Tampereen yliopisto

Olivia Saukonoja

COMPARISON OF ADOPTION AND PERFORMANCE OF SVELTE AND REACT FRAMEWORKS

Bachelor's Thesis
Faculty of Information Technology and Communication Sciences
December 2022

ABSTRACT

Olivia Saukonoja: Comparison of adoption and performance of Svelte and React frameworks
Bachelor's Thesis
Tampere University
Bachelor's Degree Programme in Information Technology
December 2022

Web development has become more common since the 1990s as the number of Internet users has increased and web technologies have developed. Nowadays, web development is strongly driven by web frameworks, which are a collection of software components. Since there are many different frameworks available today, it is important to evaluate which one is the most suitable for a particular application before starting development. The objective of this thesis is to introduce React and Svelte frontend application frameworks and to compare their adoption and performance.

The comparison of adoption was made by analyzing the syntax, number of libraries, documentation, and community according to the comparison model for agile web frameworks. It was shown that Svelte's code syntax is easier to understand and faster to learn than React's. However, the number of libraries is bigger for React than for Svelte, React's documentation is wider than Svelte's, and React's community is larger and more useful to a developer learning and using the framework. This indicates that learning Svelte is faster than learning React due to Svelte's syntax, but learning advanced web development with React is easier than learning it with Svelte due to the number of libraries and documentation, the size of the community and the maturity of the framework.

According to studies, Svelte's script execution time, when creating components as a binary tree, updating static elements, and updating a component tree with static content, is lower than React's. The reasons for this are lower fixed costs and differences in processing static content and updating the DOM. Duration of a full render cycle in creating components as a binary tree is also smaller in Svelte than in React due to smaller file sizes and lower script execution time. In conclusion, Svelte is more efficient than React.

Keywords: framework, web development, React, Svelte, performance, adoption

TIIVISTELMÄ

Olivia Saukonoja: Omaksumisen ja tehokkuuden vertailu React- ja Svelte-sovelluskehyksissä
Kandidaatintyö
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
Joulukuu 2022

Web-kehitys on yleistynyt 1990-luvulta lähtien, kun Internetin käyttäjien määrä on kasvanut ja verkkoteknologiat ovat kehittyneet. Nykyään web-kehitystä ohjaavat vahvasti sovelluskehykset, jotka ovat kokoelma ohjelmistokomponentteja. Koska nykyään on saatavilla monia erilaisia sovelluskehyksiä, on tärkeää arvioida, mikä niistä on sopivin tiettyyn sovellukseen ennen sovelluksen kehityksen aloittamista. Tämän opinnäytetyön tavoitteena on esitellä React- ja Svelte-frontend-sovelluskehykset ja vertailla niiden omaksumista ja tehokkuutta.

Omaksumisen vertailu tehtiin analysoimalla syntaksia, kirjastojen määrää, dokumentaatiota ja yhteisöä ketterien sovelluskehysten vertailumallin mukaisesti. Vertailun tulokseksi saatiin, että Svelten syntaksi on helpompi ymmärtää ja nopeampi oppia kuin Reactin. Kuitenkin Reactin kirjastojen määrä on suurempi kuin Sveltellä, Reactin dokumentaatio on laajempi kuin Svelten, ja Reactin yhteisö on isompi ja hyödyllisempi sovelluskehystä opettelevalle kehittäjälle. Tämä osoittaa, että Svelten oppiminen on nopeampaa kuin Reactin oppiminen syntaksin vuoksi, mutta edistyneen web-kehityksen oppiminen Reactilla on helpompaa kuin Sveltellä kirjastojen ja dokumentaation määrän, yhteisön ja sovelluskehyksen kypsyyden vuoksi.

Tutkimusten mukaan Svelten komentosarjan suoritusaika luotaessa komponentteja binääripuuna, päivitettäessä staattisia elementtejä ja päivittäessä staattista sisältöä sisältävää komponenttipuuta on pienempi kuin Reactilla. Tämä johtuu matalammista kiinteistä kuluista ja eroista staattisen sisällön prosessoinnissa sekä DOM:in päivittämisessä. Kokonaisen renderöintisyklin kesto luotaessa komponentteja binääripuuna on Sveltellä myös pienempi kuin Reactilla pienempien tiedostojen ja pienemmän komentosarjan suoritusajan takia. Yhteenvetona voidaan todeta, että Svelte on tehokkaampi kuin React.

Avainsanat: sovelluskehys, web-kehitys, React, Svelte, tehokkuus, omaksuminen

CONTENTS

1. INTRODUCTION 1
2. RESEARCH METHOD 3
3. WEB FRAMEWORKS 4
4. REACT 5
    4.1 React components and VDOM 5
    4.2 React hooks 6
5. SVELTE 8
    5.1 Updating the DOM in Svelte 8
    5.2 Svelte components 8
    5.3 State and reactivity in Svelte 9
6. COMPARISON OF ADOPTION 11
    6.1 Comparison of syntax 11
    6.2 Comparison of the number of libraries 12
    6.3 Comparison of the amount of documentation 13
    6.4 Community size comparison 14
7. COMPARISON OF PERFORMANCE 15
    7.1 Rendering strategies of React and Svelte 15
    7.2 Results of performance tests 17
8. CONCLUSIONS 18
9. DISCUSSION 20
REFERENCES 21

TERMS AND ACRONYMS

DOM	Document Object Model
ECMAScript	JavaScript standard
JSX	JavaScript Syntax Extension
POC	Proof Of Concept
SSR	Server-Side Rendering
UI	User Interface
VDOM	Virtual Document Object Model

1. INTRODUCTION

Web development has become more common since the 1990s as the number of Internet users has increased and web technologies have developed. Nowadays, web development is strongly driven by web frameworks, which are a collection of software components. Web frameworks provide basic functionality for web applications, such as session management, view creation, and data persistence, and they are intended to speed up and enhance web application development (Vosloo & Kourie 2008).

The objective of this thesis is to introduce React and Svelte frontend application frameworks and to compare their adoption and performance from a web developer's point of view. React, also known as React.js or ReactJS, is developed by Meta and it is the most downloaded frontend application framework in 2022 with more than 16 million downloads per week (Potter 2022). Svelte is relatively new in the field of frontend application frameworks. The first version of Svelte was released in 2016, and since version 3 was released in 2019, its downloads have been increasing constantly. In both 2020 and 2021, Svelte had the highest user satisfaction percentage compared to nine other frontend application frameworks (Greif 2022) and has emerged as the most loved application framework in Stack Overflow's 2021 survey (2021 Developer Survey 2022).

Since there are many different frameworks available today with different features, it is important to evaluate which one is the most suitable for a particular application before starting the development. Using an unsuitable framework in the application can reduce software quality, increase development costs and time, and lead to problems when required features in the application cannot be found in the framework in use. Also, avoiding changing from one framework to another in the middle of the project is important, because the time spent performing the migration is often greater than or equal to the time spent in the development using the old framework (Ferreira et al. 2022).

Application frameworks can be compared using a comparison model for agile web frameworks, that has pre-defined evaluation criteria for comparison. The criteria are security, usability, testing, presentation, domain description and persistence, service orientation, component orientation and adoption. Each criterion has a set of parameters, that define basic features of the criterion in question. (Fernández-Villamor et al. 2008) This thesis compares adoption with the parameters from the comparison model.

There is previous literature about comparing different frameworks, for example between Angular and Svelte (Tripon et al. 2021; Tyhjälä 2021; Białecki & Pańczyk 2021) and Angular, React and Vue.js (Paavilainen 2019), but there are no scientific articles in comparing React and Svelte in adoption and performance. Comparing the speed of adoption is important, because if a framework is known to be suitable for the application, but it is new for the developers, it is important to know how much effort it takes to adopt it. It is also important to evaluate if the framework suits for larger applications or applications that require high performance, which is the reason why the performance is also compared.

In this thesis web frameworks are introduced in the chapter 3, React and Svelte and their basic features are introduced in the chapters 4 and 5, then the comparison of adoption is done in the chapter 6 and the comparison of performance is done in the chapter 7. The chapter 8 presents the conclusions, and in the chapter 9, there is discussion about the most important results.

2. RESEARCH METHOD

This thesis is a literature review. It was implemented by making database searches with Andor, the search engine of Tampere University's library. The search words consisted of the key words, such as "React", "Svelte", "web development", "framework", "performance", "adoption", and "compar*" and they were chained with AND- and OR-operators. For example, information about adoption of frameworks was searched with keywords "javascript AND framework AND adoption" which gave 160 search results, information about performance of frameworks was searched with keywords ""web framework" AND performance AND react" which gave 14 search results and information about Svelte was searched with keywords "svelte AND framework", which gave 36 search results. Some of the references were also found from articles on the web, such as software developers' community forum Stack Overflow and DailyJS, that publishes news about web development.

The search results were reviewed by reading the summaries of the results. Results were filtered to articles and conference publications. Also, surveys and frameworks' official documentation are used as references. If an article was considered relevant, it was read through entirely and utilized in the thesis.

The relevance of the references was considered based on the publish time, authors, contents, and scientific nature. The publish time was considered more relevant, the newer the reference was. Most of the references were published after year 2010, when the use of web frameworks increased, and especially in the 2020s, when Svelte and React had gained popularity in the software developer community. Authors were considered relevant if they were professionals on the field, and contents were considered relevant if they handled some part of the subject in a new or different perspective. Many of the references are scientific and peer reviewed.

3. WEB FRAMEWORKS

A webpage consists of three areas: markup, styling, and functionality. For each purpose, there is a separate programming language. Markup is implemented with HTML, styling with CSS and functionality with JavaScript. JavaScript is the main programming language for making webpages interactive, and it is a lightweight, event-based, and object-oriented scripting language. Nowadays using plain JavaScript has been largely replaced with different web frameworks and libraries, that help developers to reuse code and save time.

Web frameworks can be classified as either server-centric or browser-centric. Server-centric means that the logic of the software resides on the server, while the logic of browser-centric software resides in the browser. (Vosloo & Kourie 2008) Browser-centric web frameworks can also be called frontend application frameworks, and server-centric web frameworks backend application frameworks.

The use of frontend application frameworks in web development has been growing throughout the 2010s. The number of downloads of different application frameworks has been varied constantly, and the most downloaded framework in 2022 is React (Potter 2022). Other widely downloaded frontend application frameworks include for example jQuery, Angular and Vue.js (Vailshery 2022). The application framework jQuery's downloads increased at the same pace as React's until 2016, after which the use of React has been significantly more common than jQuery's. Application framework Angular's downloads started to increase in 2014, but Vue.js application framework overtook Angular in 2016 in the number of downloads, becoming the third most downloaded application framework.

Frontend application frameworks execute scripts, that are a set of consecutive commands written with a scripting language (Tietotekniikan termitalkoot 2005). Different frameworks handle the execution in different ways. New application frameworks are constantly being developed and new ones replace older application frameworks as the technologies develop and new ways to execute scripts are implemented. Svelte is a relatively new framework, and since it has succeeded in software developers' surveys, it is likely that its use will increase in the next few years.

4. REACT

React is a frontend application framework developed by Meta, previously known as Facebook, to establish Facebook and Instagram websites with better user experience. In 2013, Meta released React as an open-source JavaScript library for developers and companies (Xing et al. 2019). Meta also released React Native framework, that is used to develop mainly mobile applications for iOS and Android (Ranisavljević et al. 2022).

In this chapter, React's main features, components and VDOM (virtual document object model) are introduced. After that, React's state- and Effect-hook are introduced and examples of them are shown. The information about components and state- and Effect-hook is utilized in the comparison of the adoption in the chapter 6, and the information about VDOM is utilized in the comparison of performance in the chapter 7.

4.1 React components and VDOM

React is a declarative and component-based programming language. Declarativity means that developers design views for each state in the application and React updates and renders the right components when the data changes. React code consists of components, that can be functional, or class based. Functional components are functions that returns JSX (JavaScript Syntax Extension), and class-based components are declared using classes from ECMAScript 6 (a JavaScript standard). (React documentation 2022a) Using components makes it easier to reuse the code and to find errors in the application.

Besides components, another main feature of React is VDOM. The application's state is stored in both JavaScript namespace and the DOM (document object model) tree of the browser window (Voutilainen et al. 2016). When a user routes to a subpage of an application, React creates an updated VDOM, and compares it with the displayed DOM. Then React re-renders only the part that has been changed, and the unchanged parts will not be re-rendered. This leads to better performance on the website. (Xing et al. 2019)

Properties, that can also be called "props", are object arguments that can be given to a component. Props are user defined JSX attributes and React passes them to the component. The component then returns the updated component with the props value included in it, and React updates the DOM to match the new value. (React documentation 2022a) Props make it easy to fit components in multiple use cases.

4.2 React hooks

A hook is a function, that allows using React features in function components without writing a class. State can be added to a function with a `useState`-function. `useState` declares a state variable, for which an initial value can be given. A name is given to the variable and to the function that updates the state. (React documentation 2022b) Hooks were added to React in version 16.8, which was released in 2019.

In the program 1, a state called `amount` is created in the function component `Example`. The state's initial value is `0`, and it is given in the `useState`-function. When a user presses the button `Increase amount`, the state is updated by calling the function `setAmount`, that was declared in the same row as the state. The change in the state can immediately be seen in the UI.

```
import React, { useState } from 'react';

function Example() {
  const [amount, setAmount] = useState(0);

    return (
      <div>
        <p>Amount: {amount}</p>
        <button onClick={() => setAmount(amount + 1)}>
          Increase amount
        </button>
      </div>
    );
}
```

Program 1. *React state-hook*

Another important hook in React is the Effect-hook. The Effect-hook allows performing side effects after React has updated the DOM. Side effects affect outside the scope of a function component. They can be for example manual DOM mutations or network requests. Effect-hooks are always run after the first render, and they can be customized to be run after every update or after the component's state has changed. (React documentation 2022c)

In the program 2, the value for the variable `total` is twice as much as the value for variable `amount`. A side effect to set the value for the dependent variable `total` is run after the first render and whenever the variable `amount` changes. It could also be customized to be run after every render by removing `[amount]` from the row 9, but it would

lead to the same output in this program, because every render happens when the `amount`-state is changed. If the braces `[]` are left empty, the `useEffect`-function would only be run after the first render.

```
import React, { useState } from 'react';

  function Example() {
    const [amount, setAmount] = useState(0);
    const [total, setTotal] = useState(0);

    useEffect(() => {
      setTotal(2*amount);
    }, [amount]);

      return (
        <div>
          <p>Amount: {amount}</p>
          <p>Total: {total}</p>
          <button onClick={() => setAmount(amount + 1)}>
            Increase amount
          </button>
        </div>
      );
  }
```

Program 2. *React Effect-hook*

Before React's version 16.8, it was required to write a class to maintain the state. Nowadays it is possible to add state to a functional component due to hooks, which makes writing code faster. Isolating the side effects' logic is also easier to implement with hooks.

5. SVELTE

Svelte is an open-source JavaScript application framework created by Rich Harris (Svelte documentation 2022). The first version of Svelte was released in 2016, and it has recently gained popularity in web development. Since version 3 was released in 2019, its downloads have been increasing constantly. (Potter 2022) Svelte also offers Svelte Native framework for building mobile applications, but it is not officially supported yet (Svelte Native documentation 2022).

In this chapter, the compiling process in Svelte is explained. After that, Svelte's components, state, and reactivity in Svelte are explained. The information about components, state and reactivity is utilized in the comparison of adoption in the chapter 6, and the information about the DOM manipulation is utilized in the comparison of performance in the chapter 7.

5.1 Updating the DOM in Svelte

Svelte is a component-based framework, and it works like a compiler. In Svelte, components are converted into imperative code at build time. The imperative code then updates the DOM. (Harris 2019b)

Because the code manipulates the DOM directly, it reduces the sizes of files, that are transferred to the browser. Smaller file sizes and compiling the code at build time lead to better performance compared to code that is compiled at run time. (Harris 2019b)

5.2 Svelte components

Components are declared in Svelte by writing one component per one `.svelte`-file. A `.svelte`-file includes three optional parts: Functionality in `<script>`-block, HTML markup, and CSS styling in `<style>`-block. (Svelte documentation 2022) By writing them in separate blocks into the same file makes the separation of functionality, markup and styling visible, and it brings together all the parts of a component. Also, the hierarchy of a webpage becomes more visible by combining components from separate files into bigger entities.

A property is a variable, that is accessible to the users of a component. In Svelte, it is declared with keyword `export`. An initial value can be given to a property, and it is the default value if no other value is given to the property. A variable, that is declared as

`const`, is not modifiable outside the component. Functions and classes can also be property values, and they are read-only outside the component. (Svelte documentation 2022)

5.3 State and reactivity in Svelte

In Svelte, a local component state is set and updated with an assignment operator (Harris 2019a). However, using array methods, such as `.push()`, will not trigger an update automatically, but it requires a subsequent assignment. In the subsequent assignment, the component will be assigned with the assignment operator, which updates the state. (Svelte documentation 2022)

In the program 3, a state called `amount` is created and initialized to value `0`. When a user clicks the button `Increase Amount`, it will be handled in the arrow function `() => amount++`, which updates the state by incrementing it with one. The functionality of the program 3's code is the same as the program 1's functionality.

```
<script>
  let amount = 0;
</script>

<p>Amount: {amount}</p>
<button on:click={() => amount++}>
  Increase amount
</button>
```

Program 3. *Svelte state in a component*

The assignments are reactive by nature, but statements are not, because `<script>`-blocks are run only when the component is created. If a statement needs to be reactive, it must be marked separately by writing `$:` before assigning it for the first time. After that, updating the variable will trigger the code to be run automatically. (Svelte documentation 2022)

In the program 4, the variable `total` is made reactive with the `$:`-expression. When the `amount`-variable's value is changed, `total`-variable's value will be updated automatically to be twice as much as `amount`. The functionality of the program 4's code is the same as the program 2's functionality.

```
<script>
  let amount = 0;

  function addToTotal(value) {
    return value + amount;
  }

  $: total = addToTotal(amount);
</script>

<p>Amount: {amount}</p>
<p>Total: {total}</p>
<button on:click={() => amount++}>
  Increase amount
</button>
```

Program 4. *Reactivity in Svelte*

When Svelte compiles the components into imperative code, it will only look at the variables that are assigned to and used within a block. This means that variables without dependent variables work correctly in any order they are declared. In case of variables with dependent variables, the order of the declaration matters: A variable with a dependent variable must be declared and called before declaring the dependent variable. (Svelte documentation 2022)

6. COMPARISON OF ADOPTION

Comparing the speed of adoption is important, because if a framework is suitable for the application, but it is new for the developers, it is important to know how much effort it takes to adopt it. Generally, it might be worth to learn a new framework if it suits well for the application to be developed. However, if the web application is going to be small or just a POC (Proof Of Concept), it might be faster to develop it with a framework that is already familiar for the developer(s). If no frameworks are familiar for the developer(s) beforehand, it might be important to start the development with a framework, that is easy to adopt.

In the comparison model, adoption is compared with a set of parameters. The parameters are documentation and code, number of libraries developed in the framework's language, community of users that provide support, number of programmers in the framework's language, maturity of framework's technologies, and maturity of the application servers that can run framework's web applications. (Fernández-Villamor et al. 2008)

React is more mature than Svelte, because it is older, and it has been developed by dozens of developers from a big company instead of an individual. In this chapter, React's and Svelte's syntax, libraries, documentation, and community are evaluated and compared from the adoption perspective. Syntax is evaluated separately because it affects on how quickly the development can be started. The parameter of maturity of the application servers is left unevaluated because it can be evaluated only for backend application frameworks.

6.1 Comparison of syntax

In the program 1, which is an example of React state, there are `import`-statements and a function declaration. Inside a function, there can be JavaScript and React statements, such as React's `useState`-hook, as in the program 1. Properties can also be given to a component as function attributes, and the styling of the component can be added by adding external `.css`-files or by adding inline styling directly to each HTML-tag (React documentation 2022a).

In the program 3, there is an example of Svelte's state. The code consists of `<script>`-block, which includes JavaScript and Svelte code, and HTML markup. There could be styling in `<style>`-block or as in React, inline styling in HTML-tag. Properties to a component can be declared with an `export`-statement. (Svelte documentation 2022)

Separation of functionality, styling and markup in both frameworks is quite clear, though Svelte's approach is more logical, as they are in different blocks, while React's functionality and markup are inside the same function. Svelte's component styling is also clearer, because it is in the same file as the rest of the code, while React's styling is in a separate file. Both frameworks support inline styling.

React declares the state with a `useState`-hook, which is a more complicated way to write code compared to Svelte, where the state is declared with a simpler `=`-mark. In React, a dependent variable can be changed with a `useEffect`-hook, while in Svelte, it is declared with a simple `$:`-expression. However, `$`-mark has different meanings in different programming languages.

Properties in React are declared as function attributes, that are accessible with `{props.[propertyname]}` outside the scope where they are declared. In Svelte, properties are declared with a keyword `export`, that is accessible directly as `{propertyname}` outside the scope. Using attributes and `export`-statement are both established ways to declare modifiable values outside the scope in many programming languages but accessing the property outside the scope is easier in Svelte because it does not require the `props`-keyword in front of it.

In conclusion, the separation of functionality, styling and markup and declaration of properties are logical in both frameworks. However, component styling is clearer, declaring the state is simpler, and the properties are easier to access in Svelte than in React. Overall, Svelte's syntax is closer to plain JavaScript than React's. That's why Svelte's code syntax might be a bit faster to learn than React's.

6.2 Comparison of the number of libraries

React has many libraries and tools due to its large number of users (Khan 2022). It helps to find additional features that are missing from React and makes the development faster, because developers can use ready-made libraries without developing the features themselves. On the other hand, it also requires more research on the available options when choosing the suitable library, and it can feel overwhelming for a developer that is new to React. Svelte has a smaller ecosystem and fewer libraries, because it is newer and less known framework than React (Khan 2022), so the options for Svelte developers are more limited compared to React.

External state management libraries are important libraries for React, because sharing state among components can be complex in bigger React applications (Carmine 2022). Some state management libraries are, for example, Redux, MobX and Recoil (Abramov

et al. 2022; MobX 2022; Recoil 2022). React also requires third-party libraries on routing and form validation. SSR (server-side rendering) can be implemented with Next.js, which is a React SSR application. (Kaluza et al. 2018) Svelte does not require an external state management library because it is built in as context and stores (Svelte documentation 2022). Svelte offers routing and SSR in SvelteKit, which is an application framework that has been built with Svelte (Libby 2021, chapter 12).

For testing, Svelte offers a testing library, that includes simple computations for unit testing (Svelte documentation 2022). React has built-in testing tools, but it also offers a React Testing Library for testing React components. Code written in React can be tested by rendering component trees or running the app in a realistic browser environment. (React documentation 2022e)

In conclusion, the number of libraries is bigger for React than for Svelte. Although it requires more research on the available options, it is still less laborious for the developers than developing the features by themselves. State management is an important part of the application and because it is built inside Svelte but not in React, starting the development with Svelte is faster than with React.

6.3 Comparison of the amount of documentation

React documentation contains installation instructions, text chapters for main concepts, advanced guides for experienced web developers, API reference, text chapters for testing and a section for frequently asked questions. There is a complete tutorial on how to make a small game in React, a blog that includes blog posts about React versions, and a list of community resources, such as team, support, courses, examples, meetups, conferences, articles, podcasts, videos, and external resources. (React documentation 2022f) In StackOverFlow, there are almost 500 000 questions, that are tagged with `reactjs`-tag in the end of 2022 (Questions tagged [reactjs] 2022).

Svelte documentation contains detailed API reference documentation. In Svelte's homepage, there are also a blog that highlights Svelte's new features, a section for frequently asked questions, and a link to Discord chatroom, where support is available. There are also interactive examples and tutorials, that can be modified directly in the browser. (Svelte documentation 2022) In StackOverFlow, there are about 4000 questions tagged with `svelte`-tag in the end of 2022 (Questions tagged [svelte] 2022).

React's documentation is longer than Svelte's, and there are more resources where it is possible to learn and find support. Svelte's documentation is comprehensive, but remarkably shorter than React's. On the other hand, because React has been developed for a

longer time and for example hooks were not introduced until version 16.8, there is also more outdated information about React than Svelte.

6.4 Community size comparison

React is the most used web frontend framework with over 16 million downloads per week (Potter 2022), and as mentioned in the chapter 6.3, React has almost 500 000 StackOverFlow questions. React is also the most wanted frontend application framework with percentage of 25 (2021 Developer Survey 2022), which means that there are new developers that are open to using React. React's community is large, and it is still growing.

Svelte has the highest user satisfaction percentage compared to nine other frontend application frameworks (Greif 2022) and is the most loved application framework in Stack Overflow's 2021 survey (2021 Developer Survey 2022). Svelte has about 300 000 downloads per week, and the number has been gradually growing over the years (Potter 2022).

In conclusion, Svelte's community members are satisfied using the framework, while React's community is bigger than Svelte's. Svelte's community has been growing gradually, but the difference is still significant. From the adoption perspective, the size of the community is the most important factor for a developer learning a new framework.

7. COMPARISON OF PERFORMANCE

Performance means how quickly web application's content loads and renders in a browser, and how well it responds to user interaction. Good performance of a web application makes the application quick and responsive, while bad performance affects negatively on the user experience and can lead to user leaving the site. At worst, bad performance can make the website inaccessible. (MDN web docs 2022)

Performance can be compared by measuring loading and rendering times of a web application. Loading means the amount of time that it takes to download the application to the browser and how quickly the application is visible for the user. Rendering means the amount of time it takes for an application to be usable for the user. (Tietotekniikan termitalkoot 2010) Comparing the performance of web frameworks is important because web frameworks have different rendering strategies and different approaches to how they manage the state of UI. In this chapter, React's and Svelte's DOM manipulation and rendering strategies are compared and results of performance tests found in the literature are described.

7.1 Rendering strategies of React and Svelte

The difference in updating the DOM, is that React uses virtual DOM (Xing et al. 2019), and Svelte manipulates the DOM directly (Harris 2019b). The figure 1 shows the diffing process for Svelte and React. Svelte's components are compiled at build time to native JavaScript, that updates the DOM directly. React's components trigger an update in the virtual DOM, that is compared with the previous virtual DOM, and then the changes are applied to the displayed DOM.

Even though React re-renders only the part that has been changed in the DOM, Svelte's approach is even more efficient. It reduces the sizes of files, that are transferred to the browser, and compiles the code at build time, which is more efficient than building it at run time (Harris 2019b). The optimization for React is minimal, because there is no HTML code that needs to be parsed, and the entire code is in JavaScript (Kaluza et al. 2018).

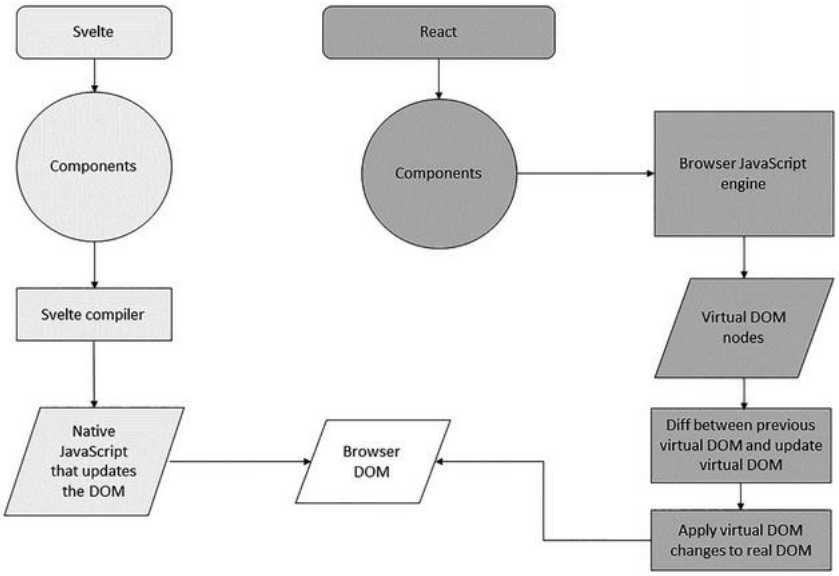

Figure 1: *The diffing process for Svelte and React (Libby 2021, chapter 1)*

The performance costs of rendering depend on input sizes and fixed costs. Input sizes are the number of components, and elements or data bindings processed per component, and they can be compared directly. When creating new components, input sizes are equivalent regardless of the rendering strategy. Fixed costs are performed for each element or data binding, and they are more complicated to compare, because their performance depends on how they are implemented. (Ollila et al. 2022)

When components are updated, the update affects a subset of components and their descendant components, because components can share their state only with descendants. React's rendering strategy walks through the subtree of the modified component. This approach avoids going through many unaffected components, but there's still a lot of unnecessary work, as it goes through all the descendants regardless of their output. Svelte processes only the components of which output has changed, which is achieved with a reactivity system. Component's output is a part of a dependency graph at compile time, and whenever a value changes, the reactivity system automatically schedules the component and all its dependencies to be re-rendered by imperative code at runtime. (Ollila et al. 2022)

A component's output can be static, which will not change after initial render, or dynamic, which depends on data bindings and may therefore change. React processes both static and dynamic content on each render, and Svelte processes static content only on the initial render. (Ollila et al. 2022)

7.2 Results of performance tests

A full render cycle consists of script execution and re-evaluation of the critical rendering path. Frameworks are responsible for script execution, which is the reason why the script execution time is measured when comparing the performance of different frameworks. When creating components as a binary tree, script execution time is remarkably smaller in Svelte than in React, as the number of elements grow, because Svelte's fixed costs are lower than React's. Also, when updating static elements and a component tree including components with primarily static content, Svelte's script execution time is much lower than React's due to the difference in processing static content and updating the DOM. Duration of a full render cycle when creating components as a binary tree, is still smaller in Svelte than in React, though not as remarkably as the duration of script execution, because the script execution time is lower in Svelte than in React. (Ollila et al. 2022)

RealWorld-app is an application, that is used for demo purposes, but it is closer to a real application than standard "to-do" demo applications. Svelte's performance is better in RealWorld-applications compared to React RealWorld-applications, that are developed with state management libraries MobX and Redux. (Schae 2020) The reasons for this can be expected to be the differences in script execution time, static content processing and fixed costs. Compiled JavaScript files' sizes are also smaller in RealWorld-applications in Svelte compared to React, and Svelte requires half as many lines of code as React (Schae 2020).

In conclusion, Svelte's approach in updating the DOM leads to smaller sizes of files and lower script execution time. Svelte's fixed costs are lower than React's, and Svelte processes static content only on the initial render, while React processes it on every render. That's why Svelte's performance is better than React's in performance tests of binary trees and RealWorld-applications.

8. CONCLUSIONS

In this thesis, React and Svelte web frontend frameworks were introduced, and they were compared in the adoption and performance point of view. There was no scientific research on the adoption of React and Svelte frameworks, so the research was made by analyzing the syntax, number of libraries, documentation, and community according to the comparison model for agile web frameworks (Fernández-Villamor et al. 2008). The result was that Svelte's code syntax is easier to understand and faster to learn than React's due to Svelte's component styling, state declaration and properties. The number of libraries is bigger for React than for Svelte, and React's documentation is longer than Svelte's. Even though Svelte's community members are satisfied of using Svelte, React's community is bigger and more useful for a developer learning and using the framework. React's technologies are also more mature than Svelte's due to the age of and the resources used to develop the framework. The conclusions are summarized in the Table 1.

Table 1. *The results of adoption in React and Svelte frameworks*

Parameter	**Svelte**	**React**
Syntax	Very easy	Easy
Number of libraries	Small	Large
Amount of documentation	Small	Large
Size of community	Growing	Big
Maturity of framework's technologies	Not mature	Very mature

This indicates that learning Svelte is faster than learning React, because Svelte's syntax is simpler than React's. However, learning advanced web development with React is easier than learning it with Svelte due to the number of libraries and documentation, the size of the community and the maturity of the technologies.

Svelte's script execution time, when creating components as a binary tree, updating static elements, and updating a component tree with primarily static content, is much lower than React's due to differences in DOM manipulation, static content processing

and fixed costs. Also, duration of a full render cycle in creating components as a binary tree is smaller in Svelte than in React due to lower script execution time. Compiled JavaScript files' sizes are smaller in Svelte than in React, and Svelte requires less lines of code than React in similar applications due to the differences in DOM manipulation. Overall, the research agrees on that Svelte is more efficient than React.

9. DISCUSSION

In conclusion, Svelte is faster to learn but it is easier to learn advanced web development in React. That's why using Svelte would be useful for applications, that need to be started quickly, and React would be better for applications, that will become large or complex. Svelte's documentation, libraries and community are growing, so in some time, Svelte might also be applicable for more complex applications. However, if performance and user experience are prioritized, Svelte would be more applicable to large web applications than React, because Svelte's performance is better than React's.

According to *On the (un-)adoption of JavaScript front-end frameworks*, popularity and learnability are the key factors that motivate the choice of frontend frameworks (Ferreira et al. 2022). This would implicate that most of the developers would choose React over Svelte, because even though there are differences in the adoption, React's learnability is still good, and its community is remarkably bigger than Svelte's.

There could be more scientific research on the adoption of different frameworks, because now the research was made by analyzing them from the resources from the web and frameworks' own documentation according to the comparison model. This is not directly applicable to the real adoption of those frameworks, because the observations made from the syntax were my own, and not a result of actual, examined adoption of the frameworks.

Research of the performance tests covers well different aspects, such as script execution and full render cycle, and different component trees. However, the components used in the comparison of script execution time are quite simple because they don't for example have any custom styling, so in a real-world application, they would be more complicated. RealWorld-application performance tests are closer to a real-world application, but on the other hand, the RealWorld-application performance tests tested only one application, and didn't test for example different component trees in the application.

Even though this thesis compares React and Svelte in adoption and performance, it is important to acknowledge, that adoption and performance are only two perspectives to compare frameworks. Depending on the project, it could also be important to compare for example security, usability, testing, presentation, domain description and persistence, service orientation and component orientation. Also, framework's support for mobile development could be important aspect to compare, when choosing a suitable framework for an application to be developed.

REFERENCES

2021 Developer Survey. (2022). Stack Exchange Inc. Updated 2022. Available (accessed 23.10.2022): https://insights.stackoverflow.com/survey/2021#section-most-loved-dreaded-and-wanted-web-frameworks

Abramov, D. & the Redux documentation authors (2022). Redux. Available (accessed 13.12.2022): https://redux.js.org/

Białecki, G. & Pańczyk, B. (2021). Performance analysis of Svelte and Angular applications. Journal of Computer Sciences Institute, Vol. 19, pp. 139–143. https://doi.org/10.35784/jcsi.2633

Carmine, B. (2022). Do you really need a React state management library? Medium, Bits and Pieces. Updated 14.3.2022. Available (accessed 5.12.2022): https://blog.bitsrc.io/react-do-you-really-need-an-external-library-for-state-management-28f67d03ebe5

Fernández-Villamor, J.I., Díaz-Casillas, L. & Iglesias, C.Á. (2008). A comparison model for agile web frameworks. EATIS '08: Proceedings of the 2008 Euro American Conference on Telematics and Information Systems. pp. 1-8. https://doi.org/10.1145/1621087.1621101

Ferreira, F., Borges, H.S., & Valente, M.T. (2022). On the (un-)adoption of JavaScript front-end frameworks. Software, Practice & Experience. Vol. 52(4), pp. 947–966. https://doi.org/10.1002/spe.3044

Greif, S. (2022). State of JS. Updated 2.2.2022. Available (accessed 23.10.2022): https://2021.stateofjs.com/en-US/libraries/front-end-frameworks

Harris, R. (2019a). Write less code. Svelte blog. Updated 20.4.2019. Available (accessed 7.11.2022): https://svelte.dev/blog/write-less-code

Harris, R. (2019b). Svelte 3: Rethinking reactivity. Svelte blog. Updated 22.4.2019. Available (accessed 7.11.2022): https://svelte.dev/blog/svelte-3-rethinking-reactivity

Kaluza, M., Troskot, K. & Vukelic, B. (2018). Comparison of Front-End Frameworks for Web applications. Journal of the Polytechnics of Rijeka. Vol. 6(1), pp. 261-282. https://doi.org/10.31784/zvr.6.1.19

Khan, A. (2022). React Vs Svelte: Which is Faster in 2022? Medium, CodeX. Updated 1.9.2022. Available (accessed 17.12.2022): https://medium.com/codex/react-vs-svelte-which-is-faster-in-2022-7c8ffd1f0988

Libby, A. (2021). Practical Svelte: Create Performant Applications with the Svelte Component Framework. Apress. https://doi.org/10.1007/978-1-4842-7374-6

MDN web docs. (2022). The Mozilla Foundation. Updated 13.10.2022. Available (accessed 25.11.2022): https://developer.mozilla.org/en-US/docs/Learn/Performance/What_is_web_performance

MobX. (2022). Available (accessed 13.12.2022): https://mobx.js.org/README.html

Ollila, R., Mäkitalo, N., & Mikkonen, T. (2022). Modern Web Frameworks: A Comparison of Rendering Performance. Journal of Web Engineering. Vol. 21(3), pp. 789–813. https://doi.org/10.13052/jwe1540-9589.21311

Paavilainen, P. (2019). Sovelluskehysten Angular, React ja Vue.js vertailua. Theseus. https://urn.fi/URN:NBN:fi:amk-2019053113620

Potter, J. (2022). NPM trends. Updated 23.10.2022. Available (accessed 23.10.2022): https://npmtrends.com/angular-vs-jquery-vs-react-vs-vue

Questions tagged [reactjs]. (2022) Stack Exchange Inc. Available (accessed 13.12.2022): https://stackoverflow.com/questions/tagged/reactjs

Questions tagged [svelte]. (2022). Stack Exchange Inc. Questions tagged [svelte]. Available (accessed 13.12.2022): https://stackoverflow.com/questions/tagged/svelte

Ranisavljević, T., Karabašević, D., Brzaković, M., Popović, G., & Stanujkić, D. (2022). React Native: A brief introduction to modern cross-platform mobile application development. Quaestus. pp. 120–136. https://www.proquest.com/docview/2715142674

React documentation. (2022a). Components and Props. Meta Platforms, Inc. Available (accessed 25.10.2022): https://reactjs.org/docs/components-and-props.html

React documentation. (2022b). Using the State Hook. Meta Platforms, Inc. Available (accessed 25.10.2022): https://reactjs.org/docs/hooks-state.html

React documentation. (2022c). Using the Effect Hook. Meta Platforms, Inc. Available (accessed 25.10.2022): https://reactjs.org/docs/hooks-effect.html

React documentation. (2022d). Virtual DOM and Internals. Meta Platforms, Inc. Available (accessed 7.11.2022): https://reactjs.org/docs/faq-internals.html

React documentation. (2022e). Testing Overview. Meta Platforms, Inc. Available (accessed 15.12.2022): https://reactjs.org/docs/testing.html

React documentation. (2022f). Where To Get Support. Meta Platforms, Inc. Available (accessed 15.12.2022): https://reactjs.org/community/support.html

Recoil. (2022). Facebook, Inc. Available (accessed 13.12.2022): https://recoiljs.org/

Schae, J. (2020). A RealWorld Comparison of Front-End Frameworks 2020. Medium, DailyJS. Updated 9.3.2020. Available (accessed 13.12.2022): https://medium.com/dailyjs/a-realworld-comparison-of-front-end-frameworks-2020-4e50655fe4c1

Svelte documentation. (2022). Svelte. Available (accessed 7.11.2022): https://svelte.dev/docs

Svelte Native documentation. (2022). Svelte Native. Available (accessed 8.12.2022): https://svelte-native.technology/docs

Tietotekniikan termitalkoot. (2005). Sanastokeskus ry. Updated 28.5.2005. Available (accessed 13.12.2022): https://sanastokeskus.fi/tsk/fi/termitalkoot/haku-266.html

Tietotekniikan termitalkoot. (2010). Sanastokeskus ry. Updated 26.4.2010. Available (accessed 13.12.2022): https://sanastokeskus.fi/tsk/fi/termitalkoot/haku-266.html

Tripon, T-D., Gabor, A.G., & Moisi, E.V. (2021). Angular and Svelte Frameworks: a Comparative Analysis. 16th International Conference on Engineering of Modern Electric Systems (EMES). pp. 1–4. https://doi.org/10.1109/EMES52337.2021.9484119

Tyhjälä, S. (2021). Svelte- ja Angular sovelluskehysten vertailu. Theseus. https://urn.fi/URN:NBN:fi:amk-2021112922282

Vailshery, L.S. (2022). Statista. Updated 9.8.2022. Available (accessed 23.10.2022): https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

Vosloo, I. & Kourie, D.G. (2008). Server-Centric Web Frameworks: An Overview. ACM Computing Surveys. Vol. 40(2), pp. 1–33. https://doi.org/10.1145/1348246.1348247

Voutilainen, J-P., Mikkonen, T. & Systä, K. (2016). Synchronizing Application State Using Virtual DOM Trees. ICWE 2016: Current Trends in Web Engineering. pp. 142-154. https://doi.org/10.1007/978-3-319-46963-8_12

Xing, Y., Huang, J. & Lai, Y. (2019). Research and Analysis of the Frontend Frameworks and Libraries in E-Business Development. ACM International Conference Proceeding Series. pp. 68–72. https://doi.org/10.1145/3313991.3314021