Rapid Web Application Development   
A Comparative Study of Next.js, Nuxt, and SvelteKit

Ryan C. Hull

The University of North Carolina at Charlotte

ITCS 5102 — Programming Languages

Dr. Mohsen Dorodchi

June 26, 2023

# Rapid Web Application Development

# A Comparative Study of Next.js, Nuxt, and SvelteKit

Web development has undergone significant advancements in recent years, with numerous frameworks and meta-frameworks emerging. Among these frameworks, Next.js, Nuxt, and SvelteKit have gained popularity for their capabilities and developer-friendly features. This paper aims to conduct a comparative study of these three frameworks, analyzing their suitability for rapid web application development and their impact on development speed, ease of use, code efficiency, deployment process, and support for external libraries. By examining the strengths and weaknesses of each framework, I hope I can provide insights into their developer experience and assess their suitability for different project requirements.

The study also delves into the underlying architectural differences among the frameworks, such as the use of Virtual-DOM or direct DOM manipulation, routing mechanisms, state management approaches, and frontend tooling. By understanding these architectural differences, developers can make informed decisions about framework selection based on project requirements and development preferences. In addition, I will complete the development of a minimum viable product (MVP) web application, and the development process will allow for practical insights into the selected framework's strengths and limitations, while documenting the steps taken and challenges encountered.

# Framework Review

## React - Next.js

[**React**](https://react.dev/) is a popular open source front-end JavaScript library for building user interfaces (Stack Overflow 2023), developed in 2013 by and currently maintained by Facebook/Meta. React has seen many updates and improvements since it first came out, looking far different then its initial conception (Hunt 2013). It provides a component-based architecture that allows developers to create reusable UI components and manage application state.

Behind the scenes, React utilizes a Virtual DOM (Document Object Model) to handle updates on the front end, which compares a virtual copy of the DOM and the real/rendered DOM to only apply necessary changes. While this improveS rendering performance and efficiency in many cases (Suthwal 2020), it may also bring increased memory usage and overhead (Suthwal 2020).

React utilizes JSX, a syntax extension to JavaScript that incorporates logic with markup (HTML) in a single file, and even markup styling with extensions and tools like [Styled Components](https://styled-components.com/), or my personal favorite [Styled JSX](https://github.com/vercel/styled-jsx). The theory behind this was to increase the separation of concerns, by reducing the coupling (modules relying on other modules), and increasing cohesion (degree of which elements in a module belong together). JSX also provides safety features out of the box, and will escape values embedded in it before rendering, helping to prevent XSS attacks. React follows a unidirectional data flow, where data will flow from parent to child component, and events can be handled and bubble up to parent components from children.

Since its conception, React has shifted from utilizing Class components to Functional components, and with this shift have introduced React Hooks, which are functions that can be imported to use state and other React features. These hooks can be confusing at first, but provide functionality that can be translated to many other web frameworks once learned.

[**Next.js**](https://nextjs.org/), on the other hand, is meta-framework built on top of React, and offers a solution that can provide many rendering solutions including Pre-Rendering (Server Side Rendering, and Static Site Generation) as well as React's default Client Side Rendering. Next.js offers a built in file-based solution for Routing, as well as API routes that can be run on a server as well as an [edge function](https://vercel.com/docs/concepts/functions/serverless-functions). Next.js offers its own basic command line interface, allowing for different commands like build, start, export, dev, lint, telemetry, and info.

## Nuxt 3 - Vue

[Vue](https://vuejs.org/), another open source front-end JavaScript library for user interfaces was created by Evan You in 2013/2014 and was meant to include parts from AngularJS that Evan liked, while still being lightweight. Vue also makes use of a Virtual DOM, and includes reactive data binding using refs, indicating a reference to a HTML or child element in the DOM/Virtual DOM, which allows tracking re-rendering, and to trigger it efficiently.

Vue makes use of Component-Based architecture much like React, and promotes Single File Components encapsulating logic, markup and styling. Vue’s templating syntax is based on HTML, and is seen as intuitive with a low learning curve. [Pinia](https://pinia.vuejs.org/) (previously Vuex) is the store (state) management library of Vue, and has also seen many variations and improvements since first introduced .

[Nuxt](https://nuxt.com/) is an open-source framework that offers many of the same features as Next.js does for React, with an integrated file-based routing system, SSR, and code splitting out of the box. Some of the unique features that Nuxt focuses on include Auto-Imports, which allow Vue composables/components to be used without direct imports, while still supporting optimized JS bundles. On the Server side, Nuxt uses Nitro, which makes use of Rollup and Node.js workers, and supports server side API routes and middleware. Nuxt also has its own extendable plugin/modules, which can allow for easy configuration for many services like hosting, database, and other external frameworks.

## SvelteKit - Svelte

[Svelte](https://svelte.dev/) is the newest frontend framework explored in this paper, first introduced by Rich Harris in 2016. Svelte takes a slightly different approach to building web applications, and aims to move some of the work from the browser to build time, and follows a reactive programming model that does not make use of a Virtual DOM like React and Vue.

Svelte takes a compile-time approach, (Karkra, 2019), compiling components to highly optimized JavaScript code with smaller bundle sizes compared to React and Vue (Sanika, 2023). Another difference is how Svelte also allows for two-way data binding, and can add reactivity without explicitly adding event handlers in components (Imfeld, 2021).

[SvelteKit](https://kit.svelte.dev/), much like the previously mentioned meta-frameworks, allows for multiple types of rendering options, integrated file based routing, a strong dynamic routing system, and the preloading/prerendering of links and pages.

One important but commonly improved on and changed aspect of all of the frameworks and libraries above are the Bundler (combining and separating of javascript files to be served to the browser) and the Compiler/Transpiler (translation of framework code to vanilla JavaScript that can be used by the browser/runtime). Over time each of the frameworks above have used many, each improving build speed and incremental building on changes for increasing speed of refreshing live applications while developing. These are subject to change, but to date the bundler and transpiler usage by each framework is the following:

| **Framework** | **Next.js** | **Nuxt3** | **SvelteKit** |
| --- | --- | --- | --- |
| **Bundler/Dev Server** | [Turbopack](https://turbo.build/pack) | [Vite (Rollup)](https://vitejs.dev/) | [Vite (Rollup)](https://vitejs.dev/) |
| **Transpiler/Compiler** | [SWC (Speedy Web Compiler)](https://swc.rs/) | [Nitro](https://nitro.unjs.io/) | [Svelte](https://svelte.dev/docs/svelte-compiler) |

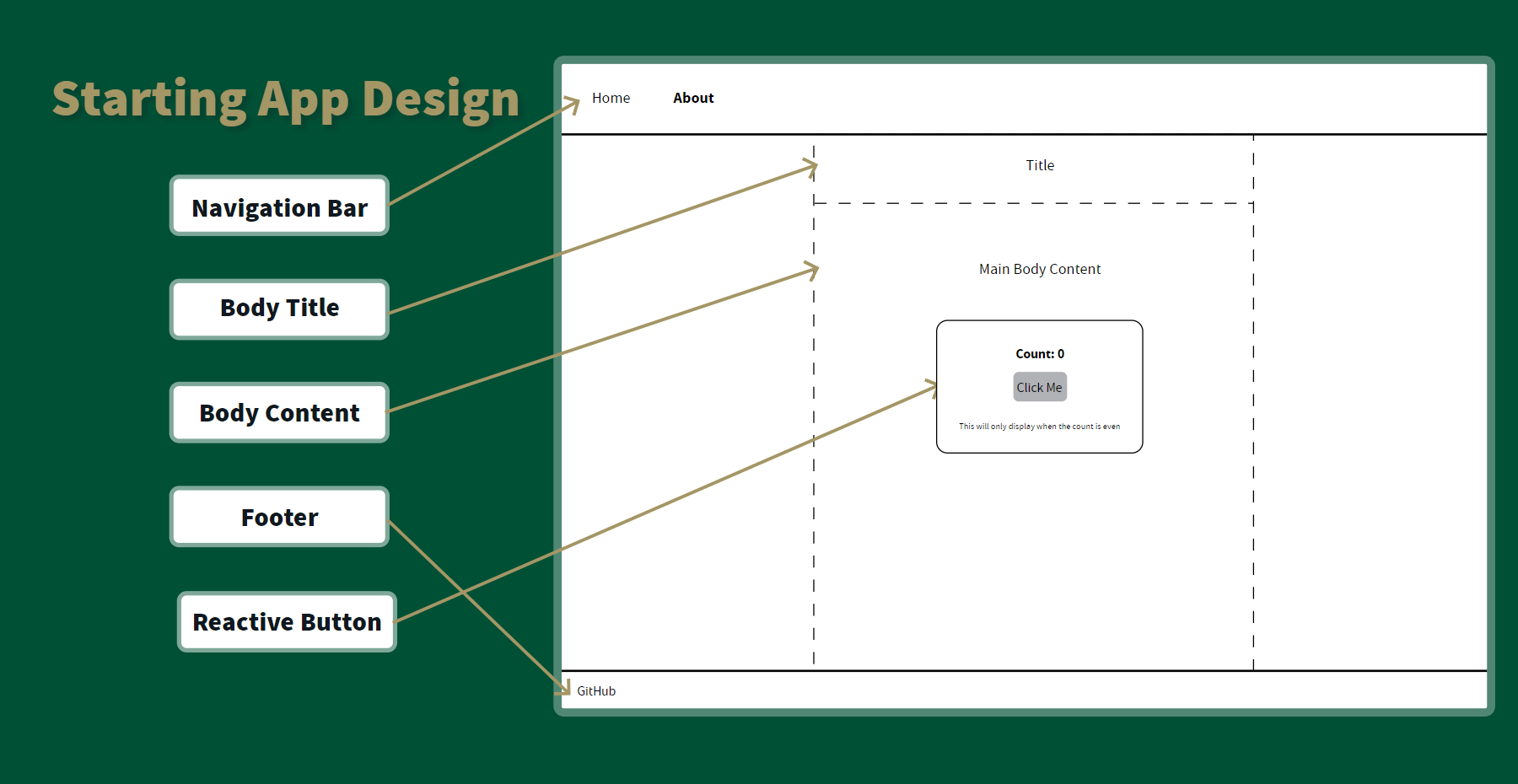
Framework Default Compiler/Bundlers

# Methodology

## Research Approach

Given the many similarities and significant speed claims of all meta-frameworks, it is difficult to provide a purely objective approach to choosing the best framework. Instead, I will create the exact same basic web application with all three frameworks, utilizing many of the concepts commonly used between all of them, while comparing the amount of time it takes to complete the application as well as the overall developer experience.

The “Starting App Design” is prototyped to include two basic pages, and include functionality like conditional rendering, conditional styling, reusable components, and state/events. The application will make use of [Tailwind CSS](https://tailwindcss.com/), and need to follow the exact same styling, shown in a prototype below. The SvelteKit implementation of the starting application is deployed and can be viewed [here](https://hullryan.github.io/sveltekit-starting-app).

Prototype Design

Overall, all frameworks were easier to use and than expected, with Next.js taking the least amount of time at 10 minutes to complete, mainly due to my previous experience with using the framework and react. Given this, I will exclude it from being used for the MVP application.

The application using Nuxt3 with no prior experience took just under 30 minutes to complete, and my only struggle was understanding the documentation about its state, due to the fracture between having both [Options](https://vuejs.org/api/options-state.html#options-state) and [Composables](https://vuejs.org/guide/reusability/composables.html) as interchangeable handlers. SvelteKit was the quickest and felt the most intuitive while learning, taking only 20 minutes to complete. They both have extremely similar component structures, and both include out of the box support for a layout page, which makes styling extremely easy. Configuring Tailwind took more time on SvelteKit, but worked without issues once setup, while Nuxt had configuration options from the CLI, same as Next.js.

Here is an example of the similarities and differences between the use of state and templating format:

Nuxt3/Vue - State and Template Structure

SvelteKit/Svelte - State and Template Structure

The example above shows how to both consume a prop passed from a parent to a child component, how to initialize a state variable and create/consume an event to update said variable, as well as conditionally render text in the DOM. This is a good example of how Svelte does not explicitly require state variables to be initialized with any special functions, and removes boilerplate and overhead when prototyping quickly. Nuxt3/Vue does include in element conditional if statements, which is similar to other HTML template languages, however I find Sveltes implementation to be intuitive and easy to understand if not the most stylish. Regarding events, I do not see any major differences in ease of use between any of the frameworks in this starting app example.

## Performance Analysis

When comparing performance metrics gathered from Lighthouse, SvelteKit had the highest performance with no optimizations made, at a perfect score of 100, Next.js following closely with 98, and Nuxt3 having a lower performance at 61. I believe this is due to the Lighthouse causing a full refresh on Nuxt, while only a quick refresh on the others. Overall, this is a synthetic test and there is no perceived difference in performance while using any of the implementations, not to mention this is not a stress test.

## MVP Requirements & Comparative Analysis

**User Story:** As a student, I need a site to display my IPython notebooks and Markdown files of my work for my Graduate classes, so that I can present and show what I have learned.

For this MVP, the application needs to be able to dynamically build a static site off of nested folders and files, being able to render Markdown and Converted Python Notebooks.

From the limited experience using both Nuxt3 and SvelteKit, I believe that SvelteKit will be the better option for this project, as it required the least amount of code, and brought the least amount of difficulties. Nuxt3 worked extremely well, and the dynamic importation of components is hard to let go, but Sveltes direct implementation of state and reactivity without needing any hooks or declarations was such a nice feature for the development experience I think it will win out on this project. React and Next.js will still be on top when it comes to external libraries and support, with Svelte having the least amount of libraries available on NPM (npm) of these frameworks, however for this project I do not believe it will come to affect it negatively.

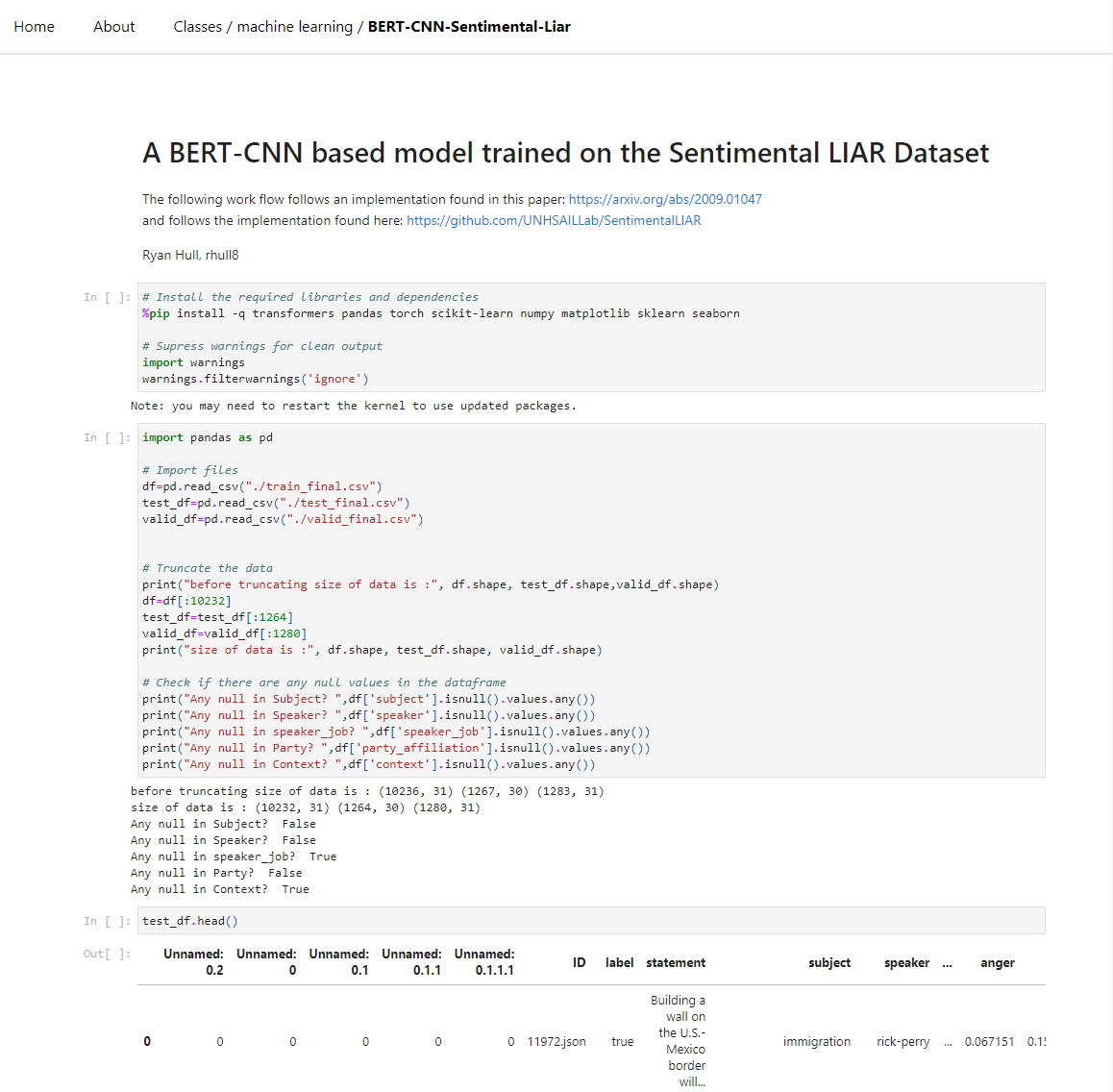
# Development Process & MVP Application

Using the Starting application as base and expanding on it, I began developing and learning the basic features of SvelteKit more specifically, focusing first on dynamic routing using [slugs](https://kit.svelte.dev/docs/routing). This also felt intuitive, and required minimal configuration, and is also similar to how dynamic routing and slugs work with Next.js. I first started by trying to render a folder of Markdown and Python notebooks converted to HTML files dynamically at different URL paths. The structure for this works with each Svelte page having a data loading JavaScript alongside it, which can take parameters from the url and query/import files in the source folder, then sending the data to the Svelte page for rendering. SvelteKit has a built-in Markdown preprocessor called MDsveX, which can convert the markup to HTML at runtime.

After I was able to have the dynamic routes base on files, using Vites built in “import.meta.glob” function, which allows for importing multiple modules, it was decently simple to then duplicate the base SvelteKit page and loading Javascript to child folders and duplicate the logic to import the folder names as Strings to be able to display links to the now child folders.

This structure looks like the following:

After completing this, I decided to render this tree style page structure into the navigation bar as well, so that the user would be able to return to any parent route without having to start at the “classes” page. To do this, I use the page store provided by default in SvelteKit, and decode the split URL path to handle logic and display the active path styling, shown below.

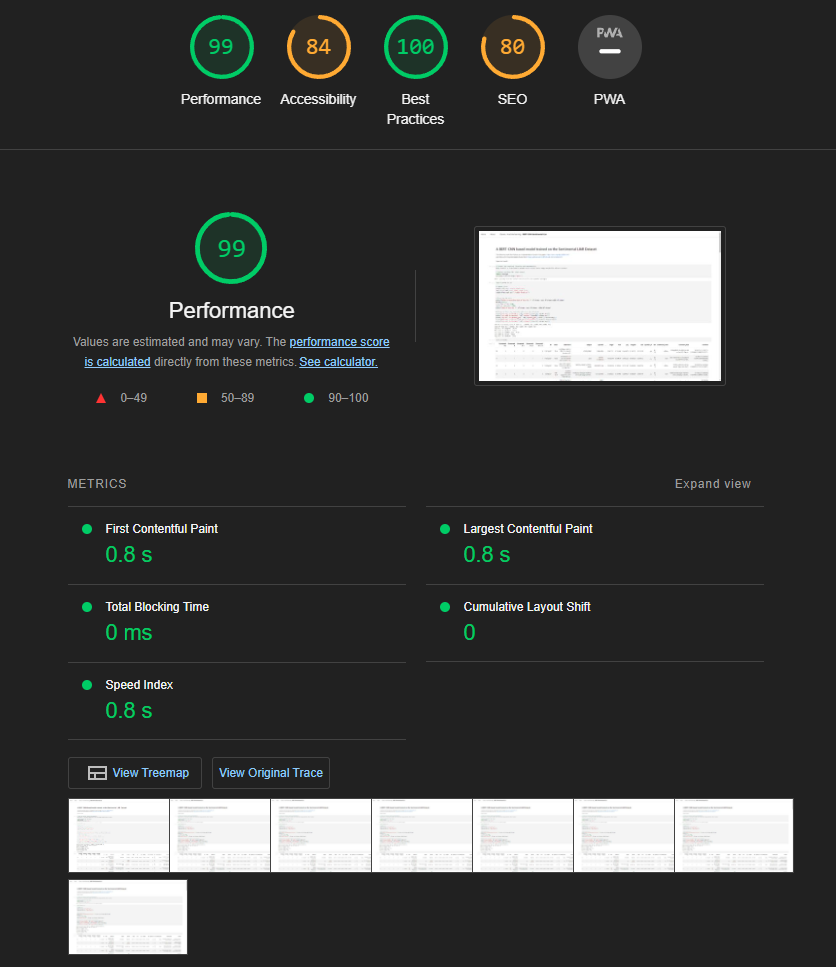


Finished MVP Web Application

The last step to complete the application was to configure the repository on GitHub to build the web app to a static site and deploy it to GitHub pages using GitHub Actions. There are many templates available online and while there was no official SvelteKit workflow on GitHub, I was able to reference other build and deploy actions to configure my own very easily. Once configured, the only issue I ran into was GitHub pages having a subroute for repositories, which broke the logic in the navigation bar. I was able to refactor and configure it to support local development and being hosted on GitHub pages, and all went fine.

# Results & Discussion

Overall I am extremely pleased with the end result and very impressed with the SvelteKit framework, and found it extremely useful and intuitive when coming from a background of React and Next.js as well as vanilla JavaScript. SvelteKit is still extremely young compared to other frameworks, and is still growing in its user base which will in turn come with increased integrations and libraries in the future. The overall performance and speed of reloading in a live preview while developing allowed for quickly testing changes, and the error handling and display when misconfiguring was decent enough to be able to figure out any issues met. The ending performance for the final built application hosted on GitHub pages shows impressive results, and you can view the deployed site [here](https://hullryan.github.io/sveltekit-mvp), as well as the entire project repository and documentation [here](https://github.com/HullRyan/ITCS5102-Final-Project). In the future I would like to add onto this MVP, and refactor my implementation further, rewriting much of it to run recursively and possible utilize a custom Rollup configuration, in order to work with any number of levels of subfolders, implement multiple file format conversions with GitHub actions automatically, and possible add fuzzy-search or a related search ideology in order to easily traverse the rendered documents.



Final MVP Lighthouse Metrics

# References

(n.d.). Stack Overflow Developer Survey 2023. <https://survey.stackoverflow.co/2023/>

Hunt, P. (2013, October 20). <https://2013.jsconf.eu/speakers/pete-hunt-react-rethinking-best-practices.html>

(2020, November 30). React Virtual DOM vs Real DOM - Devinder - Medium. Devinder. <https://medium.com/devinder/react-virtual-dom-vs-real-dom-23749ff7adc9>

karkra, N. (2019, May 25). Svelte 3 — Compile-time framework - Code To Express - Medium. Code To Express. <https://medium.com/code-to-express/svelte-3-compile-time-framework-ed8f6f158001>

Sahu, A. (2023, May 5). Svelte vs React: Which JavaScript Framework Is Better?. Turing. <https://www.turing.com/blog/svelte-vs-react-which-javascript-framework-should-you-choose/>

Imfeld, D. (2021, October 23). How Svelte Makes Two-Way Binding Safe. https://imfeld.dev/writing/how\_svelte\_makes\_two\_way\_binding\_safe