KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Computer Engineering
First cycle, 15 credits

React vs. Vue: Comparing JavaScript Frameworks from a Beginner's Perspective

Daniel Schuster

Stockholm, Sweden 2023

Abstract

JavaScript has become the dominant programming language for web application development during the last few decades, being used by 98.5% of the websites worldwide. This prominence has led to the emergence of several frameworks for this language, with React and Vue being among the popular choices. However, there is no research specifically comparing these frameworks from a beginner's perspective when creating simple web applications.

The purpose of this thesis is to map out similarities and differences between React and Vue in terms of performance and ease of use for beginners. To accomplish this, two identical web applications were developed using React and Vue, respectively. These applications were designed to require only a general understanding of programming concepts and no prior experience with the mentioned frameworks, while having a target build size ranging from 40-60 KB. The development process for each framework was documented, capturing the time invested and challenges encountered. A qualitative approach was adopted to assess performance, leveraging quantifiable data gathered from the developed applications, while ease of use was evaluated through the lens of a beginner.

The findings indicate that while React exhibited 12% faster overall execution time and 4% faster memory allocation, the performance gap was narrower than previous research suggested due to the smaller size of the applications. On the other hand, Vue was found to be more beginner-friendly, requiring 21.3% less development time to achieve the same application functionality. These findings aid beginners in selecting the most suitable framework based on their priorities and contribute to the understanding of React and Vue in the context of simple web application development.

Keywords

JavaScript, Programming, Frameworks, React, Vue

Sammanfattning

JavaScript har under de senaste decennierna vuxit till att bli det dominerande programmeringsspråket för webbutveckling och används idag av 98,5% av webbplatserna världen över. Denna utveckling har lett till skapandet av flera ramverk för JavaScript, där React och Vue är bland de populära valen. Det finns dock ingen forskning som specifikt jämför dessa ramverk från ett nybörjarperspektiv när det kommer till att skapa enkla webbapplikationer.

Syftet med denna studie är att kartlägga likheter och skillnader mellan React och Vue när det gäller prestanda och användarvänlighet för nybörjare. För att åstadkomma detta utvecklades två identiska webbapplikationer med hjälp av React respektive Vue. Dessa applikationer designades för att endast kräva en allmän förståelse av programmeringskoncept och ingen tidigare erfarenhet av de nämnda ramverken, samtidigt som de hade en storlek som sträcker sig från 40-60 KB. Utvecklingsprocessen för varje ramverk dokumenterades, bland annat med fokus på tid som investerats och de utmaningar som stöttes på. Ett kvalitativt tillvägagångssätt antogs för att bedöma prestanda, med hjälp av kvantifierbar data som samlats in från de utvecklade applikationerna, medan användarvänligheten utvärderades genom att analyseras ur en nybörjares perspektiv.

Resultaten indikerade att trots att React överlag uppvisade 12% snabbare exekveringstid och 4% snabbare minnesallokering, var prestandagapet mindre än tidigare forskning har visat främst på grund av den mindre storleken på applikationerna. Å andra sidan visade sig Vue vara mer nybörjarvänlig och krävde 21.3% mindre tid för att skapa samma funktionalitet. Dessa resultat kan hjälpa nybörjare att välja det mest lämpliga ramverket baserat på deras prioriteringar och bidrar till förståelsen av React och Vue när det kommer till att skapa enkla webbapplikationer.

Nyckelord

JavaScript, Programmering, Ramverk, React, Vue

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Mira Kajko-Mattsson, for her guidance throughout the entire process of conducting this thesis. Her constructive criticism and feedback have been crucial for the quality of this research.

Additionally, I would like to extend my thanks to Håkan Olsson, who serves as the examiner for this thesis. I appreciate his willingness to assess and evaluate my work, and I am grateful for the opportunity to present my findings to him.

Lastly, I would like to express my gratitude to KTH Royal Institute of Technology for providing me with the opportunity to pursue my bachelor studies at one of Europe's leading technical universities. The academic environment, resources, and support provided by KTH have been instrumental in my educational journey and the completion of this thesis.

Author

Daniel Schuster
Computer Engineering
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden
KTH Royal Institute of Technology
School of Electrical Engineering and Computer Science

Supervisor

Mira Kajko-Mattsson
Stockholm, Sweden
KTH Royal Institute of Technology

Examiner

Håkan Olsson
Stockholm, Sweden
KTH Royal Institute of Technology

Contents

List of Figures

List of Tables

1 Introduction

This chapter serves as an introductory overview of the thesis. Section 1.1 provides essential background information on the research topic, elaborating on the evolution of programming languages and the development of software frameworks. Section 1.2 delves into the problem statement, while Section 1.3 focuses on defining the purpose of the thesis. The goal of the research is expounded upon in Section 1.4. In Section 1.5, the research method employed in this study is outlined, highlighting the approaches utilized. Furthermore, Section 1.6 identifies the target audience of the thesis. Finally, Section 1.7 critically examines the scope and limitations of the study, along with the potential benefits, ethical considerations, and sustainability aspects. The chapter concludes in Section 1.9 by presenting an outline of the subsequent chapters, providing a glimpse into the overall structure and content of the thesis.

1.1 Background

Programming has come a long way since the first computers were developed in the mid-20th century. In the early days of computer science, programmers had to write software in low-level languages that directly communicated with the hardware. These languages were difficult to code in and required a deep understanding of machine language and the underlying hardware.[1] As computers became more powerful and widely used, high-level programming languages were developed. These languages provided abstractions that shielded the programmers from the details of the hardware, allowing them to write code which resembles human languages rather than machine languages.[2]

While high-level programming languages made programming easier, they also led to a boom in the number of software applications being developed. As software engineering became more complex, it became difficult to manage and develop the programs. Therefore, software frameworks were developed to address this issue by providing programmers with reusable and pre-built functionality.[3] Most modern languages have their own frameworks that can be used to speed up the development process and improve code quality. This results in more efficient development, easier maintenance, and better scalability in software applications.[4]

The rise of the high-level language JavaScript has led to a growing interest in comparing and contrasting its available frameworks to determine which is the best fit for a particular project. Two of the most popular frameworks in this regard are React.js and Vue.js, hereafter called React and Vue. While both frameworks share some similarities, they also have distinct differences. A comparative study of React and Vue can help developers make educated decisions about which framework to use for their web application needs.

1.2 Problem

Despite the rising popularity of both React and Vue, there is no research which maps out the similarities and differences between the two frameworks regarding performance and ease of use when it comes to developing simple web applications for beginners. This research gap is significant as it leaves developers without clear knowledge of the similarities and the differences between the two frameworks along with estimations for how long it takes to learn either framework to create an interactive web application.

1.3 Purpose

The purpose of this thesis is to map out similarities and differences between the chosen frameworks when it comes to developing simple web applications for developers who are new to these JavaScript frameworks.

1.4 Goal

The goal of the thesis is to create a simple web application where users can share workout routines, by programming it in both React and Vue to analyze performance and ease of use. This helps to map out similarities and differences and therefore serve as a guideline for new developers to choose the most suitable frameworks for those who choose to create an application of similar size and difficulty level.

1.5 Method

In this study, a comprehensive literature review is conducted to establish a solid understanding of the core concepts of React and Vue. The research approach employed is qualitative, enabling in-depth interpretations of both quantifiable data and personal observations. This holistic approach ensures that the findings of the study are applicable and relevant to similar projects in the field.

1.6 Target Audience

The target audience is primarily JavaScript developers who are looking to get started with either React or Vue to create their first web application using one of these frameworks. On top of that, companies who are looking to develop simple applications similar in size to the one developed in this thesis, or to just educate their employees, can gain knowledge that helps them decide how long it can be expected to learn either framework.

The thesis revolves around front-end development but anyone who is interested in software engineering might have an interest in this study. Since the application that is created is a simple website for sharing workout routines, anyone who wants to make a similar website might have an interest in this work.

1.7 Scope and Limitations

There are several frameworks that are used for JavaScript, both front-end and back-end oriented. This study is limited to front-end frameworks since it would be too demanding and practically impossible to finish the study on time if back-end frameworks were to be compared as well.

It is important to bear in mind that new frameworks are developed on a regular basis and it does not mean that they are not as good as the ones that are researched in this study. Since the study is conducted by a single individual, a comparison between two frameworks is a reasonable limitation.

This study revolves around a specific application, being developed by a beginner. The results that are derived from the study are therefore not necessarily representative for using chosen frameworks in all circumstances. Additionally, the results of this study are dependent on the specific set of instruments used, as mentioned in Section 3.5. These instruments play a significant role in shaping the outcomes of the research. Therefore, different tools could yield different results.

1.8 Benefits, Ethics and Sustainability

Since this thesis focuses primarily on the use of frameworks for inexperienced developers, new learners of these frameworks are going to benefit from this work.

One of the main focuses of this thesis is to compare ease of use, which includes reusing code and building separate front-end applications based on the same back end. This contributes to a sustainable development process and evaluates how adaptable different frameworks are to such processes.

In order to promote sustainability, the application that is created is a website where users can look up and share their workout routines which in turn promotes sustainable behaviours such as going to the gym and living a healthy lifestyle.

In terms of ethics, it is important to consider how the comparisons are conducted and presented. It is also important to ensure that any data collected or used in the analysis is obtained in an ethical manner, such as through user consent or through the researcher's own data. This research uses only use the researcher's data. It is encouraged through warning messages on the website to not share any personal or sensitive data when registering or submitting data on the website since little focus on this work is on website security.

1.9 Outline

Following the introduction, Chapter 2 provides a comprehensive background of the problem, delving into its theoretical aspects and exploring related work. In Chapter 3, a specific engineering-related research methodology is presented, serving as the foundation for conducting the thesis. Chapter 4 delves into the work itself, detailing its implementation and execution. Chapter 5 focuses on presenting the results obtained from the research, while Chapter 6 offers an analysis and discussion of those findings. Finally, Chapter 7 serves as the conclusion, summarizing the research, its implications and possible future work.

2 JavaScript Frameworks and their concepts

This chapter presents the background knowledge needed to follow the rest of the thesis. An overview is given in Section 2.1 before information about React and Vue is given in Section 2.2. Information about the state-management libraries Redux-React and Pinia are given in Section 2.3. Lastly, related research is discussed in Section 2.4.

2.1 Overview

The software industry has undergone a massive change since high-level programming languages were introduced in the mid-20th century. In recent years, JavaScript has emerged as the most popular high-level programming language for client-side software development. According to statistics, JavaScript is being used by 98.5% of the world's websites.[5] As a result of this, a number of frameworks have been built for this language.

A software framework typically includes a set of pre-built and reusable components, functions, data structures, libraries and toolsets, which can be integrated into an application.[3] By using frameworks, developers can reduce the amount of code that is needed to write since many common functionalities are already provided by the framework. This enables software developers to easily improve performance, maintenance and scalability while at the same time reduce the probability of bugs in the code.[4]

The JavaScript language has changed several times since it was first released in 1995. At first, it simplified adding interactive elements to web pages. Then it got more robust with dynamic HTML and AJAX. Currently, with the back-end server runtime environment Node.js, JavaScript has turned into a language that is mostly used to build full-stack web applications, developing both server and client-side parts of applications.[4] As software engineering evolves and new versions of JavaScript are released, new frameworks are developed which often replace obsolete ones.

For many years, jQuery was the go-to front-end JavaScript framework for most websites around the world. As of 2023, 77.7% of all existing websites have been built using jQuery.[6] Since it is difficult to replace a framework in a large-scale application once it has been built, jQuery is still going to be used for years to come in order to maintain the existing software. However, applications that are being built from scratch today use jQuery to a significantly lesser extent than before. Current trends on Stack Overflow indicate that newer frameworks such as React and Vue are gaining popularity while jQuery has seen a dramatic decline in usage over the last few years, as can be seen from Figure 2.1.[7]

As previously mentioned, using frameworks for software development in JavaScript has various advantages. However, there are also some drawbacks, such as increased complexity and a steeper learning curve. By understanding the similarities and differences between JavaScript frameworks, developers can make educated decisions about which framework to use for their software development needs.

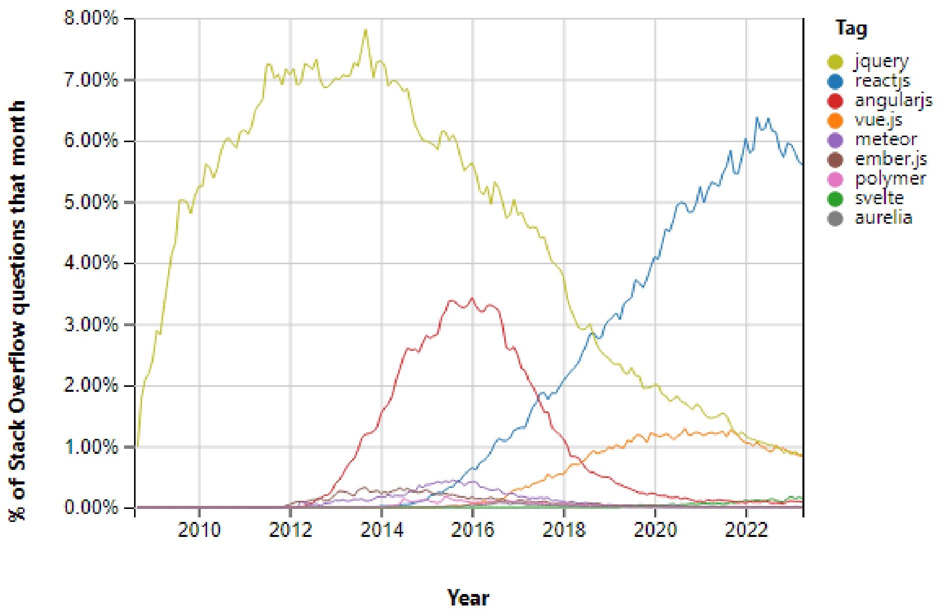


Figure 2.1: Front-End JavaScript Frameworks Trending on Stack Overflow

2.2 React and Vue

This subchapter serves as an introduction to React and Vue, delving into their fundamental principles, key features, and highlighting notable distinctions between them. Section 2.2.1 elucidates the core concepts of React, while Section 2.2.2 provides an overview of Vue's concepts.

2.2.1 Core concepts of React

React is a framework mainly used by front-end JavaScript developers for building user interfaces. It was developed by Meta (formerly Facebook) and was released in 2013. React uses a component-based approach to building user interfaces, where each component represents a part of the user interface. Components are reusable, can be made customizable and rendered in unlimited quantity in a graphical user interface if put together.[8]

A key feature of React is JSX, a syntax extension for JavaScript, which allows developers to write HTML-like elements inside JavaScript files to create user interface views by placing them in the document object model (DOM). DOM is a programming interface that represents a tree-like structure of an HTML or XML document in memory, enabling programmers to interact with web pages or XML documents programmatically to add or remove content from a page.[9]

Furthermore, React utilizes unidirectional data flow. Data moves in one direction, from parent components to child components through props, which is an array of attributes that the child component accepts from the parent component. Child components cannot change the data they get through props on their own. If a child component wants to change the data, it has communicate it to its parent component which in turn changes the data and triggers a re-render of the component tree so that the user interface can be updated.[10]

2.2.2 Core concepts of Vue

Just like React, Vue is a front-end JavaScript framework for building user interfaces. It was created by Evan You in 2014.

Vue also follows a component-based approach and utilizes a virtual DOM for efficient updates and rendering. Vue distinguishes itself from React by employing a template-based syntax instead of JSX, which closely resembles traditional HTML.[11]

A two-way data-binding approach is used in Vue, where changes to the data automatically triggers updates in the user interface. Data is defined as reactive properties, and any changes to these properties are automatically detected by Vue's reactivity system.[12] When a reactive property is modified, Vue automatically updates the corresponding part of the user interface. This allows developers to directly modify the data in a component without needing to explicitly trigger updates or communicate with parent components, making it easier to use while reducing the probability of errors in the code.

2.3 React-Redux and Pinia

In an application, it is often times crucial to manage and track various states, such as the current user logged in. This subsection delves into the state management libraries employed in this thesis. Section 2.3.1 focuses on React-Redux, while Section 2.3.2 explores Pinia.

2.3.1 React-Redux

React-Redux, henceforth called Redux, is a state management library used in conjunction with React. Redux provides a state container for managing the application's state and enables efficient data flow between components.

In Redux, the application's state is stored in a single JavaScript object called the "store." The store represents the global state of the application and is immutable, that is it cannot be modified directly. Instead, to update the state, actions are dispatched to the store. Actions are plain JavaScript objects that describe what changes need to be made to the state. These actions can be called from files outside of the store to change the state.[13]

Reducer functions are used to handle dispatched actions and producing a new state. They take the current state and the dispatched action as inputs and return a new state based on the action's logic.[14]

React components can subscribe to the Redux store to access specific parts of the state. By connecting components to the store, they can then update when the state changes. This eliminates the need for passing down props through multiple levels of components.

2.3.2 Pinia

Pinia is also a state management library, designed for Vue applications. Similar to Redux, Pinia utilizes a store to manage the application’s state. The store is a centralized container that holds the application’s states and provides methods to manipulate it.

In Pinia, stores are defined as classes that extend the base Pinia store class. These stores define the state properties and methods for manipulating the state. Vue’s reactivity system ensures that components that use the store are automatically updated when the state changes.[15]

To access the state or invoke actions within a component, a useStore function provided by Pinia is used. This function establishes a connection between the component and the store, allowing the component to access the store’s state and methods. This connection is reactive, meaning that any changes to the state will trigger reactivity in the component.[16]

2.4 Related Work

This thesis is largely inspired by a research paper titled ”Benchmark Comparison of JavaScript Frameworks”, submitted by Wenqing Xuat in 2021 at the University of Dublin. This research concluded that React is better in terms of overall performance while Vue was better in aspects such as painting time, build size and user experience.[17] However, unlike this thesis, the mentioned study did not take into account the beginner-level aspect and the creation of a smaller web-application consisting of 40-60 KB of build size. Therefore, this thesis aims to do just that while at the same time evaluating the latest versions of React and Vue, ensuring that the research reflects the current state of these frameworks.

3 Methodology

This chapter presents the research methodology used for this thesis. Section 3.1 defines the research phases and Section 3.2 defines the research method before thesis-specific conditions are explained in Section 3.3. Evaluation criteria is defined in Section 3.4. Then, research instruments are revealed in Section 3.5 before sampling is explained in Section 3.6. Lastly, validity threats are defined in Section 3.7.

3.1 Research phases

In 1945, George Polya designed a four-step method for solving mathematical problems. Subsequently, this approach became popular in other scientific fields such as computer science and engineering.[18] Polya's four step method is used in this thesis:

1. Defining the problem: The problem is identified and explained in Chapter one.

2. Planning the research: To initiate the research, a pre-study is conducted and presented in Chapter two, where related work is reviewed along with the theory necessary to establish a sufficient knowledge base for this research. Subsequently, the research methods and evaluation criteria are carefully decided and presented in this chapter.

3. Conducting the research: This is explained in Chapter four and presented in Chapter five.

4. Evaluating the research: This is presented in Chapter six.

3.2 Research method

A qualitative approach is used in this study. This allows the research to provide an understanding of reasons and motivations while using small samples of data.[19]. Despite utilizing quantifiable data in this study, the research approach cannot be categorized as quantitative since the main characteristics of the quantitative approach is having large samples of data, which is not the case for this research.[20]

Furthermore, a qualitative approach acknowledges the subjective nature of evaluation criteria such as documentation and simplicity, which cannot be solely quantitatively measured. Embracing a qualitative approach enables the exploration of subjective observations.

Additionally, a qualitative research provides flexibility in adapting the research design and data collection methods during the research process. As the field of front-end development is constantly evolving, using qualitative methods allows the researcher to capture current perspectives and experiences with React and Vue frameworks.

3.3 Defining thesis-specific conditions

Because the purpose of this thesis is map out similarities and differences between React and Vue when it comes to developing simple web applications for beginners, there has to be a definition for what a simple web application actually is and who qualifies as a beginner.

In the context of this thesis, beginners are defined as people with a basic understanding of JavaScript, CSS and HTML along with general programming principles but with no experience in using the mentioned frameworks, while simple web applications are defined as programs which a beginner can learn to code in 50-100 hours while maintaining a build size of 40-60 KB.

3.4 Evaluation criteria

This subsection provides specific details about the evaluation criteria used in this study. It is divided into two categories: performance, which encompasses execution time, memory allocation, and build size, and ease of use, which includes documentation, simplicity, and time consumption. The explanations of execution metrics are presented in Section 3.4.1, followed by memory allocation in Section 3.4.2, build size in Section 4.3.3, documentation in Section 3.4.4, and simplicity in Section 3.4.5. The approach for measuring time consumption is described in Section 3.4.6. A summary of all evaluation criteria can be found in Table 3.1.

Criteria
Execution time
Memory allocation
Build size
Documentation
Simplicity
Time consumption

Table 3.1: Evaluation Criteria

3.4.1 Execution time

In the context of this thesis, the execution metrics refer to the time it takes for the web application to load and become interactive in the user's browser. The combined average loading time for successfully logging in, registering, submitting a workout, liking the workout, deleting the workout and finally logging out is measured. Four measurements criteria are used and are measured through Google Chrome's performance tab:

1. Loading: This phase involves fetching and loading HTML, CSS, and other resources required by the web page. It includes network requests and the time taken to download and process these resources.

2. Scripting: This measures the time it takes to execute the JavaScript code of the web application. It includes parsing, compiling, and running the JavaScript code.

3. Rendering: This encompasses the time to generate and display visual elements of the user interface. It includes constructing the DOM, applying CSS styles, and calculating the layout of elements.

4. Painting: The time it takes to render pixels on the screen.

3.4.2 Memory allocation

Memory allocation plays a crucial role in determining the efficiency and performance of web applications and is therefore one of the criteria for assessing the performance aspect. To determine how well memory allocation works in both frameworks, arrays of different sizes are created with elements in each place in the array, and they are then copied to different arrays. The performance of both frameworks is monitored and compared to determine how they perform in relation to each other. These findings contribute to the thesis by providing empirical evidence and a solid foundation for evaluating the performance of React and Vue.

3.4.3 Build size

Minimizing the build size is a crucial in web development due to its impact on website performance. Compressing files effectively contributes to optimizing web application performance.[21] Therefore, the build size of both React and Vue applications are measured in order to compare the additional size requirements of each framework for accomplishing the same purpose. By examining the build sizes, insights are gained into how efficiently each framework utilizes resources and determine which one offers more compact solutions.

Moreover, the build size serves as an indicator of an application's scale, enabling comparisons with related studies. Larger applications often necessitate more resources and longer build times, whereas smaller applications can benefit from a lightweight framework. Consequently, knowing the build size allows for evaluation and comparison of frameworks utilized in applications of varying sizes and complexities.

3.4.4 Documentation

Because it is not realistic for this research to study all documentation from both React and Vue, the selected data is based on the specific programming needs of the application being developed for the comparison. The documentation is evaluated continuously, as the researcher encounters challenges during the development phase and has to look up specific parts of the documentation. This sampling approach allows for targeted and relevant data collection, as the documentation that is most relevant to the actual code and components being written is considered. The documentation is evaluated based on its quality, including factors such as clarity, comprehensiveness, and ease of use, from an inductive reasoning perspective.

3.4.5 Simplicity

When evaluating the simplicity of React and Vue frameworks for building the mentioned application, simplicity refers to how easy it is to understand, learn, and use the frameworks to achieve the desired functionality. This assessment is based on subjective factors, including code readability, ease of coding and debugging, community support, quality of state management libraries, and the overall developer experience.

3.4.6 Time consumption

To compare the time required for each framework, an alternating approach is followed. Instead of developing a complete application using one framework before moving on to the other, both frameworks are continuously used in parallel to build separate applications. Equal amounts of time are allocated to each framework until one application is completed, and then the focus shifts exclusively to the remaining framework.

This approach aims to minimize bias by ensuring that the development process for one framework is not influenced by the knowledge and experience gained while working on the other framework. By alternating between React and Vue and dedicating equal amounts of time to each, the aim is to approach each framework with a fresh perspective and avoid any undue advantage or disadvantage that might arise from the order of development. This allows for a more objective comparison of the time required and the inherent capabilities of each framework.

3.5 Research instruments

The applications in this scenario are developed using Node.js as the back-end server runtime environment and leveraging the NPM package manager for initialization. JavaScript is the programming language used, written in Visual Studio Code version 1.78. The versions utilized for React and Vue are 18.2.0 and 3.2.47, respectively. For development and performance measurement, Google Chrome version 113.0.5672.127 is selected along with its performance tab.

Regarding libraries, version 4.2.1 of Redux was used for React while version 2.1.3 of Pinia was used for Vue.

The applications are developed locally on the researcher's computer, eliminating the need for cloud services or other external environments. This approach ensures that the frameworks are tailored to the specific requirements of the research project and provides greater control over the development process. The use of local development also minimizes the impact of different hardware configurations on the research results. Furthermore, by developing applications locally, data privacy and security are maintained since sensitive research data does not need to be uploaded to external cloud services.

The entire study is done on the researcher's laptop. This computer has the following specifications:
Processor: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx, 2.10 GHz
RAM: 8,00 GB
System type: 64-bit operating system, x64-based processor

3.6 Sampling

The frameworks that are picked for this research are chosen based on the current trends on Stack Overflow, which were presented in Figure 2.1. These trends show that as of May of 2023 React and Vue are the most trending JavaScript front-end frameworks.

Regarding libraries for state management, Pinia is chosen for Vue since it is the official state management library for the Vue framework.[22] For React, the library with most NPM downloads is picked. It turns out that Redux has the most with more than twice as many as its main competitors Zustand and Xstate as of May 2023.[23][24][25]

3.7 Validity threats

The validity threats of this research are defined in this subsection, encompassing credibility in Section 3.7.1, transferability in Section 3.7.2, dependability in Section 3.7.3 and confirmability in Section 3.7.4. All validity threats are then analyzed in Section 6.3.

3.7.1 Credibility

Credibility refers to the degree of trustworthiness and reliability that can be attributed to the research findings. It reflects the confidence one can have in the accuracy and integrity of the information presented.[26]

3.7.2 Transferability

Transferability, in the context of qualitative research, refers to the extent to which the findings and interpretations can be applied or transferred to other contexts or settings beyond the specific study and participants. It is often considered the interpretive counterpart to generalizability, which is more commonly associated with quantitative research.[27]

3.7.3 Dependability

Dependability refers to the extent to which research findings can be replicated and yield consistent results. In the context of qualitative research, achieving high dependability can be challenging, as qualitative studies often emphasize rich, contextual understanding rather than strict replication. However, it is still possible to enhance the dependability of qualitative research by providing clear and transparent documentation of the research process.[28]

3.7.4 Confirmability

Confirmability refers to the extent to which the findings of an inquiry can be verified or validated by independent researchers. It emphasizes the importance of establishing the reliability and trustworthiness of research outcomes.[29] Furthermore, confirmability is "concerned with establishing that data and interpretations of the findings are not figments of the inquirer's imagination", which can be interpreted as how the researcher's inherent bias is not affecting the findings.[28]

4 Work

In this chapter, a detailed description of the degree project and how the research method is applied is provided, which form the basis of the research of this thesis. The practical aspects of the study, including the development of the application in both React and Vue, is presented. Section 4.1 provides an overview of the entire applications that were developed and Section 4.2 describes the details on how the applications were created. Subsequently, Section 4.3 explains how data was collected along with how observations were made.

4.1 Overview of the applications

This section provides a comprehensive overview of the applications' purposes and functionality. Section 4.1.1 highlights the key features and functionalities present in both the React and Vue applications, while section 4.1.2 delves into the specific files that make up the architecture of the applications.

4.1.1 Functionality

The functionality of the React and Vue applications are identical, offering users a consistent experience regardless of the framework that is used for their session.

Upon entering the home page of the application, users are presented with a login form where they can enter their username and password. Additionally, a registration link is available on this page, leading users to a separate registration page where they can sign up using their email address, username, and password. Both the login and registration processes include credential verification in the database, ensuring that logged-in users are registered and preventing duplicate login details from being submitted.

After successful authentication, users are directed to a page where they can submit workouts. They are prompted to provide details such as the workout's title, description, and type, which can only be categorized as hypertrophy or cardiovascular. Additionally, users have the option to submit their workouts anonymously, ensuring privacy if desired.

A button on the workout page allows users to view all submitted workouts along with their respective details. Users can then like specific workouts, which updates the like count displayed on the page. Furthermore, users have the possibility to delete workouts that they have submitted, granting them control over their own submissions.

To conclude their session, users can log out by clicking a logout button, which returns them to the home page for subsequent login or registration actions.

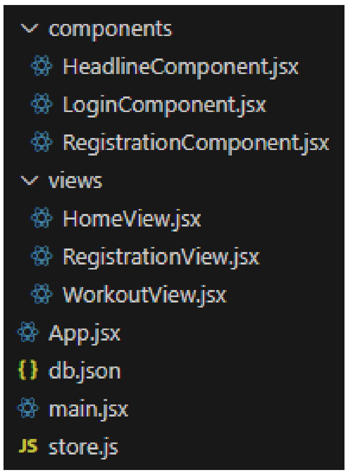
4.1.2 File structure

Figure 4.1 provides an insight into the file structure of the React application, highlighting the key files that constitute its architecture. Similarly, Figure 4.2 showcases the file structure of the Vue application, offering a visual representation of its central files.

Both architectures were designed to consist of three views that comprise the home page, the registration page, and the page where workouts are submitted and displayed to all users. The login and registration functionalities were implemented as separate components, which are rendered within their corresponding view files. Additionally, a customizable headline component was included on all pages, allowing for unique headlines while maintaining a consistent design.

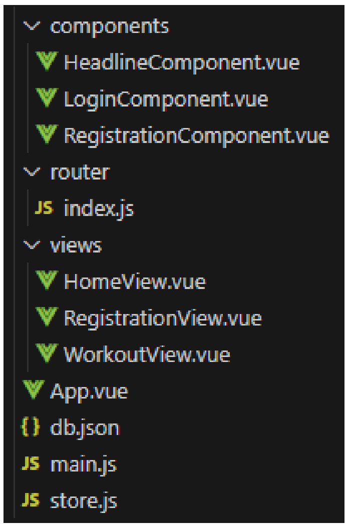
The ”db.json” file holds the database while ”store.js” helps storing the state of a logged-in user. ”App” files are responsible for routing and rendering different views in the application based on the requested URL. The ”main” files serve as the entry point for the applications and are responsible for initializing and rendering the root component.

While there are slight differences in the file structures, these variations arise from the fact that the frameworks have distinct routing implementations. The ”index.js” file in the Vue application is responsible for configuring the Vue Router and defining the available routes in the application. In React, a similar configuration is done within the App.jsx file using the ”react-router-dom” library. These differences are inherent to the frameworks themselves and are not within the control of the programmer, as they stem from the way the frameworks were designed and developed.



```
components
  HeadlineComponent.jsx
  LoginComponent.jsx
  RegistrationComponent.jsx
views
  HomeView.jsx
  RegistrationView.jsx
  WorkoutView.jsx
App.jsx
db.json
main.jsx
store.js
```

Figure 4.1: File Structure of the React Application



```
components
  HeadlineComponent.vue
  LoginComponent.vue
  RegistrationComponent.vue
router
  index.js
views
  HomeView.vue
  RegistrationView.vue
  WorkoutView.vue
App.vue
db.json
main.js
store.js
```

Figure 4.2: File Structure of the Vue Application

4.2 Implementation

This section explains in detail how each part of the application was developed, including functionality, coding and differences between the implementations of the React and Vue applications. Section 4.2.1 explains the login functionality, before diving into registration in Section 4.2.2, browsing and data sharing in Section 4.2.3, routing in Section 4.2.4, state management in Section 4.2.5 and data management in Section 4.2.6.

4.2.1 Login

The login part of the application was divided into two parts — LoginComponent and HomeView.

LoginComponent holds the HTML-like code and the logic for the login functionality. It was programmed so that when the login form is submitted, LoginComponent sends a request to the API connected to the database to fetch the list of registered users. It then compares the entered credentials with the registered user data received from the API. If a match is found, indicating valid login credentials, the LoginComponent was programmed to store the user object in the store.js file. Storing the user object in the store allows other components in the application to keep track of logged in users.

The LoginComponent file was created to be exported to the HomeView which renders the component along with other components and links, such as a headline component and a link to a registration page. This was done to avoid having too much code in one file and instead break down the view into different components.

Because Vue uses a two-way data-binding approach, changes in the login form automatically update the variables for the login credentials which can be found in the data function. This can be seen in Figure 4.3 where the v-model directive was implemented to automatically bind user input data to the relevant variables in the input object.

To achieve similar functionality in React, a React hook called useState was used. This allowed components to manage and update their own state. In the LoginComponent, useState was used to manage the username and password entered by the user. This can be seen in the code snippet from Figure 4.4, where the values of username and password are stored in state variables, and any changes to these variables trigger a re-rendering of the component, updating the corresponding inputs in the form.

By using useState, the LoginComponent in React was able to handle data flow and keep the form inputs synchronized with the component's state, similar to how two-way data binding was implemented in the Vue application.

```
<template>
  <div id="login">
    <div class="form-inputs">
      <label for="username">Username</label>
      <input type="text" id="username" name="username"
       ↪  v-model="input.username" placeholder="Username" />
    </div>
    <div class="form-inputs">
      <label for="password">Password</label>
      <input type="password" id="password" name="password"
       ↪  v-model="input.password" placeholder="Password" />
    </div>
    <button type="button" v-on:click="login()">Login</button>
  </div>
</template>

<script>
data() {
    return {
      input: {
        username: "",
        password: ""
      },
      users: []
    }
  }
(...)
```

Figure 4.3: Login Form in Vue

```
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

    const handleUsername = (event) => {
    setUsername(event.target.value);
  };

  const handlePassword = (event) => {
    setPassword(event.target.value);
  };

return (
    <div>
      <h2>Login</h2>
      <form onSubmit={handleSubmit}>
        <label>
          Username:
          <input
            type="text"
            value={username}
            onChange={handleUsername}
            placeholder="username"
          />
        </label>
        <br />
        <label>
          Password:
          <input
            type="password"
            value={password}
            onChange={handlePassword}
            placeholder="password"
          />
        </label>
        <br />
        <button type="submit">Login</button>
      </form>
    </div>
```

Figure 4.4: Login Form in React

```
fetch("http://localhost:3000/registeredUsers", {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify(newUser),
      })
        .then((response) => response.json())
        .then((responseData) => {
          console.log("Data stored successfully:");
```

Figure 4.5: Storing a Newly Registered User in the Database

4.2.2 Registration

Just like the login functionality, the registration was split into two files - RegistrationComponent and RegistrationView. RegistrationView was designed to render the RegistrationComponent along with a headline and some warnings messages that are displayed to the user so that no sensative data is submitted.

The RegistrationComponent functionality allows users to create new accounts by providing username, email address and password. The registration form captures the user input and performs validation checks to ensure the data is complete. Once the registration is successful, the user's account information is stored in the database of the application and the user is logged in to the workout page.

Similar to the login functionality, the difference between implementing the registration in React and Vue was the data-binding, which was explained in Section 4.2.1.

The main difference between implementing the login and the registration functionality was that the latter not only has to check the database but also store create a user object and store the data in the database if the registration is successful. Figure 4.5 shows how new users are registered in the JSON database. The code is written in JavaScript and is identical for both frameworks.

4.2.3 Browsing and sharing data

The applications were designed so that upon logging in, the user is presented with a new form to submit a workout. There is also a button which, when clicked, shows a list of all submitted workouts.

In React, a useEffect hook is used to keep track of changes of the states that are declared in the useState hook, that is when the submit button or the button to show all workouts is clicked, the workouts are fetched to reflect the updated database and the page re-renders to show the up-to-date data. This can be seen in Figure 4.6, where the useEffect hook, upon workout submission or when the user wants to see all workouts, fetches the new data from the JSON server.

By using the useEffect hook with these dependencies (showWorkouts and submittedWorkout), the application stays synchronized with the JSON server's database, providing users with real-time updates. This allows users to view the overall workout list accurately.

In the Vue application, the data is initialized in the data property of the component, similar to how state is initialized in React using the useState hook. When the user clicks on the ”List Workouts” button, the @click directive triggers the showWorkouts variable to be set to true, which conditionally renders the workouts component using the v-if directive.

In the Vue reactivity system, when a workout is submitted in the template, the corresponding data property is automatically updated. Vue's reactivity system tracks the dependencies between the template and the data properties, so any changes to the data are automatically detected. As a result, the user interface is re-rendered to reflect the updated data, and unlike the React program, eliminating the need for additional hooks or re-rendering logic.

While browsing through submitted workouts, the application was designed to display the number of likes for each workout. These workouts can be liked once per registered account, preventing the same user from liking the same workout multiple times. Additionally, a delete button is present on the workouts that were submitted by the currently logged-in user, requiring the website to track the user's login status and the ownership of the submitted workouts. To facilitate this

```
useEffect(() => {
    if (showWorkouts || submittedWorkout) {
      fetch(`${API_URL}/workouts`)
        .then((response) => response.json())
        .then((data) => {
          setWorkouts(data);
        })
        .catch((error) => console.log(error));
    }
  }, [showWorkouts, submittedWorkout]);
```

Figure 4.6: useEffect Hook in React

```
<Router>
        <Routes>
          <Route path="/" element={<HomeView />} />
          <Route path="/workouts" element={<WorkoutView />} />
          <Route path="/register" element={<RegistrationView />} />
          <Route path="*" element={<HomeView />} />
        </Routes>
      </Router>
```

Figure 4.7: Routing in App.jsx

functionality, the application utilized a database where the ID of each workout was stored in an array representing the workouts that a specific user has liked, and a separate array representing the workouts that the user has submitted. Consequently, the application was programmed to query the database whenever the delete or like button was pressed.

4.2.4 Routing

Routing, which is defined as navigation between different URLs within a single web-application, was an important part of both the React and the Vue application.

In the React application, the definitions for all paths were declared in the App.jsx file, as can be seen in Figure 4.7. This file serves as the entry point for the application and contains the routing configuration using the React Router library. Each defined route corresponds to a specific URL path, and when a user navigates to a particular URL, the corresponding component is rendered.

```
const router = createRouter({
  history: createWebHistory(),
  routes: [
    {
      path: "/",
      component: () => import("@/views/HomeView.vue"),
    },
    {
      path: "/signUp",
      component: () => import("@/views/RegistrationView.vue"),
    },
    {
      path: "/workouts",
      component: () => import("@/views/WorkoutView.vue"),
    },
  ],
});
```

Figure 4.8: Routing in index.js

In the Vue application, the routing was created in the index.js file, which can be seen in Figure 4.8. A router instance was created utilizing the Vue-router library. It defines the routes for different URLs within the application. The routes property is an array that contains route configurations. Each configuration specifies a path and a component. When a user visits a specific URL path, the associated component is rendered.

4.2.5 State management

To enable the application to keep track of logged-in users, state management was implemented. Redux was used in the React application while Pinia was implemented in the Vue application.

The initial state of the Redux store, when the user has not yet logged in, is an object with a loggedInUser property set to null. A reducer function, that is shown in Figure 4.9, was then implemented to modify the state. When the login action is dispatched, the reducer updates the state by setting the loggedInUser property to the payload value passed into the function. Similarly, when the logout action is dispatched, the reducer sets the loggedInUser property to null.

```
export const loginAction = (user) => ({ type: login, payload: user
↪  });
export const logoutAction = () => ({ type: logout });

const initialState = {
  loggedInUser: null,
};
const reducer = (state = initialState, action) => {
  switch (action.type) {
    case login:
      return { ...state, loggedInUser: action.payload };
    case logout:
      return { ...state, loggedInUser: null };
    default:
      return state;
  }
```

Figure 4.9: Redux React Implementation of the Reducer Function in store.js

```
dispatch({ type: "login", payload: user });
```

Figure 4.10: Storing a Logged-In User in React

Once the store file was up and running, the login actions could easily be implemented in the jsx files. Figure 4.10 shows how the program stores a logged-in user.

The store.js file written in Vue and Pinia also initiated the state as an empty string, as can be seen in Figure 4.11. The actions section defines the functions that can be used to update the state of the store and, similarly to the Redux program, two functions were implemented: loginCurrentUser and logoutCurrentUser. These functions could then be called from the other files, enabling state management in other files, as can be seen in Figure 4.12.

4.2.6 Data management

In order to manage data for the application, identical JSON servers were deployed for both applications to fetch data from REST APIs. The JSON servers served as mock APIs, storing data in JSON format and responding to requests made by the application. This approach allowed for the simulation of a back-end server, providing a realistic environment for handling user registration, login,

```
export const store = defineStore('user', {
  state: () => ({
    currentUser: '',
  }),
  actions: {
    loginCurrentUser(username) {
      this.currentUser = username;
    },
    logoutCurrentUser() {
      this.currentUser = '';
    },
  },
});
```

Figure 4.11: store.js for the Vue Application

```
const store = store()
      store.loginCurrentUser(username);
```

Figure 4.12: Storing a Logged-In User in Vue Using Pinia

and workout routine sharing. Because the focus of the research is on the front-end frameworks React and Vue rather than on back-end frameworks and website security, this was enough to conduct the study.

The database was made identical for both React and Vue and stored in a data.json file. Inside the file, the data has two JSON objects: workouts and registeredUsers. The workouts objects contain all details about workouts that the users have submitted while the registeredUsers object holds data which was submitted during successful registrations on the website along with a list of ID's of workouts that the user has liked. A snippet from the JSON file is shown in Figure 4.13, to demonstrate how the database was designed.

4.3 Data collection and observations

This section outlines the approaches employed to gather data and make observations that serve as the foundation of this thesis. The data collection process for measuring execution times is detailed in Section 4.3.1, followed by an explanation of how data was collected to measure memory allocation in Section 4.3.2. Subsequently, Section 4.3.3 elucidates the approach taken to measure build

```
{
  "workouts": [
    {
      "id": 1,
      "user": "beast2000",
      "name": "leg killer",
      "description": "leg press 5 sets x 12, squats 5 sets x 10,
      ↪  leg extensions 5 sets x 14",
      "type": "hypertrophy",
      "likes": 10,
      "isAnonymous": false
    },
  ],
  "registeredUsers": [
    {
      "id": 1,
      "email": "beasten1999@gmail.com",
      "username": "beast1999",
      "password": "krokodil123",
      "likedWorkouts": [
        1
      ]
    },
  ]
}
```

Figure 4.13: A Snippet of the Database

size. Section 4.3.4 describes the observation of documentation, while Section 4.3.5 explains how simplicity was assessed. Finally, Section 4.3.6 outlines how time consumption was measured.

4.3.1 Execution times

After starting the development server and the JSON server for both the React and Vue applications, the applications were locally run using Google Chrome as the browser. To analyze their performance, the browser's developer tools were opened, and the performance tab was accessed. The recording feature within the developer tools was activated by clicking on the record button.

Next, the login form was filled out and submitted successfully, leading to navigation to the workout page. A workout was then submitted, and upon clicking the button to show all workouts, the newly submitted workout was liked. The delete button was then clicked to remove it. Subsequently, the log out button was pressed to return to the starting page. The link to navigate to the registration page was clicked, and a new user was successfully submitted after correctly filling out the registration form. The recording was then stopped, and the performance metrics for loading, scripting, rendering and painting were observed in the performance tab. Only execution times were recorded in Google Chrome's performance tab, excluding idle time, such as the duration it took for the user to enter workout details.

By following this process, the performance of both the React and Vue applications could be assessed and compared using the recorded metrics provided by the Google Chrome's developer tools.

4.3.2 Memory allocation

To get started with assessing the memory allocation of React and Vue, a file had to be developed with identical functionality for each framework. The client-side part of this program was designed to consist of a single button and a time measurement.

When the user clicks the button, source arrays of varying sizes (10, 100,

```
1  generate() {
2          for (let i = 10; i <= 100000; i *= 10) {
3            const newArray = Array.from({ length: i }, (_, index) =>
     ↪  index + 1);
4            this.sourceArray = newArray;
5          }
6        }
```

Figure 4.14: Generating the Source Arrays

1000, 10,000, and 100,000 elements) are generated and populated with integer numbers. How the creation of these arrays was implemented in Vue can be seen in Figure 4.14. Although the code snippet is from the Vue file, the code for React is identical apart from line 4 where React instead utilizes useState to update the source array.

The UseEffect hook in React and the Vue-equivalent watch feature were implemented to make sure that when the source arrays change, a generate function is called. Each element from the source array is then copied to a into a separate destination array of the same size. The program was designed to execute 1000 iterations for each array size, aiming to provide reliable and accurate performance assessments.

Lastly, a performance API available in the Node.js v20.2.0 documentation was implemented in the code to measure the the total run time for all array sizes combined.[30]

4.3.3 Build size

To obtain the build sizes, NPM commands were executed in PowerShell. The specific command used was ”npm run build.” Running this command allowed for the generation of the build files. By using this approach, it was possible to quantify the amount of disk space each framework required when building the application.

4.3.4 Documentation

As mentioned in section 3.4.4, the evaluation of the documentation is based on the parts that are needed to develop this application. Therefore, it was natural to start looking for the parts of the manual which explain how to actually create an application with a package manager and Node modules to get started with.

After having initialized a project and created the main files, it was crucial to learn the core concepts of both frameworks. This was partly done when doing the research for the theoretical background presented in Chapter 2, albeit on a basic level. To get a firm understanding of the concepts, code examples from the documentation were read. How to manage HTML-like code, routing, data binding, hooks and components was observed from the documentation for the actual programming part of the research.

At the end, both the React and Vue documentation were thoroughly assessed and compared based on the gathered data and the experience gained from going through the relevant parts. The evaluation process involved a continuous assessment of the documentation's quality, clarity and comprehensiveness from the researcher's subjective point of view.

4.3.5 Simplicity

The simplicity of the code that was written was subjectively observed. This included the state management libraries Redux and Vue, along with re-rendering logic, hooks, component creating and overall programming experience.

4.3.6 Time consumption

To accurately measure time spent on both frameworks and at the same time limiting bias, both frameworks were used continuously until one of the applications was completed, as mentioned in section 3.4.6. In reality, this meant that during a research session, roughly half of the time would be spent on React and half on Vue until one of the applications was finished and the focus would switch to the other one.

To maintain consistency, research sessions were scheduled at the beginning of the working day, allowing for a focused and uninterrupted exploration of both frameworks. Each session consisted of a development phase with one hour allocated for each framework along with a 20 minute break between switching framework. These sessions included everything that was needed for developing the applications such as reading documentation, initializing necessary modules, writing code and debugging.

The research process followed a structured alternating pattern. The initial hour of the research was dedicated to exploring React, followed by the subsequent hour focusing on Vue. When multiple hours were allocated in a single day, each subsequent hour was dedicated to one of the frameworks, alternating between React and Vue.

To further minimize bias, the research schedule also incorporated a rotation of frameworks on a daily basis. Every day, the first hour of research was dedicated to the framework that was not explored during the initial hour of the previous day.

After each session, the time spent on a framework was recorded.

5 Result

This chapter presents the results obtained from the research, using the evaluation criteria described in Section 3.4. The evaluation criteria is divided into two main aspects: performance and ease of use. The results for performance are summarized in Section 5.1 while the results for ease of use are summarized in Section 5.2.

5.1 Performance

This subchapter presents the results related to the performance of the React and Vue frameworks. Several aspects were evaluated: Execution times in Section 5.1.1, memory allocation in Section 5.1.2 and build size in Section 5.1.3.

5.1.1 Execution times

The average execution times, defined in section 3.4.1, for ten runs, and ratios for the React and Vue frameworks were measured in Google Chrome and compared in Table 5.1.

React demonstrated better performance in loading, scripting, rendering, and total time, while Vue outperformed React in the painting metric.

Thus, considering the total ratio, React showcased a more favorable performance across the provided metrics when compared to Vue.

Criterion	**React (ms)**	**Vue (ms)**	**Ratio (React/Vue)**
Loading (ms)	6.3	6.7	0.94
Scripting (ms)	835.7	970.2	0.86
Rendering (ms)	254.2	299.5	0.85
Painting (ms)	207.8	196.5	1.06
Total (ms)	1303	1472.9	0.88

Table 5.1: Execution Times and Ratios

5.1.2 Memory allocation

The time it took to copy all generated arrays was measured and is presented in Table 5.2. The program was executed 10 times, and the average times of these runs were recorded. It is important to note that the time it took to generate the arrays or write the programming code is not included in the table.

Criterion	React (ms)	Vue (ms)
Memory Allocation	4624	4816

Table 5.2: Times to Copy Arrays for React and Vue

5.1.3 Build size

The file size for React and Vue can be found in Table 5.3. The build size for the React program was 25% larger than the Vue program.

Framework	File Size (kb)
React	54.3
Vue	43.4

Table 5.3: File Size Comparison for React and Vue

5.2 Ease of use

Results relating to ease of use are presented in this subchapter. Findings regarding documentation are presented in Section 5.2.1, simplicity in Section 5.2.2 and time consumption in Section 5.2.3.

5.2.1 Documentation

Both the React and Vue documentation provided detailed instructions on initializing an application and generating necessary modules, ensuring a smooth onboarding experience for developers. The documentation for both frameworks was beginner-friendly, offering clear and concise instructions on how to get started using various package managers and plugins. Developers were able to follow the step-by-step guides easily and set up their development environment without any issues. Thus, in terms of initialization, both React and Vue documentation were considered equal, providing excellent resources for beginner developers.

However, when it came to writing the actual programming code for the application, some differences were found. While both frameworks offered extensive examples and well-explained code snippets for various functionalities and use cases, Vue went a step further by providing a web-based integrated development environment (IDE) within its documentation.

This IDE feature of Vue's documentation allowed developers to not only run the provided code snippets but also modify the code and immediately see how the output changes. The interactive nature of Vue's IDE enhanced the learning experience, particularly for beginners. Being able to actively experiment with the code and visualize the effects of modifications facilitated a deeper understanding of Vue's features and concepts. It provided developers a more hands-on approach to learning and an environment of exploration and experimentation.

In terms of overall documentation experience, both React and Vue provide exceptional resources for developers. They offer detailed explanations, comprehensive examples, and a beginner-friendly approach, making it easy to understand and utilize the frameworks. While React's documentation provides slightly more in-depth explanations, Vue's documentation stood out as being better overall, particularly for beginner developers. The inclusion of a web-based IDE gave Vue an advantage, allowing developers to actively engage with the code and gain a deeper understanding of its features. However, React's documentation would be more suitable for developers with prior experience who can easily grasp the features without the need for an IDE-based learning environment. Since this research is primarily focused on comparing the frameworks from a beginner JavaScript developer's perspective, Vue was considered better.

5.2.2 Simplicity

Vue was found to be much simpler than React. Its two-way data-binding and automatic updates proved to be contributing to simpler and less verbose code which was considered important for a beginner developer.

Vue also offers better state management libraries compared to React. In this study, it was found that Pinia, which is only available for Vue, was easy to learn and required few code lines. On the contrary, the concepts of Redux, which was used for React, required more code and was harder to learn.

Although React has a big online community support which narrows the gap between the two frameworks in terms of learning, Vue was still easier to learn overall. This is primarily attributed to Vue's component structure, reactivity system, high-quality documentation, and two-way data-binding.

5.2.3 Time consumption

The total time spent on the React and Vue frameworks for developing the application is summarized in Table 5.4.

Framework	**Time Spent (hours)**
React	82.3
Vue	67.8
Total	150.1

Table 5.4: Time Spent for React and Vue

The time includes reading the documentation, initializing the applications, setting up APIs, coding and debugging. It does not, however, include time spent on writing the thesis or conducting any tests.

6 Analysis and Discussion

This chapter analyzes and discusses the results presented in Chapter 5. Section 6.1 touches performance while Section 6.2 is about ease of use. The chapter ends with Section 6.3 which discusses validity threats.

6.1 Performance

The findings of this study align with those mentioned in Section 2.4, which emphasized that Vue generally demonstrates faster painting times on average and has a smaller build size, while React tends to perform better overall. Thus, the results of this study reinforce the notion that Vue and React exhibit similar performance characteristics in these aspects.

The mentioned study found that Vue exhibited approximately 28.8% faster painting time. However, in this study, the observed painting time difference between Vue and React was only 6%. It is important to note that the previous research involved a significantly larger application compared to the one developed for this thesis, resulting in variations in the findings. Specifically, the file sizes in this thesis were 54.3 KB and 43.4 KB for React and Vue, respectively, while the corresponding file sizes in the previous study were 77.9 KB and 54.7 KB for React and Vue, respectively. Thus, the difference in results can be attributed to the disparity in the build sizes of the applications, as larger applications are likely to contain more components, modules, or dependencies that impact the painting time. This indicates that the execution time for a smaller beginner-level application, like the one built for this thesis, is not as important as it is for larger applications.

In terms of memory allocation, React exhibited a slightly faster performance compared to Vue, with React being 4% faster. However, the observed difference between the two frameworks in this particular aspect was relatively small. As a result, it cannot be conclusively stated that one framework was clearly superior to the other in terms of memory allocation.

Several factors contribute to the relatively small differences observed. One factor is the size of the applications, which might have minimized the variations in memory allocation performance between React and Vue. Memory allocation requirements tend to become more critical as applications grow in size and complexity. Therefore, for larger-scale applications, the differences between the frameworks in terms of memory allocation might become more obvious.

Furthermore, it is important to note that the research did not include certain optimization techniques that are commonly used in larger applications. Techniques such as code splitting, tree shaking, and other performance optimization strategies play a significant role in managing memory allocation. The absence of these optimization techniques may have contributed to the smaller observed differences between React and Vue.

Regarding loading, scripting and rendering, React demonstrated clear advantages over Vue, with ratios of 6%, 14% and 15% respectively. Even with the fact that the React file had a larger file size, it is worth noting that it does not automatically indicate inferior performance. The overall performance, such as the above mentioned aspects, play a significant role to determine how well the frameworks perform and React was 12% faster in total considering the execution metrics.

Although the study in Section 2.4 did not specifically evaluate loading, scripting and rendering, their overall evaluation indicated that React was better than Vue in terms of performance. Despite the slight differences between the studies in this regard, it can still be implied that both studies found that React performs better overall, despite the fact that Vue has a smaller build size and a faster painting time.

6.2 Ease of Use

As mentioned in Section 5.2.2, Vue was easier to learn and from observing the results presented in Section 5.2.3, React required 21.3% more time to develop the same application.

One of the main reasons was the seamless two-way data-binding and automatic updates, contributing to simpler and less verbose code. A notable illustration of this can be found in Section 4.2.3, where the React code required the inclusion of supplementary hooks and re-rendering logic, in contrast to the Vue code which re-rendered the page automatically when a value was updated.

Another reason why Vue was considered better is its approach to component creation within the .vue file. Components in Vue are organized into three distinct sections: template, script, and style. This division provides developers with a comprehensive overview of the component's structure, making the code not only easier to write but also more readable and understandable.

In particular, the template section in Vue contains the HTML-like code that defines the user interface. This aspect sets Vue apart from React, where the user interface is typically written in JSX and must be explicitly returned within the component. In Vue, the process of automatically rendering the template is seamlessly integrated into the component structure, simplifying the coding experience.

However, React was found to have a vast and active community. This strong community presence plays a crucial role in bridging the knowledge gap and providing support for React developers. The great number of unofficial guides, tutorials, and forums dedicated to React significantly aids in the learning process and facilitates the resolution of issues and debugging. The active community ensures that React developers have much resources at their disposal to help understand the core concepts of the framework. This was obvious from Figure 2.1 where React was found to have a much larger presence than Vue on Stack Overflow with recent trends indicating that roughly 5.6% of all questions asked on the website during the month of May of 2023 were related to React, while only 1.0% were related to Vue.

Despite the findings that indicate Vue's documentation favors beginners, the strong online community surrounding React narrows the gap in the learning process significantly. Documentation can only provide limited support, and developers often need to seek additional resources outside the official documentation. In the long run, the impact of documentation on the overall learning process was found to be less significant than initially thought.

One might argue that because of React's extensive community, it is easier to learn for more complex problems when the development process runs into significant problems and you need to find help and support online. However, because this thesis focuses mainly on the perspectives of React and Vue from a beginner's point of view when developing a simple web application, the findings of this report should be considered valid since the applications were developed by a beginner. On top of that, the findings that Vue's simplicity when it comes to writing code for components, state management, and, most importantly, the fact that it required 21.3% less time than React, cannot be overlooked.

6.3 Validity

The validity threats and how they were met are analyzed in this section after having previously been defined in Section 3.7. Firstly, credibility is discussed in Section 6.3.1. Then, the transferability is analyzed in Section 6.3.2 and dependability in Section 6.3.3. Lastly, the confirmability of the research is considered in Section 6.3.4.

6.3.1 Credibility

The credibility of the research findings is supported by comparing them to previous studies. As mentioned in Section 2.4, previous research has also highlighted React's better overall performance compared to Vue while also observing Vue's ease of use. While the current study found a smaller performance gap between the two frameworks in terms of execution time, the overall conclusion remains consistent with previous research. Therefore, the credibility of the findings is reinforced by their alignment with established knowledge in the field.

Furthermore, the use of established tools in the software industry enhances the credibility of the research findings. By utilizing widely recognized and reliable tools such as Google Chrome's performance tab and a performance API from Node.js, the study ensures that the measurements are accurate and can easily be comparable to similar studies in the field. These tools provide objective data, which is crucial for deriving reliable and meaningful results.

6.3.2 Transferability

The transferability of the findings is limited to the context of similar beginner-level applications. The research was conducted based on a beginner's perspective. Therefore, the findings are directly applicable to web applications with simple functionality such as registration, login and data sharing while being developed by a beginner.

6.3.3 Dependability

The dependability of the research findings is supported by the transparency and reproducibility of the research process. Chapter 3 provides a detailed description of the research method and the instruments used. An overview and a description of how data is collected along with how observations are made is described in Section 4.3. This information enables others interested in conducting similar research to understand the work and replicate the study with similar results. However, it is crucial to acknowledge that the findings of this research are specifically applicable to smaller applications (40-60 KB of build size) and from a beginner's standpoint while using the same tools that were used for this research. Consequently, it is imperative for comparative studies to be conducted under equivalent conditions to ensure that similar results are obtained.

6.3.4 Confirmability

The performance evaluation was conducted using objective and quantifiable data, ensuring a reliable basis for comparison. The findings were then compared to previous research that also reported similar trends in terms of React's overall performance superiority. This convergence strengthens the confirmability of the findings, as it indicates consistency across different studies.

In terms of ease of use, while subjective experiences played a role in assessing the frameworks, the results presented in Section 5.2.3 provide tangible evidence to support the conclusions. The observed development time difference between React and Vue directly reflects the ease of use, indicating that Vue required significantly less time to develop the same application. This objective measurement confirms the notion of Vue's simplicity discussed in Section

6.2.

7 Conclusions and Future work

This chapter sums up the entire research while reflecting on the results and discussions presented in Chapter 6. Section 7.1 presents the conclusions while Section 7.2 discusses potential future work.

7.1 Conclusions

During the last few decades JavaScript has emerged as the most popular programming language for building web applications. Because of this, several frameworks have been built for this language. JQuery was the most widely used JavaScript framework for many years but is now being replaced by new frameworks such as React and Vue. The problem in this thesis is that there is no research that maps out similarities and differences between React and Vue for beginners that are looking to create a simple web application. The purpose is therefore to conduct a study which maps out the differences between the chosen frameworks.

To achieve this, two identical web application were built with React and Vue. The applications required the developer to have a general understanding of programming concepts, albeit not previous experience using the chosen frameworks, making it possible to develop both applications in approximately 150 hours. A simple web-application where the user can login and share workout routines was built for both frameworks using the same back end and architecture. Performance and ease of use were then evaluated using a qualitative approach to determine which framework is most suited for developing a simple web application from a beginner's perspective.

The study has successfully tackled the problem by conducting thorough observations and employing predefined evaluation criteria. The analysis and discussion of the results provided valuable insights into the similarities and differences between React and Vue in terms of creating simple web applications for beginners. Regarding performance, previous studies and the current research consistently indicated that React performs better overall. While Vue demonstrated faster painting times and smaller build sizes in previous studies, the differences observed in this study were smaller, suggesting that for smaller beginner-level applications, the execution time may not be as critical as it is for larger applications. React showcased clear advantages in terms of loading, scripting and rendering performance, further reinforcing its overall performance superiority, despite the similar results regarding memory allocation which was found to be not so important for simple web applications.

In terms of ease of use, Vue turned out to be the more beginner-friendly framework. It was found to be easier to learn and required significantly less time compared to React for creating the same application. Vue's simplicity in terms of two-way data-binding, automatic updates, and clear component structure and easier state management contributed to a much quicker and easier development experience. Even though the better documentation for Vue turned out to be not as essential as first though due to the extensive online community of React, the less time spent on Vue and the simplicity and mentioned beginner-friendly aspects of the framework could not be overlooked, resulting in Vue being deemed the better framework in this research.

However, it is essential to acknowledge that the results obtained are specific to the particular application used in this study or those of similar size and functionality. These findings cannot be universally applied as a basis for all applications due to several factors that significantly influence framework selection. These factors may include the size and complexity of the application, the user's experience level, and the specific functionalities required. If a more experienced developer were to conduct a similar study with aspirations of creating a more complex application, the results would likely differ. On top of that, libraries such as Redux and Pinia were chosen based on popularity and a research with different libraries might have resulted in different findings.

Nevertheless, this research has provided beginner-level developers, who are looking to develop a simple web-application, with similarities and differences regarding React and Vue to help them choose the right framework. Additionally, despite the fact that the focus of this study has been through a beginner's point of view, companies who are looking to develop simple applications similar in size to the one developed in this thesis, or to just educate their employees, can gain knowledge that helps them decide how long it can be expected to learn either framework and which one to choose for their web development needs.

7.2 Future Work

This thesis focused primarily on building a simple web-application using two of the most popular front-end JavaScript frameworks. Even though this study has focused exclusively on frameworks for a single language, a more extensive research would be interesting to make in the future consisting of comparisons between other languages and their respective frameworks. Future work might include a more thorough research with more resources to be able to build both small and big applications, to see how the criteria differs when applications increase in size. This would include surveys and different types of applications in terms of size and functionality in order to get more accurate results and for different purposes. This would make the research more reliable and usable for more people, and not solely beginners.

In addition, future work could explore the use of additional libraries or state management solutions within React and Vue. This study focused on the core frameworks and their most popular libraries for state management. It would be interesting to investigate how the use of libraries such as Zustand or Vuex can affect the development experience and performance of the applications.

Furthermore, considering the dynamic nature of the web development ecosystem, it is essential to keep updating and reevaluating the conclusions drawn in this study. New versions of React and Vue may be released, introducing new features, performance optimizations, or changes in the development experience. Conducting a similar study in a few years' time with the latest versions of the frameworks would provide valuable insights into their current state and any potential improvements or changes.

Finally, it would be valuable to expand the study to include a larger sample size of beginner developers. This would help validate the findings and provide a more comprehensive understanding of the preferences and experiences of different individuals. Conducting surveys, interviews or other kinds of studies with beginner developers who have used React and Vue could provide additional perspectives on the ease of use and learning curves associated with each framework.

References

[1] Charles Petzold, *Code: The Hidden Language of Computer Hardware and Software*, ISBN 978-0735605053, pp. 350-354.

[2] Nathan Ensmenger, *The Computer Boys Take Over*, ISBN 9780262517966, pp. 83-97.

[3] Erich Gamma, Richard Helm, John Vlissides, Ralph Johnson, *Design Patterns: Elements of Reusable Object-Oriented Software*, ISBN 978-0201633610, pp. 26-28.

[4] David Flanagan, *JavaScript: The Definitive Guide*, 7th Edition, ISBN 9781491952023, pp. 359-389.

[5] ”Usage statistics of JavaScript as client-side programming language on websites,” W3techs, [Online]. Available: `https://w3techs.com/technologies/details/cp-javascript#:~:text=See%20technologies%20overview%20for%20explanations,98.4%25%20of%20all%20the%20websites`. [Accessed: Jun. 1, 2023]

[6] ”Usage statistics and market share of jQuery for websites,” W3techs, [Online]. Available: `https://w3techs.com/technologies/details/js-jquery`. [Accessed: Jun. 1, 2023].

[7] Stack Overflow, [Online]. Available: `https://insights.stackoverflow.com/trends?tags=jquery%2Creactjs%2Cangularjs%2Cvue.js%2Cmeteor%2Cember.js%2Cpolymer%2Csvelte`. [Accessed: May 31, 2023].

[8] React.dev, [Online]. Available: `https://react.dev/learn/describing-the-ui`. [Accessed: Jun. 1, 2023].

[9] Anthony Accomazzo, Nate Murray, Ari Lerner, Clay Allsopp, David Gutman, and Tyler McGinnis, *Fullstack React: The Complete Guide to ReactJS and Friends*, ISBN 0991344626, pp. 162-164.

[10] Alex Banks and Eve Porcello, *Learning React: Functional Web Development with React and Redux*, ISBN 9781491954621, pp. 121-122.

[11] Vue.org, [Online]. Available: `https://vuejs.org/guide/essentials/template-syntax.html`. [Accessed:

Jun. 1, 2023].

[12] Hassan Djirdeh, Nate Murray, and Ari Lerner, *Fullstack Vue 3: The Complete Guide to Vue.js and Friends*, ISBN 9781987595291, pp. 18-19.

[13] react-redux.js.org, [Online]. Available: `https://react-redux.js.org/tutorials/quick-start`. [Accessed: Jun. 1, 2023].

[14] redux.js.org, [Online]. Available: `https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers`. [Accessed: Jun. 1, 2023].

[15] pinia.vuejs.org, [Online]. Available: `https://pinia.vuejs.org/core-concepts/`. [Accessed: Jun. 1, 2023].

[16] pinia.vuejs.org, [Online]. Available: `https://pinia.vuejs.org/core-concepts/state.html`. [Accessed: Jun. 1, 2023].

[17] Wenqing Xu, *Benchmark Comparison of JavaScript Frameworks React, Vue, Angular and Svelte*, [Online]. Available: `https://www.scss.tcd.ie/publications/theses/diss/2021/TCD-SCSS-DISSERTATION-2021-020.pdf`. [Accessed: Jun. 1, 2023].

[18] George Polya, *How to Solve It*, ISBN 9780691164076, pp. 5-13.

[19] Pritha Bandari, *What Is Qualitative Research? | Methods Examples*, Scribbr, [Online]. Available: `https://www.scribbr.com/methodology/qualitative-research/`. [Accessed: Jun. 1, 2023].

[20] Pritha Bandari, *What Is Quantitative Research? | Definition, Uses Methods*, Scribbr, [Online]. Available: `https://www.scribbr.com/methodology/quantitative-research/`. [Accessed: Jun. 1, 2023].

[21] ”HTTP Compression,” developer.mozilla.org, [Online]. Available: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Compression`. [Accessed: Jun. 1, 2023].

[22] vuex.vuejs.org, [Online]. Available: `https://vuex.vuejs.org/`. [Accessed: Jun. 1, 2023].

[23] ”react-redux package,” npmjs.com, [Online]. Available: `https://www.npmjs.com/package/react-redux`. [Accessed: Jun. 1, 2023].

[24] ”zustand package,” npmjs.com, [Online]. Available: `https://www.npmjs.com/package/zustand`. [Accessed: Jun. 1, 2023].

[25] ”xstate package,” npmjs.com, [Online]. Available: `https://www.npmjs.com/package/xstate`. [Accessed: Jun. 1, 2023].

[26] Graneheim, U. H., Lundman, B. (2004). Qualitative content analysis in nursing research: Concepts, procedures and measures to achieve trustworthiness. *Nurse Education Today*, pp. 105-112. doi: 10.1016/j.nedt.2003.10.001

[27] Bitsch, V. (2005). Qualitative research: A grounded theory example and evaluation criteria. *Journal of Agribusiness*, pp. 75-91. doi: 10.22004/ag.econ.59612

[28] Tobin, G. A., Begley, C. M. (2004). Methodological rigour within a qualitative framework. *Journal of Advanced Nursing*, pp. 388-396. doi: 10.1111/j.1365-2648.2004.03207.x

[29] Baxter, J., Eyles, J. (1997). Evaluating qualitative research in social geography: Establishing ’rigour’ in interview analysis. *Transactions of the Institute of British Geographers*, pp. 505-525. doi: 10.1111/j.0020-2754.1997.00505.x

[30] ”Performance Timing API,” nodejs.org, [Online]. Available: `https://nodejs.org/api/perf_hooks.html#performancenow`. [Accessed: Jun. 1, 2023].