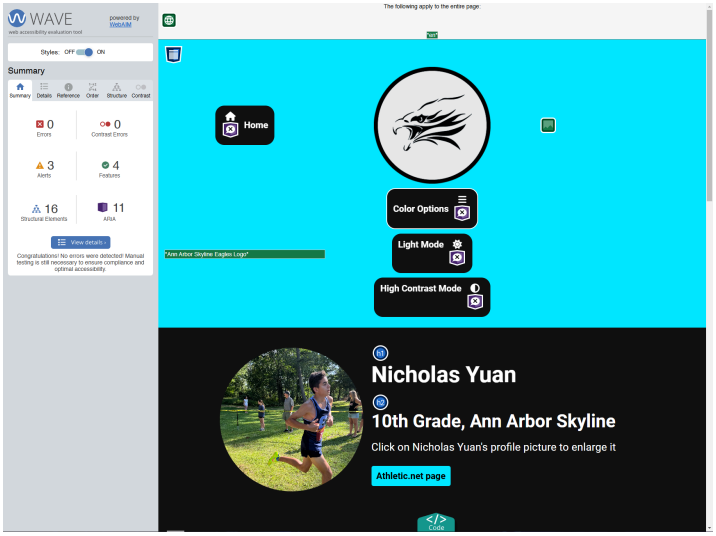


See next page for more →

WAVE - Athlete Page:

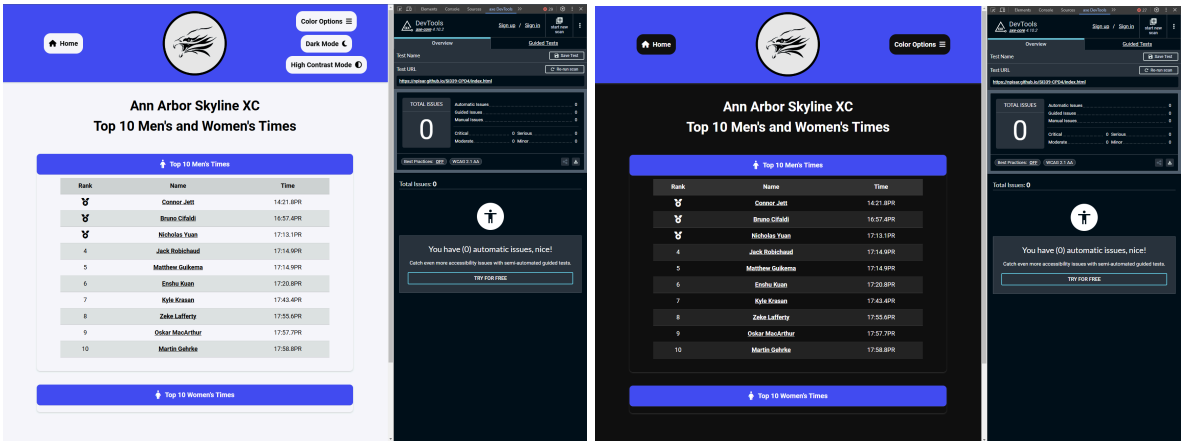


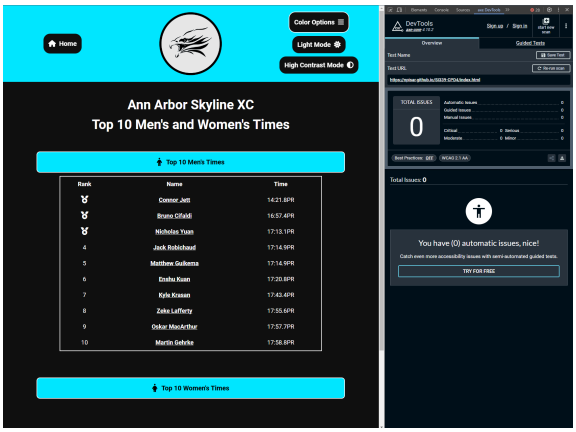




See next page for more →

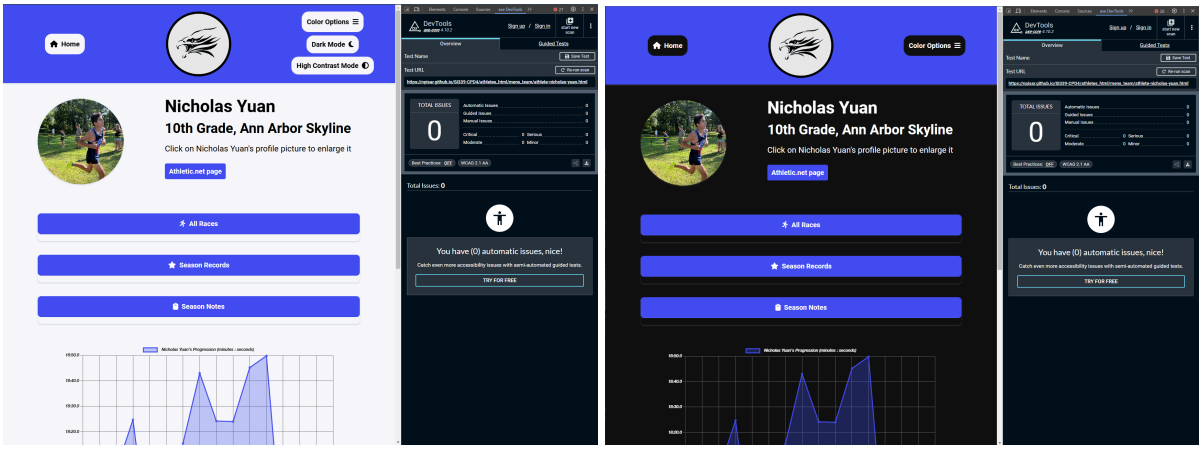
aXe - Homepage:

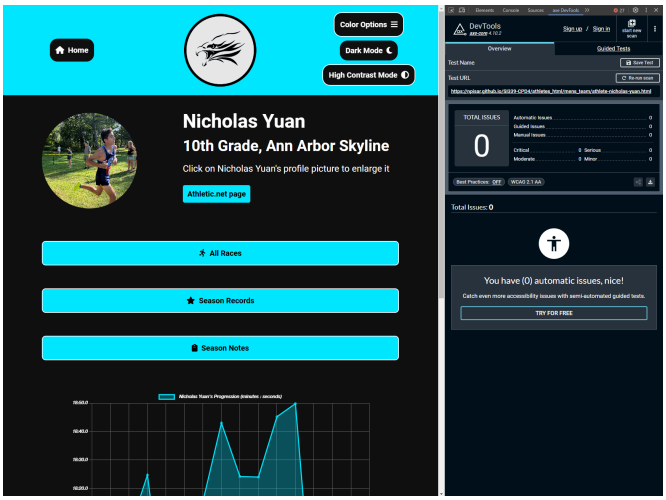




See next page for more →

aXe - Athlete Page:





See next page for more →

Addresses Prefers-Reduced Preference

I made sure to disable animations for users with reduced motion turned on. I did it with this media query:

```
@media (prefers-reduced-motion) {
   details[open] > *:not(summary) {
       animation: details-show 0ms linear;
   }

   button, .nav-home-link, .color-options-menu{
       transition: 0s color linear, 0s background-color linear;
   }

   details summary {
       transition: 0s color linear, 0s background-color linear;
   }
}
```

I also disabled the animations for lightbox with the options class within all of my HTML files, as seen below:

```
    <script src="../../dist/js/lightbox-plus-jquery.js"></script>
    <script>
        lightbox.option({
        'resizeDuration': 200,
        'wrapAround': true,
        //   'positionFromTop': 250,
        })
        if (window.matchMedia('(prefers-reduced-motion: reduce)')) {
            lightbox.option({
                'fadeDuration': 0,
                'imageFadeDuration': 0,
            })
        }
    </script>
```

See next page for more →

Fully Keyboard Accessible

My site is fully keyboard accessible. I've manually tested it and run into no issues. I defined the tabindex of navigation elements in my html headers, and used a tabindex when creating my html files with the Python script to make sure every piece of data is accessible by keyboard.

```
# Generate dynamic HTML table for 2024 races (excluding the Name column)
def races_table_maker_without_name(races_list):
    if not races_list:
        return "<p>No races available</p>"

    html_table = "<table>\n<tr>\n<th>Place</th>\n<th>Time</th>\n<th>Date</th

    tab_count = 6
    for race in races_list:
        html_table += f'<tr tabindex="{tab_count}"><td>{race['Place']}</td><
        tab_count += 1
    html_table += "</table>\n"
    return html_table
```

Presentation and Clarity

My CPD4 is very well presented and clear in the data it's displaying. There is a simple, 3-color palette, readable and understandable tables, easy-to-access options for color scheme, and all of the content fits any screen it would reasonably be displayed on. My CPD4 meets all of the criteria as listed above in a clear, professional, and easy-to-use and understand way.

Knowledgeability

My project is structured in a way that makes it extremely easy to make changes. By using Jinja2 template files, I'm able to make changes to only two template files and a CSS stylesheet, run the CSV to HTML Python file, and have all of the pages made again with all of the appropriate changes. I also know how my code works very well in general, and even though I used GenAI to help create a lot of the JS content, I've been able to learn the syntax and how JS works and have made a lot of special customizations to my JS for my specific CPD4.