



Linnæus University

Sweden

Bachelor Degree Project

Choosing a Rendering Framework

*A Comparative Evaluation of Modern JavaScript
Rendering Frameworks*



Authors: Viktor Sjölund
Denni Wernersson
Supervisor: Mattias Davidsson
Examiner: Arslan Musaddiq
Semester: VT 2023
Subject: Computer Science

Abstract

This bachelor thesis explores the rapidly evolving landscape of JavaScript rendering frameworks, comparing and evaluating Next.js, SvelteKit, and Astro. The motivation behind this investigation is to help determine the most suitable rendering framework for contemporary web developers. To address this problem, a two-pronged methodology was employed: firstly, a survey was conducted to gather insights into web developers' usage of rendering frameworks and their opinions on different aspects of the frameworks; secondly, a controlled experiment was carried out by building web applications using the three frameworks to gather data on their performance, capabilities, and developer experience. The results of the study indicate that all three evaluated frameworks are strong options, but each offers unique advantages and drawbacks that must be considered when motivating a decision between them.

Keywords: *web development, server-side rendering frameworks, full-stack frameworks, app frameworks, meta frameworks*

Contents

1	Introduction	1
1.1	Background	1
1.2	Related work	2
1.3	Problem formulation	3
1.4	Motivation	3
1.5	Results	3
1.6	Scope/Limitation	4
1.7	Target group	4
1.8	Outline	4
2	Method	5
2.1	Research Project and Design Science	5
2.2	Research methods	6
2.3	Reliability and Validity	6
2.4	Ethical considerations	8
3	Theoretical Background	9
3.1	Related Work and Knowledge Gap	9
3.2	Concepts within Web Development	9
3.3	Rendering Frameworks	11
4	Research project – Implementation	13
5	Results	15
5.1	Survey	15
5.2	Controlled Experiment	18
6	Analysis and Evaluation	27
6.1	Survey	27
6.2	Controlled Experiment	29
7	Discussion	36
7.1	Survey	36
7.2	Controlled Experiment	37
7.3	Choosing a Rendering Framework	37
8	Conclusion and Future Work	39
	References	40
	List of Terms	44
9	Appendices	A
A	Google Forms Survey	A
B	Survey Missive Letters	B
C	Application Specifications	C
D	Checklist and In-depth Questions	D
E	The Web Applications	E

1 Introduction

This paper is a 15 HEC Bachelor Thesis in Computer Science and is written as part of a Bachelor of Computer Science degree in web development at Linnaeus University. The thesis is situated within the broader field of software engineering with a focus on the field of web development, where the development and maintenance of web applications is central. A near-fundamental aspect of web development is the usage of different frameworks that make the development process both simpler and faster, allowing for a more efficient creation of high quality web applications.

One of the subsets of web development frameworks is the rendering framework, which is focused on providing both front-end and back-end functionality to the developer. The goal of this thesis is to provide a comparison of a portion of the most widely used and popular rendering frameworks in the domain of modern web development. By investigation of the frameworks' differences in performance, capabilities and features, as well as by gathering web developers opinions, we will evaluate and compare the quality of each framework.

Through a combination of practical application development and analysis of developer opinions, our goal is to deliver valuable information for web developers working with rendering frameworks, ultimately assisting them in making an informed decision when choosing the most suitable framework for their needs.

1.1 Background

Web development is a mature field within software development, and it contains an extensive range of different technologies and tools to aid different aspects of development. Web developers can of course choose to build web applications without relying on any web development framework, but nonetheless web development frameworks are widely used by web developers today. According to the State of JavaScript survey of 2022 [1], 82% of web developers have in that past year used a front-end framework like React [2], while 48% have used a rendering framework like Next.js [3].

There are a lot of different types of framework that are available to web developers, each intended to solve a different problem or enhance some aspect of the development process. For example, front-end frameworks help structure the user interface (UI), object-relational mapping (ORM) frameworks simplify database interactions, and testing frameworks assist in verifying code functionality. The attention of this thesis is concentrated on the category of rendering frameworks, also known as application, meta, or full-stack frameworks. A rendering framework is intended to handle both the front-end and back-end functionality of a web application, meaning that it is responsible for the rendering as well as the serving of that application. The potency of a rendering framework lies in its ability to offer a comprehensive web development solution, providing the developer with almost all the necessary tools to construct a robust and scalable web application.

Not only are there a lot of distinct framework categories, but each category contains a large collection of specific frameworks. This is true for rendering frameworks as well. The 2022 State of JavaScript survey collected developer statistics on 18 distinct rendering frameworks, where eight of them were considered to be of moderate to high popularity [1].

Choosing which rendering framework is best fit for a specific use case can be a very challenging task that requires thorough consideration. The ideal rendering framework for a given situation will differ depending on the requirements of the project, as well as on the developer that will be utilizing the framework. Each specific framework also comes

with its own strengths and weaknesses, inherently making them better or worse in certain situations. With the wide variety of choices available the decision-making process can become a daunting and uncomfortable experience, leaving web developers at a crossroads while looking for the best choice for their project.

1.2 Related work

There is a rather considerable amount of research making comparisons between JavaScript front-end development frameworks and rendering techniques. For example, a 2021 bachelor thesis [4] compared the front-end frameworks React [2], Angular [5], Vue.js [6] and Svelte [7]. Furthermore, another 2021 thesis compares front-end frameworks with rendering frameworks, essentially contrasting client-side rendering frameworks with server-side rendering frameworks [8]. However, there is a significant gap in the literature when it comes to formal research exclusively comparing rendering frameworks. The gap in this research could potentially be attributed to the relatively recent surge in popularity of rendering frameworks, as indicated by the State of JavaScript 2022 Survey [1].

One comparison study exists making comparisons between multiple rendering frameworks [9], specifically between the frameworks Gatsby [10] and Next.js [3]. However, the study is not strictly about rendering frameworks, and also compares them to the front-end framework React [2] as part of a larger evaluation of React-based frameworks for developing Single Page Applications, with a strict focus on performance alone.

When searching for comparisons of JavaScript rendering frameworks, developers will have an easier time finding what they're looking for in informal settings and grey literature. The list of resources that can help a developer choose a rendering framework is immense, and ranges from material such as vlog-style discussions [11] to blog posts making in-depth feature comparisons [12]. These kinds of resources are however oftentimes heavily opinionated and biased, which make them less reliable as a sole source when making a decision.

1.3 Problem formulation

While there is a considerable amount of unscientific resources available that can help web developers choose a rendering framework, there is a significant lack of scientific research in this area. The comparative analysis and evaluation of rendering frameworks is an area that remains mostly unexplored. This leaves web developers in a situation where they have little to no scientific basis to motivate a decision between different rendering frameworks. The knowledge gap can be rephrased as a set of research questions.

RQ1	What are the differences in feature-set between the different rendering frameworks?
RQ2	What factors/metrics are important to web developers when choosing/comparing rendering frameworks?
RQ3	Which of the selected rendering frameworks is the most suitable for the most common use cases, and how do they compare in terms of the determined metrics?

Table 1.1: *Research Questions*

RQ1 This research question aims to explore the various feature-sets provided by different rendering frameworks. The results of this comparison will form the basis for the metrics that the rendering frameworks will be compared against.

RQ2 This research question seeks to identify the key factors and metrics that web developers consider when choosing or comparing rendering frameworks. The results of this question will provide insight into which of the determined metrics from **RQ1** that web developers actually think are important.

RQ3 This research question aims to determine the most suitable rendering framework for common use cases by evaluating them against the key factors and metrics identified in **RQ1** and **RQ2**. The results of this question will be analysed to provide developers with a comprehensive evaluation to aid in choosing a rendering framework for their specific use case.

1.4 Motivation

This thesis will advance the domain of web development within the field of software engineering. The results will primarily advance the current scientific knowledge of qualities found in JavaScript rendering frameworks. Furthermore, from an economic point of view it will aid web developers in making well-informed decisions concerning rendering frameworks which will help to avoid the potential economic costs associated with choosing a suboptimal or unsuitable framework.

1.5 Results

We propose a new method for determining the optimal rendering framework for a given use case. The method outlines a set of criteria and significant factors to consider when making such a decision, that will serve to guide web developers in selecting the most suitable framework for their specific project requirements. The method is evaluated indirectly

by proxy, through the collection of web developers opinions on what factors they find important in a web development framework, which will be used to develop the method.

1.6 Scope/Limitation

The focus of this thesis is solely on rendering frameworks, and we make no attempt to determine whether rendering frameworks as a whole are the ideal choice over an alternative stack solution. Instead, we limit focus of this work on identifying the most suitable rendering framework for a specific use case, once it has already been decided that a rendering framework will be used.

Due to the immense number of rendering frameworks available for web developers to choose from, we cannot cover them all. The comparative analysis will be limited exclusively to three popular rendering frameworks: Next.js version 13.1.6 [3], SvelteKit version 1.5.0 [13] & Astro version 2.2.3 [14].

We also limit the scope of the project to comparing the frameworks as pure server-side rendering frameworks, only utilizing server-side rendering despite their capabilities for client-side rendering and static site generation.

Our scope for web developer opinion responses is limited as well. A perfectly representative picture cannot be captured within the scope of this thesis, as we are unable to invest in obtaining a significant amount of respondents. We rely on the goodwill of volunteers to provide their opinions, which will constrain the quantity of responses that we are able to collect. As a consequence, our sample group is unlikely to perfectly represent the broader group of web developers.

1.7 Target group

The target group for this thesis consists of web developers, specifically web developers who are considering the use of rendering frameworks for their projects.

1.8 Outline

The structure of this report is as follows. Chapter 2 delves into the project's methodology, research methods, their validity and reliability, and ethical considerations. In Chapter 3, we further explore the knowledge gap, related work, and its application in this project, as well as web development concepts central to this project. Chapter 4 outlines the data collection process for the research project. The findings from the data collection are presented in Chapter 5 and analyzed in Chapter 6, where we assess the chosen rendering frameworks. Chapter 7 further discusses the analysis and methodology, along with potential threats to the validity and reliability of the work. Finally, in Chapter 8, we conclude the project and discuss possible future work.

2 Method

Multi-Strategy Design

This research project utilizes a multi-strategy design, meaning that multiple research methods are employed to answer the projects research questions [15]. The multi-strategy design specifically employs the use of two research methods: (1) a survey using questionnaires to gather opinions and experiences from web developers regarding the use of rendering frameworks; and (2) a controlled experiment where web applications are developed using each of the selected frameworks to gather data to be used in the comparison of the frameworks.

2.1 Research Project and Design Science

Design science combines creativity and scientific methodology to solve complex problems. In this context, we formulate a design science to analyze and compare rendering frameworks, using data collection and analysis from surveys and controlled experiments. This approach enables a comprehensive and systematic evaluation, assisting in the selection a rendering framework most suitable for the situation.

Problem Identification

This project was initiated by observing the State of JavaScript survey of 2022 [1]. The survey presents data on web developers' awareness, usage, interest and retention of 8 rendering frameworks, in contrast to the 2020 survey which only presented such data on 3 rendering frameworks. The increased number of rendering frameworks covered in the 2022 survey highlights the rapid growth and diversification of the ecosystem.

A problem was identified in the fact that no formal research exists comparing and evaluating these rendering frameworks. The first step in our design science is to identify which rendering programs to evaluate and compare against each other. Using the results of the State of JavaScript survey of 2022, Next.js, SvelteKit and Astro were chosen to be evaluated as they were the top three rendering frameworks in both developer interest and retention [1].

Define Objectives of Solution

In order to establish clear evaluation criteria and determine the parameters by which the selected rendering frameworks are to be measured and compared, we must investigate their capabilities and differences. By analyzing the features and unique strengths of each framework, we can identify their distinctions and develop appropriate measuring parameters. This comprehensive approach will help in setting well-defined evaluation criteria for an effective comparison.

The finalized measuring parameters are in part based on a set of guidelines outlined in a related work of this thesis [4], where front-end frameworks are compared. The guidelines have been modified to better suit a rendering framework comparison, using the insights gained from our investigation into the rendering frameworks.

This step of the design science doubles as an opportunity for us to increase our overall comprehension of the selected rendering frameworks, which helps ensure a fair and accurate evaluation.

Design and Development

Data is collected for the evaluation of each framework using two primary methods. First, we conduct a survey using questionnaires to gather the opinions of web developers on aspects of rendering frameworks, and how they use rendering frameworks today.

Secondly, we conduct a controlled experiment where we develop a web application using each framework to gather relevant data for evaluation and comparison. The developed web applications and the rendering frameworks themselves are then measured against the criteria established in the previous section of the design science.

Analysis and Evaluation

Finally, we analyse the results of the survey and the data collected from the web applications to evaluate each of the selected rendering frameworks, and compare them against each other.

2.2 Research methods

Survey using questionnaires

Opinions and experiences of web developers concerning their use of rendering frameworks are gathered by way of survey. Our belief is that a survey-based methodology is an effective approach, as it enables us to reach a larger audience and obtain diverse perspectives. We had initially considered conducting interviews to achieve the same goal, but due to time limitations and difficulty in reaching willing and interested interviewees, we decided against this approach. The survey is largely based on the section focused on rendering frameworks of the State of JavaScript survey of 2022 [1].

Controlled Experiment

Furthermore, we conduct a controlled experiment by developing a set of web applications, one for each of the selected rendering frameworks. Our belief is that by developing web applications using the frameworks that will be evaluated we can gather important insights to the frameworks that can only be gathered from hands-on experience. By having a clearly defined set of specifications for the goal application, and established parameters to measure them by, we can compare the aspects of each framework. This controlled experiment provides us with valuable data to complement the insights gathered from the survey, leading to a more comprehensive understanding of the different rendering frameworks.

2.3 Reliability and Validity

There are many aspects to consider concerning the validity and reliability of our chosen methodology.

Validity

Large parts of the survey are based on a pre-existing survey, specifically the sections related to rendering frameworks in the State of JavaScript survey of 2022. This will positively affect several validity aspects of the survey. For example: content validity, which is the likelihood that the questions cover all relevant aspects of the topic being studied [16]; and construct validity, which refers to the effectiveness of a test in gauging

the specific aspect it was made to measure [17], are likely improved by basing the survey on an already established survey [18].

Another positive aspect of incorporating questions from an established survey is that it allows for the possibility of making comparisons between our results and those of the original survey. To facilitate this comparison, it is however important to present the questions and response alternatives identically to the original survey.

A satisficing bias is a potential issue in the survey of this thesis. This type of bias occurs when respondents provide quick and superficial answers without fully considering the question, in order to complete the survey as quickly as possible [19]. To combat satisficing bias, we limit the length of the survey and the complexity of its questions to reduce the strain on participants.

Desirability bias also presents a challenge in this thesis survey methodology. Desirability bias occurs when respondents are inclined to answer questions in a way that is perceived as socially desirable or "correct" [16]. To mitigate and minimize desirability bias in the thesis survey, we make it clear in the survey missive letter that participants are welcome regardless of their level of education, experience, and knowledge.

There is also a nonresponse bias to consider in the survey, which is when a subset of potential respondents decide not to participate in the survey at all [16]. This could affect the validity of the survey results by leading to a non-representative sample of the target group. To address this issue, we clearly communicate the survey's purpose and emphasize the significance of respondents' participation in the missive letters.

Reliability

The reliability of the methodology is influenced by our roles as web developers responsible for building the web applications in the controlled experiment, by having distinct backgrounds and qualifications. With each developer bringing their own experiences and knowledge to the table, these individual factors may affect the results found in the controlled experiment. Consequently, a different group of developers, each with their own specialized skills and perspectives, could potentially arrive at varying results when they employ the same methodology. This variation might occur, for instance, if a developer has a deeper understanding of a specific framework, enabling them to make more informed decisions.

The reliability of the method is also affected by the fact that the frameworks being evaluated are constantly changing and being updated. If the methodology were to be repeated in the future when a framework has gone through considerable changes, some aspects being compared could reasonably end up being more or less relevant than they are at present. In this thesis, we restrict ourselves to only compare and evaluate: Next.js version 13.1.6, SvelteKit version 1.5.0 & Astro version 2.2.3.

Similarly, the target group of web developers is malleable, and what is considered important within web development in the present could end up being more or less consequential in the future.

2.4 Ethical considerations

The primary ethical considerations to be aware of in this thesis are related to the survey. As it requires the participation of other people, and the collection of their information, there are some important ethical considerations to keep in mind.

Confidentiality To ensure the privacy of the participants and the confidentiality of the gathered information, all gathered information must be securely stored. To ensure this, all survey responses are stored within a Google Sheets document with access restricted to only the thesis authors. Furthermore, responses are only presented in aggregate form, which will prevent the reconstruction of any single response.

The survey is also designed to gather as little personal information as possible, to make it impossible to determine the identify of any respondent.

Consent It must also be made clear to survey participants that the survey is completely voluntary. This is clearly specified within the missive letter of the survey, and the participants consent is gathered before allowing them to participate in the survey.

Communication As survey owners, we must also make ourselves available for communication to the survey participants in case of questions or data deletion requests. To ensure this, our contact information (email addresses) is provided together with the survey, such as in the missive letter, the "response has been sent" confirmation message, and the message to respondents when the survey is closed.

Sampling Bias There is a high likelihood of there being a sampling bias present in the survey, as it is primarily distributed through channels convenient to the survey owners. As the survey is posted exclusively to a Slack channel comprised of current and former students of the Web Development programme at Linnaeus University, as well as a Reddit sub-forum, the sample group will likely skew toward a western population group. This may result in a sample that is not representative of the global group of web developers.

3 Theoretical Background

In this chapter we further discuss the knowledge gap and the related work of the project. We also discuss several concepts central to web development and rendering frameworks, to provide the necessary theoretical background to further explore the research area.

3.1 Related Work and Knowledge Gap

A considerable amount of research has been conducted comparing various front-end JavaScript frameworks and rendering methods. For example, one bachelor thesis compares and evaluates the front-end frameworks React, Angular, Vue.js, and Svelte [4]. Additionally, there is research that contrasts client-side rendering frameworks with server-side rendering frameworks [8]. However, there is a significant gap in the literature when it comes to formal research strictly focused on comparing rendering frameworks. Developers will have an easier time finding evaluations and comparisons of rendering frameworks in informal settings [11], [12], but these kinds of resources are often biased and opinionated, which can make them unreliable.

In order to address this gap, the methodology of the bachelor thesis "Hello Framework! A heuristic method for choosing front-end JavaScript frameworks" [4] will be adapted to cater to the comparison of rendering frameworks. Some modifications are necessary in order to ensure that the methodology is applicable to the new subject of rendering frameworks, as opposed to front-end frameworks.

3.2 Concepts within Web Development

Web development is a dynamic field where various types of frameworks are constantly being developed to improve the development process. A majority of these frameworks today are designed for JavaScript, the de facto standard programming language for developing web applications today [1]. Many different types of framework exist today, each intended to serve a specific purpose or improve a certain aspect of the development process. For instance, some frameworks focus on building and managing the UI, while others improve database interactions or assist in verifying code functionality.

Rendering frameworks are designed to manage both the front-end and back-end functionality of a web application, making it responsible for both the rendering and the serving of the application. One of the chief strengths of a rendering framework is its ability to offer a complete web development solution.

Document Object Model (DOM) The DOM serves as a programming interface for web documents, representing the structure and content of a webpage as a tree-like hierarchy of nodes and objects. These objects correspond to various parts of the document, such as elements, attributes, and text. By providing a connection for programming languages like JavaScript, the DOM enables developers to interact with and manipulate HTML documents. This allows them to dynamically update content, modify styles, and respond to user events [20].

Single Page Application (SPA) A Single Page Application is a web application that runs entirely in the browser, only loading a single HTML document and dynamically updating the content using JavaScript. This allows the website to be used without loading whole new pages from the server [21].

Multi-Page Application (MPA) A Multi-Page Application is a web application that primarily runs on the server side, generating and delivering multiple HTML documents to the client [22].

Client Side Rendering (CSR) Client side Rendering is a technique often used in Single Page Applications. In CSR, the server provides only a minimal HTML document while the rendering is done using JavaScript in the browser[23].

Server Side Rendering (SSR) Server Side Rendering is a technique often used in Multi-Page Applications. In SSR, the server pre-renders HTML content for each page in response to user requests [23].

Static Site Generation (SSG) Static Site Generation is the process of building a web application's content only as static files, before deploying it to a static hosting service such as a Content Delivery Network. This enables only static files to be served, potentially resulting in faster load times and improved performance [24].

Progressive Enhancement Progressive Enhancement is the strategy of prioritizing the delivery of a basic web application that functions without JavaScript, with features requiring JavaScript layered on top for clients that support it. This strategy ensures that the application remains accessible even without JavaScript or in cases of limited browser capability [25].

Markdown Markdown is a lightweight markup language that simplifies formatting text for the web using a plain-text editor. Markdown is easily converted to HTML and is commonly used for blogging, content management, documentation and readme-files [26].

JSON JSON (JavaScript Object Notation) is a syntax that uses human-readable text to store and send data objects. It is a widely used syntax within computer science fields, including within web development [27].

File-Based Routing File-based routing is an approach where routes are generated based on the file structure within a project. For example, a file found in "pages/about/index.tsx" would automatically generate a route at "/about" on the web application. This is commonly found in rendering frameworks, as opposed to the traditional routing used in back-end frameworks like Express where routes must be explicitly defined in code.

Hot Reload Hot reloading enables web developers to view code changes without manual page reloads or server restarts, as development frameworks automatically detect and inject updated code into the running development application.

JavaScript (JS) Known predominantly for its role in Web page scripting, JavaScript is a minimalistic, compiled programming language that also sees usage in non-browser environments, like in Node.js [28].

TypeScript Built on JavaScript, TypeScript is a programming language extended with a strong typing system. TypeScript is developed and maintained by Microsoft [29].

Tailwind CSS Tailwind CSS is a utility-first CSS framework that enables developers to build user interfaces rapidly. It provides pre-defined CSS classes that allow you to customize HTML elements without having to write any CSS yourself [30].

Node Package Manager (npm) Node Package Manager is a software registry for JavaScript packages, used for installing, updating and managing JavaScript package [31].

HyperText Markup Language (HTML) HyperText Markup Language is the most basic building block of the Web, establishing the structure and meaning of web content [32].

Cascading Style Sheets (CSS) Cascading Style Sheets is a language for stylesheets that specifies how HTML or XML documents should be presented visually by defining the rendering of each element on the screen [33].

3.3 Rendering Frameworks

Although there is no formal definition of what a rendering framework is, State of JS defines it as: "Frameworks focused on rendering and serving your application" [1]. There are many other contending names for the framework category. For instance, Astro refers to itself as a web framework [14], Next.js refers to itself as a React framework or a web development framework [3], and SvelteKit sometimes refers to itself as an app framework or a metaframework [24].

Next.js

Next.js is an open-source rendering framework maintained by the company Vercel. It uses React as its UI framework, while a custom-designed library is used to manage the back-end functionality [3]. The version of Next.js being evaluated in this thesis is version 13.1.6.

SvelteKit

SvelteKit is an open-source rendering framework created by Rich Harris and maintained by the Svelte core team members. It uses Svelte as its UI framework, and similar to Next.js it also has a custom-built library to manage the back-end functionality which ties together the front-end with the back-end [13]. The version of SvelteKit being evaluated in this thesis is version 1.5.0.

Astro

Astro is an open-source rendering framework maintained by the Astro Technology Company. Astro provides several alternatives for implementing a UI. While Astro uses its own UI framework, developers can also chose to utilize framework components from other UI frameworks, such as Svelte and React. Likewise, Astro also provides the option to use Node.js among other things to manage the back-end through adapters, while also offering its own built-in library to manage the back-end [14]. The version of Astro being evaluated in this thesis is version 2.2.3.

Others

Other moderately popular rendering frameworks include frameworks such as Nuxt, Gatsby, Remix and Eleventy. The reason we chose to exclude these from this study is because they are less liked/less popular according to the survey made by State of JS [1]. It is however worth to mention that State of JS is conducted in English and therefore will not reflect the opinions of the people who do not speak English, potentially impacting the results of which framework is the most popular.

4 Research project – Implementation

Two research methods were realized to collect data for the comparative evaluation of the selected rendering frameworks. First, the survey using questionnaires, secondly, the controlled experiment on web applications using the rendering frameworks. The methods were executed individually in succession, starting with the survey.

Survey

The survey was implemented as a Google Forms survey and was developed to largely mimic the State of JavaScript survey of 2022 [1]. The survey itself opens with an introductory message, explaining the context of the survey as being part of a Bachelor Thesis at Linnaeus University. The introductory text also contains a message on our ethical considerations, clarifying to the respondent that their response is anonymous and that only the thesis authors will have access to the gathered information. Lastly, it includes the contact information of both thesis authors to facilitate communication if necessary.

Continuing, the survey consists of three primary sections: (1) an "About You" section with three questions, which measured the age, educational experience and professional experience of the respondent; (2) a "Rendering Framework" section with three questions, measuring the respondents' knowledge, interest and usage of Next.js, SvelteKit and Astro; and (3) a final section measuring the respondents experience with common web development application patterns, as well as their opinions of what they find important in a rendering framework. For the full survey, see appendix item A.

The survey was published to two locations on separate dates. It was first published on April 5th 2023 in a Slack channel comprised of current and former students of the Web Development programme at Linnaeus University. At time of publishing, the channel had around 650 members. Secondly, the survey was published on April 8th 2023 to the Reddit.com sub-forum "/r/webdev", a community of people with an interest in web development. At time of publishing, the community had around 1.5 million members. The survey was published with separate missive letters shaped for the specific community it was being published to, see appendix item B.

The survey was closed on the 16th of April 2023 and had received a total of 13 responses. All survey responses were placed within a Google Sheets document, with access restricted to only the thesis authors.

Controlled Experiment

The controlled experiment began with setting up specifications for the web applications. The applications were then developed according to the specifications. After the development of each application was completed, aspects of the framework as well as the applications themselves were measured against a checklist and a collection of in-depth questions. The checklist and in-depth questions were developed in parallel with the survey, and are based on a methodology developed in one of the theses [4] in the related work of this project.

Specifications

The specifications for the web applications were developed with the goal of creating a multi-faceted web application that is diverse enough to emulate more than a single typical web application, but simple enough to remain within the scope of the thesis. The specifications call for three primary pages: (1) a main landing page containing the logo and name of the framework, as well as a title for the web application; (2) a blog page, to emulate a typical blog web application serving markdown files as blog posts; and (3) a "big data" page, to emulate a web application that needs to serve a large quantity of data to the page. For the exact web application specifications, see appendix item C.

Development

A separate web application was developed according to the specifications for each of the selected rendering frameworks, one using Next.js, one using SvelteKit, and one using Astro. The applications are version-controlled and available on separate repositories on GitHub, see appendix item E.

Checklist The frameworks and completed web applications were measured against the following checklist, which is based on a similar checklist from the methodology of a related work thesis [4].

- **Build Size:** The size of the compiled web application code
- **Loading Speeds:** The loading speeds of the different pages
- **Community:** GitHub Stars & Contributors to the framework
- **Documentation:** Is the documentation sufficient and of high quality?
- **Tutorials:** Are there any tutorials available and what do they contain?
- **UI Framework:** What UI framework does the rendering framework use?
- **Rendering Methods:** Are CSR, SSR and SSG supported?

In-depth questions Alongside the checklist, the following collection of in-depth questions were also answered in relation to each rendering framework. The questions are based on a similar collection from the methodology of a related work thesis[4].

- Is the framework mature?
- Is it being actively developed?
- Is it easy to work with?
- Is it fast to work with?

5 Results

Presented in this chapter are the results of the survey, as well as the results of the controlled experiment on web applications using the selected frameworks.

5.1 Survey

The survey received a total of 13 responses. Each question posed in the questionnaire is presented with its raw data as a table, together with an accompanying graph.

About You

Age The responses to the question "How old are you, in years?" is presented in Table 5.2. A complementary graph is shown in Figure 5.1

Age	Count
18-24 years old	1
25-34 years old	9
35-44 years old	3
45-54 years old	0
55-64 years old	0
65 or older	0

Table 5.2: Respondents age.

Q: How old are you, in years?

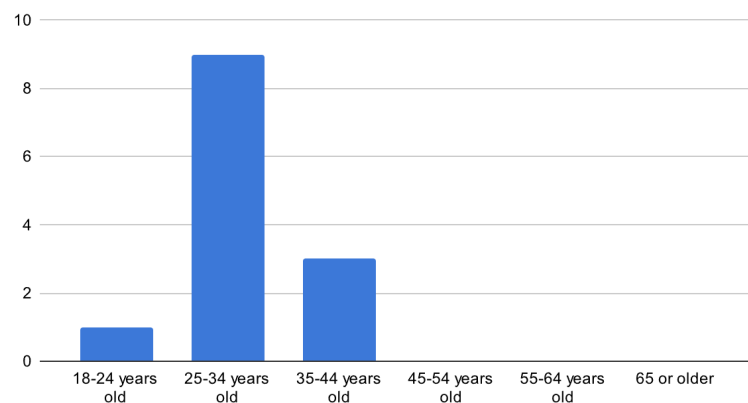


Figure 5.1: Respondents age shown as a graph.

Education The responses to the question "How many years have you spent in formal web development education?" is presented in Table 5.3. A complementary graph is shown in Figure 5.2

Time in Education	Count
None at all	0
Less than one year	1
1 to 2 years	5
3 to 5 years	6
More than 5 years	1

Table 5.3: Respondents education.

Q: How many years have you spent in formal web development education?

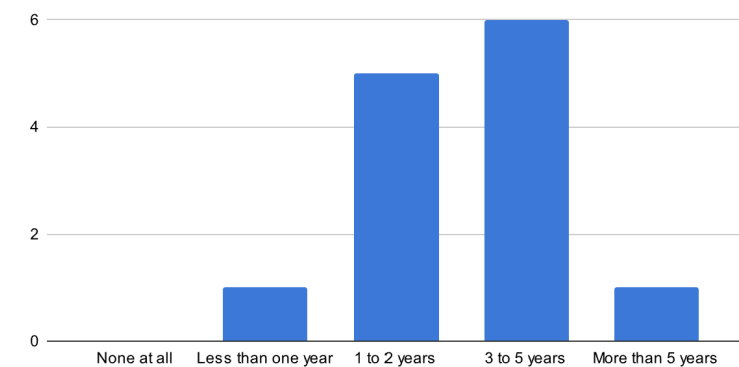


Figure 5.2: Respondents education shown as a graph.

Experience The responses to the question "How long have you been working in web development, in years?" is presented in Table 5.4. A complementary graph is shown in Figure 5.3.

Professional Experience	Count
None at all	6
Less than one year	2
1 to 2 years	0
3 to 5 years	2
6 to 10 years	2
More than 10 years	1

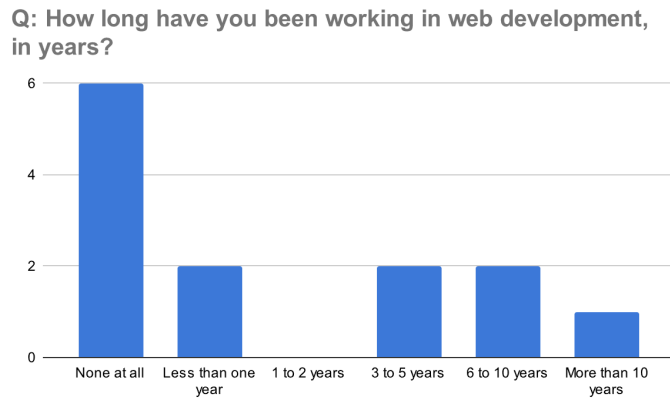


Table 5.4: Respondents experience.

Figure 5.3: Respondents experience shown as a graph.

Frameworks

The responses to the questions gauging knowledge, experience and opinions of Next.js, SvelteKit & Astro is presented in Table 5.4. A complementary graph is shown in Figure 5.5.

Option	Framework		
	Next.js	SvelteKit	Astro
Never heard of it	0	2	8
Heard of it, would like to learn	4	5	5
Heard of it, not interested	0	5	0
Used it, would use again	8	1	0
Used it, would not use again	1	0	0

Figure 5.4: Framework metrics.

- 🤔 Never heard of it/Not sure what it is
- ✅ Heard of it > Would like to learn
- ❌ Heard of it > Not interested
- 👍 Used it > Would use again
- 👎 Used it > Would not use again

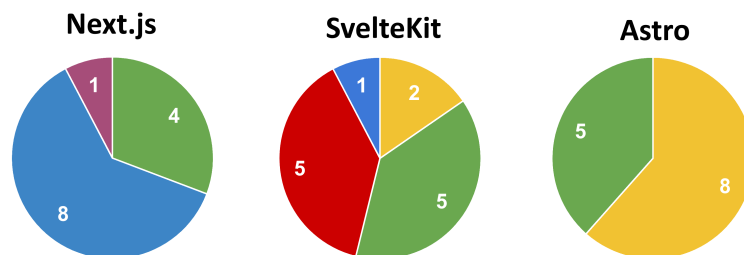
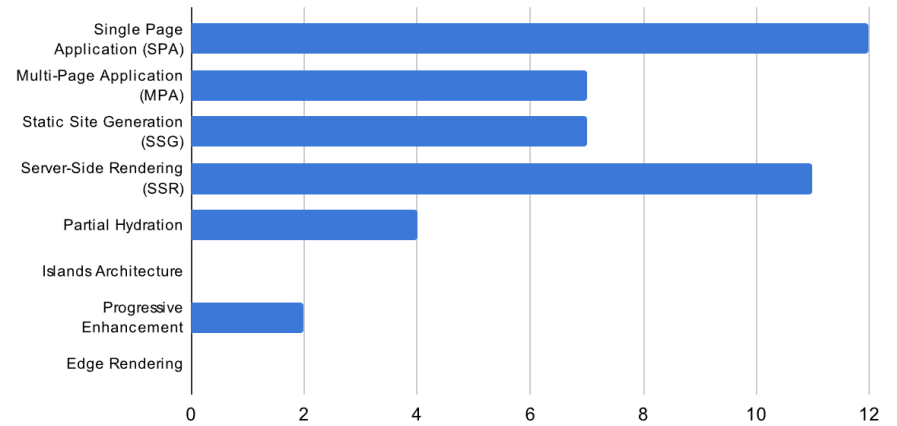


Figure 5.5: Framework metrics shown as a graph.

Architecture and Rendering Patterns

The responses to the question "Which of the following architecture and rendering patterns have you used in the last year?" is presented in Table 5.5. A complementary graph is shown in Figure 5.6.

Q: Which of the following architecture and rendering patterns have you used in the last year?



Pattern	Count
Single Page Application (SPA)	12
Multi-Page Application (MPA)	7
Static Site Generation (SSG)	7
Server-Side Rendering (SSR)	11
Partial Hydration	4
Islands Architecture	0
Progressive Enhancement	2
Edge Rendering	0

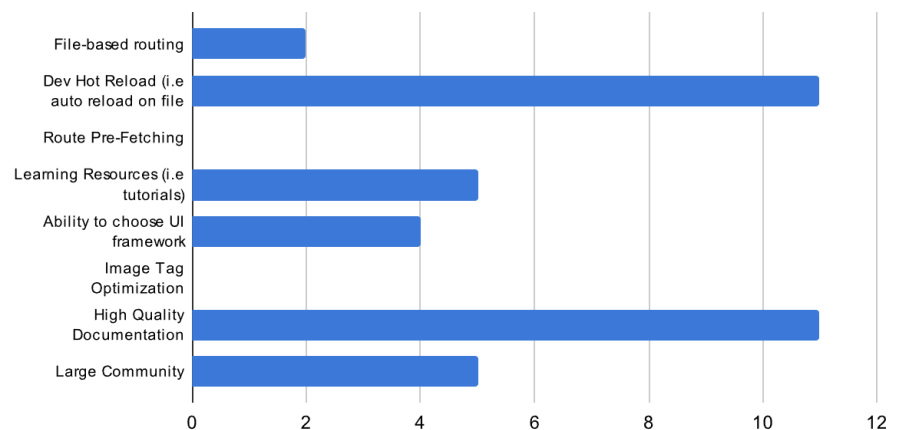
Table 5.5: Architecture and rendering pattern responses.

Figure 5.6: Architecture and rendering pattern responses shown as a graph.

Opinions

The responses to the question "Which of the following aspects of a rendering framework do you find most important?" is presented in Table 5.6. A complementary graph is shown in Figure 5.7.

Q: Which of the following aspects of a rendering framework do you find most important?



Aspect	Count
File-based routing	2
Dev Hot Reload	11
Route Pre-Fetching	0
Learning Resources	5
Ability to choose UI framework	4
Image Tag Optimization	0
High Quality Documentation	11
Large Community	5

Table 5.6: Respondents opinions.

Figure 5.7: Respondents opinions shown as a graph.

5.2 Controlled Experiment

Three web applications, one for each framework, were built in accordance with the established application specifications, see appendix item C.

Each web application is published and available for viewing in separate GitHub repositories, see appendix item E.

Next.js

The web application landing page is shown in Figure 5.8, and the "/blog" route is shown in Figure 5.9. A specific blog page at the route "/blog/nextjs-forms" is shown in Figure 5.10, and the "/big-data" route is shown in Figure 5.11. The "/big-data/people" route is shown in Figure 5.12, and a specific person at the route "big-data/people/1" is shown in Figure 5.13.

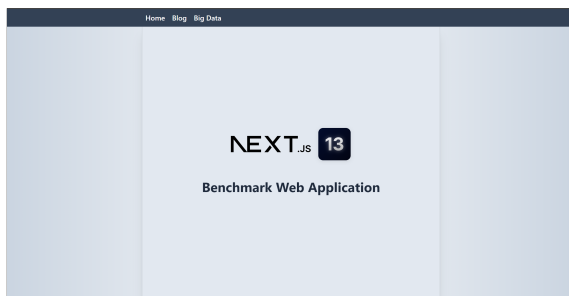


Figure 5.8: Next.js Web App Landing Page

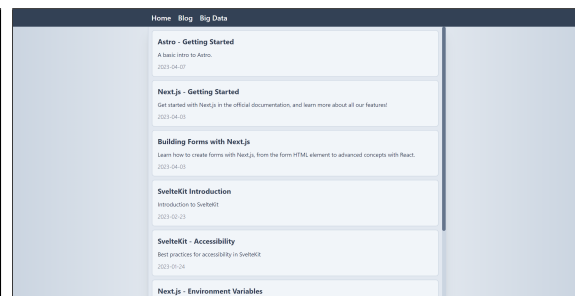


Figure 5.9: Next.js Web App Blog List

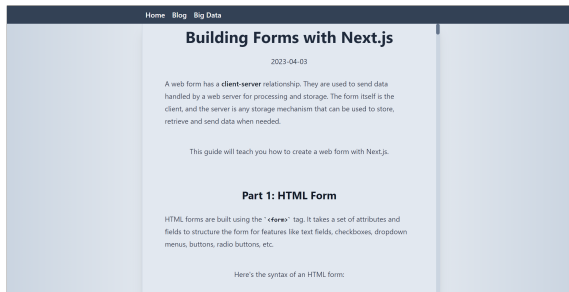


Figure 5.10: Next.js Web App Specific Blog Page

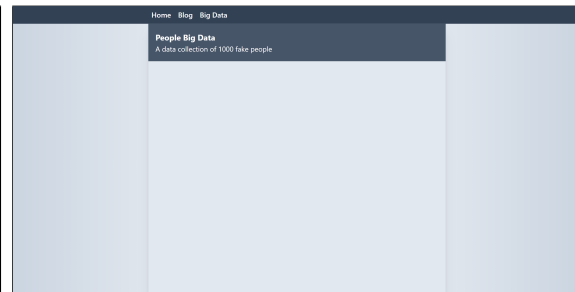


Figure 5.11: Next.js Web App "/big-data" Page

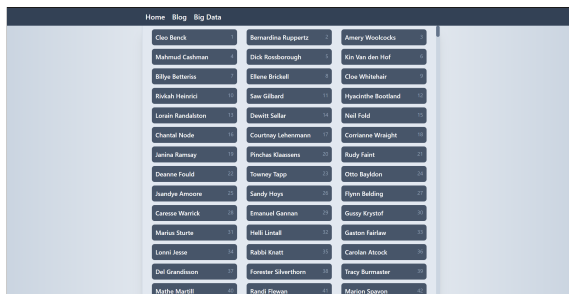


Figure 5.12: Next.js Web App "/big-data/people" Page

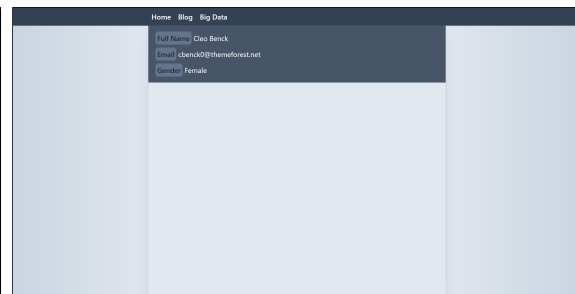


Figure 5.13: Next.js Web App Big Data Specific Person Page

Checklist

Build Size 645 KB

Loading Speeds See Tables 5.7, 5.8, 5.9 and 5.10 for the different pages loading speed measured using the browser Mozilla Firefox's built-in performance measuring tool which measures the time it takes for the DOMContentLoaded event as well as the load event to fire. The tests were run a total of four times per page by refreshing the page, cold starts were ignored. See 5.7 for the load times of the "/big-data/people" route. See 5.8 for the load times of the "/big-data/people/1" route. See 5.9 for the load times of the "/blog" route. See 5.10 for the load times of the "/blog/astro-getting-started" route.

Test #	DOMContentLoaded (ms)	Load (ms)
1	90	289
2	90	317
3	95	273
4	85	377
Average	90	314

Table 5.7: Page load times "/big-data/people"

Test #	DOMContentLoaded (ms)	Load (ms)
1	69	112
2	92	156
3	77	136
4	80	132
Average	79.5	134

Table 5.8: Page load times "/big-data/people/1"

Test #	DOMContentLoaded (ms)	Load (ms)
1	86	134
2	96	136
3	71	126
4	72	122
Average	81	129.5

Table 5.9: Page load times "/blog"

Test #	DOMContentLoaded (ms)	Load (ms)
1	79	119
2	90	146
3	75	131
4	76	138
Average	80	133.5

Table 5.10: Page load times "/blog/astro-getting-started"

Community Next.js has more than 105 000 stars on GitHub, with just over 2 600 contributors [34].

Documentation The documentation is only available in English. There is a beta version of the documentation that improves the current documentation as well as introduce features that are not yet in the latest stable version of Next.js. An explanation of most features if not every feature is easy to find within the docs by using the navigation bar [35] [36].

Tutorials The Next.js documentation offers an interactive tutorial. The tutorial is of a code-along style, where the user is required to clone a base repository in order to follow along with the code examples. It covers how to build and deploy a basic web application using Next.js [35].

UI Framework Next.js uses React [2] as its underlying UI framework [2].

Rendering Methods CSR, SSR and SSG are all supported. The framework user per-page rendering, and uses SSR if a page exports a function with the name "getServerSideProps" [37], and SSG if a page exports a function with the name "getStaticProps" [38].

In-depth Questions

Is the framework mature? Compared to other rendering frameworks, Next.js is the oldest, having had its first major release in October of 2016 [39].

Is it being actively developed? Yes, it is being actively developed. Between the 28th of March and the 28th of April in 2023, there were 451 pull requests on the framework repository, of which 370 were merged [40].

Is it easy to work with? Next.js uses React as its front-end language, which we had experience with before the controlled experiment began. The file-based routing system is very intuitive, and setting up the rendering of markdown files for the blog was simple, with an example of how to do so provided in the framework documentation.

Is it fast to work with? Tailwind CSS has to be manually installed, but there is a guide for this on the Tailwind CSS website [41]. The hot reloading feature works well, and was never perceived by us to be slow.

SvelteKit

The web application landing page is shown in Figure 5.14, and the "/blog" route is shown in Figure 5.15. A specific blog page at the route "/blog/nextjs-forms" is shown in Figure 5.16, and the "/big-data" route is shown in Figure 5.17. The "/big-data/people" route is shown in Figure 5.18, and a specific person at the route "/big-data/people/1" is shown in Figure 5.19.

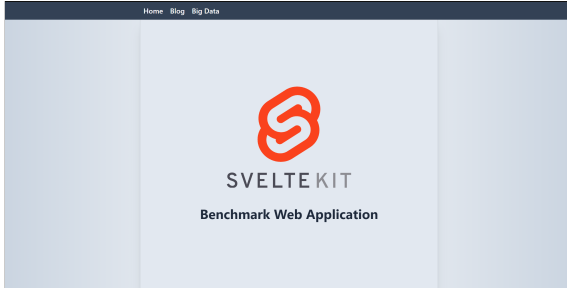


Figure 5.14: SvelteKit Web App Landing Page

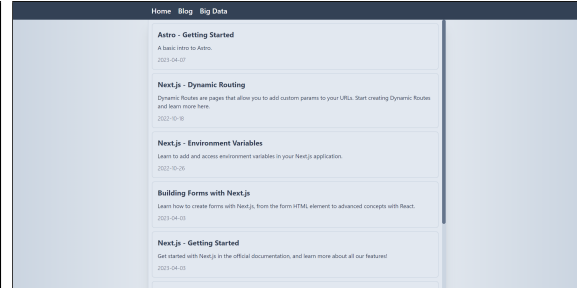


Figure 5.15: SvelteKit Web App Blog List



Figure 5.16: SvelteKit Web App Specific Blog Page

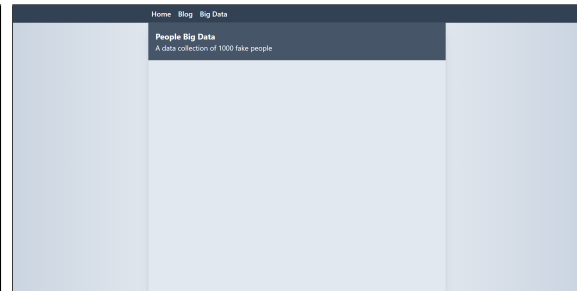


Figure 5.17: SvelteKit Web App "/big-data" Page

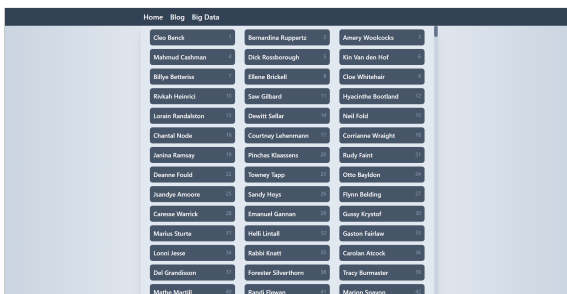


Figure 5.18: SvelteKit Web App "/big-data/people" Page

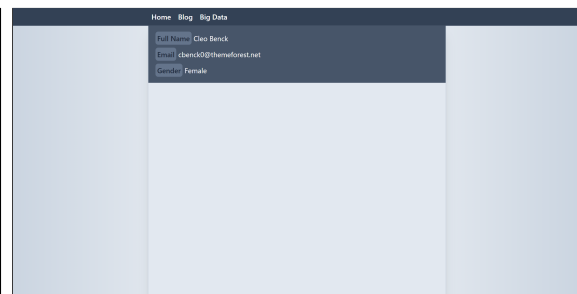


Figure 5.19: SvelteKit Web App Big Data Specific Person Page

Checklist

Build Size 565 KB

Loading Speeds See Tables 5.11, 5.12, 5.13 and 5.14 for the different pages loading speed measured using Mozilla Firefox's built in tool which measures the time it takes for the DOMContentLoaded event as well as the load event to fire. The tests were run a total of four times per page by refreshing the page, and cold starts were ignored. See 5.11 for the load times of the "/big-data/people" route. See 5.12 for the load times of the "/big-data/people/1" route. See 5.13 for the load times of the "/blog" route. See 5.14 for the load times of the "/blog/astro-getting-started" route.

Test #	DOMContentLoaded (ms)	Load (ms)
1	109	172
2	110	373
3	80	167
4	83	170
Average	95.5	220.5

Table 5.11: Page load times "/big-data/people"

Test #	DOMContentLoaded (ms)	Load (ms)
1	63	188
2	67	134
3	69	187
4	75	142
Average	68.5	162.75

Table 5.12: Page load times "/big-data/people/1"

Test #	DOMContentLoaded (ms)	Load (ms)
1	68	241
2	67	324
3	68	316
4	76	153
Average	69.75	258.25

Table 5.13: Page load times "/blog"

Test #	DOMContentLoaded (ms)	Load (ms)
1	92	274
2	71	330
3	71	467
4	81	192
Average	78.75	315.75

Table 5.14: Page load times "/blog/astro-getting-started"

Community SvelteKit has more than 14 300 stars on GitHub, with just over 400 contributors [42].

Documentation The documentation is only available in English. It explains most things in depth but it could be harder to find what you are looking for since each section within the documentation has subsections that could be moved into the main navigation bar depending on what you prefer [43].

Tutorials The SvelteKit documentation offers an interactive tutorial. The tutorial uses a web-based IDE, and doesn't require the user to clone any code to their local machine. It covers the basics of how to develop using SvelteKit, but there is no web application being built in the process. Some parts of the tutorial are incomplete, and only provide links to the relevant documentation instead [24].

UI Framework SvelteKit uses Svelte [7] as its underlying UI framework [13].

Rendering Methods CSR, SSR and SSG are all supported. The framework user SSR by default [44].

In-depth Questions

Is the framework mature? SvelteKit is the youngest of the investigated frameworks, with its first major release occurring in December of 2022 [45].

Is it being actively developed? Yes, it is being actively developed. Between the 28th of March and the 28th of April in 2023, there were 131 pull requests on the framework repository, of which 105 were merged [46].

Is it easy to work with? SvelteKit, which uses Svelte as its UI framework, felt highly intuitive to us as a way of writing code. It simplifies the rendering of HTML, making it feel like an improved version of vanilla JavaScript/HTML. SvelteKit also clearly distinguishes server and client code with different naming styles for each, with files named like “+page.server.ts” for the server code and “+page.ts” for the client code. The file-based routing is very intuitive, contributing to its ease of use.

Is it fast to work with? Like Next.js, Tailwind CSS has to be manually installed, but there is a guide for this on the TailwindCSS website [47]. The hot reloading feature works well, and was never perceived by us to be slow.

Astro

The web application landing page is shown in Figure 5.20, and the "/blog" route is shown in Figure 5.21. A specific blog page at the route "/blog/nextjs-forms" is shown in Figure 5.22, and the "/big-data" route is shown in Figure 5.23. The "/big-data/people" route is shown in Figure 5.24, and a specific person at the route "/big-data/people/1" is shown in Figure 5.25.

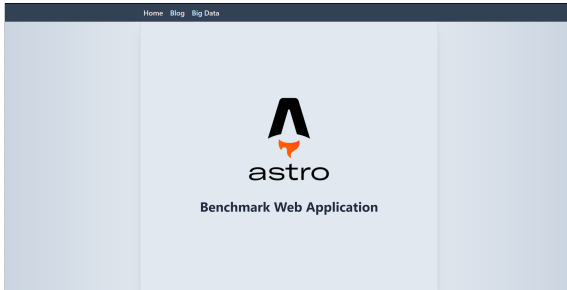


Figure 5.20: Astro Web App Landing Page

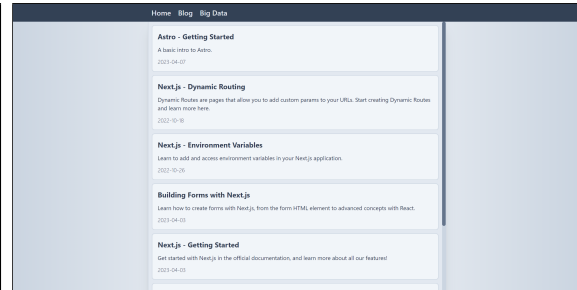


Figure 5.21: Astro Web App Blog List

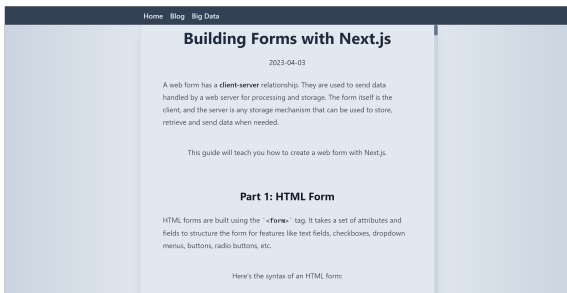


Figure 5.22: Astro Web App Specific Blog Page

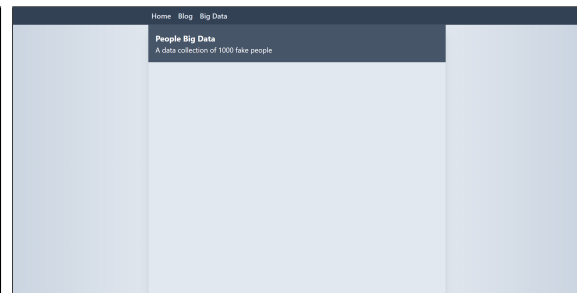


Figure 5.23: Astro Web App "/big-data" Page



Figure 5.24: Astro Web App "/big-data/people" Page

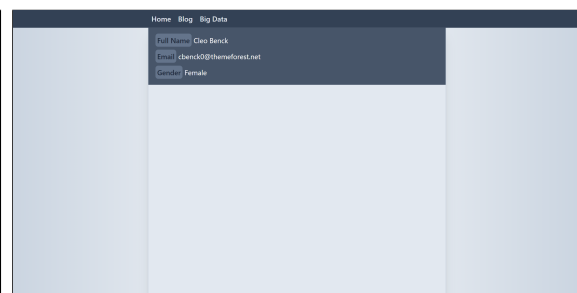


Figure 5.25: Astro Web App Big Data Specific Person Page

Checklist

Build Size 713 KB (with node adapter)

Loading Speeds See Tables 5.15, 5.16, 5.17 and 5.18 for the different pages loading speed measured using the browser Mozilla Firefox's built-in performance measuring tool which measures the time it takes for the DOMContentLoaded event as well as the load event to fire. The tests were run a total of four times per page by refreshing the page, and cold starts were ignored. See 5.15 for the load times of the "/big-data/people" route. See 5.16 for the load times of the "/big-data/people/1" route. See 5.17 for the load times of the "/blog" route. See 5.18 for the load times of the "/blog/astro-getting-started" route.

Test #	DOMContentLoaded (ms)	Load (ms)
1	161	224
2	152	207
3	174	227
4	179	360
Average	166.5	254.5

Table 5.15: Page load times "/big-data/people"

Test #	DOMContentLoaded (ms)	Load (ms)
1	61	90
2	68	106
3	52	79
4	62	106
Average	60.75	95.25

Table 5.16: Page load times "/big-data/people/1"

Test #	DOMContentLoaded (ms)	Load (ms)
1	54	80
2	70	101
3	60	82
4	70	99
Average	63.5	90.5

Table 5.17: Page load times "/blog"

Test #	DOMContentLoaded (ms)	Load (ms)
1	66	89
2	58	76
3	71	97
4	61	93
Average	64	88.75

Table 5.18: Page load times "/blog/astro-getting-started"

Community Astro has more than 29 800 stars on GitHub, with almost 500 contributors [48].

Documentation The documentation is available in 12 different languages including English, Chinese, Spanish and Russian. Some languages however do not have full support which means that some of the documentation will still be written in English. To ensure ease of use, the documentation is divided into seven categories, including Start here, Core Concepts, Tutorials, Basics, Recipes, Guides and Configuration [49].

Tutorials The Astro documentation offers an interactive tutorial. The tutorial provides the user with a choice of either cloning a repository to develop locally, or to use a web-based IDE. It covers how to build and deploy a blog web application using the Astro framework [50].

UI Framework Astro defaults to using no UI framework at all, but has support for using React, Preact, Svelte, Vue, SolidJS, AlpineJS and Lit [51]. The option for using multiple frameworks in a single project is also supported [51]. The web application built in this project used the default setting, which means no UI framework was used.

Rendering Methods CSR, SSR and SSG are all supported. The framework user SSG by default [52].

In-depth Questions

Is the framework mature? Astro is the second youngest of the investigated frameworks, having had its first major release in August of 2022 [53].

Is it being actively developed? Yes, it is being actively developed. Between the 28th of March and the 28th of April in 2023, there were 107 pull requests on the framework repository, of which 83 were merged [54].

Is it easy to work with? Astro does not require learning any specific UI framework, as it supports native HTML for front-end development. It also allows the use of other UI frameworks if desired. The file-based routing felt very intuitive to us. The framework supports rendering markdown files within the file-based routing directory without any extra configuration, which made the blog implementation very simple.

Is it fast to work with? Tailwind CSS was very fast to set up for the project, as it only requires a console command to install automatically. To enable SSR, the `astro.config.mjs` file must be configured and a server runtime adapter installed, but this process was straightforward thanks to the documentation.

6 Analysis and Evaluation

In this section of the thesis, we analyse the results of the survey and the controlled experiment, evaluating each rendering framework along the way.

6.1 Survey

In this section of the analysis we discuss the results of the survey, leading with a comparison of the demographics of the respondent sample group with that of the State of JavaScript survey of 2022.

Demographics

Starting with the demographics of the survey sample group, we can compare it against the demographics of the State of JavaScript survey of 2022 which has a considerably larger sample group of 39 471 respondents.

Of all respondents of the thesis survey sample group, 69% report being between the ages of 25 and 34 years old. This is significantly higher when compared to the State of JavaScript survey, where only 35% of the sample group belong to this age range [1]. There is a noticeable disparity between the two demographic groups.

As regards to the formal education level of the thesis survey group, 38% of respondents report having spent between 1 and 2 years in formal web development education and 46% report a range between 3 to 5 years. This contrasts with the State of JS survey, where 40% of respondents possess higher education degrees related to web development [1]. It is however important to note that a direct comparison can't be made as the survey questions are distinct from each other.

Lastly, regarding the professional experience of the survey respondents, we can see that 46% of respondents have no professional experience while a collected 38% have 3 or more years of professional experience.

The demographics of the thesis survey respondent group differ in many ways from the larger sample group of the State of JavaScript survey of 2022. This could reasonably be explained by the convenience bias discussed earlier, seeing as the survey was distributed in a channel comprised of mainly current and former students at Linnaeus University. As we continue to analyze the thesis survey results, it's important to remember that the sample group is more representative of web development students or recent graduates, rather than the global population of web developers.

Framework Section

Going into the framework section of the survey, we analyse the respondents knowledge, usage and interest of our selected frameworks. Each ratio is defined as follows:

- **Knowledge:** (total - never heard) / total
- **Usage:** (would use again + would not use again) / total
- **Interest:** want to learn / (want to learn + not interested)

Next.js, SvelteKit and Astro respectively have a knowledge ratio of 100%, 85% and 38% among the respondents of the survey. These ratios are similar to those of the State of JS survey [1], with the exception of Astro which has a considerably higher awareness rate (68%) in the State of JS sample group.

The frameworks respectively have a usage ratio of 69%, 8% and 0%. These results are also quite similar to the State of JS results [1], and indicate that Next.js is considerably more used in practice than both SvelteKit and Astro at present.

Regarding the survey groups respective interest of each framework, the numbers lie at 100%, 50% and 100% for Next.js, SvelteKit, and Astro. These interest ratios reveal that while SvelteKit has a moderate interest rate among the thesis survey respondents, both Next.js and Astro have captured the full interest of the surveyed group. Comparatively, the State of JS survey also reports rather high interest rates for Next.js (65%) and SvelteKit (66%), with Astro taking the lead at 67% [1].

The differences in knowledge, usage, and interest ratios between the two surveys can be attributed to various factors, such as sample size, demographics, and the specific focus of the respondents' work. However, we argue that the relatively high interest rate in Astro, despite its lower usage and knowledge ratios, suggests that this framework may have a significant potential for growth in the near future.

Patterns and Opinions

In this section of the survey analysis, we examine the sample groups usage of application patterns and rendering techniques, as well as their opinions of aspects of rendering frameworks.

Starting with the usage of application patterns and rendering techniques, a near universal experience is having recently used both a SPA pattern (92%) and a SSR rendering technique (85%). A lower level was observed in the experience of MPA (54%), SSG (54%), partial hydration (31%) and progressive enhancement (15%). No respondents had recently used either edge rendering or islands architecture.

These findings indicate that SPA is more commonly used than MPA, but MPA remains a relatively common pattern among the survey respondents. Similarly, SSR is more commonly used than SSG, but SSG still has a notable presence as a rendering technique. In contrast, partial hydration and progressive enhancement are not as common among the respondents, indicating that framework functionality centered on these techniques are less important to the survey sample group.

The complete absence of edge rendering and islands architecture usage in the survey group highlights that these techniques are not a priority for the respondents. It is important to note that islands architecture is a pattern specific to the Astro framework, which could attribute the result to the non-usage of Astro in the survey group. As Astro's interest rate is high among the respondents, it is possible that the adoption of islands architecture may increase if the framework gains more traction in the future.

Continuing into the survey section measuring opinions, two aspects of rendering frameworks stand out as being especially important to the survey sample group: dev hot reload (85%) and high quality documentation (85%). Other aspects such as learning resources (38%), large community (38%), ability to choose UI framework (31%) and file-based routing (15%) were also of some importance to the respondents, albeit to a lesser degree. No respondents deemed route pre-fetching or image tag optimization to be among their most important aspects.

These findings suggest a fast dev hot reload and high quality documentation are nearly universally important to web developers, and that those aspects should be inspected carefully when choosing a rendering framework. While the same can't be said for all other rendering framework aspects, many of them are still regarded important and should be considered when evaluating a framework. The significance of the "ability to choose UI

framework" implies that the underlying UI framework of a rendering framework is an important consideration for many web developers. We believe that this reflects positively on the Astro framework, as it not only allows for the possibility to choose between several UI frameworks, but also for the usage of multiple UI frameworks within a single project.

6.2 Controlled Experiment

In this section of the analysis we discuss the results of the controlled experiment, leading with an evaluation of the Next.js rendering framework.

When discussing the performance of each framework, it is crucial to note that raw loading speeds are not the primary focus of this analysis. The tests were run on a local computer, and as such, the specific numbers obtained would differ if the tests were conducted on a different machine. Instead, we will concentrate on the percentage difference between the performance of each framework, as this metric should remain consistent regardless of the machine on which the tests are run. To better visualize the results, a graph comparing the performance of each system is presented in Figure 6.26 and 6.27.

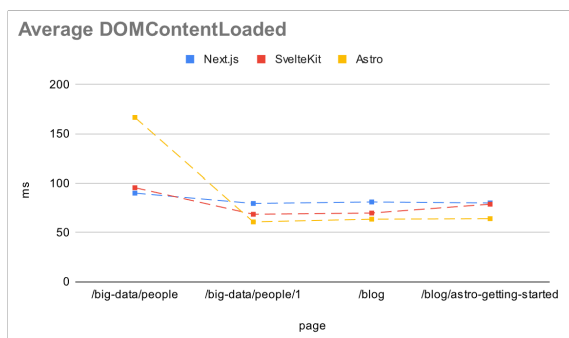


Figure 6.26: *DOMContentLoaded* page comparison

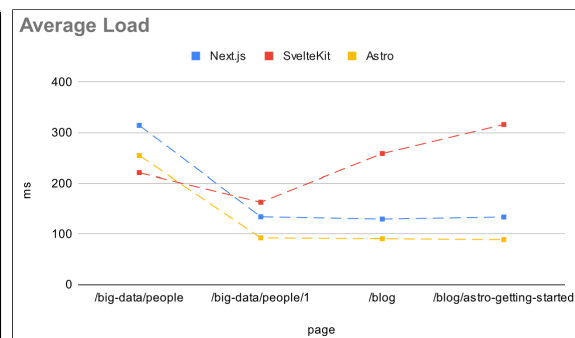


Figure 6.27: *Load* page comparison

Next.js

In this section we analyse the results of the Next.js portion of the controlled experiment, evaluating the rendering framework along the way.

Performance Factors

The total Next.js build size is ~13% larger than that of SvelteKit and ~10% smaller than that of Astro, positioning it in the middle in terms of build size for this specific web application.

We can analyze the loading speeds on the different page routes mentioned in section 5.2 to see how Next.js holds up against SvelteKit and Astro.

/big-data/people Analyzing this page shows that Next.js is ~6% faster than SvelteKit and 85% faster than Astro when comparing the `DOMContentLoaded` event, along with being ~30% slower than SvelteKit and ~19% slower than Astro when comparing the load event, see Tables 5.15, 5.7, 5.11. Ultimately, this makes it the **fastest** when comparing the `DOMContentLoaded` event times, but the **slowest** when it comes to the load event times.

/big-data/people/1 Taking a look at this page displays that Next.js is ~14% slower than SvelteKit and ~24% slower than Astro when comparing the DOMContentLoaded event, along with being ~21% faster than SvelteKit and ~29% slower than Astro when comparing the load event, see Tables 5.16, 5.8, 5.12. This makes it the **slowest** when comparing the DOMContentLoaded event times.

/blog Examining this page demonstrates that Next.js is ~14% slower than SvelteKit and ~22% slower than Astro when comparing the DOMContentLoaded event, along with being ~99% faster than SvelteKit and ~30% slower than Astro when comparing the load event, see Tables 5.17, 5.9, 5.13. This makes it the **slowest** when comparing the DOMContentLoaded event times.

/blog/astro-getting-started When evaluating this page, it shows that Next.js is ~2% slower than SvelteKit and 20% slower than Astro comparing the DOMContentLoaded event, along with being ~137% faster than SvelteKit and ~35% slower than Astro when comparing the load event, see Tables 5.18, 5.10, 5.14. This makes it the **slowest** when comparing the DOMContentLoaded event times.

Non-Performance Factors

Next.js is the most mature of the frameworks being evaluated, being not only considerably older, but is also the framework with the largest community and usage among developers according to the thesis survey and State of JS survey [1]. It is also the framework being most actively developed, going by pull requests on the framework repository. However, it is important to recognize that the number of pull requests is an imperfect indicator of development activity, as different projects can have varying standards for the size of a typical pull request. Next.js could be argued to be an industry standard, with a usage rate of 49% according to the State of JS survey. The survey also displays an interest and retention ratio are 65% and 90%, respectively, indicating a high level of satisfaction among developers who use the framework.

Next.js uses React as its UI framework which is the most popular UI framework [1]. It is also the industry norm to use React as the UI framework, even though it is not as well liked as Svelte for instance. However, its widespread use by companies worldwide compensates for some of the potential drawbacks that it has in terms of the developer experience.

Given that nearly all web developers in our sample group consider dev hot reloading an important feature, the experience with the feature during the controlled experiment are very relevant. The dev hot reloading feature provided by Next.js proved to be highly satisfactory and comparable to the other frameworks. Consequently, we believe that Next.js' hot reloading functionality is both effective and reliable, meeting the needs of most web developers.

High-quality framework documentation was also considered essential to the sample group, with tutorials being considered important as well, albeit to a lesser degree. In our evaluation, we found the documentation for Next.js to be of very high quality. Everything was easily accessible, with both the navigation and search feature functioning effectively. Specifically, for the blog implementation in the controlled experiment, we appreciated the availability of a markdown parser in the documentation. This was particularly helpful in simplifying the development process. The tutorial provided by Next.js was also beneficial, although it required local coding instead of the web-based solutions offered by SvelteKit

and Astro, which we found more preferable. We also appreciated that the Next.js tutorial guides users through creating a functional web application, as this approach allows developers to produce something practical while learning.

Since Next.js has such a large community it comes with the benefit that some very powerful libraries get developed specifically for it, such as tRPC [55] and NextAuth.js [56]. With NextAuth.js making it very easy for applications created in Next.js to implement authorization since it has built in support for all kinds of authorization methods, including username/password, magic link, passwordless, authorization using Google and much more. Furthermore, tRPC offers a better way to get end-to-end typesafe APIs by making types interlinked between the client and the server, which is great for the developers who work in Typescript. However, SvelteKit does provide something very similar out of the box by generating types from the server-side component and using them inside of the client-side component, but it comes with the downside that the application has to recompile in order to generate these types.

Next.js supports CSR, SSR, and SSG. By default, the framework uses CSR for its pages, leaving the developer to explicitly define the rendering method per page if CSR is not desired. In order to switch to SSR or SSG, the developer must explicitly define a specific method on the page. As all rendering methods are supported, the default choice is not limiting in any way, but it may suggest that the framework developers consider CSR to be the most useful, or at least most commonly used, method for framework.

A developer that wants to utilize Tailwind CSS would in Next.js be required to perform a manual installation of the CSS framework. Fortunately, there is a helpful guide on the Tailwind website to assist with this process. While the installation is not too complicated, it is still a step that web developers would need to complete. This small hurdle does not significantly impact the ease of use, but it is worth noting.

SvelteKit

In this section we analyse the results of the SvelteKit portion of the controlled experiment, evaluating the rendering framework along the way.

Performance Factors

The total SvelteKit build size is ~13% smaller than Next.js and ~23% smaller than Astro, making it the smallest in terms of build size for this specific web application.

We can analyze the loading speeds on the different page routes mentioned in section 5.2 to see how SvelteKit holds up against Next.js and Astro.

/big-data/people Analyzing this page demonstrates that SvelteKit is ~6% slower than Next.js and ~74% faster than Astro when comparing the DOMContentLoaded event, along with being ~42% faster than Next.js and ~15% faster than Astro when comparing the load event, see Tables 5.15, 5.7, 5.11. This makes it the **fastest** out of the three when comparing the load event times.

/big-data/people/1 Assessing this page shows that SvelteKit is ~16% faster than Next.js and ~11% slower than Astro when comparing the DOMContentLoaded event, in addition to being ~18% slower than Next.js and ~41% slower than Astro when comparing the load event, see Tables 5.16, 5.8, 5.12. This makes it the **slowest** out of the three when comparing the load event times.

/blog Taking a look at this page we can see that SvelteKit is ~16% faster than Next.js and ~9% slower than Astro when comparing the DOMContentLoaded event, along with being ~50% slower than Next.js and ~65% slower than Astro when comparing the load event, see Tables 5.17, 5.9, 5.13. This makes it the **slowest** out of the three when comparing the load event times.

/blog/astro-getting-started Evaluating this page displays that SvelteKit is ~2% faster than Next.js and ~19% slower than Astro when comparing the DOMContentLoaded event, as well as being ~58% slower than Next.js and ~72% slower than Astro when comparing the load event, see Tables 5.18, 5.10, 5.14. This makes it the **slowest** out of the three when comparing the load event times.

Non-Performance Factors

SvelteKit is the youngest among the analyzed frameworks, standing in contrast to the more mature Next.js. Despite its youth, SvelteKit has attracted a community, though it is the smallest in terms of GitHub stars and contributors. The framework community is substantially smaller than that of Next.js and marginally smaller than Astro. In terms of developer usage, SvelteKit trails behind Next.js with a 12% rate according to the State of JS survey, but it still outpaces Astro at 8%. SvelteKit is well-liked among web developers, with an interest ratio of 66% and a retention ratio of 93%. However, its lower usage rate compared to Next.js suggests that it is not as widely used commercially, and might be more chiefly used in hobby-like projects for some developers. Regarding development activity, there were 131 pull requests for SvelteKit, with 105 being merged. This level of activity is lower than that of Next.js, but more active than Astro. Again, it is important to note that pull request numbers are not a perfect measure of development activity since different projects might have varying standards for the size and scope of a typical pull request.

SvelteKit is not flexible when it comes to the UI framework since it uses Svelte and has no option to use any other UI framework. Svelte is however one of the most liked UI frameworks according to the State of JS survey [1], which means that the developer experience is among the best you can ask for currently for most people. Although, it is not yet an industry norm and that could play a part into developers decision to work with it.

The dev hot reloading feature provided by SvelteKit performed well and on par with the other frameworks. As such, we believe that the frameworks hot reloading feature meets the needs of most web developers.

We found the SvelteKit documentation to be of very high quality, having no difficulty in locating any information. Both the navigation and search features proved to be very effective during the controlled experiment. The SvelteKit tutorial differs from the Next.js tutorial in being focused on learning individual features in isolation. While being less practical, we believe this offers an advantage in allowing learners to dive into any part of the tutorial without following a linear progression. This enabled users to selectively explore the specific features they were interested in learning. It's unfortunate that some sections of the SvelteKit tutorial are not yet completed, but we don't consider it a very large disadvantage, as these sections seem to cover more niche features of the framework, and the pages do link to high quality documentation in the meantime. The isolated feature approach is however double-edged, as a completed tutorial does not result in the learner having created a functional web application, which we believe is less practical as a whole.

We were particularly pleased with the web-based IDE employed in the SvelteKit tutorial, as it eliminates the need for learners to download and run code on their local system.

SvelteKit also supports CSR, SSR, and SSG. Unlike Next.js, SvelteKit defaults to SSR. To use CSR or SSG, developers must define specific constants on a per-page basis, allowing them to fine-tune the rendering strategy as needed. The support for all rendering methods means that developers can choose the most suitable approach for their project, but the the default rendering method (SSR) may indicate the framework developers' preference for the framework.

SvelteKit also requires manual installation of Tailwind if the CSS framework is desired. Just like with Next.js, there is a helpful guide available on the Tailwind website to guide web developers through the process. The installation is fairly simple, and while it is not a major inconvenience, it still requires some level of effort.

Astro

In this section we analyse the results of the Astro portion of the controlled experiment, evaluating the rendering framework along the way.

Performance Factors

The total Astro build size (using the node adapter) is ~10% larger than Next.js and ~23% larger than SvelteKit, making it the largest in terms of build size for this specific web application. It's important to note that the total build size of a web application built with Astro will vary depending on the chosen server runtime adapter, and that these results are only relevant for specifically the node runtime adapter.

We can analyze the loading speeds on the different page routes mentioned in section 5.2 to see how Astro holds up against SvelteKit and Next.js.

/big-data/people Examining this page shows that Astro is ~46% slower than Next.js and ~43% slower than SvelteKit when comparing the DOMContentLoaded event, along with being ~23% faster than Next.js and ~13% slower than SvelteKit when comparing the load event, see Tables 5.15, 5.7, 5.11. This makes it the **slowest** out of the three when comparing the DOMContentLoaded event times.

/big-data/people/1 Analyzing this page demonstrates that Astro is ~31% faster than Next.js and ~13% faster than SvelteKit when comparing the DOMContentLoaded event, along with being ~41% faster than Next.js and ~71% faster than SvelteKit when comparing the load event, see Tables 5.16, 5.8, 5.12. This makes it the **fastest** out of the three when comparing the DOMContentLoaded and load event times.

/blog Evaluating this page shows that Astro is ~28% faster than Next.js and ~10% faster than SvelteKit when comparing the DOMContentLoaded event, along with being ~43% faster than Next.js and ~185% faster than SvelteKit when comparing the load event, see Tables 5.17, 5.9, 5.13. This makes it the **fastest** out of the three when comparing the DOMContentLoaded and load event times.

/blog/astro-getting-started Assessing this page displays that Astro is ~25% faster than Next.js and ~23% faster than SvelteKit when comparing the DOMContentLoaded event,

along with being ~50% faster than Next.js and ~256% faster than SvelteKit when comparing the load event, see Tables 5.18, 5.10, 5.14. This makes it the **fastest** out of the three when comparing the DOMContentLoaded and load event times.

Non-Performance Factors

Astro is an intermediate framework in terms of age and maturity, positioned between the younger SvelteKit and the more mature Next.js. While its community is by no means small, with a size slightly larger than SvelteKit's, it is still significantly smaller than the Next.js community in terms of GitHub stars and contributors. Astro's usage among web developers is also lower, with a 9% usage rate according to the State of JS survey, trailing behind SvelteKit's 12% and far behind Next.js's 49%. Astro is very well liked by developers, having an interest ratio of 67% and a retention ratio of 93% presented in the State of JS survey [1]. However, its lower usage rate suggests that it, like SvelteKit, might also be more suitable for hobby-like projects rather than more extensive commercial use cases. The lower usage rate of the framework is further reflected in the thesis survey results, as no respondents had used Astro before. In terms of development activity, Astro is considered the least active in terms of pull requests on the framework repository. Its level of activity is similar to, but slightly lower than SvelteKit and significantly lower than that of Next.js, showcasing its relative position to the two other frameworks. Again, it is important to note that the number of pull requests is non-perfect indicator of development activity, as different projects can have varying standards for the size and scope of a typical pull request.

Astro is unique in the fact that you are able to choose what UI framework you want to work with. This is a huge benefit since the UI framework can play a big part in what rendering framework you end up choosing. The available UI frameworks include React, Preact, Svelte, Vue, SolidJS, AlpineJS and Lit. This means that Astro has support for the most well liked and popular UI frameworks, going by the State of JS results on front-end frameworks [1].

Astro's dev hot reloading feature performed well in the experiment, matching the other frameworks. We believe that the feature is satisfactory to most web developers.

In our evaluation of Astro, we found its documentation to be of high quality, comparable to that of Next.js and SvelteKit. The information was easily accessible and both the navigation and search features were effective in helping us locate what we needed. In a similar vein to Next.js, the Astro documentation also provides an effective example of how to parse markdown content for blog-style web applications, which simplified our development process. The Astro tutorial stood out in our analysis, as it provides users with the option to either download code and follow along locally, like Next.js, or optionally use a web-based IDE, like SvelteKit. We consider the latter to be advantageous, as it allows users to learn the framework without needing to set up and manage a local development environment. Furthermore, the Astro tutorial aligns with the practical learning approach of the Next.js tutorial by guiding users through the creation and deployment of a functional web application. As mentioned in the Next.js tutorial evaluation, we consider this practical approach to be beneficial to learners.

Astro also supports CSR, SSR, and SSG. By default, Astro uses SSG. To use SSR, developers must enable it in the framework configuration and install a server runtime adapter. For CSR, developers specify which components to load on the client by assigning them a specific attribute, granting control over the balance between client and server responsibilities. With support for all rendering methods, developers have the freedom to

choose their preferred approach. The SSG default explicitly reflects the framework developers' belief that SSG is the most useful method for Astro's intended use, according to the framework documentation [14].

Astro, as opposed to the other rendering frameworks, simplifies the Tailwind CSS installation process by providing a short console command that automates installation. This feature makes it very easy for web developers to get started with the framework, removing even the smallest hurdle. The automatic installation in Astro is a nice touch, contributing to a more user-friendly experience for web developers.

7 Discussion

In this chapter we discuss our findings and determine if our research questions have been answered. We also discuss some areas of improvement for the methodology that were discovered after the implementation of the research project.

7.1 Survey

We believe that the survey was effective at answering **RQ2**: "What factors/metrics are important to web developers when choosing/comparing rendering frameworks?" as it allowed us to gather comprehensive data on the preferences and priorities of developers. By evaluating the demographics, we aimed to ensure that the survey sample was representative of the larger web development community. Despite the sample group being quite small, the responses were still largely similar to the State of JS survey, which is a considerably larger survey. This similarity, coupled with our survey being based on the State of JS survey, enabled us to make direct comparisons in many cases to strengthen the conclusions drawn from the thesis survey results. Assessing the knowledge, usage, and interest in various rendering frameworks provided insight into current trends and preferences, while measuring respondents' usage of common application patterns and opinions on framework aspects allowed us to understand the key factors that influence their decision-making process when choosing a rendering framework.

We could have enhanced our survey methodology by adopting a technique to manage biases through the use of repeated and flipped questions. This method entails framing questions about a particular topic in direct, and inverse ways, followed by cross-referencing the responses to minimize potential biases. We became aware of this strategy after completing the survey, and it could have potentially led to more accurate and reliable findings. However, we heavily prioritized making the survey concise and user-friendly, so it's not a given that we would have used this approach even if we were aware of it beforehand.

We also believe that the question survey question on the respondents opinions on aspects of rendering frameworks had an unclear response choice, specifically the choice of "Ability to choose UI framework". Upon reflection, we believe that the phrasing can be interpreted in multiple distinct ways. It could be taken to mean that the importance lies in the specific UI framework used by a rendering framework. Alternatively, it could be taken to mean that the importance lies in being able to choose between UI frameworks within a specific framework, such as what Astro offers. We believe respondents who considered this aspect important interpreted the choice based on the first meaning, given that no respondents had experience using the Astro framework.

7.2 Controlled Experiment

We believe that the controlled experiment was effective for providing answers to **RQ1**: "What are the differences in feature-set between the different rendering frameworks?" and **RQ3**: "Which of the selected rendering frameworks is the most suitable for the most common use cases, and how do they compare in terms of the determined metrics?". This approach allowed us to systematically evaluate the rendering frameworks under the same conditions, mitigating external factors that could introduce bias to the results. By building web applications in accordance with a mutual system specification, we were able to gather data on the frameworks performance in different scenarios, capabilities and features, as well as evaluate their respective developer experience.

7.3 Choosing a Rendering Framework

Short summaries of the frameworks, highlighting their key aspects.

Next.js

Performance wise Next.js is very similar to SvelteKit when rendering large dynamic pages, Next.js being the fastest at loading the DOM but the slowest at loading the entire page. Additionally, Next.js falls a bit short when it comes to rendering static pages, being the overall slowest at rendering the DOM.

Next.js is by far the most widely used rendering framework by present-day web developers. As it is also the framework with the largest community and history, it demonstrates a higher level of maturity compared to the other frameworks. These aspects of the framework make it the safest choice of the frameworks being evaluated.

SvelteKit

SvelteKit falls in middle of the three frameworks when it comes to rendering the DOM on both the dynamic and static pages. Notably, it is the fastest at rendering the entire dynamic page but the slowest at rendering all of the static pages.

The tutorial that SvelteKit offers is very impressive since it uses a web-based IDE making it very convenient for users to code along.

When you want to get data from the server to the client SvelteKit provides the smoothest developer experience, since it draws a clear line between the server and the client while also having a clean way of accessing the data from the server without having to do anything.

Astro

Astro is the best at rendering static pages, being a significant 256% faster than SvelteKit at a certain point. This makes Astro highly suitable for rendering static pages, like blogs or documentation. Additionally, Astro provides an unparalleled developer experience for rendering markdown files onto a page. However, it does face some challenges when the page contains a large amount of dynamic data.

Similar to SvelteKit, Astro boasts an exceptional tutorial that can also be accessed through a web-based IDE.

Given that Astro comes with multiple options of UI frameworks to work with it becomes the obvious choice for UI framework flexibility, as neither Next.js nor SvelteKit gives an option to switch.

Common Ground

The documentation is of exceptional quality for all of the frameworks, making it difficult to differentiate between them in this regard. However, if it is a requirement that the documentation should be in a language other than English, Astro has the advantage due to its support for multiple languages.

The hot reloading experience is virtually identical across all of the frameworks, with no noticeable differences in speed or reliability. Therefore, it can be said that all frameworks provide an equally impressive hot reloading developer experience.

Considering that Next.js and SvelteKit demonstrate comparable performance in generating dynamic pages, one could argue that for more dynamic websites such as e-commerce, video streaming, and social media, SvelteKit holds an edge due to its exceptional out-of-the-box developer experience.

Related Work

Due to the limited nature of formal research evaluating or comparing rendering frameworks, discussing the results in relation to such research is difficult. However, some parallels can be drawn to research that compares the UI frameworks that are utilized by the evaluated rendering frameworks. For instance, in related research React is analyzed to be the UI framework with the largest community and history [4], making it a safe choice for web development projects, akin to Next.js which utilizes React. In a similar vein, research on Svelte reveals the UI framework to have a relatively small community, but good performance, high quality tutorials and a good developer experience [4], akin to SvelteKit which is built on Svelte.

Such parallels are harder to draw for Astro, as it provides support for a wide range of UI frameworks. However, there are still some relations that can be discussed between the results and grey literature, such as community discussions and evaluations. For instance, one large community resource [11] recommend Astro for static applications over more dynamic web applications, which lines up with the results of our research project.

8 Conclusion and Future Work

This thesis has supplied a collection of questions, checklists, and specification for comparing and evaluating rendering frameworks, similar to what has been provided by others for evaluating front-end frameworks [4]. The recommended approach to evaluate a rendering framework involves implementing the specification detailed in appendix item C, along with exploring the subjects covered in the checklist in-depth questions found in appendix item D.

The web developer survey gave knowledge about what aspects of rendering frameworks are important, and what aspects are less important to the target group. Building the web applications in the controlled experiment gave knowledge about the performance, capabilities and developer experience of each framework.

Next.js establishes itself as a reliable and safe option, owing to its large community and recognition as an industry standard. Its usage of React, the leading UI framework, strengthens its position as a safe bet. However, as with all UI frameworks, developers' preferences towards them should be taken consideration during the decision-making process. SvelteKit stands as a noteworthy alternative to Next.js, demonstrating greater performance in dynamic content handling and offering a gentle learning curve with a high-quality tutorial. Astro excels in static content rendering and is particularly well-suited for blog-style content, where markdown is the given content format. It boasts developer-friendly features and an informative and practical tutorial; however, it may not be the ideal choice for projects with a strong focus on dynamic content. In the end, selecting the most appropriate rendering framework depends on the specific requirements and preferences of developers and the nature of the project at hand.

In conclusion, the selection of a rendering framework depends on the developers personal preferences, prior experience, and the specific needs of the project. Each web development project should be assessed individually. It is our hope that this thesis will assist in making informed decisions when choosing a framework.

Future Work

Although we believe this thesis tackles part of the knowledge gap, further work is needed to bridge the remaining gap.

- Additional, or alternative, frameworks could be included in the evaluation. For example, rendering frameworks like Remix, Nuxt and Docusaurus rank highly in terms of web developer interest in the State of JS survey [1].
- Evaluate Next.js utilizing the App Router (presently in beta) as opposed to the currently evaluated Pages Router. Additionally, the new beta documentation for Next.js could be evaluated instead of the current documentation that is currently set to be deprecated.
- Extend the size of the survey in terms of outreach to access a larger and more representative sample group of web developers.
- Broaden the investigation beyond exclusively SSR to also include CSR and SSG.
- Develop a more complex and diverse web application specification to enable a deeper examination of the frameworks abilities across a wider range of use cases.
- Conversely, develop a more specific and narrowed web application specification to thoroughly investigate a particular use case.

References

- [1] “The state of JS 2022”. (), [Online]. Available: <https://2022.stateofjs.com/en-US/> (visited on 01/26/2023).
- [2] “React 18: Overview | next.js”. (), [Online]. Available: <https://nextjs.org/docs/advanced-features/react-18/overview> (visited on 04/26/2023).
- [3] “Next.js by vercel - the react framework for the web”. (), [Online]. Available: <https://nextjs.org> (visited on 04/11/2023).
- [4] M. Bauer, *Hello Framework! A heuristic method for choosing front-end JavaScript frameworks*. 2021. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-452408> (visited on 01/24/2023).
- [5] “Angular”. (), [Online]. Available: <https://angular.io/> (visited on 04/11/2023).
- [6] “Vue.js - the progressive JavaScript framework | vue.js”. (), [Online]. Available: <https://vuejs.org/> (visited on 04/11/2023).
- [7] “Svelte • cybernetically enhanced web apps”. (), [Online]. Available: <https://svelte.dev/> (visited on 04/11/2023).
- [8] J. Johansson, *Create React App vs NextJS : A comparison of two ReactJS based web application frameworks*. 2021. [Online]. Available: <http://urn.kb.se/resolve?urn=urn:nbn:se:miun:diva-42390> (visited on 01/27/2023).
- [9] A. Świątkowski and K. Ścibior, “Comparative analysis of react, next and gatsby programming frameworks for creating SPA applications”, *Journal of Computer Sciences Institute*, vol. 24, pp. 224–227, Sep. 30, 2022, ISSN: 2544-0764. DOI: 10.35784/jcsi.2972. [Online]. Available: <https://ph.pollub.pl/index.php/jcsi/article/view/2972> (visited on 01/26/2023).
- [10] “Gatsby, the fastest frontend for the headless web”, Gatsby. (), [Online]. Available: <https://www.gatsbyjs.com/> (visited on 04/11/2023).
- [11] T. t3gg, *JavaScript frameworks in 2023*, Jan. 11, 2023. [Online]. Available: <https://www.youtube.com/watch?v=S7X6fLbdwlc> (visited on 04/11/2023).
- [12] Jason, *Svelte kit vs NextJS*, original-date: 2021-01-02T16:53:13Z, Apr. 8, 2023. [Online]. Available: <https://github.com/jasongitmail/svelte-vs-next> (visited on 04/11/2023).
- [13] “SvelteKit • web development, streamlined”. (), [Online]. Available: <https://kit.svelte.dev/> (visited on 04/12/2023).
- [14] “Astro”, Astro. (), [Online]. Available: <https://astro.build/> (visited on 04/12/2023).
- [15] C. Robson and K. McCartan, *Real world research: a resource for users of social research methods in applied settings*, Fourth Edition. Hoboken: Wiley, 2016, 533 pp., ISBN: 978-1-118-74523-6.
- [16] K. Nikolopoulou. “What is content validity? | definition & examples”, Scribbr. (Aug. 26, 2022), [Online]. Available: <https://www.scribbr.com/methodology/content-validity/> (visited on 04/27/2023).
- [17] P. Bhandari. “Construct validity | definition, types, & examples”, Scribbr. (Feb. 17, 2022), [Online]. Available: <https://www.scribbr.com/methodology/construct-validity/> (visited on 04/27/2023).

- [18] A. Johannessen, P. A. Tufte, and L. Christoffersen, *Introduktion till samhällsvetenskaplig metod*, Upplaga 2, trans. by B. Nilsson. Stockholm: Liber, 2020, OCLC: 1140383417, ISBN: 978-91-47-13204-1.
- [19] E. Van Susteren. “Satisficing: Learn to defeat the subtle menace in your survey data”, SurveyMonkey. (), [Online]. Available: <https://www.surveymonkey.com/curiosity/satisficing-learn-to-defeat-the-subtle-menace-in-your-survey-data/> (visited on 04/22/2023).
- [20] “Introduction to the DOM - web APIs | MDN”. (Apr. 15, 2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction (visited on 04/18/2023).
- [21] “SPA (single-page application) - MDN web docs glossary: Definitions of web-related terms | MDN”. (Feb. 21, 2023), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Glossary/SPA> (visited on 04/18/2023).
- [22] “MPAs vs. SPAs”, Astro Documentation. (), [Online]. Available: <https://docs.astro.build/en/concepts/mpa-vs-spa/> (visited on 04/18/2023).
- [23] pandaquests. “SSR vs. CSR”, Medium. (Feb. 16, 2023), [Online]. Available: <https://pandaquests.medium.com/ssr-vs-csr-5ef4b8a10498> (visited on 04/18/2023).
- [24] “What is SvelteKit? • svelte tutorial”. (), [Online]. Available: <https://learn.svelte.dev> (visited on 04/20/2023).
- [25] “Progressive enhancement - MDN web docs glossary: Definitions of web-related terms | MDN”. (Feb. 21, 2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement (visited on 04/18/2023).
- [26] “Getting started | markdown guide”. (), [Online]. Available: <https://www.markdownguide.org/getting-started/> (visited on 04/18/2023).
- [27] “JSON - JavaScript | MDN”. (Apr. 3, 2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON (visited on 04/25/2023).
- [28] “JavaScript | MDN”. (May 1, 2023), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (visited on 05/15/2023).
- [29] “JavaScript with syntax for types.” (), [Online]. Available: <https://www.typescriptlang.org/> (visited on 05/15/2023).
- [30] “Tailwind CSS - rapidly build modern websites without ever leaving your HTML.” (Nov. 15, 2020), [Online]. Available: <https://tailwindcss.com/> (visited on 05/15/2023).
- [31] “About npm | npm docs”. (), [Online]. Available: <https://docs.npmjs.com/about-npm> (visited on 05/04/2023).
- [32] “HTML: HyperText markup language | MDN”. (May 10, 2023), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/HTML> (visited on 05/13/2023).
- [33] “CSS: Cascading style sheets | MDN”. (Apr. 16, 2023), [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/CSS> (visited on 05/13/2023).

- [34] *Next.js*, original-date: 2016-10-05T23:32:51Z, Apr. 24, 2023. [Online]. Available: <https://github.com/vercel/next.js> (visited on 04/24/2023).
- [35] “Getting started | next.js”. (), [Online]. Available: <https://nextjs.org/docs/getting-started> (visited on 04/26/2023).
- [36] “Getting started | next.js”. (), [Online]. Available: <https://beta.nextjs.org/docs> (visited on 04/26/2023).
- [37] “Data fetching: getServerSideProps | next.js”. (), [Online]. Available: <https://nextjs.org/docs/basic-features/data-fetching/get-server-side-props> (visited on 04/27/2023).
- [38] “Data fetching: getStaticProps | next.js”. (), [Online]. Available: <https://nextjs.org/docs/basic-features/data-fetching/get-static-props> (visited on 04/27/2023).
- [39] “Release 1.0.0 · vercel/next.js”, GitHub. (), [Online]. Available: <https://github.com/vercel/next.js/releases/tag/1.0.0> (visited on 04/28/2023).
- [40] “Pulse · vercel/next.js”. (), [Online]. Available: <https://github.com/vercel/next.js/pulse/monthly> (visited on 04/28/2023).
- [41] “Install tailwind CSS with next.js - tailwind CSS”. (), [Online]. Available: <https://tailwindcss.com/docs/guides/nextjs> (visited on 05/06/2023).
- [42] *SvelteKit*, original-date: 2020-10-15T14:00:23Z, Apr. 24, 2023. [Online]. Available: <https://github.com/sveltejs/kit> (visited on 04/24/2023).
- [43] “Introduction • docs • SvelteKit”. (), [Online]. Available: <https://kit.svelte.dev/docs/introduction> (visited on 04/26/2023).
- [44] “Page options • docs • SvelteKit”. (), [Online]. Available: <https://kit.svelte.dev/docs/page-options> (visited on 04/27/2023).
- [45] “Release @sveltejs/kit@1.0.0 · sveltejs/kit”, GitHub. (), [Online]. Available: <https://github.com/sveltejs/kit/releases/tag/%40sveltejs%2Fkit%401.0.0> (visited on 04/28/2023).
- [46] “Pulse · sveltejs/kit”. (), [Online]. Available: <https://github.com/sveltejs/kit/pulse/monthly> (visited on 04/28/2023).
- [47] “Install tailwind CSS with SvelteKit - tailwind CSS”. (), [Online]. Available: <https://tailwindcss.com/docs/guides/sveltekit> (visited on 05/06/2023).
- [48] *Withastro/astro*, original-date: 2021-03-15T17:19:47Z, Apr. 24, 2023. [Online]. Available: <https://github.com/withastro/astro> (visited on 04/24/2023).
- [49] “Getting started”, Astro Documentation. (), [Online]. Available: <https://docs.astro.build/en/getting-started/> (visited on 04/26/2023).
- [50] “Build your first astro blog”, Astro Documentation. (), [Online]. Available: <https://docs.astro.build/en/tutorial/0-introduction/> (visited on 04/27/2023).
- [51] “Framework components”, Astro Documentation. (), [Online]. Available: <https://docs.astro.build/en/core-concepts/framework-components/> (visited on 04/26/2023).
- [52] “Routing”, Astro Documentation. (), [Online]. Available: <https://docs.astro.build/en/core-concepts/routing/> (visited on 04/27/2023).

- [53] “Release astro@1.0.0 · withastro/astro”, GitHub. (), [Online]. Available: <https://github.com/withastro/astro/releases/tag/astro%401.0.0> (visited on 04/28/2023).
- [54] “Pulse · withastro/astro”. (), [Online]. Available: <https://github.com/withastro/astro/pulse/monthly> (visited on 04/28/2023).
- [55] “tRPC - move fast and break nothing. end-to-end typesafe APIs made easy. | tRPC”. (Sep. 19, 2022), [Online]. Available: <https://trpc.io/> (visited on 05/10/2023).
- [56] “NextAuth.js”. (), [Online]. Available: <https://next-auth.js.org> (visited on 05/10/2023).
- [57] “Window: DOMContentLoaded event - web APIs | MDN”. (Apr. 8, 2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/DOMContentLoaded_event (visited on 04/25/2023).
- [58] “Window: Load event - web APIs | MDN”. (Apr. 8, 2023), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event (visited on 04/25/2023).

Glossary

Terms

DOMContentLoaded event The DOMContentLoaded event is triggered once the initial HTML document has fully loaded and parsed, without waiting for stylesheets, images, and subframes to finish loading [57]. 19, 22, 25, 29–33, 44

load event The load event is fired when the whole page has loaded, including all dependent resources such as stylesheets, scripts, iframes, and images [58]. 19, 22, 25, 29–34, 44

object-relational mapping (ORM) A programming technique that enables developers to query and manipulate data from a database using object-oriented techniques. 44

pull request A collaboration tool in version control systems for proposing and reviewing code changes before merging them into the main version of the system. 44

user interface (UI) The means by which the user and a computer system interact, in particular the use of input devices and software. 44

9 Appendices

A Google Forms Survey

Survey on Modern JavaScript Rendering Frameworks

This survey is part of a Bachelor Thesis in Computer Science at Linnæus University in Sweden. We are making a comparative evaluation of rendering/meta frameworks, and are gathering information and opinions from web developers to support the comparison. Your response is important to us!

By proceeding to take this survey, you consent to our collection of your anonymous data. This survey does not collect your email address, and all responses are anonymous. The collected data will only be used by us for the purpose of the Bachelor Thesis, and will not be made available to any third parties.

The survey is expected to take around 2-4 minutes. Thank you for your participation!

Contact details:
Denni Wernersson | dw222fy@student.lnu.se
Viktor Sjölund | vs222qj@student.lnu.se

Switch account

Not shared

About you

How old are you, in years? *

- 18-24 years old
- 25-34 years old
- 35-44 years old
- 45-54 years old
- 55-64 years old
- 65 or older

How many years have you spent in formal web development education? *

- None at all
- Less than one year
- 1 to 2 years
- 3 to 5 years
- More than 5 years

How long have you been working in web development, in years? *

- None at all
- Less than one year
- 1 to 2 years
- 3 to 5 years
- 6 to 10 years
- More than 10 years

Survey

This section of the survey is in large part sourced from the [State of JS - 2022 survey](#). This enables us to draw direct comparisons to that survey and its results.

Rendering Frameworks

Frameworks focused on rendering and serving your application

Next.js *

- 🚫 Never heard of it/Not sure what it is
- Heard of it > Would like to learn
- 🚫 Heard of it > Not interested
- 🟡 Used it > Would use again
- 🟡 Used it > Would not use again

SvelteKit *

- 🚫 Never heard of it/Not sure what it is
- Heard of it > Would like to learn
- 🚫 Heard of it > Not interested
- 🟡 Used it > Would use again
- 🟡 Used it > Would not use again

Astro *

- 🚫 Never heard of it/Not sure what it is
- Heard of it > Would like to learn
- 🚫 Heard of it > Not interested
- 🟡 Used it > Would use again
- 🟡 Used it > Would not use again

Application Patterns

Which of the following architecture and rendering patterns have you used in the last year?

<h4>SPA</h4> <p>Apps that run entirely in the browser</p> <input type="checkbox"/> Single Page Application (SPA)	<h4>MPA</h4> <p>Apps that run entirely on the server, with minimal client-side dynamic behaviour</p> <input type="checkbox"/> Multi-Page Application (MPA)
<h4>SSG</h4> <p>Pre-rendered static content, with or without a client-side dynamic element</p> <input type="checkbox"/> Static Site Generation (SSG)	<h4>SSR</h4> <p>Dynamically rendering HTML content on the server before rehydrating it on the client</p> <input type="checkbox"/> Server-Side Rendering (SSR)
<h4>Partial Hydration</h4> <p>Only hydrating some of your components on the client (e.g. React Server Components)</p> <input type="checkbox"/> Partial Hydration	<h4>Islands Architecture</h4> <p>Isolated islands of dynamic behavior with multiple entry points in an otherwise static site (Astro, Eleventy)</p> <input type="checkbox"/> Islands Architecture
<h4>Progressive Enhancement</h4> <p>Making sure an app is functional even without JavaScript</p> <input type="checkbox"/> Progressive Enhancement	<h4>Edge Rendering</h4> <p>Altering rendered HTML at the edge before sending it on to the client</p> <input type="checkbox"/> Edge Rendering

Opinions


Which of the following aspects of a rendering framework do you find most important?

Choose up to three aspects

- File-based routing
- Dev Hot Reload (i.e auto reload on file save)
- Route Pre-Fetching
- Learning Resources (i.e tutorials)
- Ability to choose UI framework
- Image Tag Optimization
- High Quality Documentation
- Large Community

B Survey Missive Letters

Slack #wp channel

 **Denni Wernersson** 15:02
Hej allesamman! / Hi everyone!

My partner Viktor Sjölund and I are working on a Bachelor Thesis making a comparative evaluation of JavaScript Rendering Frameworks (think NextJS) to help web developers choose the best framework for the situation. If you consider yourself as having any experience or interest in web development we would greatly appreciate you taking the time to answer a short survey! 🙏

👉 <https://forms.gle/qVa7xXh8ujdUfJ8U6> 👉

Survey on JavaScript Rendering Frameworks
Introduction:
The aim of this survey is to collect web developers' experience and opinions of certain aspects of modern JavaScript Rendering Frameworks, in order to support the comparative evaluation that we make in the thesis.

Your contribution:
We are interested in your opinions and experiences with rendering frameworks and software patterns. The survey should only take between 2-4 minutes to complete.

Privacy and Ethics:
This survey is anonymous and the information collected will be used the purposes of the Bachelor Thesis only, no third parties will have access to your response. The results will be stored digitally and protected from unauthorized access. The survey is entirely voluntary.

By answering the survey, you consent to us using your non-identifiable response in our thesis work. If you have any questions regarding the survey, please email us at:
Denni Wernersson (dw222fy@student.lnu.se)
Viktor Sjölund (vs222qj@student.lnu.se)

reddit.com/r/webdev

Comparing JavaScript Rendering Frameworks **Question** self.webdev
Submitted 10 days ago by [Totally_Not_A_Moose](#)

Hello everyone on [/r/webdev](#)^[1]!

Our names are Denni and Viktor and we are two students studying web development at Linnaeus University in Sweden. Currently, we are working on our Bachelor Thesis, where we are making a comparative evaluation of JavaScript Rendering Frameworks (think NextJS) to help web developers choose the best framework for their project.

We would greatly appreciate it if you, whether you are a student, self-taught, or have professional experience, could take the time to fill out our short survey. It will take between 2-4 minutes to complete, and all opinions matter to us.

Here is the link to the survey: <https://forms.gle/qVa7xXh8ujdUfJ8U6>^[2]

The purpose of the survey is to collect your opinions and experiences with modern JavaScript Rendering Frameworks and software patterns. We want your help to support the comparative evaluation in our Bachelor Thesis.

We want to emphasize that the survey is completely anonymous, and the information collected will only be used for our Bachelor Thesis. The responses will be stored digitally and protected against unauthorized access.

We are extremely grateful for your time and for your participation in our survey, and if you have any questions please feel free to contact us via email at dw222fy@student.lnu.se^[3] or vs222qj@student.lnu.se^[4].

Thank you in advance!

[comment](#) [source](#) [edit](#) [share](#) [save](#) [unhide](#) [delete](#) [nsfw](#) [spoiler](#) [flair](#) [crosspost](#) [hide all child comments](#)

C Application Specifications

Navigation Bar:

- Links to Home, Blog & Big Data page
- Home is shown at root path(/)

Landing Page (/):

- Logo
- Framework Name
- Title

Blog List (/blog):

- Lists all blog posts
- Blog posts are clickable to route to that blog/[id] page
- Each listed blog post must show:
 - Title
 - Description
 - Date

Blog Page (/blog/[id]):

- Renders the blog post belonging to the current route
- Must show:
 - Title
 - Date
 - Content

Big Data (/big-data):

- Lists all big-data datasets
- Each listed dataset must show:
 - Name
 - Description

Big Data People (/big-data/people):

- Lists people in the dataset
- Each listed person must show:
 - Full name
 - Item Index

Big Data Person (/big-data/people/[id]):

- Renders the person belonging to the current route
- Must show:
 - Full Name
 - Email
 - Gender

All Pages

- All pages are scrollable on vertical overflow
- All blog data & metadata is dynamically sourced from local markdown files
- All "big-data"-data is dynamically sourced from local JSON files

D Checklist and In-depth Questions

Checklist

- **Build Size:** The size of the compiled web application code
- **Loading Speeds:** The loading speeds of the different pages
- **Community:** GitHub Stars & Contributors to the framework
- **Documentation:** Is the documentation sufficient and of high quality?
- **Tutorials:** Are there any tutorials available and what do they contain?
- **UI Framework:** What UI framework does the rendering framework use?
- **Rendering Methods:** Are CSR, SSR and SSG supported?

In-depth questions

- Is the framework mature?
- Is it being actively developed?
- Is it easy to work with?
- Is it fast to work with?

E The Web Applications

The open-source web applications are version-controlled on separate GitHub repositories.

Next.js

<https://github.com/ViktorSjolund/nextjs-benchmark>

SvelteKit

<https://github.com/ViktorSjolund/sveltekit-benchmark>

Astro

<https://github.com/ViktorSjolund/astro-benchmark>