Rachel Ruddy
Professor Tse
CS 396: Full Stack Development
31 Jan 2024

Mini-Project 1 Explanation and Defense

Overview

For Mini-Project 1, I have created a NextJS personal portfolio comprised of three pages: Home, Portfolio, and Contact Me. The site has a consistent header and footer across all three pages, with the goal of consistent and accessible styling throughout the site. Beginning this project, I had only cursory knowledge of web development and frontend design. I have been working towards a personal portfolio of my own without any framework, so I considered this project as an opportunity to start fresh and try to learn from the mistakes I had made in my personal project.

Problem Statement

My goal is to create a personal portfolio site with effective and persistent user interaction with the site via form submission and stored user preferences. With these characteristics I intend to create a site that not only showcases my personality but also offers users a streamlined and enjoyable experience.

Design Approach

Beginning the project, I debated between using a simplified version of Next.JS (similar to our coursework in Stackblitz) or the full version offered by the "create-next-app" command. Ultimately, I chose the latter, as it offered much in terms of existing documentation as well as a clear system for file arrangement. I created three separate pages to practice linking between pages, with the links stored within the Header component I had created.

I created a Header component so that I could isolate the logic of my header and easily integrate it into my layout.js file. This way, I could make changes to the header at any time without risking compromising the layout file. This is an instance of the Singleton design pattern, as I only use the Header component once, within the layout file. I chose a similar approach for the Footer component. Together, these components (and the nav tag nested within the header) provide not only a consistent format across the site's pages, but a method of tagging meant to me more accessible to users with visual impairments, as screen readers will be able to skip over the header directly to the main content if the user desires.

The third component I created was a Contact component, meant to store the layout of the email contact form as well as the logic needed to complete the form submission. To accomplish this, I generated an API key from SendGrid and attempted several different methods to connect it to my contact component, from creating a route.js file to using the UseEffect() function. Unfortunately, due to time constraints, I chose to remove my API references and leave the form, such that the user will still see a message upon clicking submit (stored and tracked using state). The form still contains input tags with accessible label pairings.

Beyond my components, throughout my project there is use of the Module design pattern, as pages are separated into their own folders (with the exception of the home page), most with their own modular css file attached to provide unique styles to that page. When I needed to use components, I would import them into the desired file. This pattern allowed me to keep my code

files organized and separated, such that when I was solving a bug on one page, the other pages were still able to function with relative independence.

There were a number of challenges I faced during the course of this project, many of which stemmed from my previous inexperience with the framework. I initially did not realize that Next.js changed its file structure relatively recently, so I often came across tutorials referencing unfamiliar file paths which slowed my progression on the site. In addition, I spent several days attempting to connect and activate my contact form with the SendGrid API to little success, due to a lack of understanding in how best to access the API and how to communicate user inputs between server and client components. As a result, I had to remove the API connection from my project.

The time lost to my contact form API prevented me from adding another meaningful form of user interaction besides the message upon completing the form and the button that allows users to download my resume. Still, with the time I had left, I started to program what I intended to be a toggle for switching between light and dark mode on the site. For this implementation, I planned to use React context to store the current theme setting and allow it to be changed on any page (and persist from there). I integrated the toggle button into the header to ensure it would be visible on every page. This theme context would have been implemented as a client component.

Overall, my site uses SSR as the bulk of the website. The unfinished Theme Context component was intended to use the client, as was the API calling (via useEffect()). The benefits of SSR mean that my website requires minimal Javascript to be sent to the client, instead allowing it all to be rendered on the server, saving time on load. This being said, had my CSR components been fully integrated, they would not have negatively impacted server compute costs since they were planned to be computed directly on the client side.

Though time prevented me from achieving the level of user interaction I was seeking, I still dedicated much thought to ensuring the accessibility of my website. From labeled inputs, to buttons with descriptive text (e.g. “Click here to visit my repository” rather than “Click here”), I ensured that my html semantics were both descriptive and easy to follow from a coding perspective. My personal portfolio site contains accessible semantics as well as several opportunities for user interaction, effectively addressing my problem statement.