Final Project Documentation

- Page Routing:

  I set up my page routing using the react-router-dom library because it supports routes within my application. Once I installed that library, I was able to pull the components of ReactDOM, BrowserRouter, Routes, and Route from the react-router-dom library, which all help with handling routes within my application. BrowserRouter helps keep the UI content in sync with the URL, so that the correct content appears for each route. The Routes component defines a set of routes for the application, which help make it possible to have different content in separate files and load that content into the UI. The Route component renders a UI component when the URL matches the path that has been defined for that route. I set this all up within my App.js file because this is what is invoked when I start my application. I attached my Navbar component to a Route component that has 3 other Route components nested inside it. This is because the Navbar contains the navigation bar, and it needs to appear on all the routes when they are rendered. The 3 other Route components all have a specific element attached to them that will render a new IU once a user routes to that specific Route. The Navbar component is used to link to other content using two more elements from react-router-dom: Link and Outlet. The Outlet component renders the current route selected and the Link component is used to set the current URL, keep track of browsing history, and effectively links to an internal path. All of this will help my application route properly.

- Passing Via Props:

  In this project, I set up the components to use performant code, so I did not want to hard code anything within the components. This is where I used props to pass data from one component into another. Within my project, I had both data files and an API to provide data for my application. All the data was imported through App.js and then passed via props to each page. The pages would then further pass the data down to the components that were embedded into the page. This allowed the components to be dynamic and reuseable. It allows me to import my data all into one page, so then if any changes need to be made, I will only have to go to one page and make those changes.

- React Hooks:

  Throughout my project, I used a variety of React Hooks for different changes within my application. I mainly utilized useState within my application to add state management to functional components. I used it within the navbar to make the navbar collapsable. The state is attached to the button click of the hamburger menu, so when the button is clicked, it changes the state of the navbar and applies different CSS styling. I also utilized the same mechanism for the accordion. The useState was attached to the onClick function on the button. This is what creates a collapsable buttons. I also used it for the Carousel to allow it to loop though the photos. The useState was attached to the button click again, to allow the next picture to be the next one found in the data set. Lastly, I utilized both useEffect and useState within my Blog Post page. I used useEffect to fetch

the data from the API. The useEffect takes two arguments, a callback function and an optional dependency array. The callback function is used to make the HTTP request and then uses a useState to update the state with the fetched data. This allowed me to store and manage the data from the fetch. I also used two other useStates for the Blog Post page, both for a modal. These two useStates allow the modal to pop-up upon clicking a button and allows the correct data to be placed within the modal. These are all the ways I utilized React hooks within my application.

- Hamburger Menu:
  I implemented the hamburger menu inside my Navbar component using a React useState. The hamburger menu remains hidden until the screen reached a certain width. This is implemented with the TailwindCSS styling as md:hidden within the button tag. Once the screen reaches a certain width, the button becomes visible and the navbar display is changed. I added a useState hook to allow the menu options to be visible. Once the button is clicked, the menu items become visible and once it is clicked again, the menu items are hidden. This is done via styling with both TailwindCSS and typical CSS styling. The useState allows the change in state of the components which is what makes the menu responsive.
- Components:
  I was able to make my application to use performant code, so as I explained above, all the components had data passed into them. These components were then imported on the certain page that they were to appear on. I wanted all the pages to have the same header, footer and navigation bar, so I implemented these components directly in the App.js file. Since this is the file that is rendered when the application is loaded, the Header component, Footer component and Navbar component are all placed within this page. This allows all the pages to have these components within them. I also implemented Home page, Blog Posts page and Contact page within App.js via the Route tag. The routing allows the page to appear on the UI once the route is clicked. Since the Route with the Home page has the index attribute, it is defaulted as the home page. I then was able to place the rest of the components on each page, so they can appear once a user navigates to that page. The Home page has the Carousel and Accordion components embedded into it. The Blog Post page has the Modal and CardContainer component. The CardContainer component then has a Card component embedded into it. Lastly, the Contact page has the ContactForm component embedded into it. This allows all the components to be embedded into the correct page and viewed by the user.
- Styling:
  For styling, I mainly utilized the styling components of TailwindCSS. I found this to be the easiest way to style my application because of all the built-in components and I thought it was easier to add responsiveness through TailwindCSS, rather than CSS. I also utilized some basic CSS in the App.css file, this was mainly the styling for the collapsable hamburger menu. I used flexbox styling to help style my

pages to make them responsive and used the build-in sizes of sm, md, lg, xl and 2xl to make changes according to screen width.

- Difficulties/Errors:
  I luckily did not encounter many errors. One error that took me a little bit to resolve was the API call. I could not map into the response, and it took a little bit of google searching and error handling. Upon further research, I discovered that I was missing one more step into the API to grab the correct data. I had originally set the state to response.data, but I didn't realize that I needed one more step of response.data.data. Once I fixed that, my API call was working without errors. Another difficulty I faced was figuring out that the responsiveness with TailwindCSS is worked as a mobile first. Once I wrapped my mind around that fact, it was easier to style my page to be responsive. Overall, I did not encounter very many errors, the ones that I did, I was able to resolve with a quick google search.