**Enhancement One: Software Design/Engineering**

The artifact I chose for enhancement is the front-end component of my political bias tracker web application, originally created using the Angular framework during CS 465 (Full-Stack Development). The application allows users to submit article URLs, analyze their political bias based on the publication source, and track saved articles within a personalized dashboard. While the core concept and some functionality were present in the original artifact, the front end suffered from design limitations, bulky configuration, and reduced flexibility. To address these issues, I completely overhauled the front end using a modern React-based stack, incorporating **React**, **Vite**, and **Tailwind CSS**. This enhancement demonstrates an industry-relevant evolution in both design and tooling.

I included this artifact in my ePortfolio because it clearly represents my ability to work with modern frontend tools and apply best practices in software engineering. The shift from Angular to React reflects an understanding of how to evaluate trade-offs in framework selection. React’s component-based architecture enabled me to design modular, reusable elements like SearchBar, SearchResult, UserSummary, and SaveSearchButton, which significantly improved maintainability and readability. Tailwind CSS provided a utility-first styling approach that reduced the need for bloated custom stylesheets, while Vite offered a lightweight build tool that greatly sped up development and hot-reload performance.

The refactor also demonstrates an ability to work with third-party documentation and integrate modern frameworks effectively — skills that are essential in professional software development. For example, I learned to configure Tailwind’s theme system, apply responsive design breakpoints, and manage visual states using utility classes. Through the inclusion of Tailwind, I also learned about the importance and utility of capitalizing on existing React/Vite/Tailwind component libraries and adapting pre-built component code into my own. This significantly accelerated development and helped ensure design consistency across the app. The new front end is now far more responsive, visually appealing, and easier to scale. The improved architecture means that future feature expansions — like notifications, charts, or sentiment analysis — can be easily accommodated without significant rewrites.

Through this enhancement I believe I have met and exceeded the ideas I initially planned in module one. In the software industry today, React and Tailwind are widely used because they promote efficient development workflows, responsive interfaces, and scalable component design. By moving from Angular to this React/Vite/Tailwind stack, I aligned the project with contemporary industry standards and showed an ability to both adopt and effectively apply modern software development practices.

During the enhancement process, I gained significant insight into modern frontend workflows. Learning how to configure Vite taught me about bundling optimization and the importance of developer experience. Understanding Tailwind’s class-driven styling forced me to think more systematically about spacing, layout, and UI consistency. I also improved in handling React’s state management and prop-passing patterns, enabling clean communication between components such as SearchBar and UserDashboard.

The biggest challenges involved migrating out of the older project structure and defining a consistent UI system. Initially, I began the rebuild from Angular with Create-React-App (CRA) before switching to Vite. This transition caused various difficulties, such as needing to reconfigure dependencies, update build logic, and rewrite parts of the file structure. These issues taught me the importance of clearly scoping and planning technology choices before development begins. In future projects, I now understand the value of defining a concrete architectural plan upfront and sticking to it. I also learned the importance of focusing on a minimum viable product (MVP) and testing core functionality before investing time in advanced features or aesthetic polish — many of which were revised later due to structural changes.

Looking back, this enhancement served as both a technical upgrade and a learning milestone. It reaffirmed the value of modern development tooling, modular design, and streamlined styling — all of which are central to successful frontend engineering today. The final result is a polished, performant, and maintainable application interface that is easier to build upon and aligns much more closely with professional software development expectations.