Bachelor Degree Project

A Comparative Analysis of Next.js, SvelteKit, and Astro for E-commerce Web Development

Author 1: Erik Kroon Celander
Author 2: Amanda Möllestål
Supervisor: Arianit Kurti
Semester: VT 2024
Subject: Computer Science

Abstract

This thesis compares three modern JavaScript frameworks—Next.js, SvelteKit, and Astro—for e-commerce web development, focusing on how different rendering strategies (Static Site Generation, Server-Side Rendering, and Client-Side Rendering) affect performance metrics like load times, interactivity, and resource efficiency. Prototype e-commerce applications were developed using each framework, and performance testing was conducted to measure First Contentful Paint (FCP), Largest Contentful Paint (LCP), Interaction to Next Paint (INP), page size, and SEO-friendliness. Results show that Astro and SvelteKit consistently outperforms Next.js in load times and resource efficiency. However, the differences in performance metrics are not drastically large. All three frameworks are capable of delivering high-quality e-commerce applications. Therefore, decision-makers should also consider factors such as developer experience, community support, and specific project requirements when selecting a framework.

Keywords: JavaScript frameworks, e-commerce, performance optimization, rendering strategies

Preface

We would like to express our sincere gratitude to our supervisor, Arianit Kurti. His guidance, support, and insightful feedback have been invaluable throughout the entire process of conducting this thesis. We would also like to extend our gratitude to our university programming teacher, Mats Loock, and our program coordinator, Johan Leitet, for all the lessons, laughs, and encouragement.

Contents

1 Introduction

This Bachelor's thesis in Computer Science focuses on a comparative analysis of modern JavaScript frameworks—Next.js, SvelteKit, and Astro—in the context of e-commerce web development. Choosing an appropriate framework is critical as it significantly impacts the development process, performance, and user experience. With the increasing complexity of e-commerce applications, which demand fast page loads, rich interactivity, and search engine optimization (SEO), the choice of rendering strategy and its implementation is crucial for developers.

Next.js, SvelteKit, and Astro are prominent in this space, each supporting rendering strategies such as Static Site Generation (SSG), Server-Side Rendering (SSR), and Client-Side Rendering (CSR) [1]. However, the impact of these frameworks on e-commerce application performance requires further exploration and comparative analysis.

To address this knowledge gap, this thesis conducts a controlled experiment by developing a prototype e-commerce application with each framework. By maintaining consistent implementation and using the same database, this research aims to isolate the frameworks' impact on key performance metrics such as load times, interactivity, SEO, and resource efficiency. The findings will provide empirical data and insights, aiding developers and decision-makers in selecting the most suitable JavaScript frameworks and rendering strategies for e-commerce projects.

1.1 Background

As the e-commerce industry continues to grow, businesses are placing increasing emphasis on creating high-performing, user-friendly online shopping experiences. The choice of web development technologies, particularly JavaScript frameworks and rendering strategies, plays a crucial role in determining the performance and user experience of e-commerce applications [2].

While the introduction of this thesis focuses on the specific comparison of Next.js, SvelteKit, and Astro in the context of e-commerce web development, it is essential to understand the broader landscape of web development technologies and their impact on application performance. Web developers have a wide array of JavaScript frameworks and libraries at their disposal, each with its own strengths and weaknesses. The State of JavaScript survey of 2022 [1] reports that 82% of the respondents utilized a front-end framework such as React in the past year, and 48% employed a rendering framework like Next.js. This highlights the widespread adoption of frameworks in modern web development.

Application, meta, or full-stack frameworks, commonly referred to as rendering frameworks, are intended to manage both front-end and back-end functionalities of a web application. They are responsible for rendering and

serving the application, offering a comprehensive web development solution. Rendering frameworks enable web developers to build robust and scalable applications, however, selecting which rendering framework is suitable for a particular use case can be difficult due to the unique benefits and drawbacks each framework offers. Factors such as project requirements, developer expertise, and performance considerations must be taken into account when making this decision.

One of the key aspects that influence the performance of e-commerce applications is the rendering strategy employed. Server-Side Rendering (SSR), Client-Side Rendering (CSR), and Static Site Generation (SSG) are the three main rendering strategies, each with its own trade-offs in terms of performance, SEO, and user experience. SSR generates the complete HTML content on the server and sends it to the client's browser, resulting in faster initial page loads and better SEO. CSR relies on JavaScript to render the application in the user's browser, enabling faster subsequent page loads and more dynamic, interactive experiences. SSG generates static HTML, CSS, and JavaScript files at build time, which can be served directly from a content delivery network (CDN), resulting in fast page loads and excellent SEO [3].

It is important to note that Next.js, SvelteKit, and Astro are all versatile frameworks that support multiple rendering strategies, including SSR, CSR, and SSG. This flexibility allows developers to choose the most appropriate strategy for their specific needs, and even combine strategies within a single application. Therefore, our study will evaluate how each of these frameworks performs under different rendering strategies to provide a comprehensive comparison.

The choice of rendering strategy and its implementation through a specific JavaScript framework can have a significant impact on the performance of e-commerce applications. However, there is a lack of comprehensive, comparative studies that evaluate the performance of modern JavaScript frameworks and the previously mentioned rendering strategies in the context of e-commerce applications.

To address the need for comprehensive comparative studies, this research will evaluate prototype e-commerce applications built with Next.js, SvelteKit, and Astro under different rendering strategies. By examining these frameworks through a controlled experiment, the study seeks to offer practical insights and empirical data that will help developers and decision-makers make informed choices about JavaScript frameworks and rendering strategies in the e-commerce sector.

1.2 Motivation

E-commerce websites play a pivotal role in the global economy. According to BuiltWith [8], over 26 million websites were utilizing e-commerce technologies as of April this year. In 2021, online purchases accounted for

nearly 20% of global retail sales, with projections suggesting this could rise to almost 25% by 2025 [9], [10].

The performance of e-commerce websites significantly impacts user experience and conversion rates. An analysis by Portent [11] of 100 million page views from 20 e-commerce sites showed that a site loading in 1 second achieves a conversion rate 2.5 times greater than one loading in 5 seconds. Similarly, an Unbounce survey of 750 consumers found that nearly 70% admit page speed influences their likelihood to make a purchase [12]. Real-world examples, such as Vodafone's A/B test, have shown that improving page load speed can lead to substantial increases in lead-to-visit rate, cart-to-visit rate, and overall sales [13].

In addition to affecting user experience and conversion rates, page load speed is now a criterion for Google's search engine rankings [14], [15]. This highlights the necessity of optimizing e-commerce websites for performance. As the demand for high-quality e-commerce experiences grows, developers must balance server-side rendering for fast page loads and optimal SEO with client-side rendering for rich interactivity.

1.3 Related Work

There is substantial research comparing JavaScript frameworks and rendering techniques. One study [4] provides a broad overview of modern frameworks like React, Next.js, Svelte, Qwik, Astro, and HTMX, focusing on their benefits for user experience (UX), performance, and developer experience (DX). Key benefits identified include smaller runtime sizes, efficient rendering approaches, and improved DX through features like fine-grained reactivity and compiler use. However, this study does not provide empirical data or a specific focus on e-commerce, limiting its direct applicability to our research context.

A thesis [7] examines Next.js, SvelteKit, and Astro, providing insights into their general benefits and trade-offs in web development. It includes a controlled experiment and a survey, offering empirical data on general web performance. Nonetheless, it lacks a targeted analysis of e-commerce applications, which have unique requirements such as handling large product catalogs and optimizing for conversion rates. The study highlights that Next.js is robust for dynamic content, SvelteKit excels in performance for dynamic pages, and Astro is ideal for static content.

There is research [5] that offers a comprehensive performance comparison of React, Vue, Next.js, and Nuxt using metrics like Load Time, First Byte, Start Render, Speed Index, CLS, TBT, and TTI. The findings indicate that Vue consistently shows the best performance in various metrics, while Next.js performs well in rendering content quickly. However, this study does not address e-commerce-specific needs or the frameworks focused on in this thesis. The emphasis is on Vue's efficiency in responsive and efficient web

applications, but its general approach does not align directly with e-commerce requirements.

Some research [6] has been done that evaluate SSR and CSR techniques using Google Lighthouse and Google DevTools Performance Insights, finding that SSR generally provides better performance for content-rich web pages, especially under slower network conditions. This work does not examine specific frameworks or e-commerce applications in detail, limiting its applicability to our specific research focus. The study highlights SSR's advantage in resource-constrained environments, relevant but not comprehensive for e-commerce.

One paper [2] compares Angular, React, Vue, Svelte, and Blazor, providing valuable insights into rendering strategies. It finds significant performance differences, particularly in updating existing content, with Svelte consistently performing well across various benchmarks. While the study offers broad insights, it does not focus on the unique requirements of e-commerce applications, such as the need for optimized SEO and fast load times for large product catalogues. The emphasis on frameworks' rendering strategies is valuable but not directly aligned with e-commerce-specific metrics.

1.4 Problem Formulation and Research Questions

Despite our research focusing on e-commerce, the primary issue addressed is a core Computer Science problem: optimizing website performance. Existing studies have explored the performance of various JavaScript frameworks and rendering strategies [5], [7]. However, these studies either focus on other frameworks or do not specifically address the unique requirements of e-commerce applications. This thesis aims to fill this gap by examining how rendering strategies implemented in modern JavaScript frameworks impact e-commerce web applications. Our objectives are:

Focused Comparative Analysis: Unlike broader comparisons in existing research, this work specifically targets e-commerce web applications, providing insights into performance and SEO impacts relevant to this domain. Inclusion of Popular Frameworks: By analyzing Next.js, SvelteKit, and Astro, this study ensures the findings are applicable to widely used frameworks in the industry [1], making the results more relevant to real-world web development projects.

Quantitative Performance Analysis: This research extends current studies by conducting a rigorous quantitative analysis to provide concrete recommendations on the most performant framework and rendering strategy for e-commerce applications.

Addressing Rapid Technological Evolution: By focusing on the latest frameworks and rendering strategies, this thesis contributes current knowledge

to the field, recognizing the rapid pace of change noted as a limitation in previous studies.

The need for this research is underscored by the increasing complexity of web applications, particularly e-commerce platforms, which require an in-depth understanding of rendering strategies to optimize performance, user experience, and search engine visibility. This thesis aims to bridge the knowledge gap, contributing not only to academic discourse but also serving as a practical guide for developers. This exploration ensures the recommendations remain relevant and actionable amidst rapid technological advancements.

To address the problem and fill the knowledge gap, two key research questions guide this comparative analysis:

1. How do Next.js, SvelteKit, and Astro compare in terms of load time metrics (First Contentful Paint, Largest Contentful Paint, and Interaction to Next Paint) for different rendering strategies (Static Site Generation, Server-Side Rendering, and Client-Side Rendering) in an e-commerce application context?

2. How do Next.js, SvelteKit, and Astro compare in terms of resource efficiency relative to page size in an e-commerce application context?

1.5 Statement of Contribution

This thesis makes several significant contributions to the field of web development and e-commerce applications:

Empirical Data: Provides a comprehensive empirical analysis of the performance of Next.js, SvelteKit, and Astro frameworks under different rendering strategies, specifically in the context of e-commerce.

Targeted Insights: Offers targeted insights and practical recommendations for developers and decision-makers on selecting the most suitable JavaScript frameworks and rendering strategies for e-commerce projects.

Performance Metrics: Introduces a detailed evaluation of key performance metrics such as load times, interactivity, SEO, and resource efficiency, relevant to e-commerce applications.

Framework Evaluation: Enhances understanding of how modern JavaScript frameworks perform in real-world e-commerce scenarios, filling a gap in existing research.

1.6 Scope/Limitation

This thesis focuses on comparing the performance of Next.js, SvelteKit, and Astro within the context of e-commerce web applications. The scope has been limited to these three frameworks due to their popularity and representativeness in modern web development.

The controlled experiment involves developing prototype applications using each selected framework. These prototypes include core functionalities and a mix of rendering strategies representative of real-world e-commerce websites. However, these prototypes may not encompass all possible features and complexities of large-scale e-commerce platforms.

The performance evaluation focuses on specific metrics: First Contentful Paint, Largest Contentful Paint, Interaction to Next Paint, page size, and SEO-friendliness. While these metrics are widely accepted and relevant to web performance [16], they do not cover all aspects of web application performance comprehensively.

A notable limitation of this research is the sample size of the controlled experiment. Due to time constraints, a single prototype application was developed for each framework. Although these prototypes are designed to be representative of real-world e-commerce websites, a larger sample size could provide more robust and generalizable results.

Additionally, the findings may be influenced by the specific tools and technologies used in the experimental setup, such as the database, hosting environment, and performance measurement tools. Despite efforts to maintain consistency and adhere to industry best practices, different tools and technologies could potentially yield slightly different results.

1.7 Target group

The primary target group for this thesis comprises web developers and software engineers involved in building e-commerce applications using modern JavaScript frameworks. This includes:

- Front-End Developers: Responsible for implementing user interfaces and optimizing web performance.
- Full-Stack Developers: Work on both client-side and server-side aspects of e-commerce applications.
- Software Architects: Make decisions regarding the choice of frameworks and rendering strategies for e-commerce projects.
- Performance Engineers: Focus on optimizing performance and ensuring a seamless user experience.

Additionally, the research may interest the following secondary target groups:

- Researchers and Academics: Studying web performance, JavaScript frameworks, and e-commerce technologies.
- E-Commerce Business Owners and Managers: Involved in the technical decision-making process.
- Students and Aspiring Web Developers: Learning about modern JavaScript frameworks and their application in e-commerce web development.

1.8 Outline

This thesis is organized as follows. Chapter 1 introduces the background, motivation, problem formulation, and research questions. Chapter 2 covers the theoretical background on e-commerce web development, JavaScript frameworks, and rendering strategies. Chapter 3 details the research methodology, including the controlled experiment and chosen metrics. Chapter 4 describes the implementation, covering prototype design, database setup, and performance testing. Chapter 5 presents the results and analysis of performance tests on the frameworks. Chapter 6 discusses the findings, implications, and limitations. Chapter 7 suggests areas for future research.

2 Theoretical Background

2.1 Introduction to E-commerce Web Development

E-commerce has become integral to the global economy [9]. As consumers increasingly turn to online shopping, businesses invest significantly in developing high-quality e-commerce websites. Web development is crucial for creating user-friendly, secure, and performant e-commerce platforms, which contribute to the success of online businesses.

2.1.1 Importance of Web Development in E-commerce

Web development forms the foundation of e-commerce applications. A well-designed e-commerce application can enhance user experience, increase customer satisfaction, boost conversion rates, and drive greater revenue [29], [30]. Key aspects include:

Responsive Design: Ensuring the application is accessible and visually appealing across various devices and screen sizes [14], [16].

Performance Optimization: Reducing page load times and improving overall website performance to lower bounce rates and enhance user engagement [30], [31].

Security: Implementing robust measures to protect customer data and maintain trust [32], [33].

Scalability: Designing architecture to handle increased traffic and growth [34].

2.1.2 Key Challenges in E-commerce Web Development

Despite technological advancements, e-commerce web development faces several challenges:

Performance and Bandwidth: E-commerce applications are network intensive. Bandwidth limitations, especially in the “last mile,” and heavy media usage can slow page loading, affecting user experience.

User Experience (UX): Designing intuitive and user-friendly interfaces is challenging due to diverse customer preferences. Key issues include navigation ease, page loading speed, and multimedia use for visual appeal.

Search Engine Optimization (SEO): SEO is critical for visibility. Load times significantly influence search result rankings [3], [15].

2.2 Overview of Modern JavaScript Frameworks

JavaScript frameworks have transformed web development, providing powerful tools to build complex, interactive applications [1]. These frameworks have evolved, addressing modern web development challenges and offering extensive features [1], [2].

2.2.1 Evolution of JavaScript and JavaScript frameworks

JavaScript, originating in the early 1990s, has evolved from a simple client-side scripting language to a comprehensive full-stack programming language. ECMAScript 6 (ES6), introduced in 2015, added features like arrow functions, template literals, and modules, enhancing functionality and maintainability [35].

Node.js, introduced in 2009, enabled server-side JavaScript, facilitating full-stack development. Its event-driven, non-blocking I/O model supports high-performance server applications. The Node Package Manager (npm) enriches the ecosystem with numerous packages and libraries [35].

2.2.2 Types of Frameworks

JavaScript frameworks are categorized into front-end, back-end, and full-stack/rendering frameworks.

Front-End Frameworks

- React: Emphasizes component-based architecture and virtual DOM for optimized performance. Developed by Facebook, React introduced a virtual DOM and component-based architecture, revolutionizing dynamic user interface development [36].
- Vue.js: Offers simplicity and flexibility, suitable for projects of varying sizes. Vue.js is popular for its gentle learning curve and extensive documentation [37].
- Angular: Provides a robust framework for SPAs, using TypeScript for type safety and MVC architecture [38]. Backed by Google, Angular offers a comprehensive framework for building dynamic web applications.

Back-End Frameworks

- Express.js: A minimal and flexible Node.js framework, popular for developing RESTful APIs and SPAs [39].
- Koa: Developed by Express.js creators, Koa uses async functions for improved error handling and a modern server-side approach [40].

Full-Stack/Rendering Frameworks

Full-stack frameworks, also known as rendering-frameworks and meta-frameworks, handle both client-side and server-side rendering, providing a comprehensive solution for building full web applications.

Next.js: Next.js is a React-based framework developed by Vercel, designed to streamline the creation of server-rendered, statically generated, and hybrid web applications. Key features of Next.js include automatic code splitting, server-side rendering, static site generation, and a built-in API router. These features collectively enhance performance and SEO, enabling developers to create efficient and scalable web applications.

Next.js supports both server-side rendering and static site generation, allowing developers to pre-render pages at build time or request time. This

flexibility makes it suitable for a variety of use cases, from dynamic applications to static websites [41], [42].

SvelteKit: SvelteKit is a comprehensive framework built around the Svelte component framework, designed to enhance the development of web applications. Svelte, known for its compiler-based approach, enables developers to write concise, efficient code that compiles to highly optimized vanilla JavaScript. This compilation process results in fast, lightweight applications with minimal runtime overhead [43].

SvelteKit leverages Svelte's strengths by providing a full-stack solution that includes server-side rendering, client-side hydration, and offline support. This combination allows for highly performant and SEO-friendly applications. The framework simplifies many complex tasks such as routing, state management, and build optimizations, making it an attractive choice for developers who prioritize efficiency and performance [43].

Astro: Astro is a modern static site builder that combines the simplicity of static HTML with the interactivity of JavaScript. This tool allows developers to build content-focused websites using a component-based architecture while integrating JavaScript frameworks like React, Vue, or Svelte as needed [44], [45].

One of Astro's standout features is its "Islands Architecture," which enables developers to selectively hydrate interactive components on the client side. This approach minimizes the amount of JavaScript sent to the browser, leading to faster page loads and improved performance, particularly for content-heavy websites. Astro supports Markdown out of the box, making it ideal for blogs, documentation sites, and other content-driven projects [44], [46].

2.2.3 Popularity and Adoption of Different Frameworks

The popularity of JavaScript frameworks has surged. According to the State of JavaScript 2022 survey [1], React is the most widely used front-end framework, followed by Vue.js and Angular. Next.js leads among full-stack frameworks, with significant interest in Astro and SvelteKit [1].

Framework choice depends on project requirements, team expertise, performance needs, and ecosystem support. Each framework has strengths and weaknesses, and the choice should be based on specific project needs [7], [47], [48].

2.3 Rendering Strategies in Web Development

Rendering strategies play a crucial role in determining how web pages are delivered to users. The choice of rendering strategy can significantly impact the performance, user experience, and search engine visibility of a website. In this section, we will explore the three main rendering strategies: Client-Side

Rendering, Server-Side Rendering, and Static Site Generation, as well as hybrid approaches that combine these strategies.

2.3.1 Client-Side Rendering (CSR)

Client-Side Rendering is a strategy where the initial HTML sent to the browser contains minimal content, and the majority of the page is rendered dynamically using JavaScript on the client-side. In CSR, the server sends a minimal HTML file and the browser downloads the necessary JavaScript code to render the page.

Advantages of CSR include faster subsequent page loads and a more interactive user experience. However, CSR can lead to slower initial load times because the browser needs to download and execute the JavaScript code before rendering the page. Additionally, CSR can impact SEO, as search engine crawlers may have difficulty indexing the content of the page [3].

2.3.2 Server-Side Rendering (SSR)

Server-Side Rendering is a strategy where the server generates and sends a fully rendered HTML page to the browser on each request. With SSR, the server is responsible for fetching data, rendering the HTML, and sending it to the client.

The main advantage of SSR is improved performance, as the browser receives a fully rendered page, reducing the time to first contentful paint (TTFCP). SSR also benefits SEO because search engine crawlers can easily index the content of the page. However, SSR can increase the load on the server and may result in slower page loads if the server is under heavy traffic [3], [49], [50].

2.3.3 Static Site Generation (SSG)

Static Site Generation is a strategy where the server generates a fully rendered HTML page at build time, which is then served to the browser as a static file. With SSG, the HTML, CSS, and JavaScript files are pre-generated and can be served directly from a content delivery network (CDN).

The main advantage of SSG is fast page loads, as the content is pre-generated and can be served instantly. SSG also provides excellent SEO, as the content is fully accessible to search engine crawlers. However, SSG may not be suitable for websites with frequently changing or dynamic content, as the static files need to be regenerated each time the content is updated [3], [50], [51].

2.3.4 Hybrid Approaches and Their Implications

Hybrid approaches combine multiple rendering strategies to leverage the benefits of each. For example, a website may use SSR for the initial page load

to ensure fast TTFCP and good SEO, while subsequent page navigations are handled by CSR for a more interactive experience.

Another hybrid approach is Incremental Static Regeneration (ISR), introduced by frameworks like Next.js. ISR allows developers to update static pages incrementally without rebuilding the entire site, providing a balance between the performance of SSG and the flexibility of SSR [3], [51].

Hybrid approaches offer the best of both worlds but can also introduce complexity in terms of development and deployment. Developers need to carefully consider the trade-offs and choose the most appropriate approach based on the specific requirements of their project [3], [51].

3 Method

3.1 Research Project

This study utilized a controlled experiment to develop and evaluate prototype e-commerce applications built using the selected frameworks—Next.js, SvelteKit, and Astro—within a structured and thorough research process.

The project followed the design science methodology [17], [18], which involved the following steps:

Problem Identification and Motivation: The need for selecting the most suitable JavaScript framework and rendering strategy for e-commerce applications was identified. Key factors considered included performance, resource efficiency, and SEO-friendliness.

Objectives Definition: The research objectives were defined to compare the performance of Next.js, SvelteKit, and Astro in terms of load time metrics and resource efficiency. Additionally, the goal was to provide empirical data and insights to support informed decision-making for developers.

Design and Development: Prototype e-commerce applications were designed and developed using each of the selected frameworks. Core functionalities were implemented, and a mix of rendering strategies reflecting real-world architectures was applied.

Demonstration: The prototype applications were deployed under controlled conditions, ensuring they all queried the same database. This setup was crucial to isolate the effects of the frameworks and rendering strategies on application performance.

Evaluation: Performance testing was conducted using industry-standard tools and metrics, such as Lighthouse, WebPageTest, and Chrome DevTools, to measure load times, resource efficiency, and SEO-friendliness.

Communication: The findings will be presented through this thesis, providing empirical data, insights, and recommendations to the web development community. This contribution aims to enhance the ongoing discourse within the field of Computer Science.

3.2 Research Method

3.2.1 Controlled Experiment

A controlled experiment was conducted to compare the performance of Next.js, SvelteKit, and Astro in the context of e-commerce web applications. This method allowed for isolating the effects of the frameworks and rendering strategies on application performance by controlling confounding variables. According to Wohlin et al. [19], controlled experiments are essential in software engineering research for establishing causal relationships by systematically manipulating independent variables while keeping other variables constant. Kitchenham et al. [20] emphasize that controlled

experiments are particularly valuable in performance evaluation studies, providing precise measurements and facilitating reproducibility. This approach ensured that our findings on performance differences among Next.js, SvelteKit, and Astro are robust and reliable.

Prototype e-commerce applications were developed using each framework, implementing core functionalities and applying a mix of rendering strategies. By deploying these applications under the same conditions and ensuring they all queried the same database, performance differences could be attributed to the choice of framework and rendering strategy.

Key performance metrics, as well as page sizes and SEO-friendliness, were measured. The selected metrics were First Contentful Paint, Largest Contentful Paint, and Interaction to Next Paint. These metrics are widely accepted in the industry and are critical factors in user experience and search engine rankings. The following section discusses these metrics, why they were chosen, and why other metrics were not.

3.2.2 Chosen Metrics

First Contentful Paint (FCP):

Definition: FCP indicates the initial moment in the page load when the user can view any content on the page.

Relevance: The FCP metric records the interval between a user's initial navigation to the page and the display of the first piece of content. It is crucial because it provides the first visual feedback that the page is loading, essential for a good user experience [21]. A case study by Rakuten 24 found that optimizing FCP improved their revenue per visitor and conversion rate [22].

Largest Contentful Paint (LCP):

Definition: The LCP metric indicates the time at which the primary content of the page is rendered during loading.

Relevance: LCP is part of Google's Core Web Vitals, measuring when the largest content element in the viewport is fully rendered [23]. A faster LCP indicates that the page is useful sooner, leading to higher engagement and better conversion rates. Renault's case study on LCP showed that optimizing it improved bounce rates and conversion rates [24].

Interaction to Next Paint (INP):

Definition: INP measures page responsiveness, specifically how quickly the page responds to user interactions.

Relevance: INP assesses the overall responsiveness of the web page, crucial for user experience as users expect smooth interactions [16], [25]. Poor INP scores can lead to frustration and page abandonment. When redBus improved INP, they observed a 7% increase in sales [26].

Page Size:

Definition: The total size of all resources required to load the page.

Relevance: Smaller page sizes generally lead to faster load times, particularly on mobile devices or slower networks, improving the overall user experience and accessibility of the web application [14].

SEO-Friendliness:

Definition: The extent to which a web page is optimized for search engines.
Relevance: Pages that are optimized for SEO appear higher in search engine results, making them more noticeable to potential users. Metrics like FCP, LCP, and INP are used by search engines like Google to assess page performance and influence rankings [14], [15], [27].

3.2.3 Reasoning Behind Metric Selection

Previous sections in this thesis have mentioned studies showing that load time significantly impacts user experience and conversion rates in e-commerce. Therefore, focusing on load time metrics like FCP, LCP, and INP is essential. LCP and INP are recognized by Google as part of their Core Web Vitals, key indicators of page performance and search engine ranking [28]. While FCP is not classified as a Core Web Vital, it is still a significant user-centric metric for assessing perceived load speed [16], [21]. Resource efficiency, or page size, affects load times and user experience, especially on mobile devices or slower networks. SEO-friendliness is critical for attracting users through better search engine rankings [14], [15].

Many related works and industry studies also incorporate these metrics when discussing optimization and user experience, further validating their importance in our research [5], [6], [7].

3.2.4 Consideration of Additional Metrics

While the chosen metrics are crucial, other potential metrics were considered but not selected for the following reasons:

Total Blocking Time (TBT):

Definition: The TBT metric tracks the total duration a page is unresponsive to user commands.
Reason for Exclusion: INP provides a comprehensive measure of interaction delays, making TBT redundant in this study.

Cumulative Layout Shift (CLS):

Definition: CLS measures the visual stability of a page by quantifying content shifts during loading.
Reason for Exclusion: While important for visual stability, CLS was deemed less critical compared to metrics focused on load speed and interactivity for e-commerce applications.

Time to Interactive (TTI):

Definition: TTI measures the time until a page is fully interactive.
Reason for Exclusion: INP already covers the aspect of page responsiveness, providing a more detailed measure of interaction delays.

Speed Index:

Definition: The Speed Index metric assesses how fast content is visually presented during the loading of a page.

Reason for Exclusion: FCP and LCP offer clearer insights into specific stages of the loading process.

3.2.4 Tools Used for Evaluation

Industry-standard testing tools such as Lighthouse, WebPageTest, and Chrome DevTools were used to measure the performance of each framework in terms of load times, resource efficiency, and SEO-friendliness. These tools provide comprehensive insights into each framework's performance, enabling informed comparisons and conclusions.

3.3 Reliability and Validity

This section discusses the reliability and validity of the project. Reliability refers to the consistency of the research findings, ensuring that others would obtain the same results if they replicated the work. Validity concerns the accuracy and trustworthiness of the conclusions, ensuring they are supported by the research design, data collection, and analysis.

Reliability. To enhance the reliability of the research, several measures were taken. A comprehensive description of the research method and the controlled experiment, along with the code of the prototype applications, was provided. This documentation ensures that the research process is transparent and replicable. By deploying the prototype applications under the same conditions and ensuring they all queried the same database, the risk of inconsistencies in performance measurements was minimized. Widely accepted performance metrics (FCP, LCP, INP, and page size) and industry-standard testing tools (Lighthouse, WebPageTest, Chrome DevTools) were selected to ensure measurements are consistent with best practices in the field.

Validity. The controlled experiment design allows for the isolation of the effects of the frameworks and rendering strategies on application performance by controlling confounding variables. This approach strengthens the internal validity of the findings, ensuring that performance differences can be attributed to the choice of framework and rendering strategy. Although the research focuses on a specific set of JavaScript frameworks, the selection of popular and representative options widely used in the industry enhances the external validity of the findings, making them more generalizable to real-world web development projects. Performance metrics were carefully selected as key constructs to measure the performance and efficiency of the prototype applications. These constructs are well-established in the field and directly relevant to the research questions and objectives.

Despite efforts to ensure reliability and validity, the following potential threats and mitigation strategies are acknowledged.

Prototype Limitation:

Threat: The controlled experiment is limited to a single prototype application for each framework due to time constraints.

Mitigation: Prototype applications were designed to include core functionalities and a mix of rendering strategies representative of real-world e-commerce websites.

Framework and Strategy Selection:

Threat: Findings may be limited to the specific frameworks, rendering strategies, and application design chosen for this project.

Mitigation: Popular and representative options were selected, and the research process was documented in detail, allowing others to adapt and extend the work to different contexts.

Performance Measurement Factors:

Threat: Performance measurements may be affected by factors such as network conditions and device specifications.

Mitigation: Multiple rounds of testing were conducted under consistent conditions, and those conditions were reported to ensure transparency and reproducibility. By providing detailed information about the testing conditions, the context of the results can be better understood and replicated.

Availability for Independent Verification:

Threat: There may be questions about the replicability of results under different conditions.

Mitigation: The source code for the applications is publicly available, allowing others to test under different conditions if desired.

3.4 Ethical Considerations

The ethical implications of the research design and implementation were carefully considered. As the project primarily involves the development and evaluation of prototype e-commerce applications, the following ethical considerations were identified.

Although the prototype applications do not collect or store any real user data, the experiments were designed with data privacy and security in mind, considering that the applications would be publicly available after deployment. It was ensured that the applications do not include any tracking mechanisms or collect any personally identifiable information. Additionally, the applications were deployed on secure servers, and appropriate security measures were implemented to protect against potential data breaches.

In developing the prototype applications, open-source JavaScript frameworks were used, adhering to their respective licenses and terms of use. Any third-party code, libraries, or resources used in the projects were properly attributed to respect the intellectual property rights of their creators.

Transparency was maintained throughout the research process by providing detailed documentation of the methods and data collection. Any potential limitations, biases, or conflicts of interest that may have influenced

the findings were disclosed. This transparency allows others to evaluate the trustworthiness and integrity of the research. To promote reproducibility and contribute to the open sharing of knowledge, the project's materials, including code, data, and documentation, were made publicly available. This allows others to verify the findings, build upon the work, and adapt the research to different contexts.

4 Research project – Implementation

4.1 Prototype Design

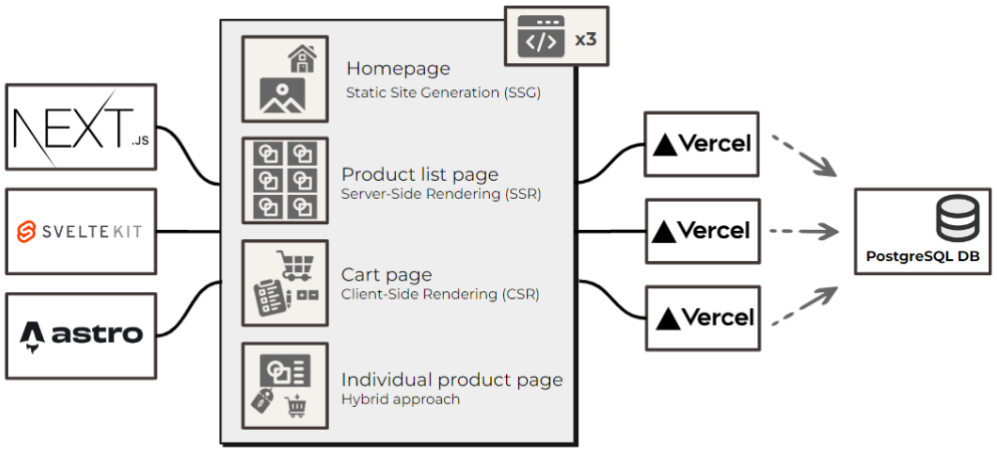

Figure 4.1: Prototype overview

The first step in the implementation process was designing the prototype e-commerce applications using each of the selected frameworks. The goal was to create applications that included code features and functionalities commonly found in real-world e-commerce websites while keeping the design consistent across all three implementations to ensure a fair comparison.

Homepage: A landing page with static assets, rendered through Static Site Generation for fast loading times and optimal SEO. SSG is ideal for pages with content that doesn't change frequently, as it allows for pre-rendering and caching, resulting in faster page loads and better search engine visibility [3].

Product catalog page: A dynamically generated page displaying 150 products in a grid, pre-rendered on the server for each request using Server-Side Rendering to ensure up-to-date content and SEO indexability. SSR is suitable for pages with frequently changing content or data that needs to be fetched from a database, as it ensures that search engines can crawl and index the most current version of the page [3].

Product detail page: A page showing detailed information about a specific product, including name, price, description, and a high-quality image. This page also includes an "Add to Cart" button and is rendered using a mix of SSR and Client-Side Rendering, also known as hybrid rendering, to balance performance and interactivity. Hybrid rendering allows for the initial page load to be fast while enabling rich interactivity once the page is loaded [3].

Shopping cart page: A page with components that display the current items in the user's cart, allow for quantity modification and item removal, and show the total price of the items. These components are rendered in the user's

browser through CSR to provide a dynamic, user-specific experience. CSR is well-suited for highly interactive components that don't require SEO indexing, as it allows for real-time updates without the need for full page reloads [3], [50].

4.2 Database and Hosting

To ensure that the only variable in the controlled experiment is the choice of framework and rendering strategy, a shared database that all three prototype applications utilize was set up. A Supabase PostgresSQL database hosted on a cloud platform in Frankfurt, Germany was chosen. The applications were hosted on the same Hosting-as-a-Service platform, Vercel and in the same region, eu-west-1.

4.3 Application Development

With the prototype designs and backend infrastructure in place, the development of the e-commerce applications using Next.js, SvelteKit, and Astro proceeded. For each framework, the core functionalities identified in the prototype design were implemented, following the designated rendering strategies for each page and component. The framework versions used was Next.js 14.1.1, SvelteKit 2.0.0 and Astro 4.6.2.

To ensure consistent state management across all three frameworks, Nano Stores were used for handling the cart items. Nano Stores is a tiny state manager designed to be fast and efficient, suitable for both small and large-scale applications. Its lightweight nature and ease of integration made it an ideal choice for maintaining consistent state across the different frameworks used in this project [52], [53].

Nano Stores facilitated a uniform approach to managing the shopping cart's state, ensuring that changes to cart items were instantly reflected across the entire application, regardless of the framework. This consistency was crucial for delivering a seamless user experience, as it allowed users to add, remove, and update cart items without discrepancies between different parts of the application.

After the initial development, the focus was on optimizing and refining each implementation to ensure the best possible performance. This involved initial performance tests to identify potential bottlenecks and areas for improvement. The full code is publicly available in these repositories:

1. https://github.com/erikcelander/e-commerce-next
2. https://github.com/erikcelander/e-commerce-svelte
3. https://github.com/erikcelander/e-commerce-astro

The following figures 4.2–4.5 shows screenshots of four different pages in the structurally identical applications. The only visual difference between the applications are the colour schemes.

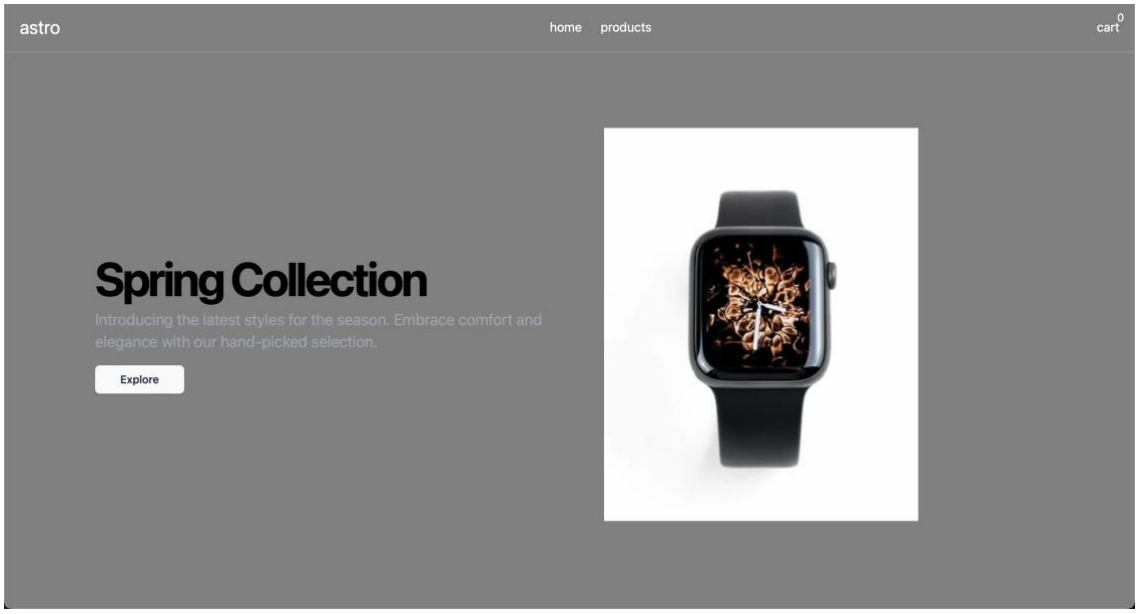

Figure 4.2: Screenshot of the homepage in the application built with Astro

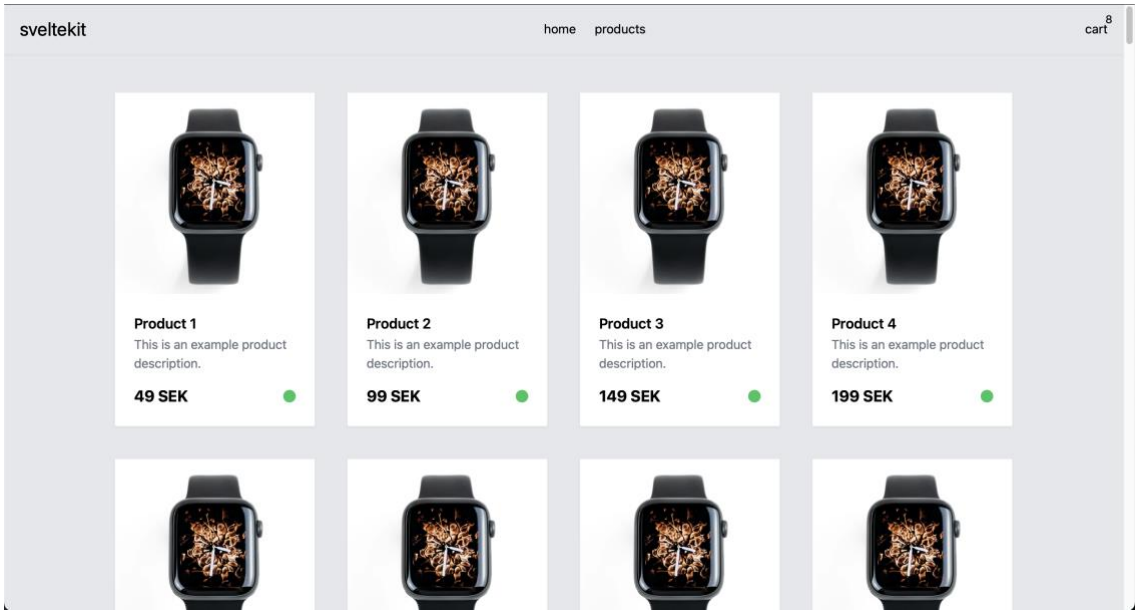

Figure 4.3: Screenshot of the product catalogue page in the application built with SvelteKit

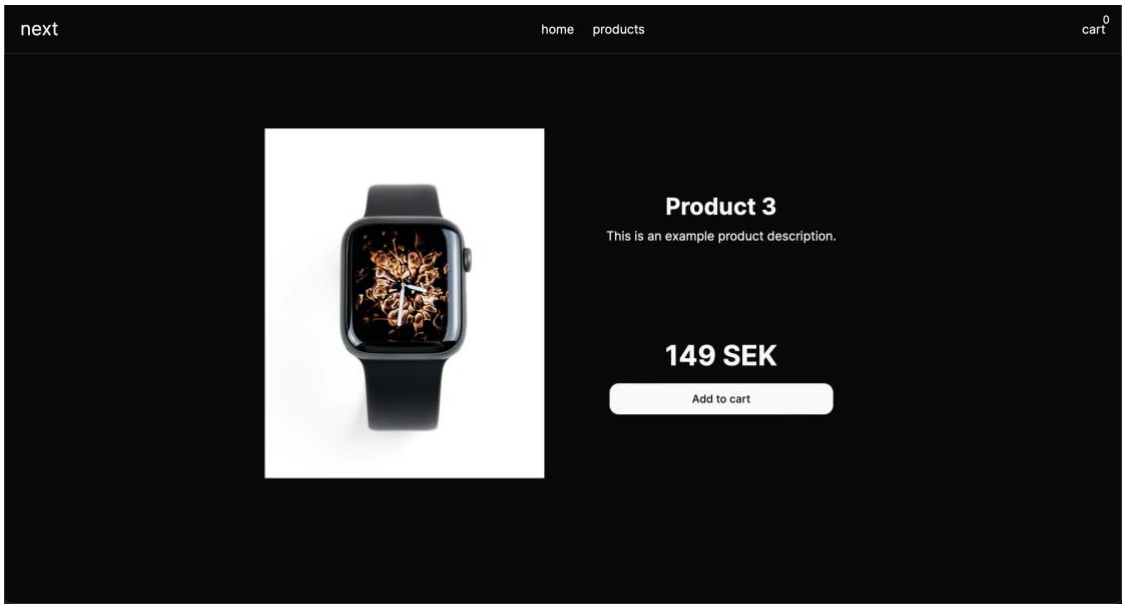

Figure 4.4: Screenshot of the product detail page in the application built with Next.js

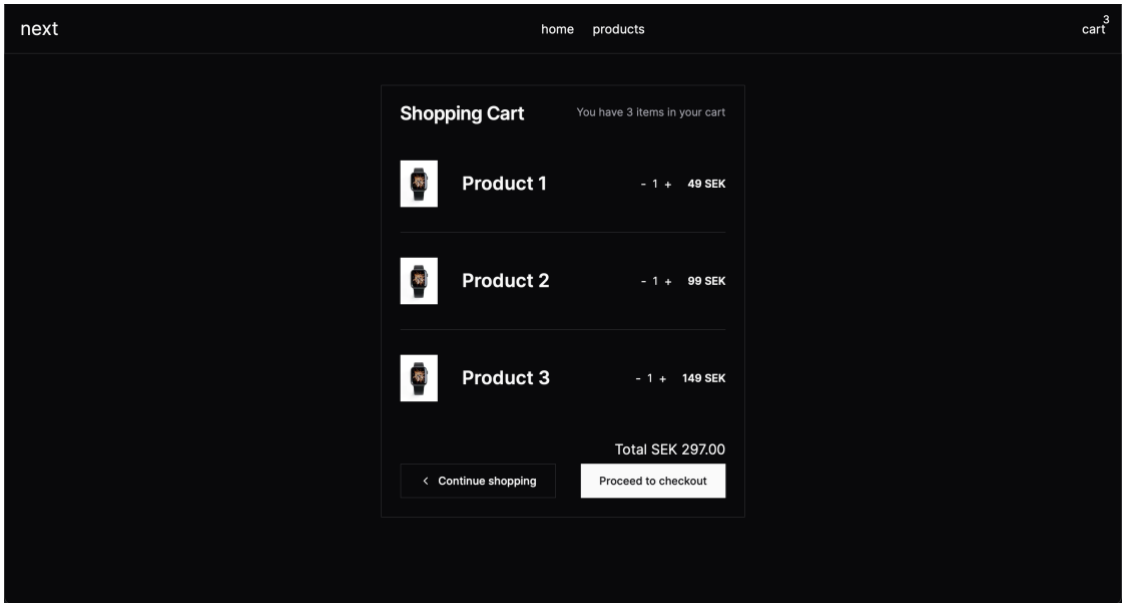

Figure 4.5: Screenshot of the cart page in the application built with Next.js

4.4 Performance Testing

To evaluate the performance of the prototype applications, structured performance tests were conducted using industry-standard tools and metrics. The tests measured key performance indicators such as load times, resource efficiency, and SEO-friendliness.

WebPageTest was used to assess the performance of each application under simulated network conditions. The tests were run using the “Simple

Configuration" setting, which includes three test runs to allow for the median result, helping to remove any anomalies. The tests were conducted using Chrome Desktop from Dulles, Virginia, US, on a 5 Mbps (5000/1000 Kbps) cable connection with 28ms of latency. The "Run Lighthouse Audit" option was also added to check the SEO score of each application.

To measure the Interaction to Next Paint metric, Lighthouse in Chrome DevTools was used. For the product detail page, user interaction was simulated by pressing the "Add to Cart" button four times during the test timespan. For the shopping cart page, the cart was pre-filled with two of each product (products 1-4), and user interaction was simulated by increasing the quantity of each item to three and then reducing it back to zero, effectively emptying the cart.

By conducting these performance tests, valuable data was collected to form the basis of the comparative analysis and help answer the research questions.

5 Results and Analysis

In this chapter, the results of the performance tests conducted on Next.js, SvelteKit, and Astro for various e-commerce application pages are presented. The tests were performed using WebPageTest and Lighthouse, measuring key performance metrics such as First Contentful Paint, Largest Contentful Paint, Interaction to Next Paint, page size, and SEO score.

5.1 Load Time Metrics

Table 5.1 presents the FCP and LCP values for each framework and page type. Figure 5.1-5.4 shows visual comparison between each framework for each page.

Framework	**Rendering Strategy**	**Page**	**FCP (s)**	**LCP (s)**
Next.js	SSG	/	0.554	0.833
SvelteKit	SSG	/	0.315	0.428
Astro	SSG	/	0.349	0.483
Next.js	SSR	/products	1.212	1.413
SvelteKit	SSR	/products	0.822	1.015
Astro	SSR	/products	0.368	1.007
Next.js	Hybrid SSR/CSR	/products/ID	0.796	1.075
SvelteKit	Hybrid SSR/CSR	/products/ID	0.714	0.935
Astro	Hybrid SSR/CSR	/products/ID	0.702	1.087
Next.js	CSR	/cart	0.512	0.716
SvelteKit	CSR	/cart	0.525	0.525
Astro	CSR	/cart	0.303	0.303

Table 5.1: FCP and LCP values for each framework and applicable page types.

Figure 5.1.1: Visual comparison between each framework for homepage.

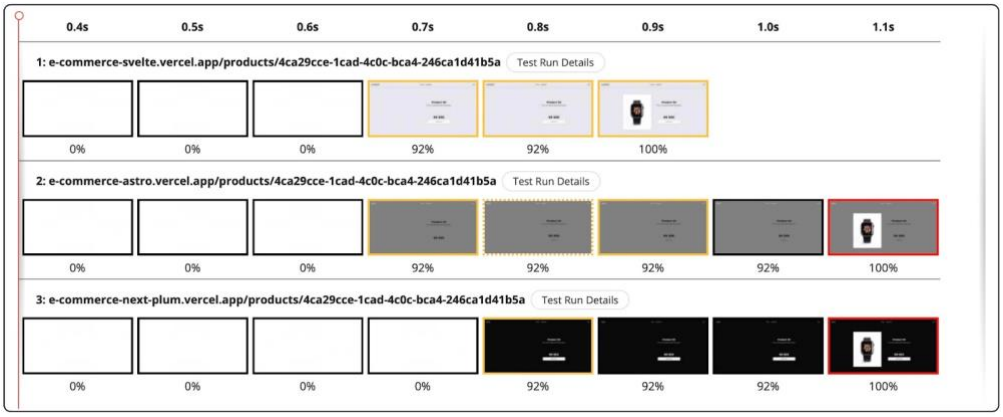

Figure 5.1.2: Visual comparison between each framework for product catalog page.

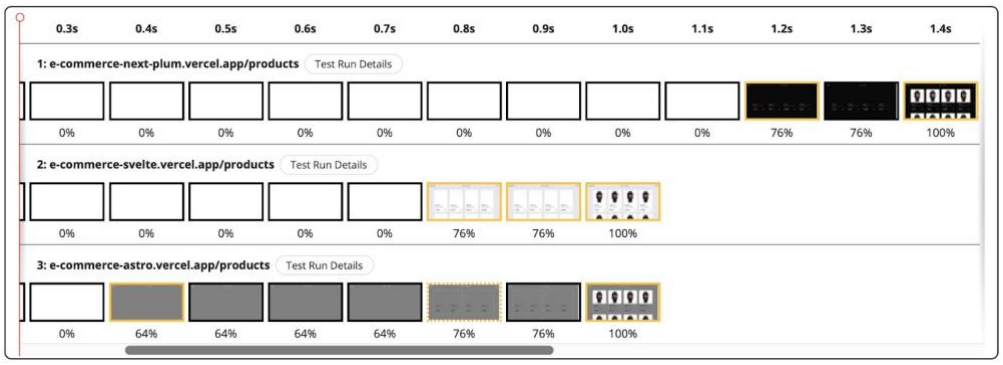

Figure 5.1.3: Visual comparison between each framework for product detail page.

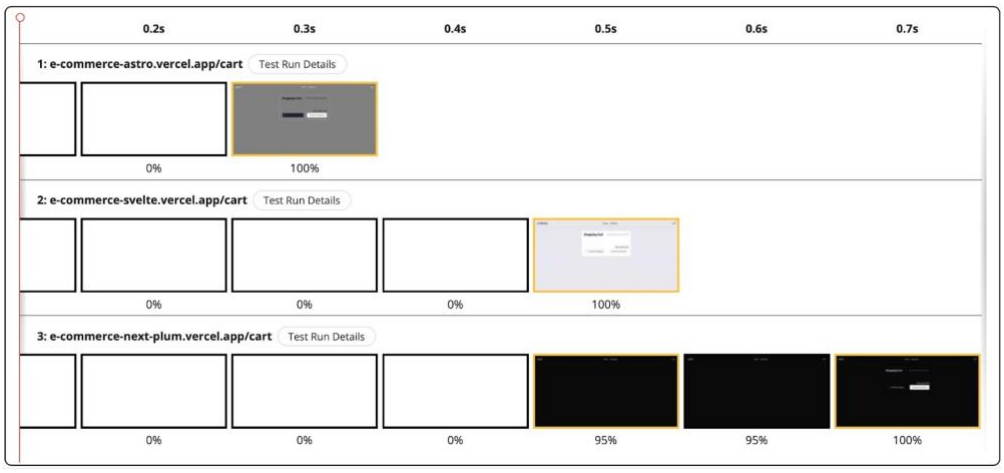

Figure 5.1.4: Visual comparison between each framework for cart page.

Astro consistently achieves the fastest FCP across all rendering strategies, followed closely by SvelteKit, with Next.js generally performing the slowest. This suggests that Astro may be more optimized for quick initial content rendering, which is critical for improving user experience, particularly in e-commerce applications where load times directly affect conversion rates.

Similar to the FCP results, Astro demonstrates superior performance in terms of LCP across most rendering strategies, particularly excelling in Client-Side Rendering. SvelteKit also performs well, maintaining competitive LCP times, while Next.js shows slower LCP times. These findings highlight Astro and SvelteKit's efficiency in rendering significant content elements quickly, which is essential for maintaining user engagement and optimizing search engine rankings.

5.2 Interaction to Next Paint (INP)

Table 5.2 presents the INP values for each framework and applicable page types.

Framework	**Rendering Strategy**	**Page**	**INP (s)**
Next.js	Hybrid SSR/CSR	/products/ID	0.016
SvelteKit	Hybrid SSR/CSR	/products/ID	0.024
Astro	Hybrid SSR/CSR	/products/ID	0.008
Next.js	CSR	/cart	0.032
SvelteKit	CSR	/cart	0.016
Astro	CSR	/cart	0.016

Table 5.2: INP values for each framework and applicable page types.

The results demonstrate that Astro has the best performance with the lowest INP values, particularly for the hybrid SSR/CSR strategy on the /products/ID page, achieving an impressive 0.008 seconds. SvelteKit offers competitive INP values, while Next.js, although still reasonable, has slightly higher INP times.

5.3 Page Size

Table 5.3 presents the page sizes (in kilobytes) for each framework and page type.

Framework	Rendering Strategy	Page	Size (kb)
Next.js	SSG	/	188
SvelteKit	SSG	/	49
Astro	SSG	/	16
Next.js	SSR	/products	185
SvelteKit	SSR	/products	66
Astro	SSR	/products	43
Next.js	Hybrid SSR/CSR	/products/ID	191
SvelteKit	Hybrid SSR/CSR	/products/ID	61
Astro	Hybrid SSR/CSR	/products/ID	90
Next.js	CSR	/cart	169
SvelteKit	CSR	/cart	31
Astro	CSR	/cart	59

Table 5.3: Page sizes for each framework and page type.

Astro and SvelteKit consistently achieve smaller page sizes compared to Next.js, with both being the smallest in half of the pages tested. This indicates that Astro and SvelteKit are more efficient in terms of resource usage, which is crucial for optimizing load times and enhancing user experience.

5.4 SEO Scores

Table 5.4 presents the SEO scores for each framework and page type.

Framework	Rendering Strategy	Page	SEO Score (0-100)
Next.js	SSG	/	100

SvelteKit	SSG	/	100
Astro	SSG	/	100
Next.js	SSR	/products	100
SvelteKit	SSR	/products	100
Astro	SSR	/products	100
Next.js	Hybrid SSR/CSR	/products/ID	100
SvelteKit	Hybrid SSR/CSR	/products/ID	100
Astro	Hybrid SSR/CSR	/products/ID	100
Next.js	CSR	/cart	100
SvelteKit	CSR	/cart	100
Astro	CSR	/cart	100

Table 5.4: SEO scores for each framework and page type.

All three frameworks—Next.js, SvelteKit, and Astro—achieved perfect SEO scores of 100 across all rendering strategies and page types. This demonstrates that all frameworks are highly effective in optimizing web pages for search engines, ensuring that web pages are well-optimized and likely to rank highly in search engine results.

6 Discussion and Conclusions

6.1 Addressing the Research Questions

RQ1: How do Next.js, SvelteKit, and Astro compare in terms of load time metrics (First Contentful Paint, Largest Contentful Paint, and Interaction to Next Paint) for different rendering strategies (Static Site Generation, Server-Side Rendering, and Client-Side Rendering) in an e-commerce application context?

Based on the results, Astro consistently achieves the fastest load times across majority of rendering strategies. SvelteKit follows closely behind, showing competitive performance, especially in FCP and LCP. However, Next.js lags, especially under SSR conditions, where its load times and responsiveness are notably slower.

For INP, Astro also shows the best responsiveness, particularly in hybrid SSR/CSR scenarios. SvelteKit performs well, providing almost as low INP values as Astro, while Next.js, although still reasonable, has slightly higher INP times.

These results indicate that Astro's architecture and optimization strategies are highly effective for e-commerce applications, where quick load times and responsiveness are critical. SvelteKit also proves to be a strong contender, offering good performance. Next.js, while robust and versatile, shows slower load times and responsiveness, potentially due to its more extensive feature set and higher overhead.

RQ2: How do Next.js, SvelteKit, and Astro compare in terms of resource efficiency in relation to page size in an e-commerce application context?

The analysis of page size across different rendering strategies reveals that Astro and SvelteKit consistently achieve smaller page sizes than Next.js. Both Astro and SvelteKit have the smallest page sizes for half of the pages, while Next.js consistently has the largest page sizes for all pages.

Astro's and SvelteKit's designs focus on minimal resource usage and efficient builds, resulting in smaller page sizes, which are advantageous for faster load times and better performance on slower networks. In contrast, Next.js, while offering extensive features, results in larger page sizes, negatively impacting load times and overall performance. This indicates that Astro and SvelteKit are more efficient in terms of resource usage, which is crucial for optimizing load times and enhancing user experience.

6.2 Discussing the Findings

6.2.1 Comparison with Related Findings

Our findings align with some aspects of the related work discussed in Chapter 1. By focusing on e-commerce applications, we provide additional insights for developers and decision-makers working in this domain.

The thesis [7] that provided a comprehensive evaluation of various JavaScript rendering frameworks, focusing on aspects such as load time metrics and developer experience, concluded that while frameworks like Next.js offer robust features and flexibility, newer frameworks like SvelteKit and Astro might offer better performance due to their innovative approaches. Our research corroborates these findings, particularly in the context of e-commerce, where Astro and SvelteKit demonstrate superior load time metrics compared to Next.js. We contribute further by emphasizing the specific advantages of these frameworks in terms of FCP, LCP, and INP.

Research [6] focused on the performance implications of SSR and CSR, particularly under varying network conditions and used Next.js as framework concluded that SSR generally provides better performance for content-rich web pages. Our study supports this conclusion, additionally showing that Astro and SvelteKit outperform Next.js in SSR scenarios, especially in terms of FCP and LCP. We expand on their work by providing a detailed comparison across multiple rendering strategies (SSR, SSG, CSR) as well as frameworks, and highlighting the specific performance advantages of each framework in e-commerce applications.

6.2.2 Limitations and Generalizability of Findings

The limitations of our study, such as the use of a single prototype application for each framework and specific tools and technologies, may influence the generalizability of our findings. While the prototype applications included core functionalities representative of real-world e-commerce websites, a larger sample size and more diverse application designs could provide more robust results. Despite these limitations, the consistency in testing conditions helps ensure that the observed performance differences are attributable to the frameworks and rendering strategies themselves.

While our findings are specific to Next.js, SvelteKit, and Astro in the context of e-commerce web development, the principles and performance metrics discussed can be applied to other web development scenarios. The emphasis on load time metrics, resource efficiency, and SEO performance is relevant to a wide range of web applications. These insights can guide developers in making informed decisions when selecting JavaScript frameworks and rendering strategies for various projects.

6.2.3 Beyond Performance Metrics

While Astro demonstrates the best performance in load time metrics and tied for resource efficiency, the differences between the frameworks are not overwhelmingly large. Both SvelteKit and Next.js also perform well, with SvelteKit being particularly close in many metrics. Therefore, decision-makers should also consider other factors such as developer experience, community support, and specific project requirements when choosing a framework.

Developer experience: The ease of use, learning curve, and available tooling can significantly impact development speed and efficiency. For instance, Next.js is built on top of React, which has a vast ecosystem and a large community, making it easier for developers familiar with React to get started. SvelteKit offers a more modern and concise syntax, leading to cleaner and more maintainable code.

Community and ecosystem: The size and activity of the community surrounding a framework can be crucial. A larger community often means more third-party plugins, comprehensive documentation, and quicker resolution of issues. Next.js, with its strong backing from Vercel and integration with React, has a significant advantage in this area.

Specific project requirements: Certain projects may have unique requirements that make one framework more suitable than others. For example, if server-side rendering and static site generation are critical, Next.js might be preferred due to its robust support for these features. If performance and minimal JavaScript payload are crucial, Astro or SvelteKit might be the better choice.

6.3 Conclusions

- Astro is the most performant framework in terms of FCP, LCP, and INP across all rendering strategies, making it highly suitable for e-commerce applications that prioritize fast load times and responsiveness. SvelteKit also performs well, particularly in FCP and LCP, but lags slightly behind Astro in INP. Next.js, while feature-rich and versatile, shows slower performance in load times and responsiveness, particularly under Server-Side Rendering conditions.
- Astro and SvelteKit demonstrate the most efficient use of resources, achieving the smallest page sizes for half of the pages tested. This efficiency contributes to their superior performance in load time metrics. Next.js, despite its robustness and flexibility, consistently has larger page sizes, which can negatively impact its performance, especially on slower networks.
- Overall Astro and SvelteKit consistently outperforms Next.js in terms of load time metrics and resource efficiency. However, the differences in performance metrics are not drastically large. All three frameworks are capable of delivering high-quality e-commerce applications. Therefore, decision-

makers should also consider factors such as developer experience, community support, and specific project requirements when selecting a framework.

7 Future Work

While the study provides significant insights, there are areas that could be improved. The primary limitation is the use of a single prototype application for each framework, which may influence the generalizability of the findings. Expanding the study to include a larger sample size with more diverse application designs could provide more robust results.

Given the scope and limitations of this thesis, there are multiple options for future research:

1. Explore Other Frameworks: Expanding the research to include additional modern JavaScript frameworks, such as Nuxt.js, SolidStart, or Qwik City, could provide a broader comparison and reveal how newer or less mainstream frameworks perform in the context of e-commerce applications. This could help identify whether the observed trends hold true across a wider array of frameworks or if certain frameworks offer unique advantages.
2. Exploring Additional Metrics: Investigating additional performance metrics, such as Total Blocking Time (TBT) and Cumulative Layout Shift (CLS), would help in creating a more comprehensive performance profile for each framework.
3. Qualitative Analysis: Incorporating qualitative aspects such as developer experience, ease of use, and community support could provide a more holistic view of the frameworks' effectiveness and adoption in real-world projects.
4. Real-world Implementations: Applying the frameworks to real-world e-commerce projects and measuring their impact on business metrics such as conversion rates, customer satisfaction, and SEO performance would be valuable.

By addressing these areas, future research can build on the findings of this thesis, contributing further to the understanding and optimization of web development frameworks for e-commerce and beyond.

References

[1] State of JavaScript, "Rendering Frameworks" 2022. [Online]. Available: https://2022.stateofjs.com/en-US/libraries/rendering-frameworks/. [Accessed: Mar. 19, 2024].

[2] R. Ollila, N. Mäkitalo, and T. Mikkonen, "Modern Web Frameworks: A Comparison of Rendering Performance," *Journal of Web Engineering (JWE)*, vol. 21, no. 3, pp. 789-814, Mar. 2022. [Online]. Available: https://doi.org/10.13052/jwe1540-9589.21311. [Accessed: May 15, 2024].

[3] A. Osmani and J. Miller, "Rendering on the Web," web.dev. [Online]. Available: https://web.dev/articles/rendering-on-the-web. [Accessed: May 20, 2024].

[4] BuiltWith, "eCommerce Usage Distribution on the Entire Internet," [Online]. Available: https://trends.builtwith.com/shop/traffic/Entire-Internet. [Accessed: May 20, 2024].

[5] "What is e-commerce?" McKinsey & Company, Jun. 29, 2023. [Online]. Available: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-e-commerce. [Accessed: May 20, 2024].

[6] "E-commerce as share of total retail sales worldwide 2021-2027," Statista, Feb. 28, 2024. [Online]. Available: https://www-statista-com.proxy.lnu.se/statistics/534123/e-commerce-share-of-retail-sales-worldwide/. [Accessed: May 20, 2024].

[7] ARVO, "Site Speed is (Still) Impacting Your Conversion Rate," Portent, Apr. 20, 2022. [Online]. Available: https://www.portent.com/blog/analytics/research-site-speed-hurting-everyones-revenue.htm. [Accessed: May 20, 2024].

[8] "The Page Speed Report Stats & Trends for Marketers," Unbounce. [Online]. Available: https://unbounce.com/page-speed-report/.

[9] Vodafone, "A 31% improvement in LCP increased sales by 8%," web.dev, Mar. 17, 2021. [Online]. Available: https://web.dev/case-studies/vodafone. [Accessed: May 20, 2024].

[10] S. Subramanian, "Evaluating page experience for a better web," Google Developers Blog, May 28, 2020. Updated Mar. 12, 2024. [Online]. Available: https://developers.google.com/search/blog/2020/05/evaluating-page-experience. [Accessed: May 20, 2024].

[11] "How does website speed boost search engine optimization (SEO)?" Cloudflare. [Online]. Available: https://www.cloudflare.com/en-gb/learning/performance/how-website-speed-boosts-seo/. [Accessed: May 20, 2024].

[12] J. Paakkanen, "Upcoming JavaScript web frameworks and their techniques," Bachelor's thesis, Aalto University, School Of Science, Computer Science, Dec. 2023. [Online]. Available:

https://www.researchgate.net/publication/376835044_Upcoming_JavaScript_web_frameworks_and_their_techniques. [Accessed: May 20, 2024].
[13] D. Wernersson and V. Sjölund, "Choosing a Rendering Framework: A Comparative Evaluation of Modern JavaScript Rendering Frameworks," Bachelor's thesis, Linnaeus University, Faculty of Technology, Department of Computer Science and Media Technology (CM), 2023. [Online]. Available: https://lnu.diva-portal.org/smash/record.jsf?pid=diva2%3A1764151&dswid=8604. [Accessed: May 20, 2024].
[14] M. Siahaan and R. Kenidy, "Rendering performance comparison of React, Vue, Next, and Nuxt," *Jurnal Mantik*, vol. 7, no. 3, pp. 1851-1860, Oct. 2023. Available: https://www.iocscience.org/ejournal/index.php/mantik/article/view/4242/3010.
[15] C. Nordström and A. Dixelius, "Comparisons of Server-side Rendering and Client-side Rendering for Web Pages," Examensarbete, Uppsala Universitet, Uppsala, Sweden, UPTEC IT 23031, Sept. 2023. Available: https://www.diva-portal.org/smash/get/diva2:1797261/FULLTEXT02.pdf.
[16] P. Walton, "User-centric performance metrics," web.dev. [Online]. Available: https://web.dev/articles/user-centric-performance-metrics. [Accessed: May 20, 2024].
[17] S. Rasli, N. Khairi, H. Ayathuray, and M. Syafiq, "The Impact of E-Business Website Quality on Customer Satisfaction," Faculty of Business and Accountancy, Universiti Selangor, Dec. 2018. [Online]. Available: https://www.researchgate.net/publication/329916119_THE_IMPACT_OF_E-BUSINESS_WEBSITE_QUALITY_ON_CUSTOMER_SATISFACTION. [Accessed: May 20, 2024].
[18] "Why does speed matter?" web.dev. [Online]. Available: https://web.dev/learn/performance/why-speed-matters. [Accessed: May 20, 2024].
[19] W. Stadnik and Z. Nowak, "The Impact of Web Pages' Load Time on the Conversion Rate of an E-Commerce Platform," in *Advances in Intelligent Systems and Computing, International Conference on Information Systems Architecture and Technology,* Sept. 2018. [Online]. Available: https://www.researchgate.net/publication/319449830_The_Impact_of_Web_Pages'_Load_Time_on_the_Conversion_Rate_of_an_E-Commerce_Platform. [Accessed: May 20, 2024].
[20] S. Dakov and A. Malinova, "A Survey of E-Commerce Security Threats and Solutions," *Proceedings of CBU in Natural Sciences and ICT*, vol. 2, 2021. [Online]. Available:

https://ojs.cbuic.cz/index.php/pns/article/view/135. [Accessed: May 20, 2024].
[21] S. Ehikioya and E. Guillemot, "A Critical Assessment of the Design Issues in E-commerce Systems Development," *Engineering Reports*, vol. 2, no. 3, Mar. 2020. [Online]. Available: https://www.researchgate.net/publication/340299002_A_Critical_Assessment_of_the_Design_Issues_in_E-commerce_Systems_Development. [Accessed: May 20, 2024].
[22] S. Dakov and A. Malinova, "A Survey of E-commerce Security Threats and Solutions," *PNS*, vol. 2, pp. 1-9, Oct. 2021. [Online]. Available: https://ojs.cbuic.cz/index.php/pns/article/view/135. [Accessed: May 21, 2024].
[23] A. Shukla, "Modern JavaScript Frameworks and JavaScript's Future as a Full-Stack Programming Language," *Journal of Artificial Intelligence & Cloud Computing*, 2023. [Online]. Available: https://www.researchgate.net/publication/377629693_Modern_JavaScript_Frameworks_and_JavaScript's_Future_as_a_Full-Stack_Programming_Language/citation/download. [Accessed: May 21, 2024].
[24] React. [Online]. Available: https://reactjs.org/. [Accessed: May 21, 2024].
[25] Vue.js. [Online]. Available: https://vuejs.org/. [Accessed: May 21, 2024].
[26] Angular. [Online]. Available: https://angular.io/. [Accessed: May 21, 2024].
[27] Express. [Online]. Available: https://expressjs.com/. [Accessed: May 21, 2024].
[28] Koa. [Online]. Available: https://koajs.com/. [Accessed: May 21, 2024].
[29] Next.js. [Online]. Available: https://nextjs.org/. [Accessed: May 21, 2024].
[30] Next.js Documentation. [Online]. Available: https://nextjs.org/docs. [Accessed: May 21, 2024].
[31] "Introduction • Docs • SvelteKit," SvelteKit, 2024. [Online]. Available: https://kit.svelte.dev/docs/introduction. [Accessed: May 14, 2024].
[32] "Introducing Astro: Ship Less JavaScript," Astro, 2024. [Online]. Available: https://astro.build. [Accessed: May 14, 2024].
[33] "What Is Astro? An Introduction To the Popular Static Site Generator," Kinsta, 2024. [Online]. Available: https://kinsta.com/blog/astro. [Accessed: May 14, 2024].
[34] "Why Astro? | Docs," Astro Documentation, 2024. [Online]. Available: https://docs.astro.build/en/getting-started/. [Accessed: May 14, 2024].
[35] J. McMahon, "Astro vs. Next.js," CloudCannon, Apr. 10, 2023. [Online]. Available: https://cloudcannon.com/blog/astro-vs-next-js/. [Accessed: May 21, 2024].

[36] C. Gawin, "How Astro compares to Next.js for React apps," LogRocket, May 4, 2022. [Online]. Available: https://blog.logrocket.com/how-astro-compares-next-js-react-apps/. [Accessed: May 21, 2024].
[37] I. Grigorik, "Critical Rendering Path," web.dev. [Online]. Available: https://web.dev/articles/critical-rendering-path?hl=en. [Accessed: May 21, 2024].
[38] J. Wagner, "Client-side rendering of HTML and interactivity," web.dev. [Online]. Available: https://web.dev/articles/client-side-rendering-of-html-and-interactivity?hl=en. [Accessed: May 21, 2024].
[39] P. Lewis, "Rendering performance," web.dev. [Online]. Available: https://web.dev/articles/rendering-performance?hl=en. [Accessed: May 21, 2024].
[40] V. Vaishnavi and W. Kuechler, “Design Science Research in Information Systems,” Jan. 20, 2004. Updated by V. Vaishnavi and P. Stacey in 2017 and 2019, and by V. Vaishnavi and S. Duraisamy on Dec. 15, 2023. [Online]. Available: http://www.desrist.org/design-research-in-information-systems/. [Accessed: May 20, 2024].
[41] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, Jan. 2007. [Online]. Available: https://www.researchgate.net/publication/284503626_A_design_science_research_methodology_for_information_systems_research. [Accessed: May 20, 2024].
[42] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in Software Engineering. *Springer Science & Business Media*, 2012.
[43] B. Kitchenham, L. Pickard, and S. L. Pfleeger, “Case studies for method and tool evaluation,” *IEEE Software*, vol. 12, no. 4, pp. 52-62, 2002.
[44] P. Walton, “First Contentful Paint (FCP)," web.dev. [Online]. Available: https://web.dev/articles/fcp/. [Accessed: May 20, 2024].
[45] H. Lee, L. Doung, R. Akiba and S. Kashiwase, “How Rakuten 24's investment in Core Web Vitals increased revenue per visitor by 53.37% and conversion rate by 33.13%,” web.dev. [Online]. Available: https://web.dev/case-studies/rakuten. [Accessed: May 20, 2024].
[46] P. Walton and B. Pollard, "Largest Contentful Paint (LCP)," web.dev. [Online]. Available: https://web.dev/articles/lcp/. [Accessed: May 20, 2024].
[47] A. Bisch, C. Bazureau, and T. Coustillac, "How Renault improved its bounce and conversion rates by measuring and optimizing Largest Contentful Paint," web.dev. [Online]. Available: https://web.dev/case-studies/renault. [Accessed: May 20, 2024].

[48] J. Wagner, “Interaction to Next Paint (INP)," web.dev. [Online]. Available: https://web.dev/articles/inp/. [Accessed: May 20, 2024].
[49] A. Kumar and S. Rajpal, “How redBus improved their website's Interaction to Next Paint (INP) and increased sales by 7%,” web.dev. [Online]. Available: https://web.dev/case-studies/redbus-inp. [Accessed: May 20, 2024].
[50] “Understanding page experience in Google Search results," Google Developers. [Online]. Available: https://developers.google.com/search/docs/appearance/page-experience. [Accessed: May 20, 2024].
[51] P. Walton, “Web Vitals,” web.dev. [Online]. Available: https://web.dev/articles/vitals#core-web-vitals. [Accessed: May 20, 2024].
[52] "Nano Stores - A Tiny State Manager for JavaScript Applications." [Online]. Available: https://github.com/nanostores/nanostores. [Accessed: May 21, 2024].
[53] “State Management with Nano Stores." [Online]. Available: https://dev.to/yet_another_dev/state-management-with-nano-stores-4f8a. [Accessed: May 21, 2024].

Appendix

A Full test data in JSON

https://drive.google.com/drive/folders/1n0CHJBvu6v0344ios-k5d72ZggAFcb3o?usp=sharing

B Application Source Code

https://github.com/erikcelander/e-commerce-next
https://github.com/erikcelander/e-commerce-svelte
https://github.com/erikcelander/e-commerce-astro

C WebPageTest Visual Comparisons between frameworks for each page

/
https://www.webpagetest.org/video/compare.php?tests=240427_AiDcQX_6S7%2C240427_BiDcMJ_71Q%2C240427_BiDcWK_7FM&thumbSize=150&ival=100&end=visual

/products
https://www.webpagetest.org/video/compare.php?tests=240427_AiDc2B_6SY%2C240427_BiDc43_7G8%2C240427_BiDcW5_71X&thumbSize=100&ival=100&end=visual

/products/id
https://www.webpagetest.org/video/compare.php?tests=240427_AiDcB9_7F1,240427_AiDcKB_71V,240427_BiDcBB_6TV

/cart
https://www.webpagetest.org/video/compare.php?tests=240427_AiDcXK_724%2C240427_BiDcKD_7GX%2C240427_BiDcR2_6VA&thumbSize=200&ival=100&end=visual