

Comparison of Modern JavaScript Web Frameworks

Sara Arghwan Jameel

Bachelor's thesis
2023



Tomas Bata University in Zlín
Faculty of Applied Informatics

Tomas Bata University in Zlín
Faculty of Applied Informatics
Department of Informatics and Artificial Intelligence

Academic year: 2022/2023

ASSIGNMENT OF BACHELOR THESIS

(project, art work, art performance)

Name and surname: Sara Arghwan Jameel
Personal number: A19901
Study programme: B0613A140021 Software Engineering
Type of Study: Full-time
Work topic: Srovnání moderních JavaScript frameworků pro web
Work topic in English: Comparison of Modern JavaScript Web Frameworks

Theses guidelines

1. Prepare the literature review of the React and Svelte frameworks.
2. Describe these frameworks' main concepts, architecture, and components.
3. Design a simple application that will contain commonly used user interface elements and will be used to compare frameworks.
4. Implement the application using both frameworks and briefly describe the implementation process.
5. Compare the implementation process of created applications using subjective parameters (evaluation of implementation simplicity, quality of documentation, etc.) and objective parameters (application performance, application size, etc.).

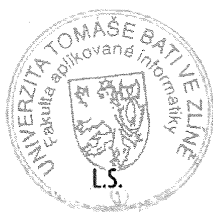
Form processing of bachelor thesis: **printed/electronic**

Recommended resources:

1. HOLTHAUSEN, Simon. Svelte a beginner's guide. B.m.: Sitepoint, 2022.
2. LIBBY, Alex. Practical svelte: Create performant applications with the Svelte Component Framework. Berkeley: Apress, 2022.
3. SEGALA, Alessandro. Svelte 3 Up and Running: A fast-paced introductory guide to building high-performance web applications with SvelteJS. B.m.: Packt Publishing Ltd, 2020.
4. BODUCH, Adam and Roy DERKS. React and react native. Birmingham, England: Packt, 2020.
5. React – a JavaScript library for building user interfaces- A JavaScript library for building user interfaces [online]. [accessed. 19. September 2022]. Retrieved z: <https://reactjs.org/>
6. Svelte – Cybernetically enhanced web apps [online]. [accessed. 19. September 2022]. Retrieved z: <https://svelte.dev/>

Supervisors of bachelor thesis: **Ing. Radek Vala, Ph.D.**
Department of Informatics and Artificial Intelligence

Date of assignment of bachelor thesis: **October 5, 2022**
Submission deadline of bachelor thesis: **May 12, 2023**



doc. Ing. Jiří Vojtěšek, Ph.D. m.p.
Dean

prof. Mgr. Roman Jašek, Ph.D., DBA m.p.
Head of Department

In Zlín October 10, 2022

I hereby declare that:

- I understand that by submitting my Bachelor's Thesis, I agree to the publication of my work according to Law No. 111/1998, Coll., On Universities and on changes and amendments to other acts (e.g. the Universities Act), as amended by subsequent legislation, without regard to the results of the defence of the thesis.
- I understand that my Bachelor's Thesis will be stored electronically in the university information system and be made available for on-site inspection, and that a copy of the Bachelor's Thesis will be stored in the Reference Library of the Faculty of Applied Informatics, Tomas Bata University in Zlín, and that a copy shall be deposited with my Supervisor.
- I am aware of the fact that my Bachelor's Thesis is fully covered by Act No. 121/2000 Coll. On Copyright, and Rights Related to Copyright, as amended by some other laws (e.g. the Copyright Act), as amended by subsequent legislation; and especially, by §35, Para. 3.
- I understand that, according to §60, Para. 1 of the Copyright Act, TBU in Zlín has the right to conclude licensing agreements relating to the use of scholastic work within the full extent of §12, Para. 4, of the Copyright Act.
- I understand that, according to §60, Para. 2, and Para. 3, of the Copyright Act, I may use my work - Bachelor's Thesis, or grant a license for its use, only if permitted by the licensing agreement concluded between myself and Tomas Bata University in Zlín with a view to the fact that Tomas Bata University in Zlín must be compensated for any reasonable contribution to covering such expenses/costs as invested by them in the creation of the thesis (up until the full actual amount) shall also be a subject of this licensing agreement.
- I understand that, should the elaboration of the Bachelor's Thesis include the use of software provided by Tomas Bata University in Zlín or other such entities strictly for study and research purposes (i.e. only for non-commercial use), the results of my Bachelor's Thesis cannot be used for commercial purposes.
- I understand that, if the output of my Bachelor's Thesis is any software product(s), this/these shall equally be considered as part of the thesis, as well as any source codes, or files from which the project is composed. Not submitting any part of this/these component(s) may be a reason for the non-defence of my thesis.

I herewith declare that:

- I have worked on my thesis alone and duly cited any literature I have used. In the case of the publication of the results of my thesis, I shall be listed as co-author.
- That the submitted version of the thesis and its electronic version uploaded to IS/STAG are both identical.

In Zlín; dated:26/5/2023.....

Student's Signature Sara Arghwan Jameel m.p.

ABSTRAKT

Tato práce porovnává frameworky React a Svelte z hlediska front-end vývoje. React používá virtuální DOM pro efektivní vykreslování, zatímco Svelte kompiluje komponenty do optimalizovaného JavaScript kódu. Práce zkoumá jejich architekturu, syntaxi a výkon. Svelte je považován za srozumitelnější, vyžaduje méně kódu a má lepší využití paměti a dobu načítání. React vyniká v rychlosti vykreslování. Průzkumy a komunity naznačují rostoucí popularitu Svelte. Na základě těchto zjištění je Svelte považován za framework se slibnou budoucností v oblasti vývoje front-endu. Cílem práce je pomoci vývojářům při výběru frameworku, který nejlépe vyhovuje požadavkům a preferencím jejich projektu

Klíčová slova: Svelte, React, frameworky, front-end, výkon

ABSTRACT

This thesis compares the React and Svelte frameworks for front-end development. React uses a virtual DOM for efficient rendering, while Svelte compiles components to optimized JavaScript code. The research examines their architecture, syntax, and performance optimizations. Svelte is considered easier to understand, requires less code, and has better memory usage and load times. React excels in rendering speed. Surveys and communities indicate Svelte's growing popularity. Based on the findings, Svelte can be a framework with a promising future in front-end development. The research aids developers in selecting the framework that best suits their project requirements and preferences.

Keywords: Svelte, React, frameworks, front-end, performance

ACKNOWLEDGMENTS

I would like to take this opportunity to express my deepest appreciation and gratitude to my esteemed supervisor, Dr. Radek Vala, for his unwavering support and invaluable guidance throughout every step of my thesis writing journey. His profound expertise, insightful feedback, and continuous encouragement have played an integral role in shaping the quality and direction of my research. I am truly thankful for the countless hours he dedicated to mentoring me and for his unwavering belief in my capabilities.

Furthermore, I would like to extend my sincere thanks to Dr. Tomas Vogeltanz for his ongoing support and guidance. His valuable input and constructive suggestions have been instrumental in refining my research approach and broadening my understanding of the subject matter. I am truly grateful for his mentorship and the depth of knowledge he shared with me.

I am also deeply indebted to my family for their unwavering love, encouragement, and belief in my abilities. Their constant support has been a pillar of strength throughout my academic journey. Their sacrifices and understanding have provided me with the freedom to pursue my educational goals wholeheartedly.

Moreover, I would like to acknowledge and express my gratitude to the university for providing me with the opportunity to pursue my studies. The exceptional learning environment, resources, and academic community have nurtured my growth and provided me with an enriching platform to expand my knowledge and skills.

In summary, I extend my heartfelt appreciation to Mr. Radek Vala, Mr. Tomas Vogeltanz, my family, and the university for their unwavering support, invaluable contributions, and belief in my academic and personal growth. Their collective guidance and encouragement have shaped me into the researcher and individual I am today, and I am truly grateful for their presence in my academic journey.

INTRODUCTION	10
THEORY	11
1 JAVASCRIPT WEB FRAMEWORKS.....	12
1.1 REACT.....	12
1.2 SVELTE	13
1.3 DOM	13
1.4 JAVASCRIPT FRAMEWORKS.....	15
1.5 TAILWIND CSS.....	16
1.6 PERFORMANCE API.....	17
2 MAIN CONCEPTS OF FRAMEWORKS	18
2.1 REACT	18
2.1.1 KEY FEATURES.....	18
2.1.2 COMPONENTS	18
2.1.3 RENDERING	19
2.1.4 COMMUNITY AND SUPPORTING LIBRARIES	19
2.1.5 JSX	19
2.1.6 LIFECYCLE	20
2.2 SVELTE	20
2.2.1 UNIQUE PERFORMANCE OPTIMIZATION APPROACH	20
2.2.2 UNUSED CSS STYLE CHECKING.....	21
2.2.3 EASE OF USE	21
2.2.4 EFFICIENCY FOR LARGE-SCALE APPLICATIONS	21
2.2.5 LIFECYCLE	21
ANALYSIS	23
3 DESIGN OF APPLICATIONS	24
3.1 REQUIREMENTS	24
3.1.1 FUNCTIONAL REQUIREMENTS	24
3.1.2 NON-FUNCTIONAL REQUIREMENTS.....	25
3.2 USE CASE DIAGRAM.....	26
3.3 WIREFRAMES	27
3.3.1 HOME WIREFRAME	28
3.3.2 BOOKLIST WIREFRAME.....	28
3.3.3 BOOKDEATIL WIREFRAME.....	29
3.3.4 FAVORITE WIREFRAME	30
4 APPLICATION IMPLEMENTATIONS.....	31

4.1	REACT INSTALLATION	31
4.2	SVELTE INSTALLATION.....	32
4.3	FOLDER STRUCTURE.....	32
4.3.1	REACT FOLDERS.....	33
4.3.2	SVELTE FOLDERS.....	33
4.4	CODE STRUCTURE.....	34
4.4.1	REACT CODE STRUCTURE	35
4.4.2	SVELTE CODE STRUCTURE	35
4.5	ROUTING.....	36
4.5.1	ROUTING IN REACT	36
4.5.2	ROUTING IN SVELTE.....	37
4.6	FUNCTIONALITIES	37
4.6.1	API FETCHING	37
4.6.2	FAVORITE.....	38
4.6.3	FAVORITE CHECKER	40
4.6.4	SEARCH.....	41
4.6.5	NAVIGATION TO BOOK DETAILS.....	41
4.6.6	BOOKDETAILS FETCH	42
4.6.7	ADDING COMMENTS AND RATING.....	43
4.6.8	ADDITIONAL INFORMATION	44
4.7	HTML TEMPLATES CREATION.....	45
4.7.1	CONDITIONAL STATEMENTS IN HTML	45
4.8	LOOPING AND ARRAY ITERATION IN HTML	46
4.9	USER INTERFACE.....	47
4.9.1	HOME PAGE USER INTERFACE.....	47
4.9.2	BOOKLIST PAGE USER INTERFACE	49
4.9.3	FAVORITE USER INTERFACE.....	51
4.9.4	BOOKDETAILS USER INTERFACE.....	53
5	PERFORMACE.....	55
5.1	NAVIGATION TIMING	55
5.1.1	NAVIGATION TIMING OF REACT.....	56
5.1.2	NAVIGATION TIMING OF SVELTE.....	57
5.1.3	OVERALL OF NAVIGATION TIMING	58
5.2	USER TIMING.....	59
5.2.1	REACT USER TIMING	60

5.2.2	SVELTE USER TIMING.....	61
5.2.3	OVERALL OF USER TIMING	62
5.3	PERFORMANCE MEMORY	62
5.3.1	PERFORMANCE MEMORY OF REACT.....	63
5.3.2	PERFORMANCE MEMORY OF SVELTE	64
5.3.3	OVERALL OF MEMORY PERFORMANCE	65
5.4	SIZE OF THE APPLICATION.....	66
5.5	FRAMEWORK USAGES	66
5.5.1	RETENTION.....	67
5.5.2	INTERESTED	68
5.5.3	AWARENESS.....	68
5.6	SURVEY.....	69
5.7	GIT COMMUNITY	70
5.7.1	REACT GITHUB COMMUNITY	70
5.7.2	SVELTE GITHUB COMMUNITY	71
5.8	PERSONAL EXPERIENCE AND BACKGROUND	71
5.9	RESEARCH COMPARISON	72
	CONCLUSION	75
	BIBLIOGRAPHY.....	76
	LIST OF ABBREVIATIONS	81
	APPENDICES.....	85
	APPENDIX P I: APPENDIX TITLE.....	86

INTRODUCTION

Front-end development has become a fundamental aspect of creating interactive user interfaces on the web. As the demand for dynamic web applications continues to rise, developers are constantly seeking efficient frameworks, technologies, and languages to streamline their development process. Among the plethora of options available, React and Svelte have emerged as popular frameworks worth considering. React, developed by Facebook, has gained widespread adoption as a JavaScript library for building user interfaces. With its virtual DOM approach, React efficiently manages component state and handles rendering updates. Its mature ecosystem provides a wide array of libraries, tools, and community support, making it a robust choice for developers. In contrast, Svelte is a relatively new framework that takes a unique approach to web application development. By compiling components to highly optimized JavaScript code during the build process, Svelte eliminates the need for a virtual DOM at runtime. This compilation step results in enhanced performance and reduced bundle sizes compared to frameworks like React. Moreover, Svelte's simpler and concise syntax facilitates easier code writing and learning.

The aim of this thesis is to provide valuable insights for individuals interested in understanding these prominent frameworks, React and Svelte. Through an in-depth analysis and comparison, we will explore their key features, advantages, and performance optimizations. By delving into their architectural differences, syntax characteristics, and development efficiencies, readers will gain a comprehensive understanding of these frameworks. This research seeks to empower developers to make informed decisions when selecting a framework for their projects. By examining the strengths, weaknesses, and unique features of React and Svelte, readers will have the necessary information to align their framework choice with project requirements, development preferences, and performance goals. With a thorough review of the frameworks' documentation, practical use in real-world scenarios, and the underlying theory behind them, this thesis aims to present an objective analysis and comparison of React and Svelte. By considering factors such as performance, development efficiency, community support, and learning curve, developers can make well-informed decisions that lead to successful and efficient front-end development.

In the following chapters, we will delve into the detailed analysis of React and Svelte, exploring their capabilities, use cases, and practical implications. Through this exploration, we aim to shed light on the strengths and advantages of these frameworks, helping developers navigate the ever-evolving front-end development landscape.

I. THEORY

1 JAVASCRIPT WEB FRAMEWORKS

JavaScript is a widely used programming language in web development, known for its versatility and compatibility. Svelte, a relatively new JavaScript framework, has gained popularity among web developers due to its ability to reduce code verbosity. [6] On the other hand, React is a widely adopted JavaScript library that focuses on building user interfaces. The Document Object Model (DOM) serves as a programming interface for web browsers, allowing developers to access and manipulate web page elements. [10] React utilizes the DOM to efficiently manage and update components,[1] while Svelte takes a different approach to optimize performance. The objective of this research is to provide valuable insights and recommendations on effectively utilizing the Svelte and React frameworks in various development scenarios. The research will highlight the unique features, strengths, and weaknesses of each framework, aiming to assist developers in making informed decisions when choosing between the two. By exploring their capabilities, practical use cases, and performance optimizations, developers will gain a comprehensive understanding of Svelte and React. This knowledge will enable them to select the most suitable framework based on project requirements, development preferences, and performance objectives. In the following chapters, this research will delve into a detailed analysis of Svelte and React, shedding light on their distinct characteristics. By presenting objective comparisons and recommendations, developers will be equipped with the necessary information to leverage these frameworks effectively in their web development endeavors.

1.1 React

React is a widely popular JavaScript library utilized for building user interface pages with a range of features that make it a preferred choice among developers. One of its most noteworthy features is the React component, which allows for the use of the same user interface for multiple pages. [1] Additionally, the virtual DOM of React is also highly valued, as it streamlines the development process. [2] It is noteworthy that React was initially created by a Facebook engineer, Jordan Walke in 2011, with support from Instagram in 2012. While the library was first released to the public in 2013, it has continued to evolve over the years, [3] with the latest version released in 2022. [4] React is utilized by many renowned companies, including Facebook, Instagram, WhatsApp, Atlassian, Uber Eats, Airbnb, Dropbox, Netflix, Codecademy, and Skyscanner, to name a few. [5] In summary, Reacts

popularity and utility are evident in its features and adoption by many high-profile companies, making it a widely used and respected library for web development.

1.2 Svelte

Svelte is a modern JavaScript framework that has garnered considerable attention from developers due to its intuitive development approach and efficient code-writing capabilities. Unlike other popular JavaScript frameworks like React and Vue, Svelte does not use a virtual DOM, instead relying on direct reflection to the DOM, leading to surgical updates at runtime. [6] Created in 2016 by Rich Harris, a software developer on the New York Times graphics team, Svelte is written in TypeScript and has continued to evolve, with the most recent version released in 2023. The framework has gained popularity among React and Vue developers due to its ease of use and reduced development time, leading to lightweight programs that do not use a virtual DOM. [7] Svelte has gained widespread adoption by prominent companies, with its usage doubling in 2021, as reported on its website. Notable companies such as Apple, Spotify, Square, Rakuten, Bloomberg, Reuters, Ikea, and Brave have all incorporated Svelte into their development process. [6] Apple's use of Svelte for the development of the Apple Music Beta website is an excellent example of the framework's capabilities. [8] Overall, Svelte's popularity is increasing, and it is quickly becoming a preferred choice for developers who desire an efficient and lightweight framework for building web applications. With its intuitive approach, rapid development time, and surgical updates, Svelte is poised to continue its upward trajectory as a viable and powerful framework in the world of web development.

1.3 DOM

The World Wide Web Consortium (W3C) developed the Document Object Model (DOM) in 1998 to provide a standardized programming interface that web browsers can easily understand and utilize. [10] The DOM's primary purpose is to create a representation of the web page, allowing all available data to be accessed quickly and conveniently through three programs. The beauty of the DOM's design is its language-agnostic nature, allowing it to be utilized with any programming language. [9] It is not designed for any language or framework, and developers can use it either manually or through a library or framework that includes it in a JavaScript file. It is advisable to use a separate JavaScript file for manual code addition. The process involves writing JavaScript code that targets different parts of

the document, such as elements, attributes, and text nodes, to access and manipulate the DOM.

This makes it possible to create dynamic web pages with user-page interactions, such as updating page content based on user input or handling asynchronous server requests. Accessing the information through DOM functions [10] further enhances its functionality. The example below demonstrates how the Document Object Model (DOM) interacts with HTML by dividing it into sections and adding subparts, which allows for quicker and easier accessibility. This approach provides a more organized and structured way of working with HTML code. The example below demonstrates how the Document Object Model (DOM) interacts with HTML by dividing it into sections and adding subparts, which allows for quicker and easier accessibility. This approach provides a more organized and structured way of working with HTML code.

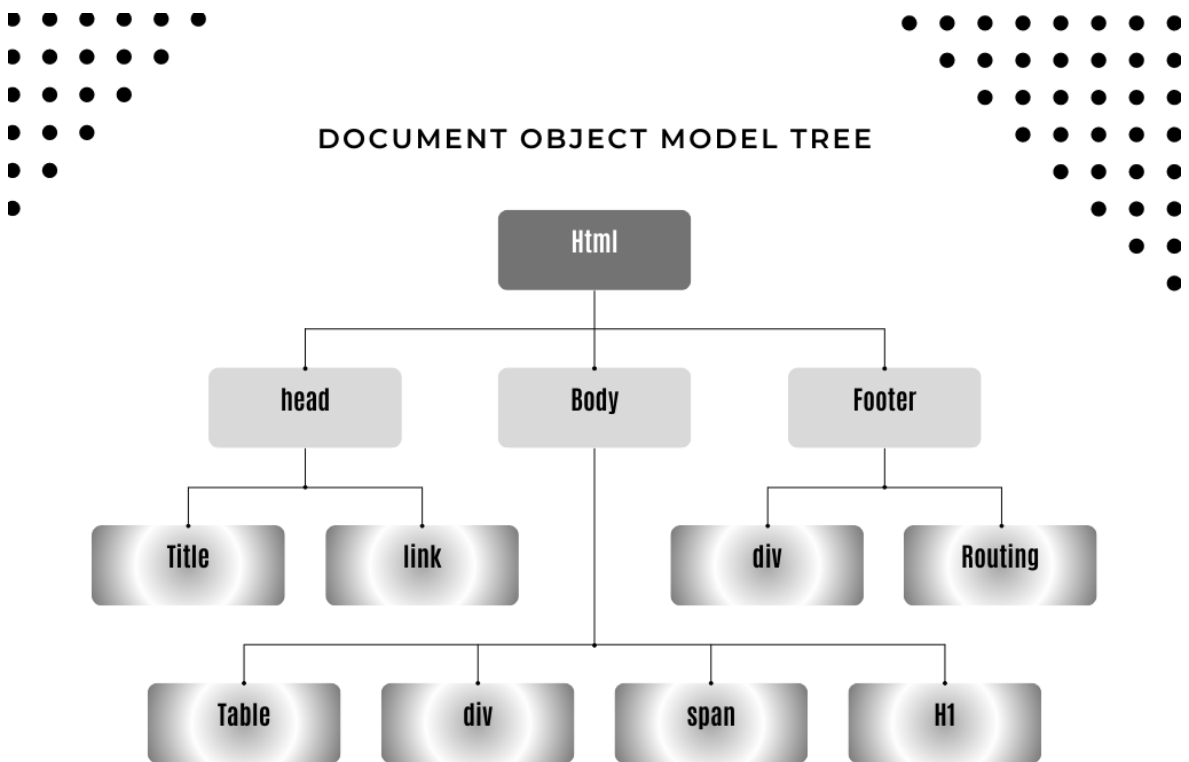


Figure 1: Represent of Document Object Model

To summarize, the Document Object Model (DOM) is a fundamental concept for web developers, as it offers a standardized interface for manipulating web documents that have been widely adopted and implemented in web browsers. The DOM's implementation allows for faster and easier access to web page data and empowers developers to create dynamic and interactive web pages. Understanding the DOM and its functions is therefore essential for web development, as it enables developers to leverage the full potential of this powerful tool.

1.4 JavaScript Frameworks

JavaScript has experienced a surge in popularity among developers, leading to the creation and development of numerous frameworks. [11] As there are multiple frameworks available, developers may find it challenging to select the best one, given that they all aim to offer similar features and approaches, yet each has its unique characteristics. Brendan Eich invented JavaScript in 1995, and it gained popularity in the early 2000s as a programming language for web development. [12] JavaScript enabled developers to perform tasks that were not possible with HTML and CSS alone. With time, many new features were added to JavaScript, simplifying its usage for developers. JavaScript frameworks are collections of pre-written JavaScript libraries that provide a structure for developing programs. Adding these libraries can be difficult and time-consuming, and they may not meet the application's standards being developed. The use of a framework streamlines the development process and maintains consistency in the application's structure. [13] In 2023, the world of web and mobile application development is dominated by multiple JavaScript frameworks, each offering unique features and advantages to developers. Frameworks have become an essential tool for developers, facilitating the creation of websites, web applications, mobile applications, and games with ease. [14]

React, Angular, Vue, Ember, and Backbone are some of the most famous and widely used JavaScript frameworks in 2023. React, developed by Facebook, is a popular choice for building complex and dynamic user interfaces. Angular, developed by Google, is a powerful and feature-rich framework for building web applications.

Vue, developed by Evan You, is a lightweight and flexible framework that is easy to learn and use. Ember, developed by a team of developers, is a full-featured framework that emphasizes convention over configuration. Finally, Backbone, developed by Jeremy Ashkenas, is a lightweight and minimalist framework that provides a solid foundation for building web applications. Given that the popularity of JavaScript frameworks continues to grow, new frameworks are likely to emerge, each with its unique features and advantages. As such, developers must remain up to date with the latest trends and technologies to remain competitive in the fast-paced world of web and mobile application development. [14]

1.5 Tailwind CSS

Since the Tailwind is used for the design aspect, concise description of the framework is provided. Tailwind is a CSS library that is open-source and can be used to design applications. Its purpose is to provide a clearer and more efficient way of writing code by using only the necessary styles. This results in improved performance, especially for responsive web applications. Essentially, Tailwind enables developers to streamline their CSS code and create more efficient and effective web applications. [24] The difference between using Tailwind CSS and writing plain CSS is significant as seen in the chart. With plain CSS, there are many individual classes that must be parsed by the application, whereas with Tailwind CSS, all classes come from a single source and only the necessary classes are used. This greatly reduces the amount of parsing the application must do and results in faster load times and improved performance. In other words, using Tailwind CSS streamlines the process of styling and improves the efficiency of the application. [25]

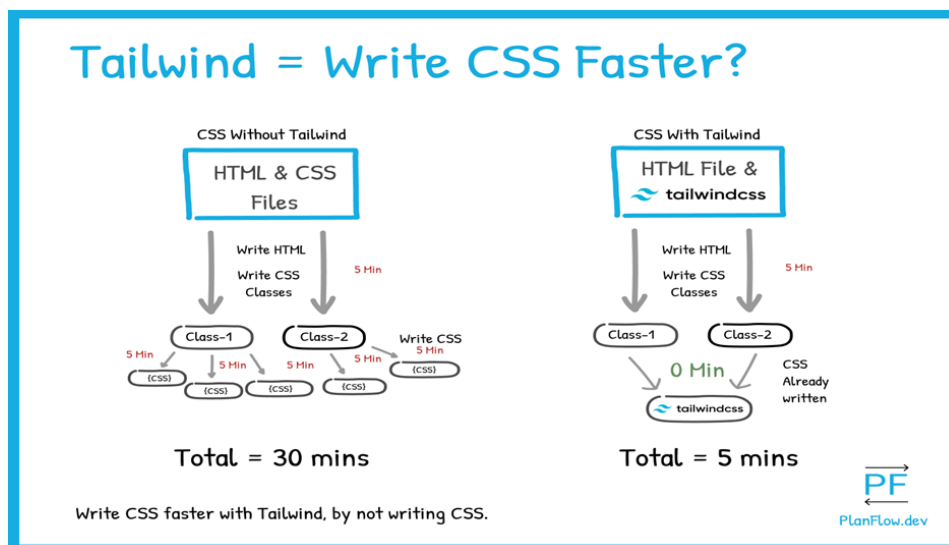


Figure 2: Difference between TailwindCSS and CSS [25]

1.6 Performance API

To accurately measure the performance of an application and account for various factors that can affect it, the Performance API can be used. This API allows us to measure the performance of an application and view the measurements and marks within the developer tools. There are four types of measurement available with this API, which can help to accurately measure the performance of an application regardless of device model or connection quality. By utilizing the Performance API, we can better optimize our application and ensure a smoother user experience. [33]

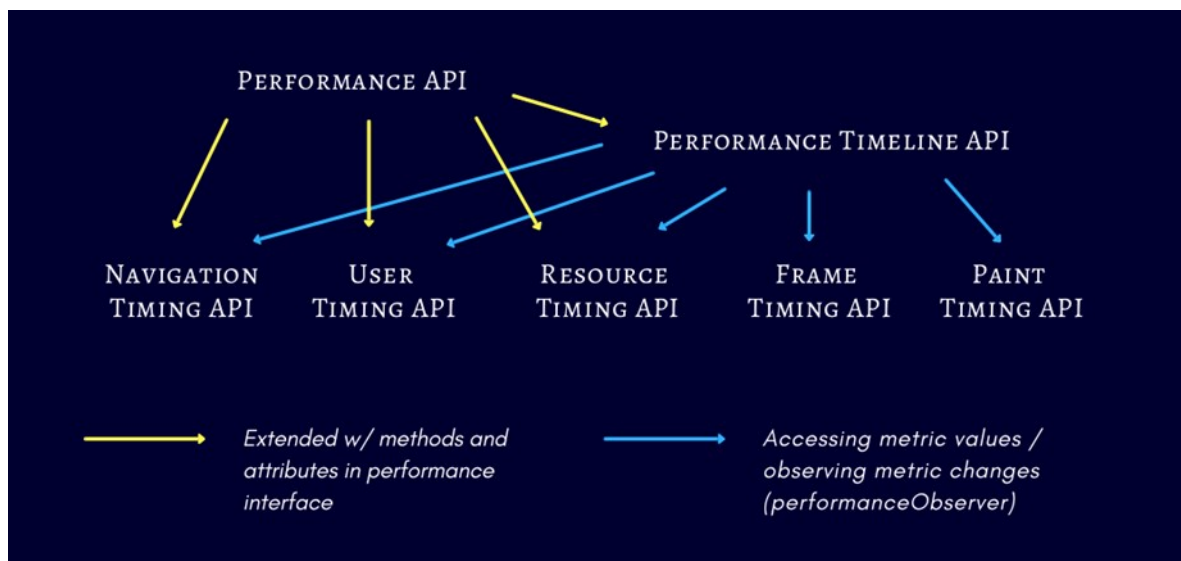


Figure 3: Performance API [33]

2 MAIN CONCEPTS OF FRAMEWORKS

In order to gain a comprehensive understanding of two frameworks or JavaScript library along with their underlying principles and functional capabilities, it is imperative to delve into the key features of each framework and its architecture. By exploring the fundamental concepts and components that define each framework, one can ascertain the various resources and tools that are available for use. Through this approach, a deeper understanding of the framework's structure and functionality can be achieved.

2.1 React

React is a widely used JavaScript library that has gained popularity for its ability to build user interfaces based on components. It follows a one-way data flow model where data is passed down from parent components to child components through props. This approach helps in organizing and managing code effectively.[1]

2.1.1 Key Features

One of the key features of React is its virtual DOM. It provides a lightweight representation of the real DOM, enabling efficient updates by rendering only the necessary components instead of the entire page. This feature is particularly useful for building large and complex applications that require frequent updates and changes.[18]

2.1.2 Components

React components can be categorized into two types: Functional Components and Class Components. Functional Components are simpler and easier to understand as they use ES6 syntax and pure JavaScript to return React elements. On the other hand, Class Components are more complex and offer additional functionality such as state management and lifecycle methods. Both types of components are used to build independent UI components that are not directly connected to each other. This modular approach facilitates easier testing and maintenance of the application.[17]

2.1.3 Rendering

React provides the ability to conditionally render components, allowing them to be displayed only when needed. This feature is crucial for reducing the size of the application and improving its overall performance. Each React component has a rendering function that accepts the React element and specifies where it should be added or mounted in the DOM.[16][18]

2.1.4 Community and Supporting Libraries

React has a thriving community and is supported by numerous libraries and frameworks. For example, React Loadable is a library that enhances application performance by loading resources faster. React Responsive provides a way to create breakpoints in applications using the useMediaQuery hook and Media Query, making it easier to build responsive applications that adapt to different screen sizes and devices. React Beautiful DnD is a library that offers drag-and-drop functionality, enabling users to interact with the application in a more intuitive and engaging manner.[15]

2.1.5 JSX

React utilizes JSX syntax, which stands for JavaScript Extension. JSX allows developers to write UI code and JavaScript code together in a single file. This feature is highly beneficial for users as it improves the readability and maintainability of the code. Although using JSX is not mandatory, it is widely preferred by most programmers.[36]

One significant advantage of JSX is its performance. When compared to normal JavaScript, JSX is faster. It achieves this by leveraging the efficient rendering capabilities of the virtual DOM. By representing the UI elements in a declarative manner, JSX enables React to optimize the rendering process, resulting in improved performance and responsiveness of the application. [37]

2.1.6 Lifecycle

React components go through three main lifecycle phases: mounting, updating, and unmounting. During the mounting phase, a component is created and inserted into the DOM. This involves initializing state, setting up event listeners, and rendering the component's UI. In the updating phase, changes to the component's state or props trigger a re-rendering process. React efficiently updates only the necessary parts of the DOM. The unmounting phase occurs when a component is removed from the project or the DOM. It allows for any necessary cleanup tasks before the component is completely removed.

In conclusion, React is a powerful and flexible library for building high-quality user interfaces. Its component-based architecture, virtual DOM, and conditional rendering make it easy to develop complex applications that are efficient and maintainable. With its large and active community, as well as a wide range of supporting libraries and frameworks, React is an excellent choice for building modern web applications.

2.2 Svelte

Svelte has emerged as a cutting-edge front-end framework that has disrupted the development landscape. It is known for delivering exceptional performance while minimizing code usage, making it a popular choice among developers. The framework draws insights from leading frameworks like React, Angular, and Vue, and innovates their ideas to create a more efficient and streamlined development experience.

2.2.1 Unique Performance Optimization Approach

A key distinguishing feature of Svelte is its unique approach to optimizing performance and usage. Unlike React, Svelte doesn't rely on a virtual DOM. Instead, it compiles code to JavaScript at build time. This approach, inspired by Angular, allows Svelte to perform updates and interactions directly with the DOM. As a result, Svelte offers faster and more efficient operations, especially for large applications.

2.2.2 Unused CSS Style Checking

Svelte offers a standout feature of building and checking unused CSS styles. This capability allows developers to eliminate unnecessary CSS styles that can slow down performance. Svelte ensures that only necessary CSS styles are included, leading to faster loading times and a more streamlined application.

2.2.3 Ease of Use

Svelte's appeal also lies in its ease of use. Developers can write programs in Svelte with basic knowledge of CSS, HTML, and JavaScript. The framework uses classic languages and adds JavaScript markup, enabling developers to add CSS to each component separately without interference. This approach simplifies the debugging process, reducing the time and resources required to resolve issues.

2.2.4 Efficiency for Large-scale Applications

For developers working on large-scale applications, Svelte proves to be an excellent choice. Compared to React projects, Svelte requires 40% less code, resulting in significantly faster and more efficient development. Minimizing unnecessary code simplifies the debugging process and allows developers to focus on building high-quality applications without unnecessary complexity.

2.2.5 Lifecycle

In Svelte, the lifecycle of a component can be divided into three main phases: creation/initialization, update, and destruction/cleanup.

During the creation/initialization phase, the component is set up, and initial values are assigned to data properties.

In the update phase, any changes to the component's state or props are detected, and the component's template is re-evaluated and updated accordingly.

Finally, in the destruction/cleanup phase, the component is removed or destroyed, and any necessary cleanup tasks can be performed, such as unsubscribing from event listeners or releasing resources.

In conclusion, Svelte is a powerful and innovative front-end framework that is revolutionizing the development landscape. Its unique approach to performance optimization and usage, along with its ease of use, makes it an excellent choice for developers of all skill levels. Svelte is rapidly becoming the future of front-end development, and its popularity is expected to continue to grow in the years to come.

II. ANALYSIS

3 DESIGN OF APPLICATIONS

This chapter focuses on the design of an application for the purpose of comparing frameworks. The application will be designed in a straightforward manner, utilizing both frameworks to showcase their differences. The main goal of this chapter is to provide a practical demonstration of the distinctions between the frameworks.

3.1 Requirements

This section aims to examine the criteria that applications should fulfill and the expected outcomes based on their functional and non-functional requirements. These criteria will be categorized into two distinct areas: functional needs and non-functional needs.

3.1.1 Functional Requirements

Functional requirements play a vital role in application development by defining the specific functionalities and capabilities that an application should possess. These requirements serve as a blueprint, outlining the intended behaviors and features that align with the objectives and user needs. By providing clear guidance on the desired functionalities, functional requirements assist developers in effectively shaping the application's development process. Moreover, functional requirements serve as a means of communication between stakeholders, developers, and project participants. They facilitate a shared understanding of the application's scope and purpose, ensuring that all parties involved have a common vision of the expected functionalities. This alignment helps streamline the development process and mitigate misunderstandings, ultimately leading to a more successful outcome. By focusing on user needs and aligning functionalities accordingly, functional requirements contribute to user satisfaction. They help developers prioritize and implement the features that are most important to the target audience, enhancing the overall user experience. By meeting these expectations, applications can increase user engagement and adoption, leading to higher levels of user satisfaction and long-term success.

The inclusion of a rating and comment feature is intended for testing user inputs and is not connected to any server-side functionality. It serves as a simulated representation of such a feature, allowing users to provide ratings and comments as part of their interaction with the application. It is important to note that the rating and comment functionality is solely for demonstration purposes and does not involve any real-time data processing or storage on a server.

Functional Requirements	Description
FR-01-BookList	Users will be able to view a comprehensive list of all books that are currently available.
FR-02-Search	The user will have the capability to look up a specific book by its name.
FR-03-AddFavorite	Users have the option to add their preferred books to a "Favorites" category, which will be saved locally.
FR-04-RemoveFavorite	Users can delete their preferred books from the "Favorites" category, which will be removed locally.
FR-05-BookDetails	Providing information for each book.
FR-06-AddingComment	The user will have the ability to provide comments or reviews on the website.
FR-07-AddRating	The user will be able to assign a rating to the website while leaving their comment.

Table 1: Functional Requirements

3.1.2 Non-Functional Requirements

In order to enhance the performance of the application's functional requirements, the implementation of non-functional requirements will be undertaken. These non-functional requirements focus on aspects beyond the specific functionalities and capabilities of the application, aiming to optimize its overall user experience. By addressing factors such as reliability, scalability, usability, and performance, the application can deliver an enhanced level of performance and meet the expectations and needs of its users.

None-Functional Requirements	Description
NF-01-Styling	Tailwind CSS will be utilized in the design of the application to enhance the user interface.
NF-02-Resposive	The program will be functional on all browsers and able to adjust to various screen sizes.
NF-03-User-Friendly	The application will have a user interface that is easy to use and navigate for all users.
NF-04-Web-Browser	The implementation of the application should be focused on web browser applications.
NF-05-Storage	The capacity to store data locally within a device.

Table 2: None Functional Requirements

3.2 Use Case Diagram

The provided diagram depicts the use case scenario of the application, involving an individual referred to as the User. The User possesses the capability to engage with the application and initiate various actions. These actions include navigating through book details, managing a personalized list of favorite books by adding or removing them, and providing feedback in the form of comments and ratings for the website .

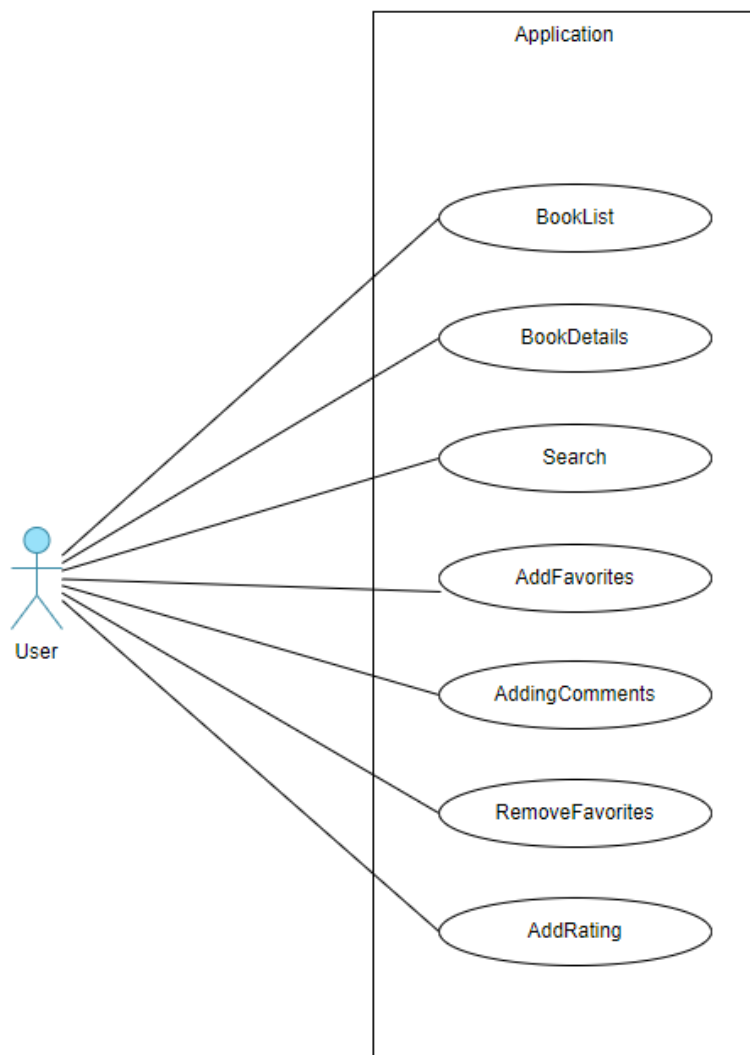


Figure 4: Use Case Diagram

3.3 Wireframes

In this section, a wireframe will be utilized to present the implementation of the application design. The design encompasses four pages, each accompanied by a menu, in order to fulfill various requirements. The first page, known as the home page, serves as the main interface of the application. Furthermore, individual wireframes will be provided for each of the available pages, offering a visual representation of the design and layout of these respective components.

3.3.1 Home Wireframe

This wireframe illustrates the homepage, which functions as the central page of the application. The homepage comprises various elements, including a section for comments and ratings. Additionally, it features a navigation link to the booklist page.

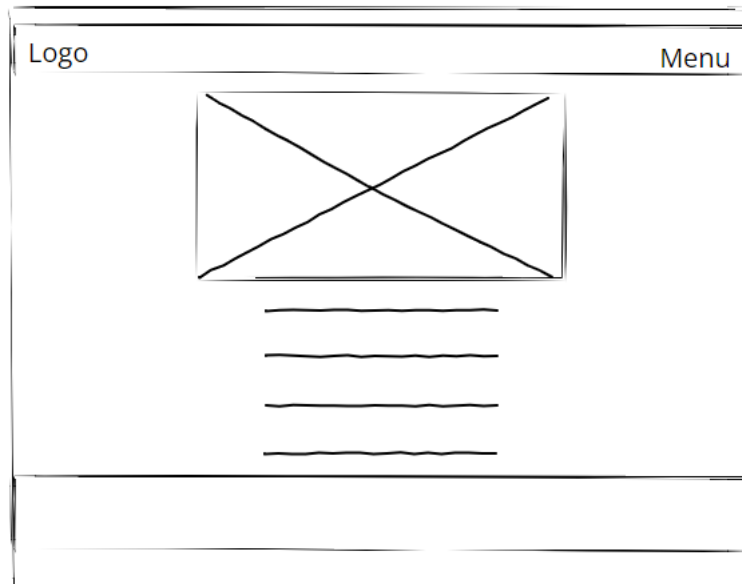


Figure 5: Home Wireframe

3.3.2 BookList Wireframe

This wireframe presents the booklist page, which serves as a comprehensive listing of all available books within the application. Users have the ability to browse through the entire collection of books. Furthermore, the wireframe includes a search functionality that allows users to conveniently search for specific books based on their preferences or criteria.

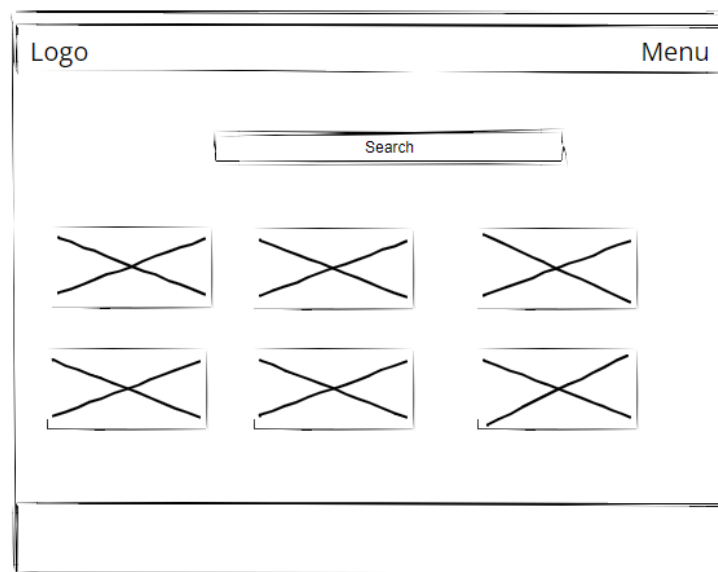


Figure 6: BookList Wireframe

3.3.3 BookDetail Wireframe

This wireframe represents a book details page utilized for the purpose of presenting a comprehensive description pertaining to the selected book.

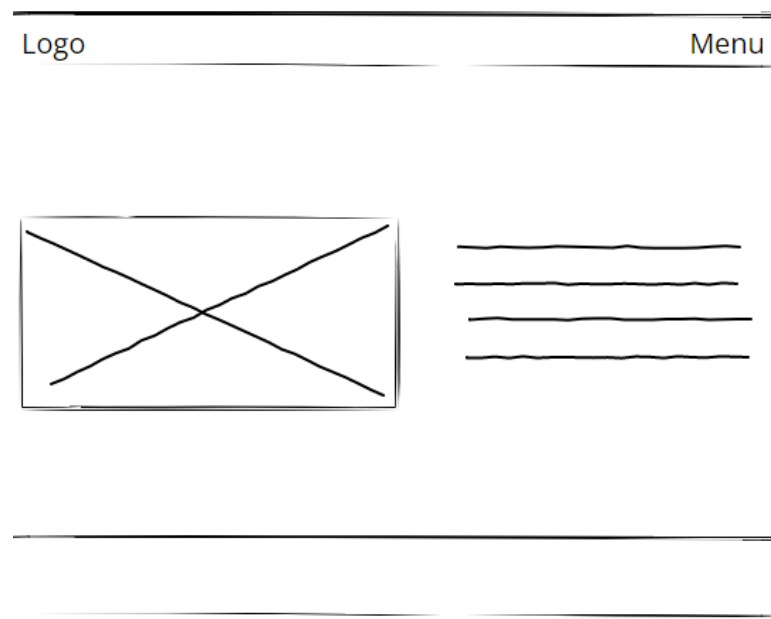


Figure 7: BookDetail Wireframe

3.3.4 Favorite Wireframe

This wireframe illustrates the favorites page, which exhibits a curated compilation of books that users have chosen to include in their personal favorites list.

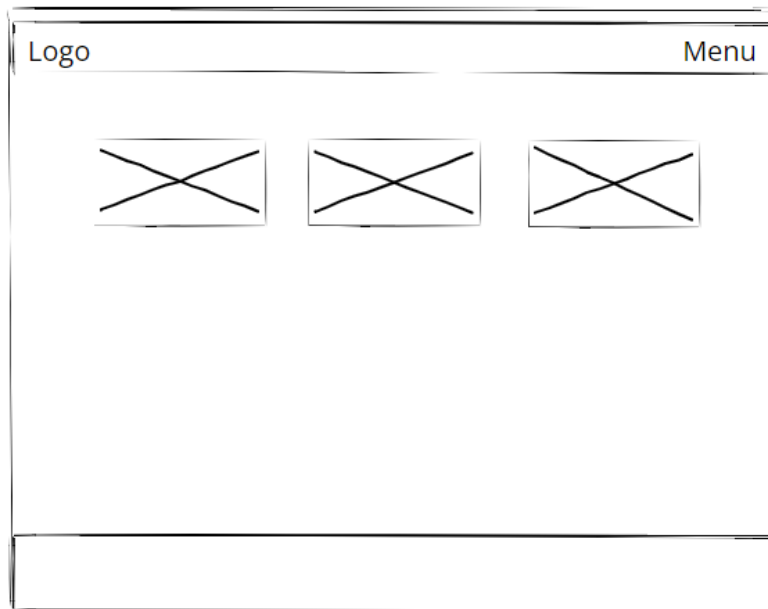


Figure 8: Favorite Wireframe

4 APPLICATION IMPLEMENTATIONS

Two software applications have been developed using different frameworks to highlight their unique features and facilitate user understanding. The aim of this exercise is to compare and contrast the characteristics and functionalities of the two frameworks utilized in the development of these software applications.

4.1 React Installation

This section provides detailed instructions on how to set up and configure a React framework to effectively integrate with the application.

Installation of Framework	
Steps	React + TailwindCSS
1. Installation	<code>npx create-react-app [appName]</code>
2. Redirecting to the project	<code>cd my-project</code>
3. Install Tailwind CSS	<code>npm install -D tailwindcss</code>
4. Generate config.js for tailwind	<code>npx tailwindcss init</code>
5. Customize Tailwind.config.js	content: [<code>"/src/**/*.{js,jsx,ts,tsx}"</code> , <code>]</code> ,
6. Create a CSS file And add to the file	<code>@tailwind base;</code> <code>@tailwind components;</code> <code>@tailwind utilities;</code>
7. Run the Application	<code>npm run start</code>

Table 3: Installation of React

4.2 Svelte Installation

This section offers step-by-step instructions on configuring a Svelte framework to seamlessly integrate with your application and ensure smooth functionality.

Installation of Framework	
Steps	Svelte + TailwindCSS
1. Installation	<code>npm create svelte@latest [appName]</code>
2. Redirecting to the project	<code>cd my-app</code>
3. Install Tailwind CSS	<code>npm install -D tailwindcss postcss autoprefixer</code>
4. Generate config.js for tailwind	<code>npx tailwindcss init -p</code>
5. Customize Tailwind.config.js	<code>content: ['./src/**/*.{html,js,svelte,ts}'],</code>
6. Create CSS File And add it to the file	<code>@tailwind base;</code> <code>@tailwind components;</code> <code>@tailwind utilities;</code>
7. Run the Application	<code>npm run dev</code>

Table 4: Installation of Svelte

4.3 Folder Structure

Comprehending the file structure is vital for effective framework utilization. Each file within the structure holds a distinct purpose, making it essential to provide a brief overview and discuss their significance. Understanding this is of paramount importance.

4.3.1 React Folders

In a React application, the file structure typically consists of two primary sections: the "src" folder and the "public" folder. The "src" folder encompasses the core components, routing configuration, and CSS stylesheets of the application. It serves as the main area for building the application's functionality.

On the other hand, the "public" folder is primarily utilized for displaying icons, images, or other static assets within the application. It acts as a repository for files that are directly served to the user without undergoing any specific processing.

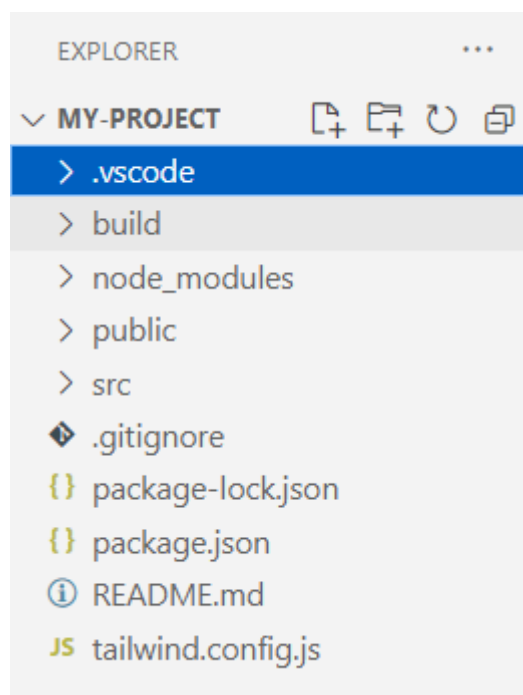


Figure 9: React Folders

4.3.2 Svelte folders

Both Svelte and React exhibit similarities in their folder structures. In Svelte, similar to React, it is common to find an "src" folder that houses components, routing configuration, and CSS styling. This folder serves as the primary location for constructing the application's functionality in Svelte.

To handle the display of images and icons, Svelte developers can employ either a "public" folder, akin to React, or a "static" folder. Both options allow for the inclusion of static files such as images and icons in the application.

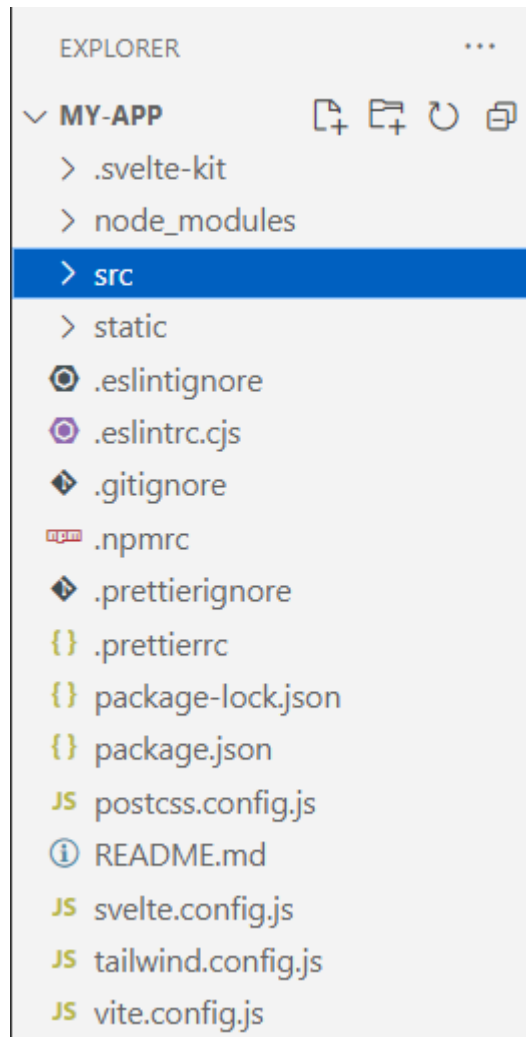


Figure 10:Svelte Folders

4.4 Code structure

To enhance comprehension of the implementation process, it is essential to elucidate the code structure employed by each framework, as they possess distinct styles..

4.4.1 React code structure

React employs a code structure wherein every component is implemented using the JSX extension. JSX enables the combination of HTML-like syntax within JavaScript for enhanced component definition and rendering.

```
// Importing libraries

const ComponentName = () => {

  // Javascript Code
  return (
    <div>
    /*
     |   Html Code
     |   */
    </div>
  );
}
export default ComponentName;
```

Figure 11: Base code of React

4.4.2 Svelte code structure

The code structure utilized for writing Svelte components aligns with the exemplified format, while Svelte components are conventionally stored with the file extension ".svelte".

```
<script>
  // Importing libraries
  // Javascript Code
</script>

/* Html Code */
```

Figure 12: Base code of Svelte

4.5 Routing

Both React and Svelte support routing, and there are similar ways to implement it in both frameworks. In React, you can use "react-router-dom" and import the "Route" and "Routes" components. In Svelte, you can use "svelte-routing" and import the "Router" and "Route" components. With these components, you can navigate between pages in your application. The code blocks for implementing routing in both frameworks are similar, with the only difference being the syntax used to write the code.

4.5.1 Routing in React

In React, you can utilize routing to enable navigation between pages. Essentially, you define a unique route for each page, and you can access a specific page by using its corresponding route name.

```
return (  
  <div className='App'>  
    <NavigationMenu />  
    <Routes>  
      <Route path="/" element={<Main />} />  
      <Route path="/books" element={<Books />} />  
      <Route path="/favorites" element={<Favorites />} />  
      <Route path="/books/:id" element={<BookDetails />} />  
    </Routes>  
    <Footer />  
  </div>  
}
```

Figure 13: Routing in React

4.5.2 Routing in Svelte

In Svelte, a similar approach is followed for routing by importing the necessary routing components or libraries. This allows you to define routes for your pages and navigate between them.

```
<div>
  <Menu />
  <Router>
    <Route path="/" component={Main} />
    <Route path="/Home" component={Home} />
    <Route path="/favorite" component={Favorite} />
    <Route path="/books/:id" component={BookDetails} />
  </Router>
  <Footer />
</div>
```

Figure 14: Routing in Svelte

4.6 Functionalities

There are multiple functions created to perform different tasks in my applications, and I will be discussing the variations between them.

4.6.1 API fetching

When fetching data from an API in both React and Svelte, I used Axios and followed the same functionality and methods. However, the main difference between the two frameworks was in how I kept track of changes in the API data. In React, I used hooks like `useState` and `useEffect` to manage state and lifecycle events and keep track of the data changes. In contrast, in Svelte I simply declared the variable and relied on Svelte's reactivity system to automatically update the data. This difference in approach helps to highlight the different philosophies and design patterns of the two frameworks.

```
const [books, setBooks] = useState([]);
useEffect(() => {
  axios
    .get(API_URL)
    .then((res) => {
      console.log(res.data.books);
      setBooks(res.data.books);
    })
    .catch((err) => console.log(err));
}, []);
```

Figure 15: Fetching in React

```
let books = [];
axios
  .get(API_URL)
  .then((response) => {
    books = response.data.books;
    console.log(books);
  })
  .catch((error) => {
    console.log(error);
  });
});
```

Figure 16: Fetching in Svelte

4.6.2 Favorite

In the application analysis, it was noted that users can add and remove books to and from their favorites list. To implement this functionality, different approaches were taken in React and Svelte. In React, the "context" technique was utilized to pass data and prevent code duplication. Meanwhile, in Svelte, parameters were used to share information between components. Additionally, in both frameworks, the favorite books array was stored in local storage to ensure that the list was saved and persisted even after the user closed or refreshed the page.

4.6.2.1 Adding and Removing Favorites in React

The "addToFavorites" function allows the user to add a book to their favorites array. It creates a new array called "newFavorites" by combining the existing favorites with the new book, and then stores this updated array in the local storage.

```
const addToFavorites = (book) => {  
  const newFavorites = [...favorites, book];  
  setFavorites(newFavorites);  
  localStorage.setItem('favorites', JSON.stringify(newFavorites));  
};
```

Figure 17: Adding to Favorites React

The "removeFromFavorites" function is implemented by checking the book's ID in the favorites array. If a book with a matching ID is found, it is removed from the favorites array. Then, the new favorites array is set, and the updated array is saved in the local storage.

```
const removeFromFavorites = (_id) => {  
  const newFavorites = favorites.filter((book) => book._id !== _id);  
  setFavorites(newFavorites);  
  localStorage.setItem('favorites', JSON.stringify(newFavorites));  
};
```

Figure 18: Removing from Favorites React

4.6.2.2 Adding and Removing Favorites in Svelte

The process of adding and removing books from the favorites is implemented by retrieving the book's ID and comparing it within the favorites array. If the ID exists in the array, the book is removed using the splice method. This ensures that the remove function works correctly. On the other hand, if the ID does not exist, the book is pushed to the favorites array. Finally, the updated favorites array is saved locally in the browser's local storage.

```
function toggleFavorite(book) {
  const updatedFavorites = [...favorites];
  const index = updatedFavorites.findIndex((fav) => fav._id === book._id);

  if (index > -1) {
    updatedFavorites.splice(index, 1);
  } else {
    updatedFavorites.push(book);
  }

  favorites = updatedFavorites;
  localStorage.setItem('favorites', JSON.stringify(updatedFavorites));
}
```

Figure 19: Favorite Adding and Removing in Svelte

4.6.3 Favorite checker

To prevent duplicate entries on the favorites page, I implemented a function that checks if a book already exists in the favorites list. If the book exists in the list, the function allows the user to remove it. If not, the user can add it to the list. This is achieved by checking the book ID to ensure that each book is unique in the list.

```
const favoritesChecker = (_id) => {
  return favorites.some((book) => book._id === _id);
};
```

Figure 20: Checking Favorite list in React

The development of the same functionality in Svelte was done as follows:

```
function favoritesChecker(bookId) {  
  return favorites.some((fav) => fav.id === bookId);  
}
```

Figure 21: Checking Favorite list in Svelte

4.6.4 Search

Our applications have a feature that allows users to search for books by their titles. This feature was implemented in both frameworks in a similar manner.

```
let filteredBooks = [];  
$: {  
  filteredBooks =  
    searchQuery === ''  
      ? books  
      : books.filter((book) => {  
        const title = book.title.toLowerCase();  
        const query = searchQuery.toLowerCase();  
        return title.includes(query);  
      });  
}
```

Figure 23: Search in Svelte

4.6.5 Navigation to book Details

To enable users to view book details, another function had to be implemented. This was done by creating a new page and passing the book's information based on its ID when it was clicked. In both React and Svelte, navigation was used to accomplish this.

```
const navigate = useNavigate();
```

Figure 24: Navigation to BookDetails in both Frameworks

When the book is clicked, the button's event handler triggers a navigation action by passing the book's ID. This navigation action directs the user to a specific page or view associated with the clicked book's ID.

```
navigate(`/books/${book._id}`)
```

Figure 25: Button Event handler for Navigation

4.6.6 BookDetails Fetch

In the book details component, the book's ID is shared to fetch data from the API. This was achieved by passing parameters. updating data in React requires `useEffect`, but in Svelte there is only assignment to reactive propert

```
useEffect(() => {
  axios
    .get(`${BookDetails_URL}/${id}`)
    .then((res) => {
      console.log(res.data);
      setBook(res.data['book']);
      setIsLoading(false);
    })
    .catch((err) => {
      console.log(err);
      setError(err.message);
      setIsLoading(false);
    });
}, [id]);
```

Figure 26: Fetching BookDetail in React

```
onMount(async () => {
  try {
    const response = await axios.get(`${BookDetails_URL}/${id}`);
    book = response.data['book'];
    console.log(book);
  } catch (error) {
    console.log(error);
  }
});
```

Figure 27: Fetching BookDetails in Svelte

4.6.7 Adding Comments and Rating

One of the functionalities of our application is the ability for users to leave comments. This feature allows users to share their opinions, provide feedback, and engage in discussions within the application. By leaving comments, users can interact with content and communicate their thoughts effectively.

4.6.7.1 Commenting and Rating in React

In the React implementation for adding and rating comments, the code checks if the comment is not empty before submitting it. If the comment is not empty, it will be submitted. Otherwise, if the comment is empty, it will not be submitted. After submitting, the value of the comment is set to null using the `useState` hook, preparing it for the next comment entry.

```
const handleSubmitComment = (event) => {
  event.preventDefault();
  if (comment.trim() !== '') {
    setComments([...comments, { comment, rating }]);
    setComment('');
    setRating(0);
  }
};
```

Figure 28: SubmittingComment and Rating in React

4.6.7.2 *Commenting and Rating in Svelte*

The same concept was applied when implementing the functionality in Svelte, following a similar approach. However, instead of using the `useState` hook in React, an empty array was used for storing comments, and separate variables were used for rating and comment in Svelte. The overall logic and behavior remain consistent, with the difference lying in the specific syntax and conventions of each framework.

```
function handleSubmitComment(event) {
  event.preventDefault();
  if (comment.trim() !== '') {
    comments = [...comments, { comment, rating }];
    comment = '';
    rating = 0;
  }
}
```

Figure 29: Submitting Comment and Rating in Svelte

4.6.8 **Additional Information**

Another functionality implemented after displaying the description of each book is the ability for users to expand or show less of the description. This allows for a more user-friendly experience, especially for longer descriptions.

4.6.8.1 *Additional Information in React*

The `toggleDescription` function updates the state of `showFullDescription` by toggling its value. It uses the `setShowFullDescription` function provided by React's `useState` hook. When called, it inverses the current value of `showFullDescription`, effectively showing or hiding the full description depending on its previous state.

```
const toggleDescription = () => {
  setShowFullDescription(!showFullDescription);
};
```

Figure 30: Additional Information in React

4.6.8.2 Additional Information in Svelte

The code allows users to toggle the display of the book description. When the `toggleDescription` function is called, it flips the value of the `showFullDescription` variable. After the update, if `showFullDescription` is true, the page smoothly scrolls to the 'full-description' element.

```
const toggleDescription = () => {
  showFullDescription = !showFullDescription;
};
afterUpdate(() => {
  if (showFullDescription) {
    const descriptionElement = document.getElementById('full-description');
    descriptionElement.scrollIntoView({ behavior: 'smooth' });
  }
});
```

Figure 31: Additional Information in Svelte

4.7 HTML Templates Creation

Given the extensive amount of HTML and CSS code, will focus on the essential elements and their implementation in both frameworks to enhance comprehension. By highlighting these key aspects, readers will gain a better understanding of how HTML and CSS are utilized in each framework.

4.7.1 Conditional Statements in HTML

To implement the functionality where the color and functionality of the button change based on whether a book is already added to the favorite list, conditional statements such as "if" and "else" can be utilized. Using these statements, you can determine if a book exists in the favorite list and adjust the appearance and behavior of the button accordingly. If the book is already added to the list, the button will be displayed as a "Remove" button. Conversely, if the book is not in the favorite list, the button will be displayed as an "Add" button. By employing "if" and "else" statements, you can dynamically modify the button based on the book's presence or absence in the favorite list.

```
{favoritesChecker(book._id) ? (...
) : (...
)}
```

Figure 32: Conditioning in Html Tags React

```
{#if favoritesChecker(book._id)}
<button ...
</button>
{:else}
<button ...
</button>
{/if}
```

Figure 33; Conditioning in Html Tags Svelte

4.8 Looping and Array Iteration in HTML

In programming, loops are employed to enhance efficiency and customize code. They allow us to iterate through an array until the last index, repeating specific operations until that point is reached. To achieve this, I utilized the "map" function in React, which enables iteration through an array and displaying missing values. On the other hand, in Svelte, I utilized the "forEach" method, which allows looping through an array and returning the modified array. By utilizing these functionalities, the code becomes more concise and tailored to the specific requirements of the program.

```
<div className="container mx-auto px-4 py-6">
  <div className="max-w-screen-xl mx-auto">
    <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-8">
      {filteredBooks.map((book) => (
        <div ...
      </div>
      ))}
    </div>
  </div>
```

Figure 34: Mapping in React

```
<div class="container mx-auto px-4 py-6">
  <div class="max-w-screen-xl mx-auto">
    <div class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-8">
      {#each filteredBooks as book}...
    </div>
  </div>
</div>
```

Figure 35: Looping in Svelte

4.9 User Interface

The purpose of developing two applications with identical functionality and user interfaces was to examine their differences in terms of operation and ensure compatibility across multiple platforms. The main goal was to create user-friendly applications that could seamlessly function on any platform.

4.9.1 Home page User Interface

The home page is designed to provide a welcoming experience for the user. It includes a brief description of the application and a menu bar for easy navigation. Additionally, there is a section dedicated to displaying a list of books for quick access. The design is user-friendly and responsive, meaning it adapts well to different screen sizes and devices. Furthermore, there is a section where users can leave comments and ratings for the website, enhancing interactivity and user engagement.

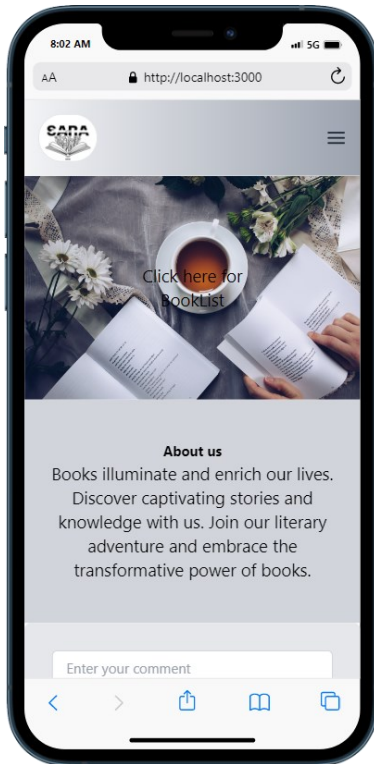


Figure 36:Home page “iPhone”

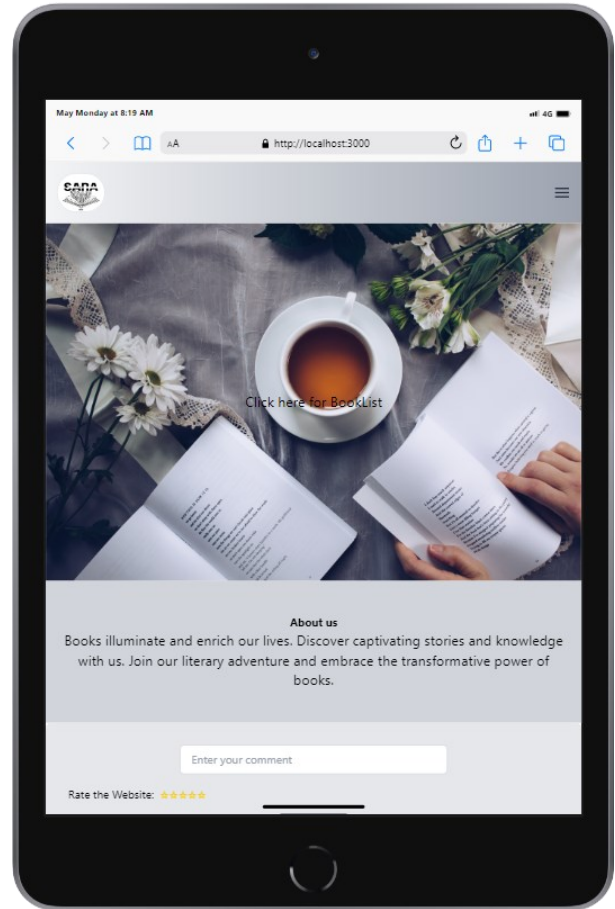


Figure 37:Home page “iPad”

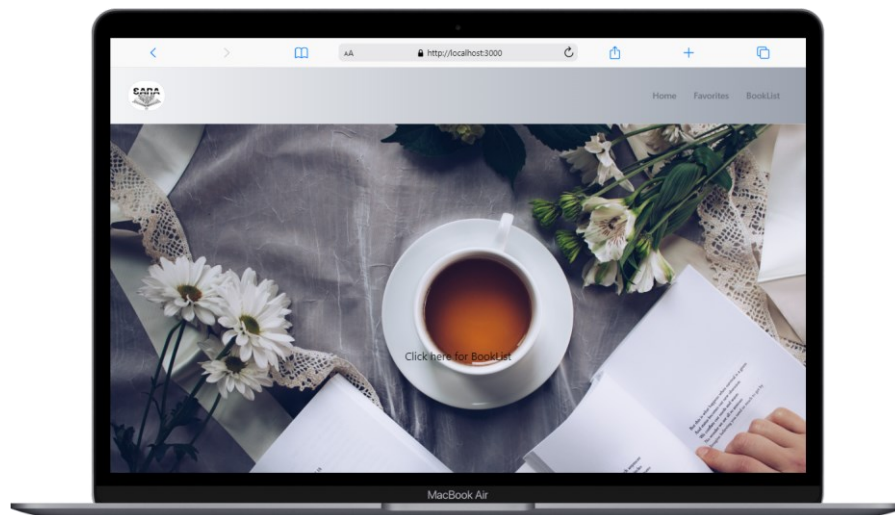


Figure 38: Home page “MacBook Air”

4.9.2 BookList page User Interface

We have designed a BookList page that displays a list of books fetched from an API. The page includes a menu, a search bar for users to search for books, and buttons to add or remove books. Additionally, there is a details button to provide more information about each book. We have ensured that the UI of the Home page is optimized for various landscapes, including iPad, iPhone, and desktop, allowing users to have a consistent and user-friendly experience across different devices.

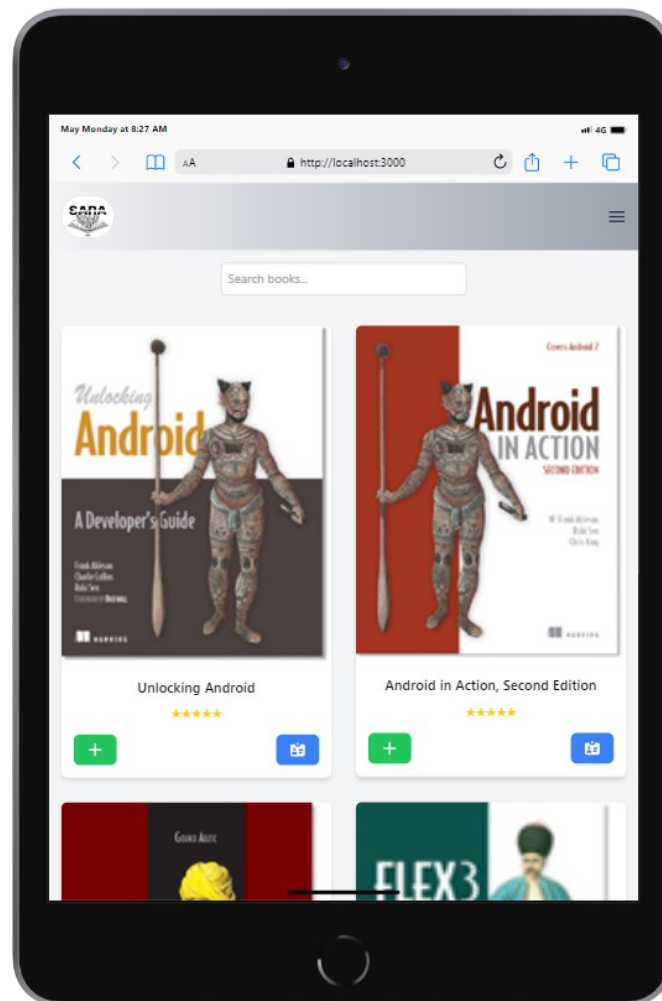


Figure 39: BookList “iPad”

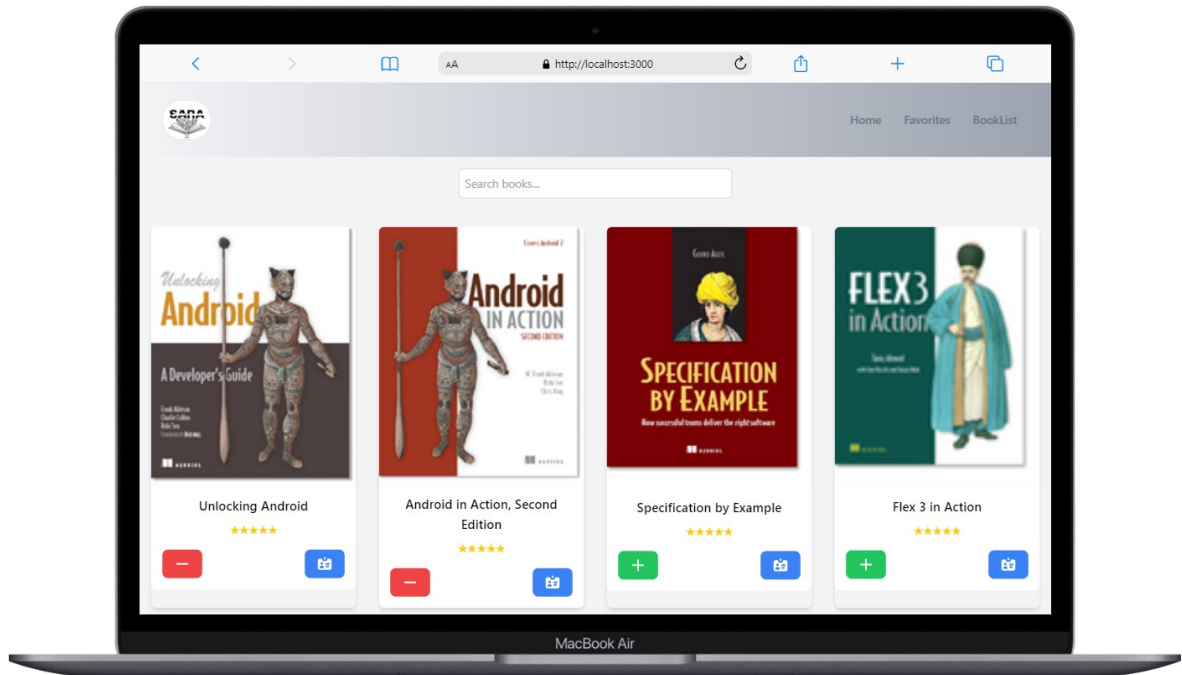


Figure 41: BookList “MacBook Air”

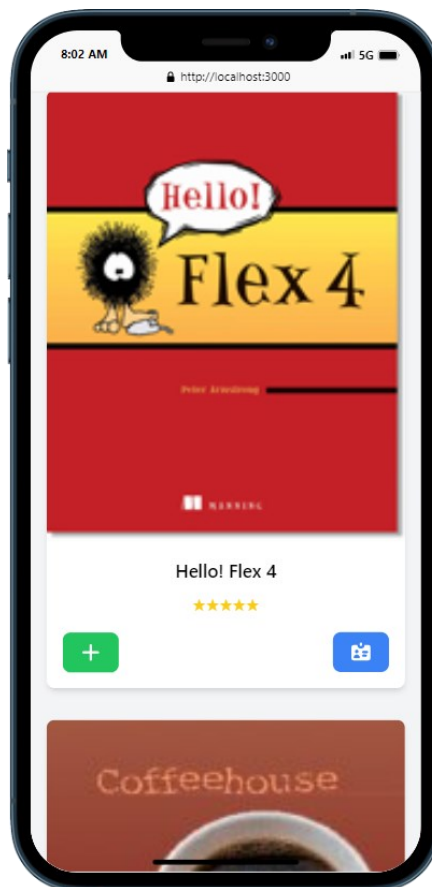


Figure 40: BookList “iPhone”

4.9.3 Favorite User Interface

Following the development of the Home page, a Favorites page was also created. On this page, users have the ability to store a list of favorite books in an array. When a book is added to the favorites, the corresponding button's state is changed to a "remove" button, allowing users to easily remove the book from their favorites. Additionally, users can access detailed information about each book on this page. The user interface of the Favorites page has been designed to be compatible and visually appealing across different landscape orientations, ensuring a consistent and enjoyable user experience on various devices.

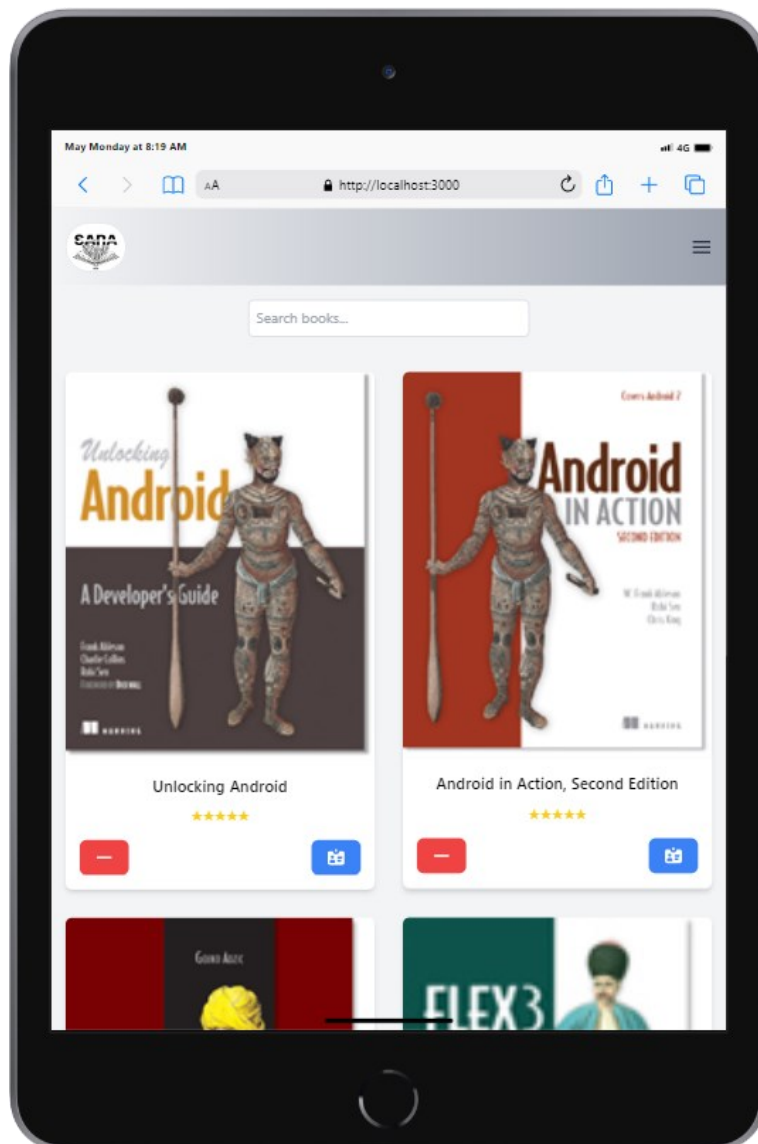


Figure 42: Favorites “iPad”

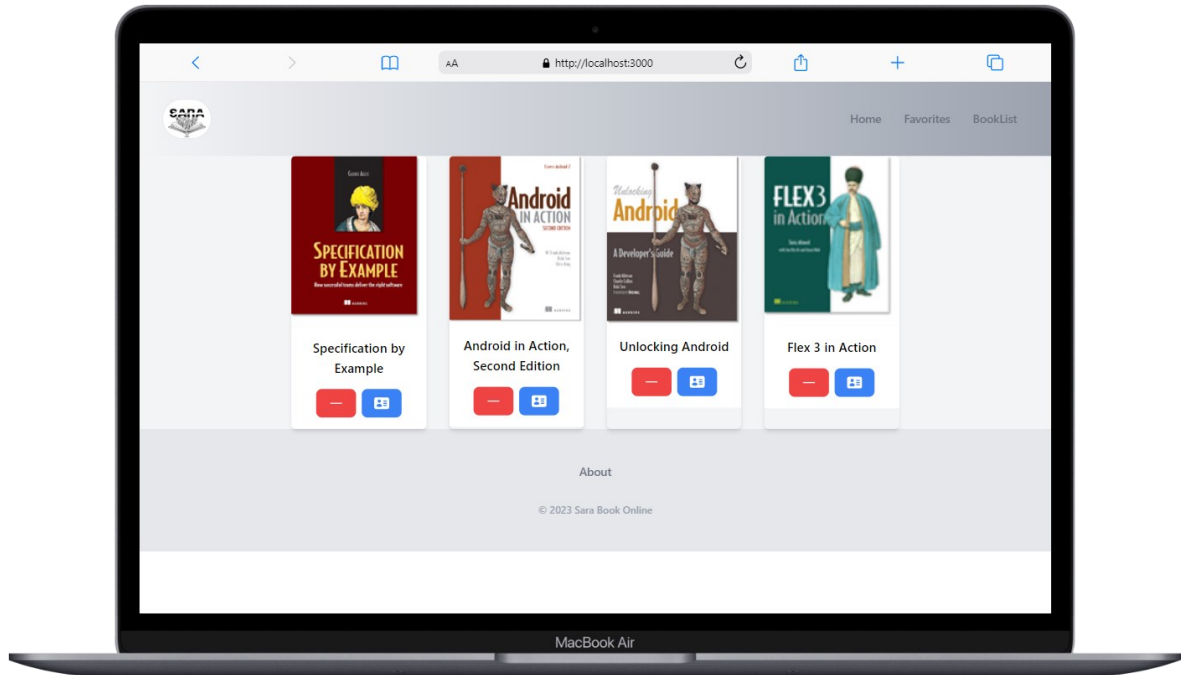


Figure 43: Favorites “MacBook Air”



Figure 44: Favorites “iPhone”

4.9.4 BookDetails User interface

There is a button responsible for navigating to the book details page. When this button is clicked, the user is directed to a page that provides additional information about the book. The book details page includes a concise description of the book, the author's name, the book's title, and the number of pages it contains.

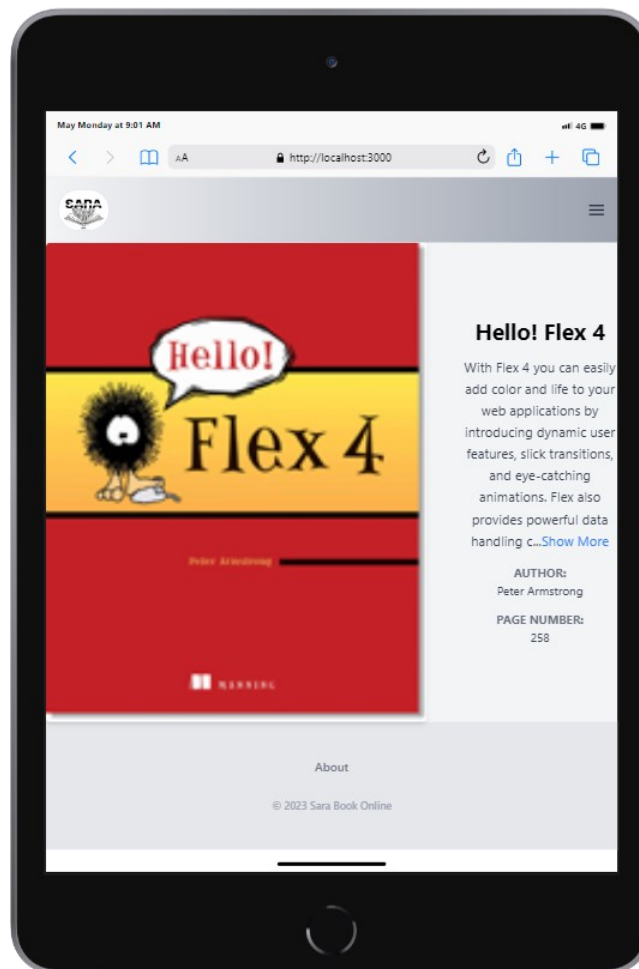


Figure 45: BookDetails “iPad”

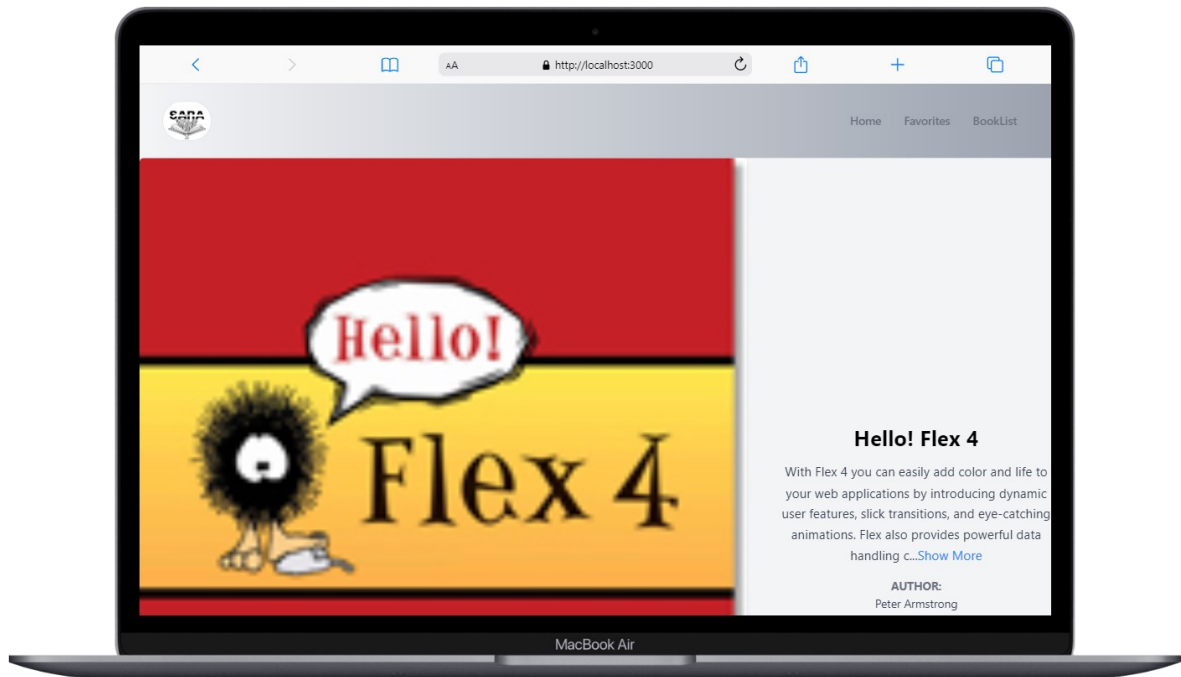


Figure 46: BookDetails “MacBook Air”

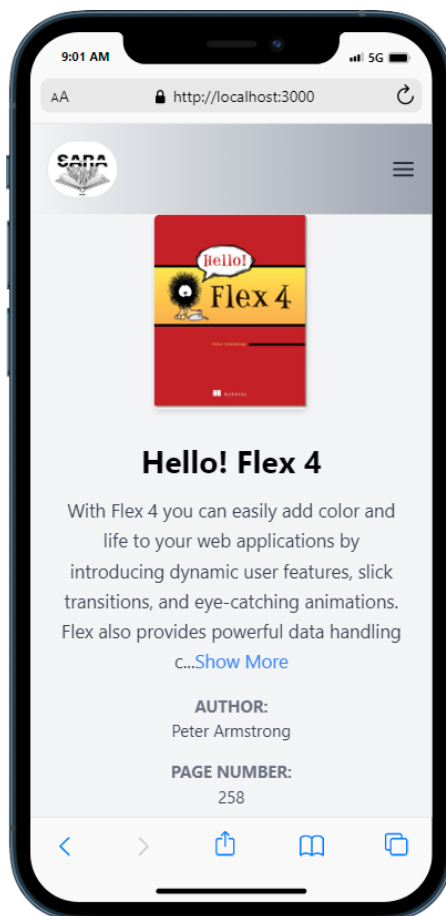


Figure 47: BookDetails “iPhone”

5 PERFORMANCE

To evaluate the performance of different frameworks and assess their size and loading, 20 tests were conducted for each framework and calculated the average values for each metric.

5.1 Navigation Timing

To measure the performance of the application objectively, our focus is on evaluating the loading and processing times of each framework. Specifically, we are interested in analyzing the time it takes for the DOM to become interactive and for processing to complete, as shown in the accompanying diagram. The DOM becomes interactive when it is ready for manipulation, regardless of any ongoing data loading. Processing is considered complete when the DOM has finished rendering.

We will also compare the overall loading time of the application, which is calculated as the difference between the start and end times of the load event. This provides a standardized measure of the time required for the application to be fully loaded and ready for use. The purpose of examining the load time and processing performance is to objectively compare how the real DOM and virtual DOM are updated. While the Svelte framework does not employ a virtual DOM, React utilizes a virtual DOM. The objective is to measure and analyze the speed at which these frameworks can load and process data when implementing the same functionality within the application. This analysis aims to determine the relative efficiency and speed of data loading and processing between the two frameworks, without introducing any subjective bias.[26]

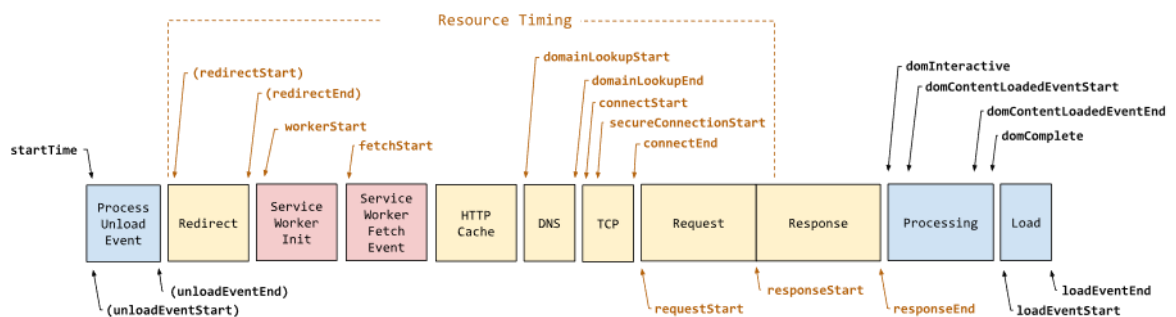


Figure 48: Load and Process counting [26]

5.1.1 Navigation Timing of React

The following table displays the results obtained from testing a React application using Navigation Timing to load 250 books and measure the application's performance 20 times. By calculating the average of the results, we can obtain more accurate current results for the loading and processing of the React application .results for each variable is available by getting the difference between each start and end time.

TestNumber	DomInteractive [ms]	DomComplete [ms]	LoadEventEnd [ms]	LoadEventStart [ms]
1	503.4	719.5	719.8	719.8
2	398.7	1285.7	1285.9	1285.9
3	391.4	829.5	829.7	829.7
4	399	713.7	714	714
5	400.5	782.1	782.7	782.7
6	400.2	825.5	825.9	825.9
7	409.3	744.3	744.7	744.7
8	394.2	819.2	819.4	819.4
9	400	731.3	731.3	731.3
10	396	692.9	693.2	693.2
11	428.3	1593	1593.3	1593.3
12	392.8	721.7	721.8	721.8
13	397.1	728.2	728.5	728.4
14	413	788.8	789.1	789.1
15	91.9	596.7	597	596.9
16	405.7	735.7	736	736
17	121.5	477.6	477.9	477.9
18	401.1	788.9	789.2	789.2
19	106.3	456.3	456.5	456.5
20	404.3	855	855.2	855.1
Average	352.78	794.28	794.56	794.54

Table 5: Navigation Timing for React

Variable	Calculations
Process= DOM Complete - DOM Interactive	441.5 ms
Load= Load Event End - Load Event Start	0.02 ms

Table 6: Navigation Results in React

5.1.2 Navigation Timing of Svelte

The performance timing for the Svelte application was also calculated using the same method as the React application. One observation that caught my attention was that the load timing was very close to when the DOM is complete, with only a few milliseconds

TestNumber	DomInteractive [ms]	DomComplete [ms]	LoadEventStart [ms]	LoadEventEnd [ms]
1	126.2	145.5	145.5	145.5
2	149.8	233.5	233.5	233.5
3	143.1	165.5	165.5	165.5
4	138.7	202.3	202.3	202.3
5	164.7	293.9	293.9	293.9
6	145.3	173.1	173.1	173.1
7	150.9	193.1	193.2	193.2
8	161	183.8	183.8	183.8
9	131.8	244.4	244.5	244.5
10	140.7	209.8	209.8	209.8
11	173.1	219.1	219.1	219.1
12	154.5	232.3	232.3	232.3
13	174	227.3	227.3	227.3
14	168.4	205.4	205.4	205.4
15	143.8	229.8	229.8	229.8
16	137	162.2	162.2	162.2
17	134.5	157.8	157.8	157.8
18	122.7	253.2	253.3	253.3
19	134	314	314	314
20	132.7	153.7	153.7	153.7
Average	146.35	210	210	210

Table 7: Navigation Timing for Svelte

Variable	Calculations
Process= DOM Complete - DOM Interactive	63.65 ms
Load= Load Event End - Load Event Start	0 ms

Table 8: Navigation Results in Svelte

5.1.3 Overall of Navigation Timing

After obtaining the results, we can visualize the disparity in loading and processing times of each framework using a chart table. This table will provide a clear representation of the differences observed.

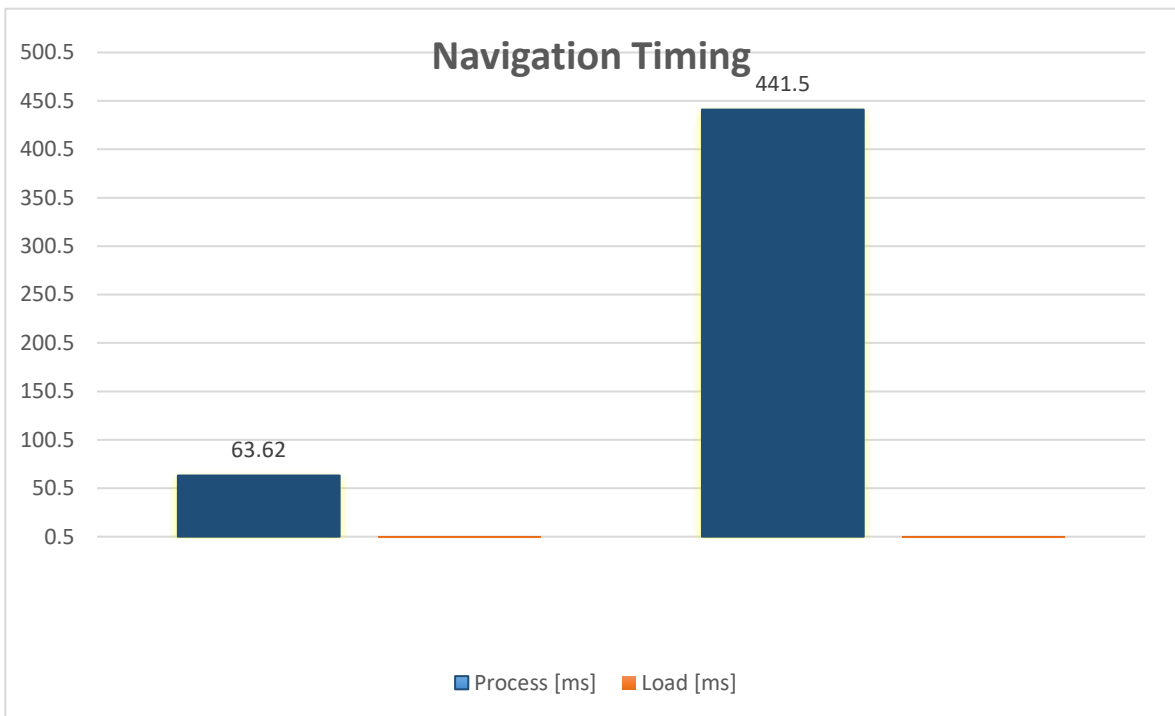


Table 9: Difference between Frameworks for Navigation Timing

5.2 User Timing

To test the timing of each framework while performing a task, you can utilize the "User Timing" API, which allows you to measure the timing of a specific task. This involves creating marks at the start and end of the task, and then using the "performance.measure()" method to measure the duration between the marks. By implementing this approach, you can accurately measure the execution time of different tasks within each framework and compare their performance.[34] After completing the navigation timing analysis, another aspect that caught my attention was the time required to display all the books. To measure this, I began utilizing user timing by tracking the rendering time of the last element in the array. This element typically includes an image, the book's title, and buttons for adding, removing, and accessing details. By measuring the time it takes to render this final element, I can gain insights into the overall performance of displaying the entire collection of books.

5.2.1 React user Timing

The timing measurement was performed for the last index of the array during the book display process. This measurement was conducted using the same method, which involved having a total of 250 books and conducting 20 tests per framework.

TestNumber	Duration [ms]
1	9.6
2	8.3
3	11.3
4	14.5
5	8.5
6	8.8
7	8.6
8	6.4
9	10.3
10	12.6
11	8.5
12	11.4
13	10.4
14	13.1
15	15.1
16	12.8
17	10.1
18	7.7
19	7.8
20	8.2
Average	10.2

Table 10: User Timing in React

5.2.2 Svelte user timing

The identical method used in React to measure User Timing for a specific task was also employed in Svelte. This allowed for consistent measurement and comparison of Timing between the two frameworks for the same task.

TestNumber	Durattion [ms]
1	91.3
2	98.5
3	91.8
4	104.1
5	57.2
6	92.3
7	113.1
8	106.7
9	64.6
10	94
11	105.1
12	97.1
13	66.1
14	75
15	57.6
16	113.3
17	109.2
18	92.2
19	63.2
20	70.5
Avarage	88.15

Table 11: User Timing in Svelte

5.2.3 Overall of User Timing

After conducting 20 tests and calculating the average duration for each framework, two results were obtained. The difference between these results is presented in the chart below:

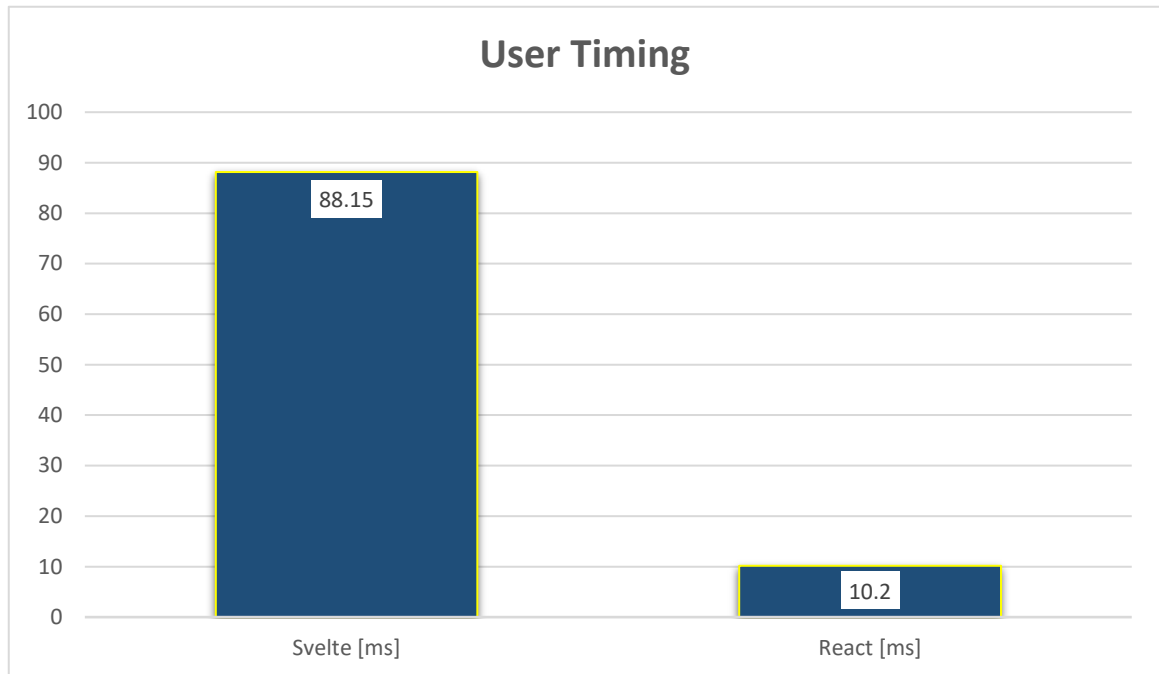


Figure 49: Difference between Frameworks for User Timing

5.3 Performace Memory

In order to obtain more precise information about the actual size of JavaScript in each application and gain deeper insights into memory usage, the Memory API performance tool can be utilized. By comparing the obtained results, we can extract valuable data such as the 'jsHeapSizeLimit', which represents the overall size of data utilized by JavaScript. Additionally, the 'totalJSHeapSize' provides the current total memory consumption in bytes by JavaScript. Lastly, the 'usedJSHeapSize' indicates the specific portion of the 'totalJSHeapSize' that is actively being utilized. These metrics obtained from the Memory API performance tool enable developers to assess and optimize memory usage, leading to more efficient JavaScript applications. [35]

5.3.1 Performance Memory of React

The memory usage of React was measured by running 20 tests and calculating the average value. The average memory usage for React was found to be approximately 0.032367 GB for the total heap and 0.030245 GB for the used heap. These values indicate the amount of memory allocated and utilized by React during the tests.

TestNumber	jsHeapSizeLimit [bytes]	totalJSHeapSize [bytes]	usedJSHeapSize [bytes]
1	4294705152	18095915	14505331
2	4294705152	27406048	25255740
3	4294705152	30408217	28259333
4	4294705152	37006443	34873411
5	4294705152	37214333	34830109
6	4294705152	36952619	34770228
7	4294705152	43669632	41404360
8	4294705152	30551225	28437397
9	4294705152	37268238	35179870
10	4294705152	30551021	28505225
11	4294705152	37005888	34881096
12	4294705152	37214965	34972171
13	4294705152	30813851	34820109
14	4294705152	30813892	28589023
15	4294705152	37530903	28507044
16	4294705152	37476542	35029699
17	4294705152	37214535	35293138
18	4294705152	43669267	41445915
19	4294705152	30813419	28640707
20	4294705152	43407247	41308011
Average	4294705152	34754210	32475393.8

Table 12: Performance Memory of React

5.3.2 Performance Memory of Svelte

By employing the same method in a Svelte application, a significant difference in memory usage compared to React was observed. The total heap usage for Svelte was approximately 0.012581 GB, while the used heap amounted to 0.011572 GB. These values indicate a lower memory footprint for Svelte compared to React, suggesting more efficient memory utilization in the Svelte application.

TestNumber	jsHeapSizeLimit [bytes]	totalJSHeapSize [bytes]	usedJSHeapSize [bytes]
1	4294705	7425495	6906185
2	4294705152	10057388	9390516
3	4294705152	11487457	10727229
4	4294705152	15846175	14649342
5	4294705152	11829341	110298869
6	4294705152	16188065	14929909
7	4294705152	11752117	10951965
8	4294705152	16110836	14833744
9	4294705152	11753375	11055831
10	4294705152	16112096	14958576
11	4294705152	11833115	11170476
12	4294705152	16453980	15170476
13	4294705152	12018037	11269417
14	4294705152	12018662	11181822
15	4294705152	17425957	15230337
16	4294705152	12018662	11313902
17	4294705152	17427219	15148491
18	4294705152	12361808	11573904
19	4294705152	17769103	15466451
20	4294705152	12284582	11453358
Average	4294705152	13508736.55	12424989.55

Table 13: Perfomace Memory of Svelte

5.3.3 Overall of Memory Performace

Svelte:

- Total heap usage: 0.012581 GB
- Used heap: 0.011572 GB

React:

- Total heap usage: 0.032367 GB
- Used heap: 0.030245 GB

Comparing these values, it is evident that Svelte consumes significantly less memory compared to React. The total heap usage and used heap in Svelte are approximately one-third of the corresponding values in React. This suggests that Svelte is more efficient in terms of memory utilization, resulting in a lower memory footprint for Svelte applications compared to React applications.

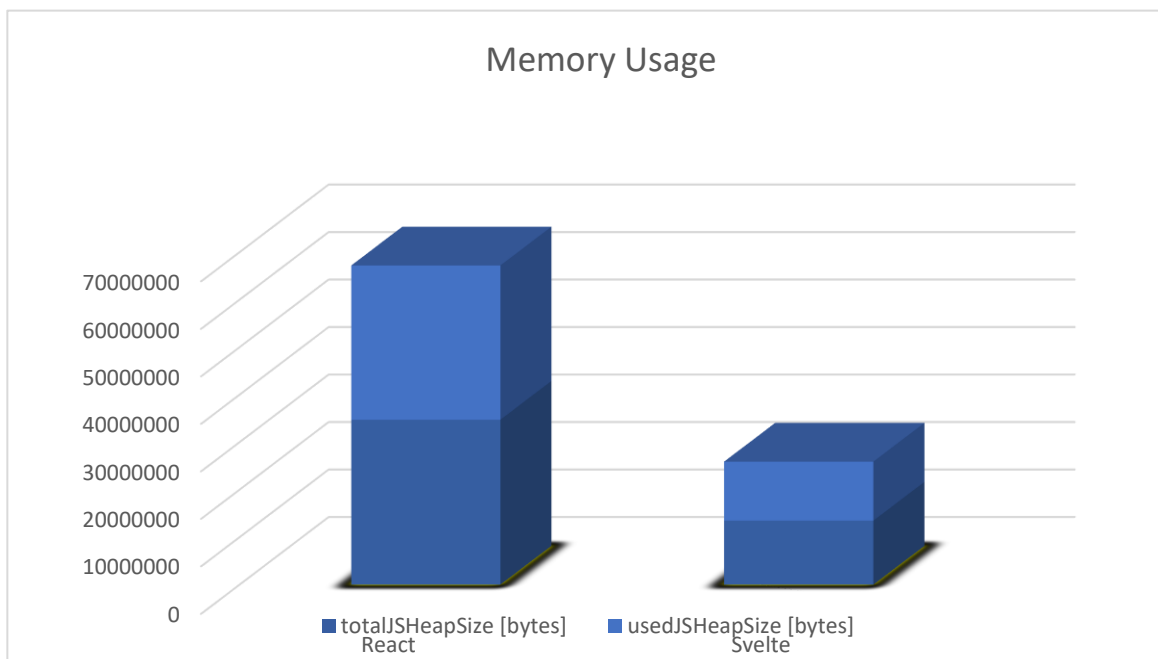
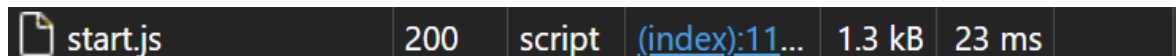


Figure 50: Difference between Frameworks for Memory Performance

5.4 Size of the application

The size of an application has a significant impact on its performance, determining whether it will run faster or slower. While comparing the application sizes of different frameworks, there is some intriguing discoveries. Interestingly, the size of the same application developed in another framework was only one-fifth of the size of the React application. This significant reduction in size could be the primary reason behind the faster performance observed in the alternative framework.

A screenshot of a file explorer or terminal showing application details for Svelte. The file is named 'start.js', has a size of 200 bytes, is a script type, and is located at '(index):11...'. The application size is 1.3 kB and the load time is 23 ms.

start.js	200	script	(index):11...	1.3 kB	23 ms
----------	-----	--------	---------------	--------	-------

Figure 51: Application size in Svelte

A screenshot of a file explorer or terminal showing application details for React. The file is named 'bundle.js', has a size of 200 bytes, is a script type, and is located at '(index)'. The application size is 547 kB and the load time is 110 ms.

bundle.js	200	script	(index)	547 kB	110 ms
-----------	-----	--------	---------	--------	--------

Figure 52: Application size in React

5.5 framework Usages

Based on another survey conducted by the State of JS, there is potential for further exploration and discovery of differences by delving deeper into the results. [28]

5.5.1 Retention

The survey conducted to determine the usage of retention was based on a calculation that involved dividing the total usage by the sum of users who expressed a desire to use it again and those who said they would not use it again. Looking at the graph, we can observe that React was the leading framework until 2019. However, after the introduction of Svelte, there was a significant drop in the number of users for React, indicating that developers started to prefer other frameworks, including Svelte. It is not clear whether Svelte was the sole reason

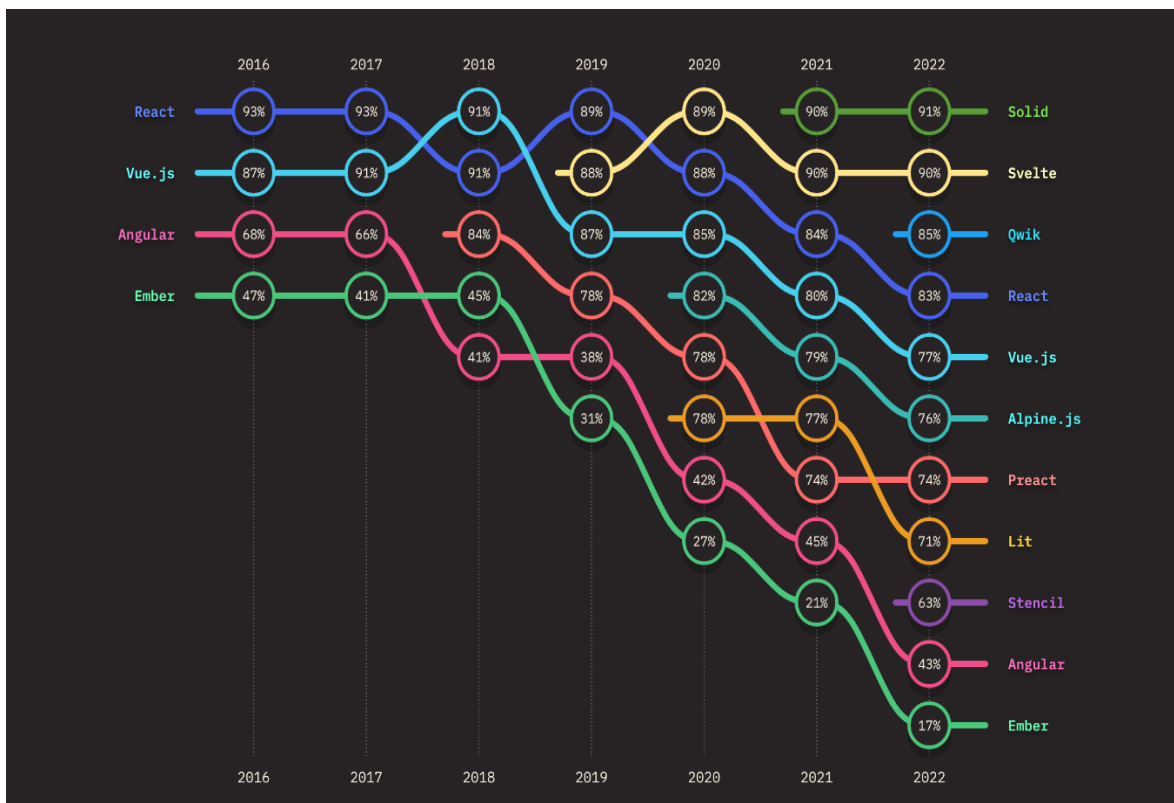


Figure 53: Retention of Frameworks [28]

For this shift, as other frameworks could have also played a role. Although Svelte's popularity has been affected by the emergence of other frameworks like Solid, it still ranks higher than React based on user rankings.

5.5.2 Interested

For the next survey or question, the focus was on people or groups who were interested in learning the framework. The calculation involved dividing the number of people who wanted to learn the framework by the total number of people who expressed interest in learning it, including those who were not interested.

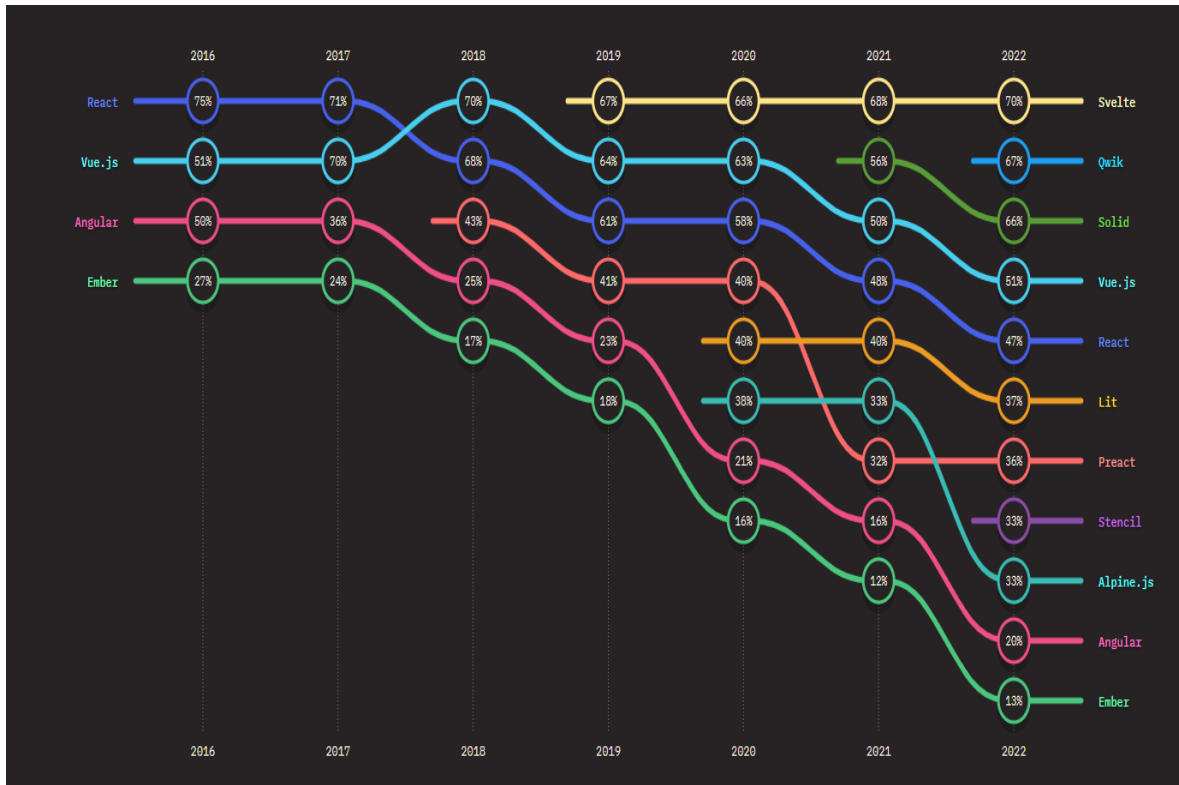


Figure 54: Interest in Frameworks [28]

Based on the results, we can observe that the interest in learning the framework increased steadily from the beginning to the introduction of Svelte. However, there was a significant drop in interest in React, with almost half of the users expressing no interest in learning the framework anymore.

5.5.3 Awareness

Several factors can affect the popularity of a framework, such as the knowledge and awareness of the technology within the programming community. Therefore, a survey was conducted to assess the awareness of the technology. The calculation involved subtracting the number of respondents who answered "never heard" from the total number of answers, and then dividing by the total number of respondents.

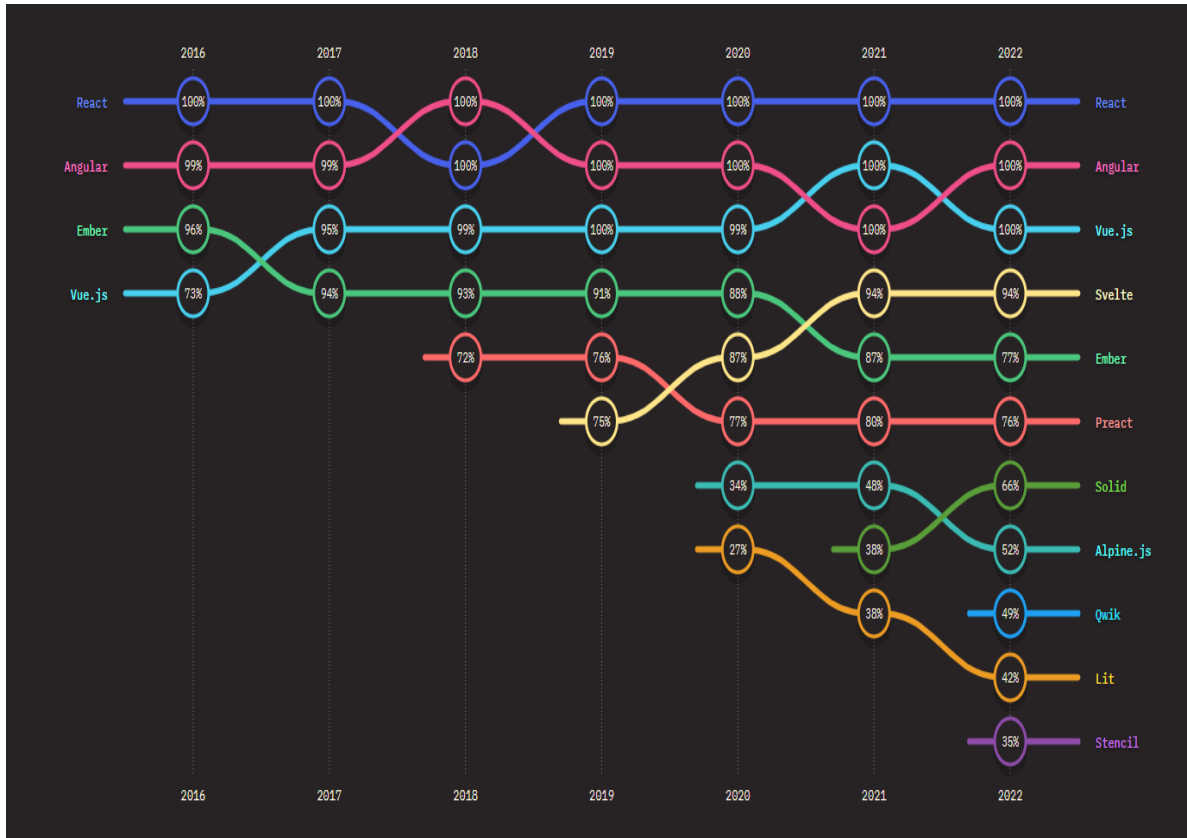


Figure 55: Awareness of Frameworks [28]

In the awareness chart, it is evident that React gained significant awareness right from the beginning of its launch as a framework. On the other hand, the situation is different for Svelte. While its average awareness has been improving over time, it still falls short compared to React. This doesn't necessarily mean that Svelte is not competing with React, as there are other frameworks in the market that are also vying for attention and adoption.

5.6 Survey

Frontend development frameworks have their own communities and supporters, and two well-known and trustworthy websites for programmers are used to gather data about their popularity. A survey conducted on Stack Overflow in 2022 shows that React is currently the most widely used framework, with a growing community of developers. While React has the fastest-growing community among other frameworks, Svelte is also gaining in usage and development. According to research, both frameworks are proving to be versatile and useful

to programmers, and as a result, their communities are expected to grow even more in the future. Overall, both React and Svelte have promising futures.[29]

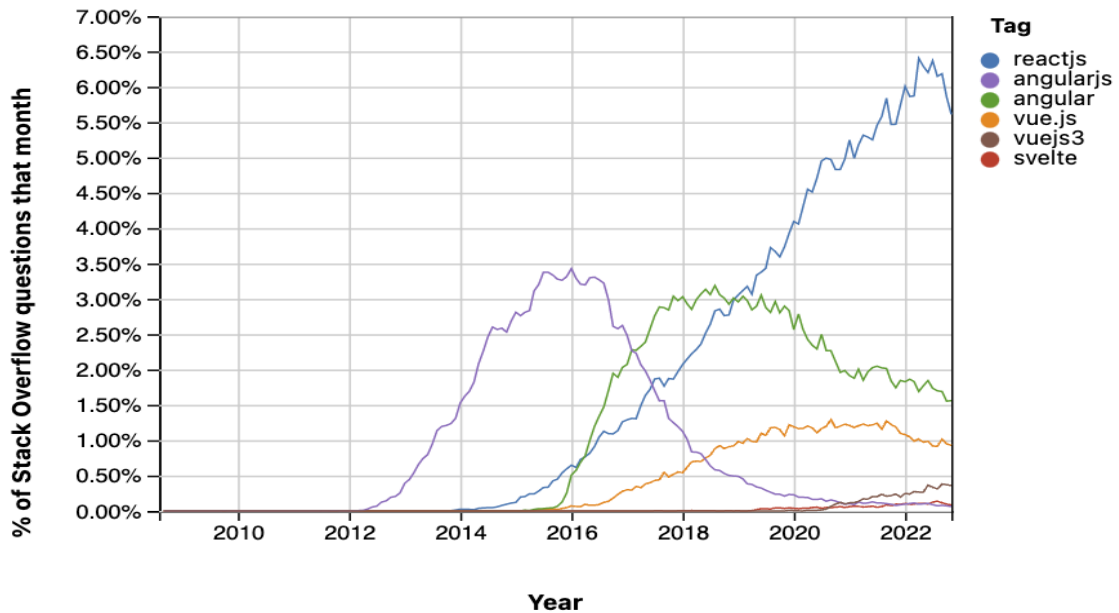


Figure 56: Trend questions of frameworks in 2022 [29]

5.7 Git Community

As we are aware, every technology has its own community, such as GitHub, which consists of a diverse range of individuals from various areas and backgrounds across the globe. [30] These communities have different objectives, with some aiming to learn and acquire new skills, while others focus on development and gaining more experience. Now, let's discuss the communities associated with different frameworks, including the issues they encounter and the commits they have made up until this point.

5.7.1 React GitHub Community

In the React community, there is a large following of individuals actively engaged in working on various projects and resolving issues. Currently, there are 2.6k followers dedicated to different projects within React. Additionally, there are 56 repositories that are actively being worked on, with each repository having a few open issues that require attention and resolution. [31]

5.7.2 Svelte GitHub Community

Despite being a relatively newer framework and not as widely known as others, the Svelte community has garnered a substantial number of followers. Considering its launch time, the community's size of 1.6k followers is quite commendable. Svelte can even be seen as a competitor to React, given its comparable number of followers. Furthermore, Svelte boasts an even higher number of repositories compared to React, with a total of 71 repositories actively maintained by the community. [32]

5.8 Personal Experience and Background

Starting from scratch and delving into two new frameworks was a challenging experience for me. I had no prior background in either of them, making a fair comparison between them more appropriate towards the end. Personally, I preferred using React due to its concept being easier for me to grasp. React also benefits from extensive documentation, numerous available sources, and a strong support community, which made the learning process smoother. Additionally, my previous experience working with Blazor contributed to my better understanding of React's concepts. On the other hand, Svelte, as mentioned on its official site, offers an easy and fast learning curve, even with a limited background in front-end technologies. While Svelte may have less support and available learning resources compared to React, its simplicity and quick start-up potential make it an appealing framework to consider.

A crucial aspect for beginners in any technology is the availability of learning resources to enhance their knowledge. Limited sources can make it challenging to acquire new skills, often leading individuals to choose easier alternatives. In terms of code implementation, I personally found React to be more straightforward, despite that Svelte is claiming that it requires 40% less code. From my experience, this statement did not hold true, as the availability of comprehensive sources was lacking for React. On the other hand, having multiple frameworks with abundant sources and support is preferable for users. In the case of Svelte, while the official documentation is well done, it lacks crucial information that users may require. In contrast, React offers well-documented resources, and even if the answers are not found in the official documentation, other platforms like Stack Overflow provide better support for React. This demonstrates the importance of comprehensive and accessible sources for users, which can significantly impact their choice of framework.

5.9 Research comparison

Several research studies have been conducted to compare various JavaScript frameworks and their related research works in terms of usage, performance, memory usage, and other relevant features. These studies aim to provide insights into the current research trends and advancements in JavaScript.

I have conducted thorough research and obtained test results that strongly demonstrate the correlation between rendering performance and memory usage. These results provide solid evidence to support the findings and conclusions of the study, offering valuable insights for optimizing rendering algorithms and resource allocation.

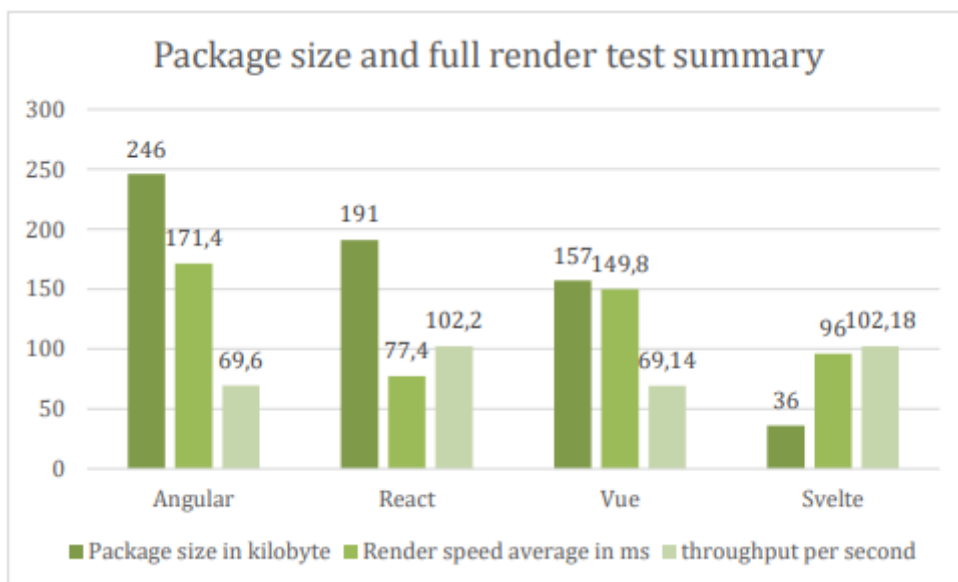


Figure 57: Render of other Researches [38]

In the conducted research comparing rendering times between React, Svelte, Angular, and Vue, the results show similarities between Svelte and React, where React exhibited shorter rendering times while Svelte had slightly higher rendering times. Additionally, the memory usage in Svelte was significantly lower compared to React. These findings provide evidence that Svelte's rendering is slower than React, despite its lower memory usage. Contrary to expectations, the lower memory usage does not positively impact the rendering speed in Svelte.

Several research studies have been conducted to compare the Svelte and React frameworks, shedding light on their respective strengths and characteristics. One approach employed in these studies involves examining the frameworks as JavaScript application frameworks, specifically their roles and functionalities in the development of web application user interfaces. A noteworthy research endeavor conducted at Turun Ammattikorkeakoulu delved into the implementation aspects of Svelte and React. The study aimed to analyze how these frameworks can be effectively implemented in real-world applications, taking into consideration factors such as code structure, ease of use, and development workflow. By scrutinizing the implementation intricacies, the researchers sought to gain a comprehensive understanding of the frameworks' practical implications and their suitability for different project requirements.[40]

In Greece, another insightful research project was carried out, focusing on the performance aspect of Svelte and React in terms of DOM manipulation. Specifically, the study sought to ascertain how efficiently each framework could handle tasks such as updating, deleting, and modifying elements within the Document Object Model (DOM). The aim was to identify any significant discrepancies in the performance of these frameworks when it comes to dynamic UI updates and interactions. By quantifying the performance metrics and conducting comparative analyses, the researchers aimed to provide developers with valuable insights for making informed decisions regarding framework selection based on their specific performance requirements.[41]

Moreover, there was an intriguing research investigation that centered around the cross-platform UI development capabilities of React and Svelte. With the ever-increasing variety of devices and screen sizes, ensuring optimal user experiences across platforms has become crucial. This research study aimed to evaluate how React and Svelte address the challenges of creating responsive user interfaces that adapt seamlessly to diverse screen dimensions. By assessing factors such as layout responsiveness, component adaptability, and rendering efficiency, the researchers aimed to determine the strengths and weaknesses of each framework in accommodating the needs of modern multi-device environments.[39]

In summary, these research studies provide valuable insights into the Svelte and React frameworks, exploring their implementation aspects, performance in DOM manipulation, and cross-platform UI development capabilities. By examining and comparing these frameworks in various contexts, the research community aims to advance our understanding of JavaScript frameworks and assist developers in making informed choices based on specific project requirements and objectives.

CONCLUSION

After the application designs were completed, they were implemented and subsequent measurements were conducted to evaluate their performance and functionality. Each application underwent rigorous testing, and the obtained results were subjected to detailed analysis. During the evaluation process, various factors such as available resources, community support, and functional capabilities of different frameworks were taken into consideration, leading to the determination that React enjoys wider usage and support.

Upon careful examination of the provided materials, it becomes apparent that both Svelte and React possess distinct advantages and disadvantages. Svelte is renowned for its user-friendly nature, as emphasized in its documentation, whereas React also offers a manageable learning curve. In terms of implementation, Svelte necessitates less code but imposes certain restrictions on writing, while React provides greater flexibility albeit at the potential cost of increased code volume.

With respect to performance, noteworthy discrepancies exist between the two frameworks. Svelte exhibits exceptional memory utilization, owing to its lightweight and efficient nature. Moreover, it demonstrates superior loading and processing speeds, thereby positioning itself as a standout performer in these domains. Conversely, React excels in rendering speed, with its rendering timing estimated to be approximately seven times faster than that of Svelte.

Considering the surveys conducted and the communities surrounding each framework, Svelte is currently experiencing a surge in popularity while React continues to be acknowledged as a robust and influential framework. Consequently, based on a comprehensive analysis, it can be inferred that Svelte represents a more promising framework with considerable potential for future advancement.

In conclusion, Svelte offers a more intuitive learning experience, requires less code while incorporating certain limitations, showcases commendable memory utilization and expeditious loading times, whereas React distinguishes itself through its rendering speed. Both frameworks boast vibrant communities, yet Svelte is emerging as a formidable contender. The research findings collectively support the notion that Svelte is a more auspicious framework, poised for significant contributions to the realm of front-end development..

BIBLIOGRAPHY

- [1] React.component. React [online]. [Accessed 14 March 2023]. Available from: <https://reactjs.org/docs/react-component.html>.
- [2] OKORO, Alvin. React virtual DOM - using virtual dom in react. KnowledgeHut [online]. 12 February 2023. [Accessed 16 March 2023]. Available from: <https://www.knowledgehut.com/blog/web-development/react-virtual-dom>.
- [3] The history of react native: Facebook's open source app development framework. TechAhead [online]. 13 May 2021. [Accessed 16 March 2023]. Available from: <https://www.techaheadcorp.com/blog/history-of-react-native/>.
- [4] Versions. React [online]. [Accessed 16 March 2023]. Available from: <https://reactjs.org/versions/>.
- [5] CHAVAN, Bakuli. Top companies using react JS services to their best! Bigscal [online]. 23 January 2023. [Accessed 16 March 2023]. Available from: <https://www.bigscal.com/blogs/frontend-technology/top-companies-using-react-js-services-to-their-best/>.
- [6] Svelte. • Cybernetically enhanced web apps [online]. [Accessed 16 March 2023]. Available from: <https://svelte.dev/>.
- [7] BEATA TWARDOWSKA MAR 22. Why Svelte is the next big thing in javascript development. Naturaily [online]. 22 March 2022. [Accessed 16 March 2023]. Available from: <https://naturaily.com/blog/why-svelte-is-next-big-thing-javascript-development>.
- [8] R/sveltejs - apple beta music uses Svelte. reddit [online]. [Accessed 16 March 2023]. Available from: https://www.reddit.com/r/sveltejs/comments/v7ic2s/apple_beta_music_uses_svelte/.
- [9] Introduction to the DOM - web apis: MDN. Web APIs | MDN [online]. [Accessed 16 March 2023]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

- [10] Document object model (DOM) level 1 specification. Dokument Object Model (DOM) Level 1 Specification [Vidur Apparao, Steven Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Arnaud Le Hors, Gavin Thomas Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, Lauren Wood] [online]. [Accessed 16 March 2023]. Available from: <http://www.dret.net/biblio/reference/dom>.
- [11] How javascript programming language is becoming a market leader. mobileLIVE [online]. 18 November 2022. [Accessed 16 March 2023]. Available from: <https://www.mobilelive.ca/blog/javascript-leader>.
- [12] Loops in JavaScript - performing repeated operations on a data set. [online]. [Accessed 16 March 2023]. Available from: https://launchschool.com/books/javascript/read/loops_iterating.
- [13] SIMPLILEARN. JavaScript frameworks: What are they and how do they work?: Simplilearn. Simplilearn.com [online]. 1 March 2023. [Accessed 16 March 2023]. Available from:
- [14] ADMIN. The future of javascript: What to expect in 2023? Programmer Force [online]. 23 January 2023. [Accessed 16 March 2023]. Available from: <https://pf.com.pk/blogs/the-future-of-javascript-what-to-expect-in-2023/#:~:text=Even%20after%20such%20a%20long,scalable%20and%20user%2Dfriendly%20applications>.
- [15] DAAN. 8 react libraries that I'd like to introduce to you. Medium [online]. 15 February 2022. [Accessed 3 March 2023]. Available from: <https://levelup.gitconnected.com/8-react-libraries-that-id-like-to-introduce-to-you-3802770b3952>.
- [16] GOSPEL, Darlington. React.js architecture pattern: Implementation + best practices . [online]. 30 January 2023. [Accessed 1 March 2023]. Available from: <https://www.knowledgehut.com/blog/web-development/react-js-architecture>.
- [17] React. – The library for web and native user interfaces [online]. [Accessed 18 March 2023]. Available from: <https://react.dev/>.

- [18] GACKENHEIMER, Cory. The Core Of React. In : Introduction to react: Using react to build scalable and efficient user interfaces. New York : Apress, 2015. p. 21–27.
- [19] LEVLIN, Mattias. DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte. thesis. 2020.
- [20] Úvod do Technologie Svelte. SvelteJS [online]. [Accessed 21 March 2023]. Available from: <https://www.sveltejs.cz/>.
- [21] Getting started with Svelte - learn web development: MDN. Learn web development | MDN [online]. [Accessed 21 March 2023]. Available from: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_getting_started.
- [22] SVELTEJS. Unused CSS selector warning - disable specific parts? · issue #1594 · sveltejs/svelte. GitHub [online]. [Accessed 21 March 2023]. Available from: <https://github.com/sveltejs/svelte/issues/1594>
- [23] MELTZER, Rachel. What is JavaScript used for? Lighthouse Labs [online]. 3 December 2020. [Accessed 30 March 2023]. Available from: <https://www.lighthouselabs.ca/en/blog/what-is-javascript-used-for#:~:text=Javascript%20is%20used%20by%20programmers,by%2097.0%25%20of%20all%20websites>.
- [24] Rapidly build modern websites without ever leaving your HTML. Tailwind CSS [online]. [Accessed 27 April 2023]. Available from: <https://tailwindcss.com/>
- [25] The Main Advantage Of TailwindCSS [online]. 6 August 2021. How To Debug CSS. [Accessed 2023]. Available from: <https://planflow.dev/blog/the-main-advantage-of-tailwindcss>.
- [26] MOZDEVNET. Navigation Timing - web apis: MDN. Web APIs | MDN [online]. [Accessed 20 April 2023]. Available from: https://developer.mozilla.org/en-US/docs/Web/API/Performance_API/Navigation_timing.

- [27] BOTHRA, Komal. What are User Timing Marks and measures? Seahawk [online]. 24 April 2023. [Accessed 2 May 2023]. Available from: <https://seahawkmedia.com/site-speed-glossary/what-are-user-timing-marks-and-measures/>.
- [28] State of JavaScript 2022. Front-end Frameworks [online]. [Accessed 3 May 2023]. Available from: <https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/>.
- [29] Stack Overflow trends [online]. <https://gist.github.com/eyesofkids/14d94c8da3953440bd6e4bb46ed4220c>. [Accessed 1 May 2023]. Available from: <https://insights.stackoverflow.com/trends?tags=reactjs%2Cvue.js%2Cangular%2Csvelte%2Cangularjs>.
- [30] GitHub Community guidelines. GitHub Docs [online]. [Accessed 4 May 2023]. Available from: <https://docs.github.com/en/site-policy/github-terms/github-community-guidelines>.
- [31] React community. GitHub [online]. [Accessed 1 May 2023]. Available from: <https://github.com/reactjs>
- [32] Svelte. *GitHub* [online]. [Accessed 6 May 2023]. Available from: <https://github.com/sveltejs>
- [33] PREETHI ON DEC 20 and PREETHI. Breaking down the performance API: CSS-tricks. CSS [online]. 20 December 2017. [Accessed 10 May 2023]. Available from: <https://css-tricks.com/breaking-performance-api/>.
- [34] MOZDEVNET. Performance - web apis: MDN. Web APIs | MDN [online]. [Accessed 9 May 2023]. Available from: <https://developer.mozilla.org/en-US/docs/Web/API/Performance>.
- [35] MOZDEVNET. Performance: Memory property - web apis: MDN. Web APIs | MDN [online]. [Accessed 16 May 2023]. Available from: <https://developer.mozilla.org/en-US/docs/Web/API/Performance/memory>.

- [36] Rendering elements. *React* [online]. [Accessed 4 April 2023]. Available from: <https://legacy.reactjs.org/docs/rendering-elements.html>
- [37] ReactJS JSX - javatpoint. *www.javatpoint.com* [online]. [Accessed 15 May 2023]. Available from: <https://www.javatpoint.com/react-jsx>.
- [38] TOMAS MARX-RAACZ VON HIDVÉG. *Are the frameworks good enough?* thesis. 2022.
- [39] OKSANEN, Miikka. *Javascript frontend web app frameworks React vs Svelte.* thesis. 2021.
- [40] HAAPASALO, Karel. *JAVASCRIPTIN SOVELLUSKEHYS VERKKOSOVELLUKSEN KÄYTTÖLIITTYMÄN TOTEUTUKSESSA.* thesis. 2021.
- [41] HAIDER, Ali. *Javascript frontend web app frameworks React vs Svelte.* thesis. [no date].

LIST OF ABBREVIATIONS

API - Application programming interface

CSS - Cascading Style Sheets

DOM - Document Object Model

DnD - Drag and Drop

GB - GigaByte

HTML - HyperText Markup Language

JSX - JavaScript XML

UI - User Interface

XML - Extensible Markup Language

LIST OF FIGURES

<u>Figure 1: Represent of Document Object Model</u>	14
<u>Figure 2: Difference between TailwindCSS and CSS [25]</u>	16
<u>Figure 3: Performace API [33]</u>	17
<u>Figure 4: Use Case Diagram</u>	27
<u>Figure 5: Home Wireframe</u>	28
<u>Figure 6: BookList Wireframe</u>	29
<u>Figure 7: BookDetail Wireframe</u>	29
<u>Figure 8: Favorite Wireframe</u>	30
<u>Figure 9: React Folders</u>	33
<u>Figure 10:Svelte Folders</u>	34
<u>Figure 11: Base code of React</u>	35
<u>Figure 12: Base code of Svelte</u>	35
<u>Figure 13: Routing in React</u>	36
<u>Figure 14: Routing in Svelte</u>	37
<u>Figure 15: Fetching in React</u>	38
<u>Figure 16: Fetching in Svelte</u>	38
<u>Figure 17: Adding to Favorites React</u>	39
<u>Figure 18: Removing from Favorites React</u>	39
<u>Figure 19: Favorite Adding and Removing in Svelte</u>	40
<u>Figure 20: Checking Favorite list in React</u>	40
<u>Figure 21: Checking Favorite list in Svelte</u>	41
<u>Figure 22: Search in React</u>	41
<u>Figure 23: Search in Svelte</u>	41
<u>Figure 24: Navigation to BookDetails in both Frameworks</u>	41
<u>Figure 25: Button Event handeral for Navigation</u>	42
<u>Figure 26: Fetching BookDetail in React</u>	42
<u>Figure 27: Fetching BookDetails in Svelte</u>	43
<u>Figure 28: SubmittingComment and Rating in React</u>	43
<u>Figure 29: Submitting Comment and Rating in Svelte</u>	44
<u>Figure 30: Additional Information in React</u>	44
<u>Figure 31: Additional Information in Svelte</u>	45
<u>Figure 32: Conditioning in Html Tags React</u>	46
<u>Figure 33; Conditioning in Html Tags Svelte</u>	46
<u>Figure 34: Mapping in react</u>	46

<u>Figure 35: Looping in Svelte</u>	47
<u>Figure 36:Home page “iPhone”</u>	48
<u>Figure 37:Home page “iPad”</u>	48
<u>Figure 38: Home page “MacBook Air”</u>	48
<u>Figure 39: BookList “iPad”</u>	49
<u>Figure 40: BookList “iPhone”</u>	50
<u>Figure 41: BookList “MacBook Air”</u>	50
<u>Figure 42: Favorites “iPad”</u>	51
<u>Figure 43: Favorites “MacBook Air”</u>	52
<u>Figure 44: Favorites “iPhone“</u>	52
<u>Figure 45: BookDetails “iPad”</u>	53
<u>Figure 46: BookDetails “MacBook Air”</u>	54
<u>Figure 47: BookDetails “iPhone”</u>	54
<u>Figure 48: Load and Process counting [26]</u>	55
<u>Figure 49: Difference between Frameworks for User Timing</u>	62
<u>Figure 50: Difference between Frameworks for Memory Performance</u>	65
<u>Figure 51: Application size in Svelte</u>	66
<u>Figure 52: Application size in React</u>	66
<u>Figure 53: Retention of Frameworks [28]</u>	67
<u>Figure 54: Interest in Frameworks [28]</u>	68
<u>Figure 55: Awareness of Frameworks [28]</u>	69
<u>Figure 56: Trend questions of frameworks in 2022 [29]</u>	70
<u>Figure 57: Render of other Researches [38]</u>	72

LIST OF TABLES

<u>Table 1: Functional Requirements</u>	25
<u>Table 2: None Functional Requirements</u>	26
<u>Table 3: Installation of React</u>	31
<u>Table 4: Installation of Svelte</u>	32
<u>Table 5: Navigation Timing for React</u>	56
<u>Table 6: Navigation Results in React</u>	57
<u>Table 7: Navigation Timing for Svelte</u>	57
<u>Table 8: Navigation Results in Svelte</u>	58
<u>Table 9: Difference between Frameworks for Navigation Timing</u>	58
<u>Table 10: User Timing in React</u>	60
<u>Table 11: User Timing in Svelte</u>	61
<u>Table 12: Performance Memory of React</u>	63
<u>Table 13: Performance Memory of Svelte</u>	64

APPENDICES

Appendix P I: fulltext.pdf

Appendix P II: Thesis Project

APPENDIX P I: APPENDIX TITLE

Appendix P I:

- Contains the pdf file of mt thesis project

Appendix P II:

Contains the source Code

- Svelte

Contains the source code for Svelte project

- React

Contains the source code for React project