Zerihun Dinku

React.js vs. Next.js

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor’s Thesis

05 May 2022

Abstract

Author:	Zerihun Dinku
Title:	React.js vs. Next.js
Number of Pages:	36 pages
Date:	05 May 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Mobile Solutions
Supervisors:	Patrick Ausderau, Principal Lecturer

React.js and Next.js are among the JavaScript frameworks for building web applications. This thesis compares them in popularity, availability of documentation, and performance. Through a literature review in the theory part, the thesis will show the pros and cons of using each, thereby giving a starting point in decision making on choosing the proper framework based on the scale and intended purpose of the application.

Furthermore, this thesis analyzes the popularity and availability of documentation of the frameworks. Data from GitHub, Stackoverflow, NPM trends, and Google trends were analyzed to see their popularity. The materials, manuals, and tutorials provided by the official React.js and Next.js websites provide adequate documentation for both frameworks.

This paper comes with two similar applications made by React.js and Next.js for performance comparison. Lighthouse performance metrics from Google Chrome were used to measure the performance of each framework.

In conclusion, React.js outperformed Next.js based on the comparison criteria. Despite React.js popularity and better performance, Next.js offers server-side rendering that makes fast initial loading possible and improves the user experience. Overall, developing large-scale applications and conducting extensive concurrent user load testing should be performed to reach a more sensible deduction.

Keywords: React.js, Next.js, JavaScript frameworks

Contents

List of Abbreviations

API:	Application Programming Interface
CSS:	Cascading Style Sheet
CDN:	Content Delivery Network
DOM:	Document Object Model
ES6:	ECMAScript 6
HTML:	HyperText Markup Language.
HTTP:	Hypertext Transfer Protocol
JSON:	JavaScript Object Notation
JSX:	JavaScript XML
JVM:	JavaScript Virtual machine
MVC:	Model View Controller
NPM:	Node Package manager
SEO:	Search Engine Optimization
SVG:	Scalable Vector Graphics
UI:	User Interface
XML:	Extensible Markup Language

1 Introduction

JavaScript Frameworks were created to make the work of web programmers smoother, for example, by providing ready-made libraries, an active supporting community, and benefiting developers from reusable components. They must select the framework that best meets the purposes of their applications and provides high-quality code and performance. Maintainability, validity, an active and supportive community, and other criteria may affect their decision [1].

This thesis aims to compare React.js and Next.js frameworks to show the advantages and disadvantages of using them. It is also accompanied by two similar web applications built with these two frameworks. The comparison will be made by analyzing the popularity, availability of documentation, and performance of the frameworks.

The thesis begins by presenting a theoretical background. It explores the concept of JavaScript, JavaScript frameworks, and a more detailed theoretical explanation of React.js and Next.js and the tools used. The thesis also investigates the popularity of the frameworks based on the data available from popular online developers' tools. After popularity evaluation, the availability of official documentation will be summarized. Then comes the performance comparison section, where a detailed analysis will be made of the accompanied applications developed in this project.

Finally, this thesis will try to summarize the result of the comparison criteria set in the conclusion section. Based on the outcome of the comparison, it will try to show the pros and cons of these two frameworks. It will also give some recommendations on improving the result of the comparisons by using additional tests.

2 Theory

2.1 JavaScript

JavaScript came into existence in 1995. The original purpose of JavaScript was to enable interactiveness to Web pages in the Netscape Navigator browser and eventually with other browsers [2]. Initially, it was the primary scripting language integrated into browsers for implementing simple scripts that improve client-side web applications. In recent years, the language has become one of the popular programming languages. JavaScript is commonly utilized in all client-side web applications and mobile, desktop, and server applications [3].

JavaScript is known as a scripting programming language. Programs written in JavaScript can be written in the source code of a HyperText Markup Language (HTML) web page and run during the page load. Today, JavaScript can also execute on the server or any other device that has the JavaScript engine. In general, all browsers have an embedded engine called JavaScript Virtual machine (JVM). The engine reads the script and converts it to the machine language, enabling the code to run fast.

A web browser was the first host environment for JavaScript, and it is still the most frequent execution environment for JavaScript code today. Due to the introduction to the web browser environment, JavaScript code may get input from the mouse and keyboard of the user and make Hypertext Transfer Protocol (HTTP) requests. It also enables JavaScript code to display HTML and Cascading Style Sheet (CSS) output.

JavaScript code has become capable of running in a different host environment since 2010. Node.js grants JavaScript access to the complete operating system, authorizing written programs in JavaScript to write and read files, send and receive data over the network, and create and serve HTTP requests, rather than restricting JavaScript to the Application Programming Interface (API)

offered by a web browser. Node.js is a popular choice for building web servers and a handy tool for developing basic utility scripts instead of shell scripts [4].

The standardization first started for JavaScript in November 1996. The standard was designated as ECMA-262, and the standardization committee was known as technical committee -39. ECMAScript is the name of the language defined by the standard, while JavaScript is essentially a brand name for ECMAScript. JavaScript is evolving with each version of ECMAScript. At the time of writing this thesis, it reached ECMAScript 2021 edition [5].

2.1.1 JavaScript in Modern Web Development

JavaScript is one of the reasons for the shift of the modern, highly interactive web from the static read-only simple web. JavaScript can alter the content and the presentation, making it a good tool for the web. While HTML specifies the text and CSS defines the look and feel, JavaScript is in charge of determining the functionality of the application. It is what gives the internet its dynamism and interactivity.

JavaScript is used on the web, whether client-side code or server-side logic. It handles everything. It is part of the entire web market in various frameworks and libraries thus ensuring that the web development process is more manageable.

Feature-rich libraries and frameworks like Angular, React.js, and Vue facilitate web development by allowing developers to streamline complex commands into simple blocks of JavaScript code, thus making the programming process easier and faster [5].

2.1.2 JavaScript Libraries and Frameworks

Many popular front-end frameworks have been used widely in the web development process. These include React.js from Facebook, Angular from Google, Vue, and other open-source frameworks.

In addition to the front-end frameworks, JavaScript provides backend frameworks such as Node.js, Express, and Meteor.js. Node.js is a server-side JavaScript built on top of the Chrome browser JavaScript engine, designed to use JavaScript to create backend applications. Express is a server-side JavaScript framework that runs on top of Node.js which makes web application server building structured. Metero.js is another open-source real-time back-end framework built on Node.js. It functions with MongoDB and other open-source databases.

Besides frontend and backend frameworks, JavaScript provides data layer frameworks, JavaScript automation test frameworks, and test runtime environments. JavaScript is present throughout the web in the form of these powerful frameworks that offer a wealth of functionality for the entire web development process [5].

2.2 React.js

The introduction of React.js altered the development of web applications [6]. Based on the React.js official documentation, React.js allows developers to develop large, web-based applications that alter data without subsequent page updates. It is utilized as a view in Model View Controller (MVC) architectural pattern. React.js abstracts the Document Object Model (DOM) for a good experience of developers. React.js is primarily server-side rendered using Node.js, and React.js Native provides native mobile app support. React.js implements a one-way data flow, simplifying the boilerplate and proving simpler than traditional data binding [7].

The principal foundation behind React.js is the concept of virtual DOM. React.js makes effective use of the virtual DOM. Virtual DOMs can be rendered on either the client-side or the server-side and communicate with each other. The virtual DOM generates a subtree of nodes based on state changes and performs on as few DOM operations as possible to keep the component up to date instead of explicit updates [8].

2.2.1 Virtual DOM

The virtual DOM is an abstraction that allows React.js developers to handle JavaScript as reactive. It is a quick, in-memory replica of the real DOM [9]. The Virtual DOM makes it possible to rerender every state change, reflecting user activities and providing a fluid user experience.

For various reasons, rerendering the actual entire DOM upon each change is difficult. First, repainting takes a long time. Each change necessitates recalculating and reapplying styles and sizes, resulting in cascading CSS reapplications. Working directly with the DOM is considered slow due to the layout process. Secondly, data will be obscured. Re-rendering the DOM, for example, will mess with form fields and reset the scrollbar position, and animations will get reset. Finally, without buffering, re-rendering will result in screen-flashing.

Working on a virtual DOM, on the other hand, is different from working on the real DOM because layouting is not required. React.js creates patches for the browser DOM by calculating the difference between the virtual and modified subtrees. The virtual DOM can use selective subtree rerendering by calling `setState()` on particular components. A component of React.js `setState` method labels it as impure.

To write to the browser DOM, React.js utilizes batched updates. Using batched updates means the DOM is only updated once each event loop. Updating once every event loop makes re-rendering with React.js quick right out of the box.

Developers are freed of the stress of synchronous and asynchronous re-rendering and the associated complexity. The `shouldComponentUpdate()` hook informs React.js whether or not a component should be updated based on its current state, which improves the operations of the virtual DOM even more.

Although React.js primarily uses virtual DOM, some data, such as form field value, is stored in the physical DOM. Because the virtual DOM is unaware of modifications made by those libraries, React.js seldom compares its tree to the actual DOM tree. Manipulating the browser DOM with JavaScript frameworks like jQuery, which work directly on the DOM, will result in unpredictable behavior. This unwanted behavior can be avoided by using lifecycle hooks when interacting with third-party APIs [10].

2.2.2 JSX

JavaScript XML (JSX) is a JavaScript syntax that resembles Extensible Markup Language (XML). In React.js, it is utilized to make User Interface (UI) components. It is extremely similar to HTML but with minor modifications. JSX extends JavaScript to develop React.js components as simple as building HTML pages [11].

Although JSX is not required to build React.js, it makes the language more cumbersome and complex without it. Furthermore, because most developers will already be using tools like Babel to transform their ECMAScript 6 (ES6) code to JavaScript, writing in JSX is not that difficult because most tools include built-in support for it. JSX appears to be almost identical to HTML, for example `<h1>Greeting</h1>` is the same in both languages.

JSX and HTML differ because of the strictness of JSX inherited from XML. For example, in the case of a one-sided tag like an image `<img>` tag, HTML does not require a closing forward slash like `<img src="image/image1.jpg">`, while in JSX the forward-slash cannot be omitted `<img src="image/image1.jpg"/>`. Other

distinctions between JSX and HTML reflect that JSX is written in the context of JavaScript [12].

2.2.3 React.js Components

Components in React.js are encapsulated. They are reusable and depending on the situation, they can vary their behavior. Each component represents how a particular bit of markup should appear at any time [13].

The React.js component provides the three most common functionalities which are initial rendering of the user interface, event management or handling, and updating the user interface anytime the internal state changes. There are two types of components in the React.js library. The function component uses a standard JavaScript function. On the other hand, the ES6 class component utilizes ES6 class. Function components are very minimal code-wise. Its sole requirement is to return a React.js element. Examples of function and class components respectively are shown in listing 1.

```
function Greeting () {
            return <div>Greeting</div>
}

class Greeting extends React.Components {
            render() {
                        return(<div>Greeting<div/>
             );
       }
}
```

Listing 1 React.js function and class component example code

Class components support state management out of the box. However, it is not supported by function components. Meanwhile, React.js provides a hook called `useState()` that allows functional components to keep their state. Class components have a life cycle, with separate callback APIs for each life cycle

event. There is no life cycle for the function component. For the function component, React.js offers a hook, `useEffect()`, that allows access to the different stages of the component [14].

2.2.4 React.js Hooks

React.js 16.8 introduced React.js Hooks, which allows React.js developers to leverage state and other React.js capabilities in the absence of a class. The essential notion of React.js Hooks is the same as that of React.js. They use existing JavaScript features to try to encapsulate state management. As a result, it is not necessary to acquire and comprehend specialized React.js capabilities anymore; instead, it is possible to use Hooks with existing JavaScript skills.

Before React.js Hooks, to deal with state changes, Class components were used with the particular function called life cycle methods, such as `componentDidUpdate()`, and unique state-handling methods, like `this.setState`. React.js classes, particularly `this` context, a JavaScript object, are challenging to read and comprehend [15].

Hooks can handle all the issues, and it is not needed to utilize class components and is no longer required to utilize higher-order components or render props for context because one can access the data needed with a context Hook. Props are properties in React.js used for passing data between components. Hooks also give the power to reuse stateful behavior across components without having to create higher-order components.

Hooks are adaptable. Nevertheless, there are certain drawbacks to using Hooks. They are limited to function components, but they cannot be used in class components. The sequence in which hook definitions are defined is essential and must be maintained; consequently, hooks cannot be used in conditionals, loops, or nested functions [16].

Hooks for various functions are already available in React.js. There are three basic Hooks as well as a few more. The most often used functionality in stateful React.js projects are provided by the basic Hooks, `useState()`, `useEffect()`, and `useContext()` [15].

2.3 Next.js

Next.js incorporates developer experience with production-ready features like server-side rendering, TypeScript capability, smart bundling, route prefetching, etc. [17]. It gives React.js functionalities structure while also introducing a few of its own. It has an opinion about how the application should be organized, routing pages through a simple file folder system [18].

Next.js has several essential features. The first feature is hot code reloads, which refer to the ability of the Next.js server to identify modified files and instantly reload them. Automatic routing is another feature that dismisses the need to set up URLs for routing since the files should go in the pages folder and be mapped to all URLs with the possibility of customization. Next.js also gives the advantage of using component-specific styles because global and component-specific styles are supported by styled-JSX. The other important feature is server-side rendering which comes with pre-rendered on the server, making the client-side loading quicker. Loading JavaScript modules and React.js components dynamically are another advantage of Next.js. Typescript support and being able to export static sites are worth mentioning as distinct features of Next.js [19].

Next.js is used to create landing pages, Search Engine Optimization (SEO) friendly websites, e-commerce stores, and other web applications that require quick load time. Netflix, TikTok, Hulu, Uber, and many other websites and platforms involving many users are some of the examples of Next.js in action [18].

2.3.1 Pages

Pages are React.js Components in Next.js exported from a file in the page directory. A route is assigned to each page depending on its file name. For example, it will be available at `/Contact` if a page is created at `pages/Contact.js` that exports a React.js component as shown in listing 2. Next.js supports dynamic page routes. For instance, if you create a file called `pages/posts/[id].js`, then it will be available at `posts/1`, `posts/2`, etc.

```
function Contact () {
            return <div>Contact</div>
}
export default Contact
```

Listing 2. Contact page for Next.js with function component

Pre-rendering can improve performance and search engine optimization. Next.js renders each page in advance by default. Instead of relying on client-side JavaScript to generate HTML for each page.

Each rendered HTML page is accompanied by the bare minimum of JavaScript code needed for that page. The JavaScript code runs when a browser loads a website, making the page interactive. This is referred to as hydration.

Static Generation and Server-side Rendering are the two types of pre-rendering available in Next.js. The difference between these two is when the HTML for a page is generated. The first type is a static generation in which the HTML is created during the build process and reused for each request. The other type is server-side rendering, where each request generates HTML.

For performance reasons, static generation is preferred over server-side rendering. Content Delivery Network (CDN) can cache statically created pages without any additional configuration to improve performance. In other

circumstances, though, server-side rendering may be the only alternative, for example when fast page load speed is needed [20].

2.3.2 Routing

The file-system-based routing in Next.js is designed on the notion of pages. A file is automatically available as a route when it is added to the directory of a page. The most common patterns can be defined using the files in the directory of the page [21].

Routing in Next.js follows three important rules. The first rule is index routes; that is, the `index.js` file links to the root of the directory in a folder. For instance, `pages/index.js` maps to `'/'` and `pages/blogs/index.js` maps to `/blogs`. Nested routes are the second rule. The router URL automatically detects nested folder structures in the directory pages. For example, `pages/user/dashboard/Contact.js` maps to `/user/dashboard/Contact` and `pages/blogs/article.js` maps to `/blogs/article`. The last rule is dynamic routes. A named parameter can be utilized to match URLs. For instance, `pages/[user]/info.js` maps to `/:user/info` where we can use URLs like `/xyz/info` [22].

2.3.3 API Routes

API routes offer a way to use Next.js to create an API. A file in the `pages/api` folder is mapped to `/api/*` and regarded as an API endpoint rather than a page. They are exclusively server-side bundles; therefore, they will not add to the size of client-side bundles. The API route `pages/api/product.js`, for example, provides a JavaScript Object Notation (JSON) response with a 200 status code as shown in listing 3.

```
function productHandler (request, response) {
            response.status(200).json({ title: 'Slim Shirt' })
}
Export default productHandler
```

Listing 3. JSON response with status code of 200

To make an API route work, it must first export a default function that gets the `req` and `res` inputs. The `req` parameter contains an `http.incomingMessage` instance as well as some pre-built middleware. The res parameter is an `http.ServerResponse` objects with several helper functions. `A request.method` can be used in a request handler to control several HTTP methods in an API route as shown in listing 4.

```
function handleRequest (request, response) {
            if (request.method === 'POST') {
            // handle a POST request
            } else {
            // process any other HTTP method
            }
}
}
export default requestHandler
```

Listing 4. Request handler to handle several HTTP requests

Other implementations using API Routes include masking an external URL service, for example, using `/api/pass` instead of https://entitiy.com/pass-url. To securely access external services, using environment variables on the server is good [23].

2.3.4 SEO in Next.js

The method of improving a website or web page to boost the frequency and portion of unpaid visit from search engines is called SEO. It is the process of assisting in improving a ranking of a website on Google and other search engines. A webpage with effective SEO has a better chance of appearing at the top of a search engine's result page. Even though Google is the well-known search engine, other like Bing, Yahoo, DuckDuckGo, have their own web page crawling algorithms that yield the most acceptable search outcomes. [24].

Understanding how to build websites and pages so that they benefit rather than undermine their search engine ranking has evolved into a significant part of SEO advancement. This primary SEO objective is frequently referred to as core SEO.

Pages must appear higher up and at the top of the list when a search engine returns them in response to a direct query. The spot of a web page in a sorted list of search query results is referred to as search engine placement. [25].

Only a single HTML file is generated when using typical React.js Single Page Applications (SPAs). Afterward individual page is loaded into the browser, simulating client-side page navigation during exploring time. The pages that make up the website do not exist until the client renders them. In other words, any web crawler will not be able to find them because they do not technically exist. In terms of SEO, this is a significant issue. The web crawler is a program used by the search engine to gather data for indexing purposes [26].

The server-side rendered applications are designed to have one file per page. Each page exists before it is rendered on the client side by the browser. This means, any web crawler can index all of them and consider them individually depending on their content to achieve SEO by default [27].

Although using Next.js will increase the SEO results of a website, one must still pay attention to other parts of the application. Some of the factors are metatag, accessibility, and performance [28].

2.4 Tools

2.4.1 Git and GitHub

GitHub is a valuable DevOps tool for managing source code. DevOps is a combination of development and operation. GitHub is version control system that helps developers keep track of changes and collaborate with others.

Compared to other version control systems like Perforce, the popularity of the git arises from its simplicity, efficiency, and low entry barriers. Git is efficient not just because it merges and branches well but also because it distributes, allowing developers to commit changes without connecting to a central server.

GitHub was established in 2008. It was one of the earliest sites to host Git repositories. The open-source community soon used it for code exchange. GitHub became an overnight sensation as a result of this. Therefore, the platform has begun to attract many users [29].

2.4.2 Lighthouse

Lighthouse is an open-source tool that provides performance, accessibility, SEO, progressive web apps assessments, and recommendations for how to enhance these elements of websites [30]. This thesis only focuses on six performance audits described in this section. These performance indicators show how the web application performs in loading speed.

The First content paint occurs as soon as the web browser displays the first content from the DOM. Its timestamp is set when the web browser begins to render any text, picture, non-white canvas, or Scalable Vector Graphics (SVG) [31].

The Speed Index is a metric used to measure how rapidly content appears visually on a page after it loads. Lighthouse generates the Speed Index score with the Speedline Node.js module. It begins by recording a video of the browser page loading and calculating the visual transformation between frames. [32].

The time it requires for the largest content element in the viewport to be portrayed on the screen is measured by Largest Content Paint. It estimates when the main page content is visible to users [33].

Time to Interactive is the total duration it requires for a website page to become fully responsive. When a page displays relevant content, event handlers for the visible page elements are registered, and the page reacts to user interactions [34].

The total period a page is prevented from reacting to user input, such as mouse click, screen taps, or inputs from keyboard, is measured as Total Blocking Time. Total Blocking Time measured by millisecond [35].

Cumulative Layout Shift determines how much material moves about on a page after it has been produced. The Cumulative Layout Shift should be as low as possible. A value of 0 indicates that the layout is perfectly stable [36].

3 Framework popularity and Documentation

3.1 Popularity

This section will elaborate on the popularity of React.js and Next.js based on available data from different sources used by developers in their daily development process. GitHub, Stack overflow, Node Package Manager (NPM) Trends, and Google Trends are selected. Collected and analyzed data from these sources gives some idea of the popularity of the frameworks.

3.1.1 GitHub

There are three indicators of the popularity of a repository on GitHub. These are the number of stars, watches, and forks. Each of these gives valuable data that can be used to evaluate the status of a given repository.

The number of stars in a repository influences the rankings of the GitHub repository. It is easy to find it later if it is a repository or a subject with a star. GitHub users can find similar projects by starring repositories. GitHub might suggest relevant information on the dashboard if they starred repositories.

Adding a star to a repository also conveys an appreciation message to the work of the maintainer of the repository [37].

Forking a repository is typically used to suggest changes to the project or utilize the work of other people as a springboard for new ideas. One can fork a repository to make a clone that can be modified without impacting the upstream repository [38]. The number of the fork could show the popularity of a given repository.

Table 1. The number of stars, forks, and watch counts of React.js and NexJs in GitHub repositories as of April 2022.

Popularity indicators	React.js	Next.js
Stars	185000	83700
Forks	37900	17500
Watch counts	6600	1200

When users wish to be notified of all activity in a repository, they can utilize the Watch feature [39]. The number of people watching indicates the interest in each repository. The popularity of these two frameworks based on the star, the fork, and the watching count is shown in table 1.

3.1.2 Stack Overflow

Stack Overflow is an online tool where developers may ask questions related to programming, answer queries of other people, and find solutions to problems they are having while programming. A developer must include tags to help other users figure out the inquiry when publishing a question. If a given answer solves the problem of a user, the questioner can select that answer as the accepted response to that inquiry. Any User of the platform can vote on queries and solutions. Positive votes are known as upvotes, and negative votes are downvotes, and they indicate how helpful the question and answer were to the users [40].

Stack overflow is a good way to measure the popularity of these frameworks. Stack Overflow has more than 100 million visitors monthly and more than 21 million questions since its establishment [41]. The percentage of questions asked about React.js and Next.js is shown in figure 1.

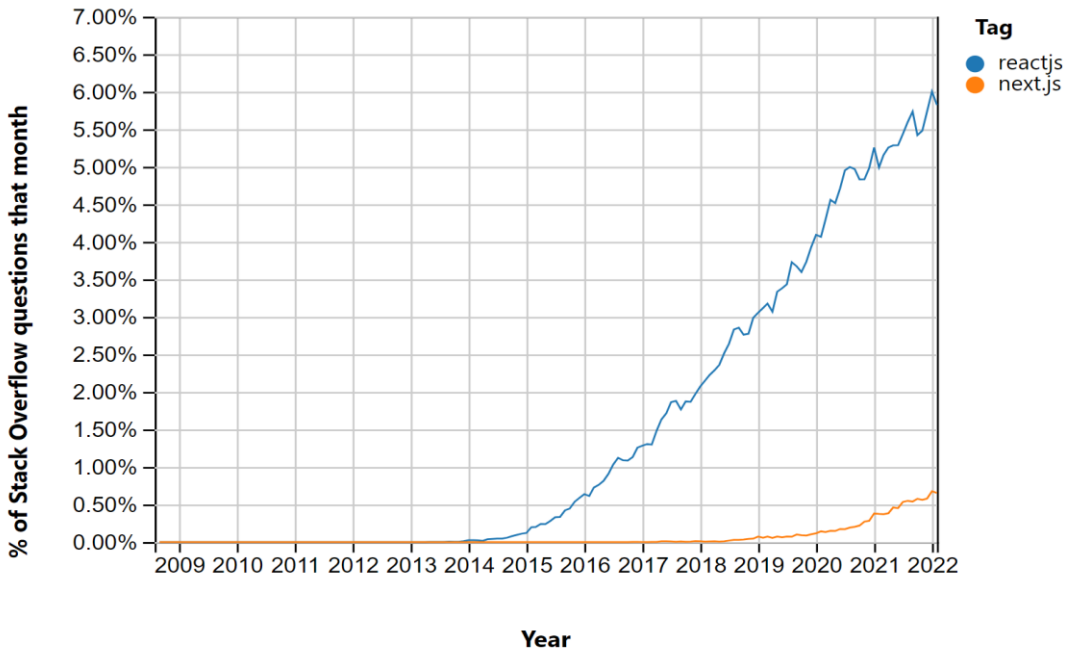

Figure 1. Percentage of questions asked on stack overflow about React.js and Next.js from 2009 to 2022 [42].

The result shown in figure 1 indicates that the percentage of questions asked about React.js has been steadily increasing since 2014. Next.js percentage also has been increasing steadily since mid-2018. These trends show that React.js percentage of questions asked is relatively high, but for better comparison, it is a good idea to consider maturity level.

3.1.3 NPM Trends

NPM is one of the software registries in the world. Numerous corporations use NPM to manage private development, and open-source developers worldwide use it to borrow and exchange packages. NPM Trend gives data on the number

of packages downloaded from the library. The NPM package download data of React.js and Next.js are indicated in figure 2.

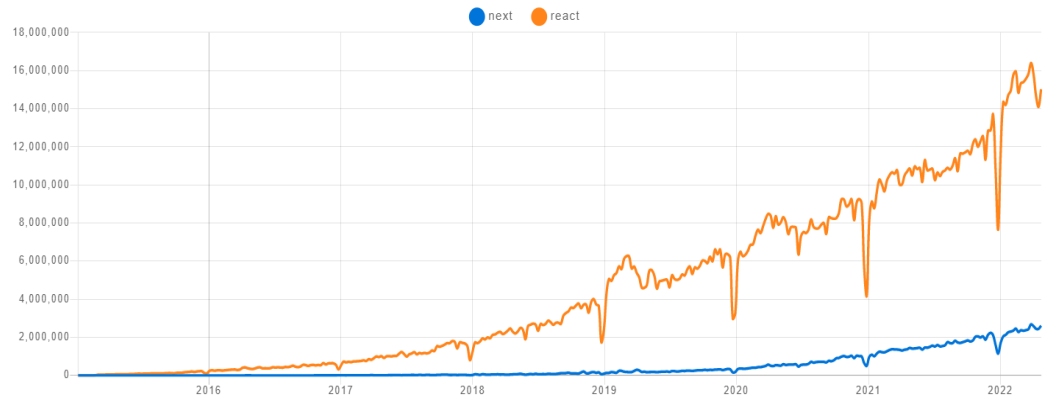

Figure 2. The NPM Trends for Next.js and React.js from 2015 to 2022 [43]

The NPM Trends in figure 2 indicates that the NPM Package download data of React.js is about 5.8 times bigger as of April 24, 2022. Comparing five years after their existence will tell that React.js is higher than Next.js by a million download package data.

3.1.4 Google Trends

On May 11, 2006, Google Trends was launched for the first time. On August 5, 2008, Google announced Google Insights for Search, an enhanced and more extensive tool that supplied users with search trends. Google incorporated Google Insights for Search with Google Trends on September 27, 2012. It mainly gives data on Google Search searches [44].

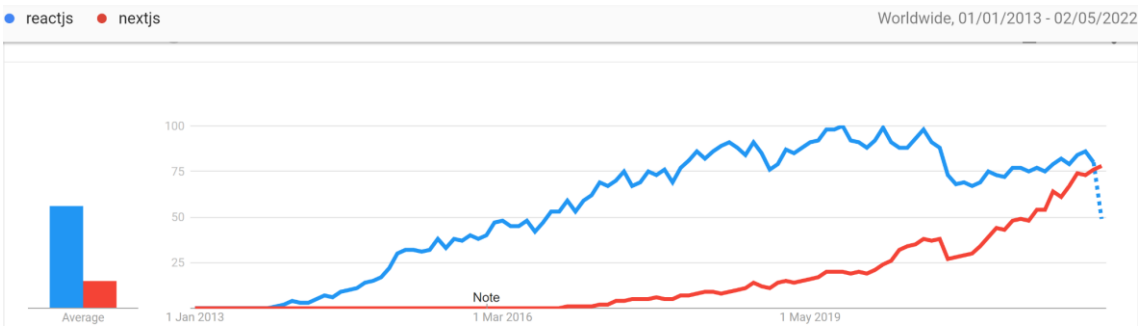

Figure 3 Worldwide Google Trends of React.js and Next.js from the beginning of 2013 until May 2022 [45].

The worldwide Google Trends of React.js and Next.js since 2103 are shown in figure 3. The popularity of Next.js is growing quite fast in Google search and is catching up with React.js. Contrary to the other sources of data, this data clearly shows the gained popularity by Next.js quite as much as React.js as of today.

3.2 Availability of Documentation

For any project development, good documentation may assist users in learning how to use tools, which libraries to utilize, and many other things. This section will discuss the availability of documentation for React.js and Next.js.

The official React.js website includes excellent documentation for the developers. There is a tutorial section for newcomers with JavaScript backgrounds where a hands-on way of learning is explained. The current tutorial gives a deep understanding of how to build any React.js application. All the fundamentals are covered in detail. The tutorial section is a good starting point for anyone who wants to learn by doing React.js. Developers have a chance to go through the documentation to strengthen their knowledge or solve the problem they face during the development.

Next.js official website also comes with a tutorial and documentation section. The tutorial section is beneficial to grasp the main points of developing with Next.js. It covers all the basic concepts of the framework. Developers can also study the documentation to enhance their knowledge in Next.js and tackle issues they face in the development process.

In terms of the official documentation, React.js and Next.js are equally good. In addition, if developers want to learn more, they should visit the official documentation pages for both React.js and Next.js [46].

At the time of writing this thesis, it has been noticed that finding formal books, journals, articles, and other formal resources was not easy for Next.js than React.js. This could be due to the age and maturity differences.

When it comes to informal resources like blog posts, YouTube tutorials, etc., React.js has an enormous amount compared to Next.js. Although React.js has more resources compared to Next.js due to its maturation, Next.js is also gaining more popularity based on the data from Google Trends.

4 Performance

To evaluate the performance difference and speed load test between the frameworks, a simple client-side e-commerce application is built in each framework. Both applications have the same user interface, business logic, and design.

4.1 Application design and structure

The e-commerce application is a performance testing subject built by React.js and Next.js separately. In both cases, random data is used for the products displayed on the front page. The application fetches the product data from a local MongoDB database. In addition to the product data, the local MongoDB database stores user data.

Though Next.js is a React.js framework, setting up the application is different in the two frameworks. In this project, the following methods were utilized. For setting up the React.js project `Npx create-react-app frontend` and `Npx create-next-app` for Next.js application. In the case of Next.js application the name of the application prompt after putting the above command.

Unlike the Next.js, the React.js application required setting up the backend separately. Due to this reason, in the React.js project, the Node.js server was

created at the root level in a different folder called the backend. The `npm init` command is used to set up a backend server. Besides initializing the Node.js within the React.js application, the Express is also installed.

The folder and file structure of the React.js and Next.js e-commerce applications developed for this thesis project are indicated in figure 4. On the left side of figure 4 is the structure of the React.js application, whereas on the right side, is the folder structure of the Next.js application. Achieving a full-stack web application in React.js, requires multiple setups. On the other hand, Next.js comes with full capability for creating a full-stack application. The React.js application is divided into Frontend and backend parts.

Figure 4. The folder structure of React.js on the left and Next.js applications on the right

The two identical React.js and Next.js applications from a features perspective are shown in figure 5. A user can browse through the list of products and products and add them to the cart without logging in. A user needs to login to navigate to the checkout page. The checkout page checks if the user is logged in. After login, the user will be prompted to enter a shipping address, select a payment method, and confirm the order.

All the order data is also stored in the local MongoDB database like the user and product data. The application is limited to the client-side. The admin part of the application is not included in this project.

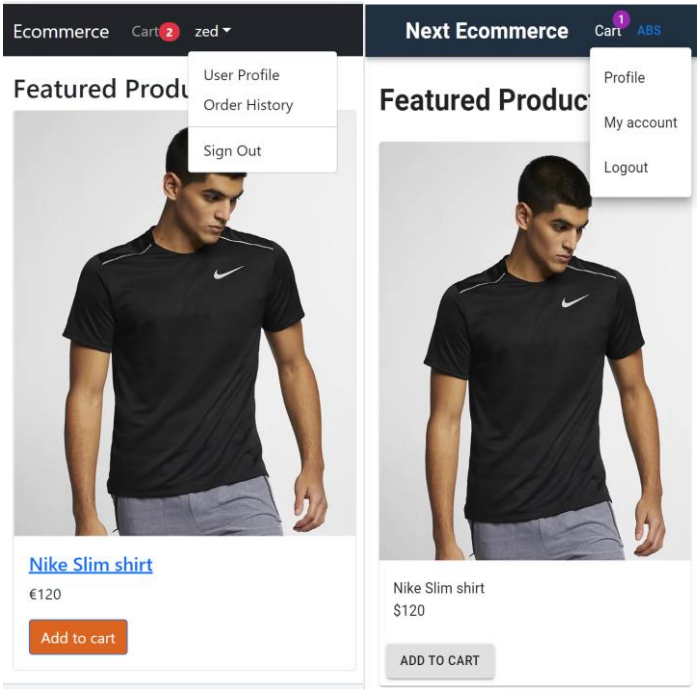

Figure 5. Home page of React.js on the left and Next.js application on the right

The styling and making of a user-friendly UI for the React.js application is done by using react-bootstrap. React-Bootstrap is a toolkit that yields native Bootstrap elements in pure React.js. [47]. On the contrary, material UI is used for styling and building the user interface of the Next.js application. Material-UI is tool with React.js components that obey Google's Material Design approaches [48].

4.2 Number of Lines of Code

The number of lines of code is not a direct performance indicator, but it will give some indication in choosing a framework. The number of lines of code smaller

could mean that a certain framework is using ready-made libraries to make the coding easier.

The number of lines of code and the number of files of the applications developed for this project in React.js and Next.js are shown in table 2. The results in table 2 clearly show that the amount of code required to build a Next.js application is less than one-third of the amount of code line required to build the same React.js application.

Table 2. Number of lines of code for React.js and Next.js for the same e-commerce web application

	React.js	Next.js
Number of lines of code	36625	10983
Files	44	38

It might be interesting to see immense differences in application build sizes, 2.87 MB and 51.0 MB for React.js and Next.js applications. The application build size of Next.js is almost 18 times bigger than the React.js application production build size.

This difference in size could be because Next.js is built on top of React.js with the additional power of server-side rendering. This comparison might not give a complete picture of what could be the application production build size in real-world applications. However, it could indicate how to choose the proper framework based on the complexity of the application project.

4.3 Loading Speed Test

The main performance indicator chosen in this thesis project is the loading speed test. Web performance is evaluated by the page load time from the client point of view. This is the elapsed time from when the user requests a new page until the browser entirely renders the page. Fast Web pages are rendered in

stages. The content is added in stages as the browser loads it. Progressively rendering web pages gives feedback that the page is loading, and Users can get the requested information momentarily [49].

The loading speed test was conducted locally on a Windows 10 machine and Google Chrome browser. The applications were made available through localhost. The loading speed tests were conducted using Lighthouse from a tool that is available in the Chrome browser DevTools.

The audit result for the React.js application is illustrated in table 3. Five consecutive tests were done to get more reliable results. The average results are recorded in the last column. There is an evident variation in Total Bocking Time results contrary to the other performance metrics. The variation in the results could be due to internet speed. Further analysis should be done to arrive at a better conclusion.

Table 3. React.js e-commerce application Lighthouse five test runs, and average performance metrics.

React.js	Test 1	Test 2	Test 3	Test 4	Test 5	Average
First Contentful Paint	0.4 s	0.4 s	0.4 s	0.4 s	0.4 s	0.4 s
Speed Index	0.6 s	0.8 s	0.7 s	0.6 s	0.7 s	0.7 s
Largest contentful paint	1. 7 s	1. 6 s	1. 6 s	1. 6 s	1. 5 s	1. 6 s
Time to Interactive	1.1 s	1.1 s	1.1 s	1.0 s	1.1 s	1.08 s
Total Blocking Time	160 ms	140 ms	130 ms	130 ms	160 ms	144 ms
Cumulative Layout Shift	0.335	0.403	0.403	0.403	0.403	0.389

Five different test runs for the Next.js application for six different performance metrics can be seen in table 4. The last column also calculates the average results to make a conclusive enough comparison. The First Contentful Paint, Speed Index, and Cumulative Layout Shift stayed the same in all test cases, unlike the React.js application. In this case, some variation can be observed in all other performance metrics.

Table 4. Next.js e-commerce application Lighthouse five test runs, and average performance metrics.

Next.js	Test 1	Test 2	Test 3	Test 4	Test 5	Average
First Contentful Paint	0.4 s	0.4 s	0.4 s	0.4 s	0.4 s	0.4 s
Speed Index	1.1 s	1.1 s	1.1 s	1.1 s	1.1 s	1.1 s
Largest contentful paint	4. 4 s	4.5 s	4.6 s	4. 6 s	4.6 s	4.54 s
Time to Interactive	4.5 s	4.6 s	4.6 s	4.7 s	4.7 s	4.62 s
Total Blocking Time	640 ms	660 ms	680 ms	690 ms	670 ms	668 ms
Cumulative Layout Shift	0	0	0	0	0	0

The combined Lighthouse performance metrics for React.js and Next.js applications are shown in table 5. The First Contentful Paint seems to be the same in both cases. In all other metrics other than the Cumulative Layout Shift, the React.js application outperforms the Next.js application. The Cumulative Layout Shift is stable for the Next.js but not in React.js.

Table 5. The combined result of Lighthouse performance metrics for the React.js and Next.js

	React.js	Next.js
First Contentful Paint	400 ms	400 ms
Speed Index	700 ms	1100 ms
Largest contentful paint	1600 ms	4.54 s
Time to Interactive	1080 ms	4620 ms
Total Blocking Time	144 ms	668 ms
Cumulative Layout Shift	0.389	0

The graphic comparison with Lighthouse performance is indicated in figure 6. The difference in Speed Index is not significant. On the other hand, the

difference between React.js and Next.js in Large Contentful Paint and time to interactive is considerably huge.

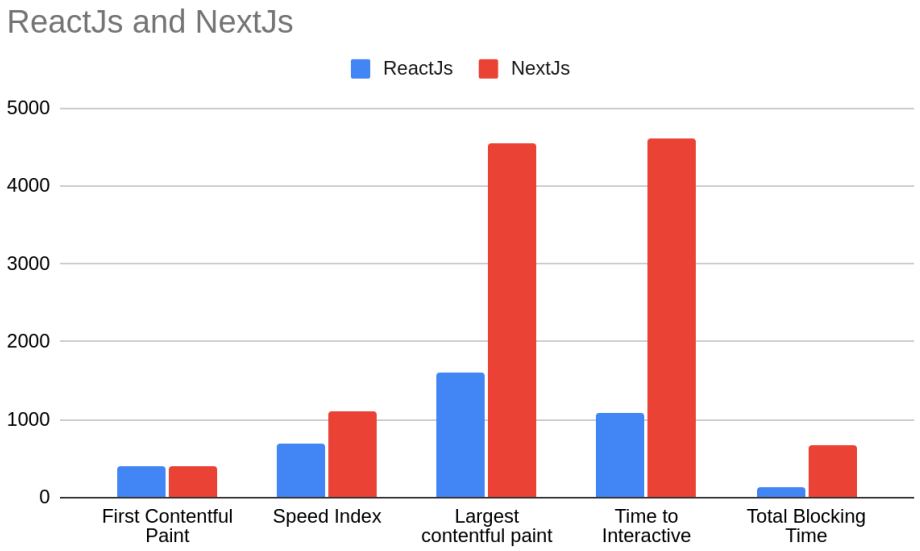

Figure 6 Lighthouse performance metrics graphic comparison between the React.js and Next.js application

The performance metrics results are limited to small-scale applications built along with this project. For more reliable results building large-scale applications with multiple features might be required. Including other Lighthouse metrics like accessibility and best practices might also be essential.

5 Conclusion

This paper compares React.js and Next.js, thereby pointing out the advantages and disadvantages of using either of these frameworks. It compares the page rendering performances and the popularity and availability of documentation of the two frameworks. Two similar e-commerce web applications were built to evaluate the performance of each framework.

Data from GitHub, Stack Overflow, NPM Trends, and Google Trends was utilized to compare popularity differences. The result of the popularity

comparison indicated that React.js is more prevalent than Next.js. Although React.js is more popular than Next.js, the popularity trend of Next.js also increased. The big difference between their popularity might be due to the age difference. For example, the Google Trends showed how Next.js is also gaining popularity despite its age and maturity.

The method by which the performance comparison was conducted was by using Lighthouse performance metrics from Google Chrome. These metrics are First Contentful Paint, Speed Index, Largest contentful paint, Time to Interactive, Total Blocking Time, and Cumulative Layout Shift. Except Cumulative Layout Shift, all metrics are speed measurements. The results from the comparison showed that React.js performed better in most aspects. Next.js performed better only in Cumulative Layout Shift.

In conclusion, there are some limitations to the results of this study, namely the availability of literature and time constraints to develop multiple applications for a better outcome. Elevation of these challenges in the future may impact the result comparison. For example, developing large-scale applications might be necessary to reach a definitive conclusion. Loading test, simulating large numbers of concurrent users with multiple requests per second, is another thing to consider. Even with the mentioned shortcomings, this paper could be a starting point for deciding between the two frameworks.

References

1 Gizas, Andreas, Christodoulou, Sotiris & Papatheodorou, Theodore. 2012. Comparative evaluation of javascript frameworks. Proceedings of the 21st international conference companion on World Wide Web - WWW ’12 Companion. Lyon, France: ACM Press.

2 Haverbeke, Marijn. 2018. Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming. No Starch Press.

3 Andreasen, E., Gong, L., Møller, A., Pradel, M., Selakovic, M., Sen, K. & Staicu, C.-A. 2018. A Survey of Dynamic Analysis and Test Generation for JavaScript. ACM Computing Surveys, 50, 5, pp. 1–36. URL: https://doi.org/10.1145/3106739. Accessed: 27 April 2022.

4 Flanagan, David. 2002. JavaScript: the definitive guide. Sebastopol, CA: O’Reilly.

5 Ranjan, Alok, Sinha, Abhilasha & Battewad, Ranjit. 2020. JavaScript for Modern Web Development: Building a Web Application Using HTML, CSS, and JavaScript. BPB Publications.

6 Rawat, P. & Mahajan, A.N. 2020. React.js: A Modern Web Development Framework. 5, 11, pp. 5.

7 Aggarwal, Sanchit. 2018. Modern Web-Development using React.js. 5(1), 5.

8 Kumar, Anurag & Singh, Ravi Kumar. n.d. Comparative analysis of angularjs and React.js. International Journal of Latest Trends in Engineering and Technology. Volume 7(Issue 4).

9 Fedosejev, Artemij. 2015. React.js Essentials. Packt Publishing Ltd.).

10 Theel, Oliver. n.d. Rich Internet Applications w/HTML and Javascript. , 36.

11 Sengupta, Doel, Singhal, Manu & Corvalan, Danillo. 2016. Getting Started with React. Packt Publishing Ltd.

12 Masiello, Eric & Friedmann, Jacob. 2017. Mastering React Native. Packt Publishing Ltd.

13 Pitt, Christopher. 2016. React Components. Packt Publishing Ltd.

14 React.js – Component. n.d. URL: https://www.tutorialspoint.com/React.js/React.js_components.htm. Accessed: 22 March 2022.

15 Banks, A. & Porcello, E. 2020. Learning React: Modern Patterns for Developing React Apps. O’Reilly Media, Inc

16 Bugl, D. 2019. Learn React Hooks: Build and refactor modern React.js applications using Hooks. Packt Publishing Ltd.

17 Next.js by Vercel - The React Framework. n.d. URL: https://Next.js.org. Accessed: 23 March 2022.

18 Next.js vs React: What are the differences? 2021. URL: https://www.imaginarycloud.com/blog/next-js-vs-react/. Accessed: 23 March 2022.

19 Next.js – Overview. n.d. URL: https://www.tutorialspoint.com/Next.js/Next.js_overview.htm. Accessed: 23 March 2022.

20 Next.js – Pages. n.d. URL: https://www.tutorialspoint.com/Next.js/Next.js_pages.htm. Accessed: 24 March 2022.

21 Routing: Introduction | Next.js s.a. URL: https://Next.js.org/docs/routing/introduction. Accessed: 25 March 2022.

22 Next.js - Routing n.d. URL: https://www.tutorialspoint.com/Next.js/Next.js_routing.htm. Accessed: 25 March 2022.

23 Routing: Introduction | Next.js. n.d. URL: https://Next.js.org/docs/routing/introduction. Accessed: 25 March 2022.

24 Mahmood, N. 2021. SEARCH ENGINE OPTIMIZATION.

25 Davis, H. 2006. Search Engine Optimization. O'Reilly Media, Inc.

26 Dhenakaran, S.S. & Sambanthan, K.T. n.d. WEB CRAWLER - AN OVERVIEW. pp. 3.

27 Next.js SEO made easy for headless CMS. 2021. URL: https://www.datocms.com/blog/dealing-with-Next.js-seo. Accessed: 26 March 2022.

28 How Next.js can help improve SEO 2020. URL: https://blog.logrocket.com/how-next-js-can-help-improve-seo/. Accessed: 26 March 2022.

29 GitHub vs Gitlab vs Bitbucket | Disbug Blog. n.d. URL: https://disbug.io/en/blog/GitHub-vs-gitlab-vs-bitbucket. Accessed: 28 March 2022.

30 Heričko, T., Šumak, B. & Brdnik, S. 2021. Towards Representative Web Performance Measurements with Google Lighthouse. University of Maribor Press.

31 First contentful paint - MDN Web Docs Glossary: Definitions of Web-related terms | MDN. n.d. URL: https://developer.mozilla.org/en-US/docs/Glossary/First_contentful_paint. Accessed: 22 April 2022.

32 Speed Index. n.d. URL: https://web.dev/speed-index/. Accessed: 22 April 2022.

33 Largest Contentful Paint. n.d. URL: https://web.dev/lighthouse-largest-contentful-paint/. Accessed: 22 April 2022.

34 Time to Interactive. n.d. URL: https://web.dev/interactive/. Accessed: 22 April 2022.

35 Total Blocking Time. n.d. URL: https://web.dev/lighthouse-total-blocking-time/. Accessed: 22 April 2022.

36 Cumulative Layout Shift | DebugBear. n.d. URL: https://www.debugbear.com/docs/metrics/cumulative-layout-shift. Accessed: 22 April 2022.

37 Saving repositories with stars. n.d. URL: https://docs.GitHub.com/en/get-started/exploring-projects-on-GitHub/saving-repositories-with-stars. Accessed: 28 March 2022.

38 Fork a repo. n.d. URL: https://docs.GitHub.com/en/get-started/quickstart/fork-a-repo. Accessed: 28 March 2022.

39 Analyzing popular repositories on GitHub. 2021. URL: https://www.analyticsvidhya.com/blog/2021/07/analyzing-popular-repositories-on-GitHub/. Accessed: 28 March 2022.

40 Ranjitha, R.K. & Singh, S. 2015. Is Stack Overflow Overflowing With Questions and Tags. Proceedings of the Third International Symposium on Women in Computing and Informatics - WCI

41 Empowering the world to develop technology through collective knowledge. n.d. URL: https://stackoverflow.co/. Accessed: 29 March 2022.

42 Stack Overflow Trends. n.d. URL: https://insights.stackoverflow.com/trends?tags=React.js%2Cnext.js. Accessed: 29 March 2022.

43 next vs react | npm trends. n.d. URL: https://www.npmtrends.com/react-vs-next. Accessed: 4 May 2022.

44 Jun, S.-P., Yoo, H.S. & Choi, S. 2017. Ten years of research change using Google Trends: From the perspective of big data utilizations and applications. Technological Forecasting and Social Change, 130.

45 Google Trends. n.d. URL: https://trends.google.com/trends/explore?date=2013-01-01%202022-05-02&q=React.js,Next.js. Accessed: 5 May 2022.

46 React vs Next js 2022: Which JS Framework your Project Requires? n.d. URL: https://www.thirdrocktechkno.com/blog/comparison-between-next-js-vs-react//. Accessed: 31 March 2022.

47 React-Bootstrap. n.d. URL: https://react-bootstrap.GitHub.io/. Accessed: 4 May 2022.

48 Meet Material-UI —your new favorite user interface library. 2018. URL: https://www.freecodecamp.org/news/meet-your-material-ui-your-new-favorite-user-interface-library-6349a1c88a8c/. Accessed: 4 May 2022.

49 Manhas, D. 2013. A Study of Factors Affecting Websites Page Loading Speed for Efficient Web Performance. International Journal of Computer Sciences and Engineering.