HÖGSKOLAN I BORÅS

Exploring Benefits and Disadvantages with SvelteKit Through the Development of a Dungeons and Dragons Game Tracker

Albin Cedenblad

2025-10-23

Abstract

This study evaluates the SvelteKit framework by developing a Dungeons and Dragons game tracker to explore its benefits and disadvantages in full-stack web development. The research purpose was to assess SvelteKit’s practicality, performance, and developer experience through hands-on implementation. The project involved building a functional web application that allowed users to manage campaigns with, NPCs, characters, and scenes within an interactive interface and the performance was measured using both Lighthouse and Sitespeed.io to gather insights into loading speed, responsiveness, and stability of the web application.

The result showed that SvelteKit delivers high technical efficiency, achieving fast load times, stable rendering, and minimal blocking. From the developer’s perspective, the framework offered an intuitive structure, modern tooling, and a fast development workflow. The study also identified certain limitations with the lack of community with incomplete documentation, and fewer third-party integrations compared to more established frameworks such as Angular and Vue.js

Even though SvelteKit is a strong candidate for small to medium sised applications that prioritise performance, maintainability, and developer productivity. The study contributes a practical evaluation of SvelteKit’s realworld applicability and provides insights that may assist developers and researchers in selecting suitable frameworks for future web development projects.

1. Introduction

1.1 Background

Modern web development relies on frameworks that simplify the process of building interactive and maintainable applications. Frameworks such as Next.js, Angular, and Vue.js have become a big part of how developers design web applications (2021, pp. 34-36), enabling faster development and improved performance. Svelte and its full-stack framework SvelteKit have provided innovative alternatives for developers, and unlike traditional frameworks that rely on a virtual DOM, Svelte compiles components into highly efficient vanilla JavaScript at build time, resulting in smaller bundle sizes and faster load times(Svelte, 2025b). SvelteKit extends this model by introducing features such as server-side rendering, file-based routing, and integration with modern tooling.

As the web development platforms continue to evolve, Svelte is gaining popularity among developers seeking a simpler yet powerful approach to building applications. But the limited research and practical evaluations make it unclear how well SvelteKit performs in real-world, full-stack contexts. Therefore, this study aims to explore the benefits and disadvantages of using the SvelteKit framework by developing a full-stack web application to assess its practicality, performance, and developer experience.

1.2 Problem Statement:

Despite the growing number of modern web frameworks, there remains limited empirical research on how emerging frameworks like SvelteKit perform in real-world, full-stack development contexts. Most existing studies focus on well-established frameworks noted by both Xu and Putra (Xu, 2021, pp. 34-36; Putra et al., 2025, pp. 355-364), leaving a gap in understanding regarding SvelteKit’s practical applicability, performance characteristics, and developer experience. Particularly developers and researchers lack evaluations that combine hands-on implementation with theoretical comparison. This absence makes it difficult to determine whether SvelteKit is a viable alternative for building complex, interactive web applications as those that demand real-time updates, data persistence, and dynamic user interfaces.

1.3 Objective

The objective of this study is to evaluate the SvelteKit framework through the development of a practical web application Dungeons and Dragons game tracker, to analyse its benefits and disadvantages. The study aims to achieve this by implementing a fully functional web application using SvelteKit to observe its architecture, tooling, and performance in practice, assessing technical performance and developer experience, and creating literature-based comparisons and empirical observations to identify when and why SvelteKit may or may not be suitable for production-grade web applications.

1.4 Research Question

What benefits and disadvantages are there in the SvelteKit framework?

1.5 Significance

The significance of this study lies in providing a practical and systematic exploration of the SvelteKit framework as a full-stack framework. SvelteKit represents a modern approach to web development that emphasises simplicity, performance, and developer efficiency. Because SvelteKit was released in 2022 (Svelte, 2022) limited documentation and few case studies demonstrating how it performs. By developing a fully functional web application using SvelteKit, this study aims to generate practical insights into its strengths, limitations, and suitability for building interactive, data-driven web systems. The findings can support developers and students who wish to understand SvelteKit's architecture, development workflow, and best practices.

1.6 Expected Contribution

This study is expected to contribute by providing a documented case study of a full-stack application built entirely with SvelteKit. The project will demonstrate how the framework can be applied in practice, including its setup, performance behaviour, and integration of key features such as routing, state management, and database connectivity. The resulting analysis will offer concrete examples of SvelteKit's real-world use, highlighting both its capabilities and challenges. This report will contribute to the academic discussion on modern web development frameworks by presenting empirical findings.

1.7 Target Group

This study is primarily intended for web developers, computer science students, and educators interested in modern full-stack frameworks. Developers can benefit from the practical insights and implementation details presented in this report when adopting SvelteKit in their own projects. For students, the study offers a concrete example of how theoretical knowledge of web technologies can be applied to real-world development, and educators and researchers may also find the documentation useful as a reference for comparison or investigation into the performance and usability of emerging web frameworks such as SvelteKit.

1.8 Literature Review

The evaluation frameworks through practical case studies is well-established in software engineering research. Kitchenham et al. (2002) demonstrate that case studies provide valuable empirical data for technology assessment, which justifies the hands-on approach taken in this study. Recent comparative studies consistently highlight Svelte's performance advantages. Xu (2021, pp.34-36) conducted systematic benchmarks that show that Svelte outperforms React, Vue, and Angular in runtime efficiency and bundle size due to its compiler-based architecture. These findings are reinforced by Putra et al. (2025, pp. 355-364), who observed similar performance benefits in their technical evaluation. This is supported by Celander and Möllestål (2024, pp.32-34), whose bachelor's thesis on e-commerce frameworks found that SvelteKit consistently outperformed Next.js in load time metrics and resource efficiency. Community-driven benchmarks align with academic research; Jangra (2025) and Bennett (2025) both report faster load times and smaller bundle sizes for Svelte compared to traditional virtual DOM-based frameworks.

While performance advantages are clear, researchers note ecosystem limitations. Dékány (2025, pp. 31-35) found that Svelte's smaller community and fewer third-party libraries present practical challenges compared to more established frameworks. This aligns with observations by Eze (2024), who noted that Next.js offers more mature tooling and enterprise-ready features despite SvelteKit's performance edge.

The official Svelte documentation (Svelte, 2025b) positions SvelteKit as a full-stack framework with server-side rendering, file-based routing, and adapter support. This places it in direct competition with established solutions like Next.js (Next.js, 2025), which boasts extensive community support and production-ready features.

The literature reveals consistent performance advantages for Svelte and SvelteKit but identifies ecosystem maturity as a limitation. But most studies focus on technical benchmarks rather than practical implementation experiences. This study addresses this gap by documenting the end-to-end development of a full-stack application, providing insights into both technical performance and real-world developer experience.

To ensure an evaluation of the SvelteKit application, a set of industry-standard performance metrics was selected. These metrics, defined by Google's Web Vitals initiative and measured using the Lighthouse auditing tool, provide a user-centric view of application quality (Google Chrome, 2019). The following metrics were chosen for their direct impact on user experience.

First Contentful Paint (FCP) measures the time from when the page starts loading to when any part of the page's content is first rendered on the screen. It is a key metric for perceived loading speed and initial user feedback (Google Chrome, 2019).

Largest Contentful Paint (LCP) marks the point during loading when the page's main content—typically a large image or a block of text has become fully visible. It indicates when the user feels the page has substantially loaded and is a Core Web Vital (Google Chrome, 2019).

Time To First Byte (TTFB) is a measure of server responsiveness, quantifying the time between the browser requesting a page and receiving the very first byte of data from the server. It is a foundational metric for diagnosing backend or network-level delays (Google Chrome, 2019).

Speed Index (SI) measures how quickly the contents of a page are visibly populated. It calculates the average time at which visible parts of the page are displayed, providing a score for the overall perceptual loading speed (Google Chrome, 2019).

Total Blocking Time (TBT) measures the page's responsiveness during load by adding all periods where the main thread was blocked by long-running JavaScript tasks, preventing the page from responding to user input. Optimising TBT directly improves the Interaction to Next Paint (INP) metric (Google Chrome, 2019).

Cumulative Layout Shift (CLS) measures visual stability by summing the score of all unexpected layout shifts during the page's lifespan. A low CLS is critical for a non-frustrating user experience, ensuring that content does not jump around as the page loads (Google Chrome, 2019).

The consistent use of these metrics across academic research (Xu, 2021; Celander & Möllestål, 2024) validates their importance and allows for comparison with established benchmarks.

, to see how robust the developed website is. While Lightehouse provides a view of how the pages are performing while loading in the open-source Sitespeed.io toolkit is designed specifically for continuous and automated web performance testing. Its primary strength lies in its ability to run performance tests repeatedly to understand how the site runs over time(Sitesoeedi.io, 2024).

2. Method

2.1 Scope

This study focuses on evaluating the SvelteKit framework through the development of a practical web application, a Dungeons and Dragons game tracker. The project aims to explore key features of SvelteKit, including its server-side rendering capabilities, file-based routing, state management, and TypeScript integration. The tracker allows users to use a web-based tabletop DnD campaign manager. It lets the user organise and run game sessions, manage scenes, NPCs, items, and players, and visually track connections and events on a map. Features include:

- Database-backed storage of all game content using MongoDB, chosen for its flexible schema, which accommodates the varied and dynamic structure of game elements.
- Visual scene management with drag-and-drop connections
- Map tools for placing and managing markers, NPCs, and players
- Team and battle management for encounters
- Image upload for scenes and maps
- Flexible UI for running and tracking DnD sessions

These features provide a real-world scenario to test the framework's performance, developer experience, and suitability for building interactive applications.

The study emphasises both technical performance and developer experience. The project deliberately targets a manageable scope to allow for exploration of these core aspects without introducing unnecessary complexity. Comparative insights with other frameworks, threw literature and community benchmarks to contextualise SvelteKit’s strengths and limitations, rather than implementing a direct side-by-side comparison.

2.2 Literature Review Method

The literature review was conducted using a combination of academic databases, official documentation, and credible online sources to ensure both scholarly rigour and practical relevance. Academic searches were first carried out in Google Scholar, the search terms included: “Svelte performance comparison”, “SvelteKit vs Next.js”, “JavaScript framework benchmarks”, and “frontend framework evaluation”.

This search yielded several peer-reviewed and academic sources that were directly relevant to the study. Xu (2021) provides a benchmark comparison of React, Vue, Angular, and Svelte, highlighting performance differences and bundle size advantages. Putra et al. (2025) investigated technical performance metrics across Svelte, React, and Vue, including runtime efficiency and resource usage. Dékány (2025) evaluates React, Vue.js, and Svelte with a focus on technical performance and developer experience, offering complementary insights into usability and practical challenges. Kitchenham et al. (2002) was included as a foundational source, offering methodological guidance for evaluating software tools and frameworks through case studies.

Given that SvelteKit is a relatively new framework with limited coverage in traditional academic literature, complementary sources were used to provide practical and technical insights. These include the official Svelte documentation (Svelte, 2025a), which details the framework’s architecture and features, and industry articles such as Eze(2024), which compares SvelteKit to Next.js. Community-driven benchmarks and discussions were also reviewed to capture real-world developer experiences. For example, blogs by Jangra (2025) and Bennett (2025) offered performance comparisons and developer perspectives on Svelte versus React and Vue. Only sources from well-regarded platforms were included to ensure credibility.

Selection criteria focused on recency, relevance to the research question, and reliability of the source. Older foundational works, such as Kitchenham et al. (2002), were included for methodological grounding. This systematic approach ensured the literature review was based on trustworthy, up-to-date, and contextually relevant materials, providing a strong foundation for evaluating SvelteKit in a practical web application.

2.3 Materials and Tools Required

The development of a DnD Game Tracker relies on a combination of modern web technologies and supporting tools. At the core of the project is SvelteKit, a contemporary framework that enables developers to build both frontend and backend logic within a unified application (Svelte, 2025a). SvelteKit supports essential features such as server-side rendering, file-based routing, and integration with TypeScript, making it wellsuited for building interactive and data-driven web applications.

Data storage is managed using MongoDB, a flexible NOSQL database capable of storing the necessary data. By using a cloud-based database, the project benefits from scalability, availability and easy access from multiple devices. Image and file uploads are supported through native SvelteKit endpoints, allowing media assets to be stored either in the database or in the file system. The frontend of the application is built using Svelte, the core UI library, with custom CSS for styling and interface layout, eliminating the need for additional UI frameworks.

2.4 Procedures

The project development started with the selection of the technological foundation. SvelteKit was chosen as the primary framework due to its modern architecture that unifies frontend and backend development providing built-in routing capabilities and full-stack development features within a single codebase. This decision was complemented by selecting MongoDB as the database solution, chosen for its document-based NoSQL structure that offers flexibility in data modelling and seamless integration with JavaScript-based applications. The project was initialised with a GitHub repository to maintain proper version control and document the development progress throughout the entire process.

The initial setup phase involved updating Node.js and npm to their latest versions, followed by creating a new SvelteKit project using the official starter template. TypeScript support was immediately enabled to enhance type safety and improve long-term code maintainability. During the dependency installation phase, essential packages were added, including the MongoDB client for database connectivity, bcryptjs for secure password hashing, and jsonwebtoken for authentication token management. While authentication modules such as @auth/sveltekit were initially evaluated and installed, the development process ultimately led to implementing a custom JWT-based authentication system that provided superior control and better integration with the planned API-driven architecture.

The database architecture development began with creating a foundation that would support the relationships inherent in the systems. A reusable database connection utility was implemented in db.js to establish persistent MongoDB connections accessible throughout the application. Seven primary collections were designed to represent the core entities of a DnD campaign. Games representing individual campaigns, sessions for tracking game meetings, NPCs for non-player characters, players for player characters, items for equipment and treasure, scenes for locations and maps, and notes for contextual information. To ensure consistency and maintainability, a centralised schema system was developed in entities.js that serves as the single source of truth for all entity structures. This schema system defines base fields common to all entities, individual schemas for each entity type, and relationship mappings that enable complex interactions such as items belonging to players or NPCs, NPCs existing within specific scenes, and detailed session tracking with connections to all game entities.

The backend development utilised SvelteKit's file-based routing system to create a API structure within the /routes/api/ directory. Custom authentication middleware was implemented in auth.js, providing essential functions for JWT token extraction through getUserIdFromRequest, standardised response helpers including createJsonResponse and createErrorResponse, and consistent authorisation validation across all protected endpoints. A model layer architecture was established in the /lib/models/ directory, with individual files for each entity type that handle specific CRUD operations and business logic while maintaining the relationships defined in the schema system.

The API endpoints were designed to provide complete game management functionality across multiple domains. Game management endpoints handle campaign creation, user game retrieval, and detailed game data fetching that includes all associated entities. Content management systems provide full CRUD operations for NPCs, players, items, and scenes while maintaining proper relationship management between entities. The session system enables creation with structured note categories, session status updates, and dynamic connections between session data and game entities. Entity relationship endpoints facilitate complex operations such as connecting and disconnecting items to players or NPCs, placing NPCs within scenes, and managing item locations across the game world. A note system allows contextual information to be attached to any entity with full CRUD capabilities, while specialised file upload endpoints handle scene images through base64 storage and retrieval systems.

Frontend development used a component-based architecture with reusable Svelte components housed in the /lib/ directory. The EntityCardList component provides dynamic display capabilities for any entity type with integrated relationship visualisation. GameContentForm serves as a universal creation and editing interface for NPCs, players, items, and scenes, while SessionForm offers structured session creation with organised note categories and entity connection management. Interactive relationship management is handled through ConnectionBox and ConnectModal components that provide intuitive interfaces for managing entity relationships. SceneFullscreenView creates immersive viewing experiences with entity overlay capabilities, and SidebarSessions provides efficient session navigation with visual status indicators.

State management throughout the application relies on Svelte's reactive store system and component props, enabling real-time updates whenever entities are created, modified, or connected to other entities. Authentication state is managed through localStorage JWT tokens with automatic inclusion in API requests via a centralised API client implemented in api.js. The page routing structure follows SvelteKit's file-based system, beginning with a landing page that showcases system features, followed by dedicated authentication pages for login and signup that use the custom JWT implementation. The main application flow includes a game dashboard at /my-games displaying user campaigns, detailed game views at /game/[id] providing entity management capabilities, and structured session creation workflows accessible through /session/new.

The testing and optimisation phase involved validation using SvelteKit's built-in development server, with iterative debugging facilitated by browser developer tools and server-side console logging. Each API endpoint underwent thorough validation to ensure proper authentication mechanisms, robust error handling, and data consistency maintenance. Particular attention was paid to entity relationship integrity testing, verifying that connections and disconnections between NPCs, players, items, and scenes maintained proper database consistency and application state. Performance optimisation was achieved through detailed analysis using Lighthouse and Sitespeed.io tools, measuring critical metrics including load times, accessibility scores, and JavaScript bundle sizes. Image optimisation was implemented through base64 encoding for scene graphics, while the component architecture ensures efficient rendering patterns with minimal computational overhead. The MongoDB schema design was optimised for query performance through strategic indexing on frequently accessed fields such as gameId and userId, ensuring responsive user experiences even with large datasets.

The completed implementation represents a DnD session management system that integrates secure JWT-based authentication, entity relationship management, structured session tracking capabilities, and an intuitive user interface designed to support both game preparation activities and live gameplay scenarios. The system's architecture supports scalable growth while maintaining code organisation and data integrity throughout all user interactions.

2.5 Data Collection and Analysis

The data collection phase relied on both developer experience observations and performance measurements throughout the development process. Working with SvelteKit presented a relatively smooth and accessible learning experience that validated the framework's design philosophy of emphasising simplicity and minimal boilerplate code. The built-in file-based routing system and documentation contributed to an intuitive setup process that allowed for rapid understanding of the framework's structure and workflow patterns.

The framework's commitment to simplicity became evident through concrete implementation examples, particularly in the authentication system development. Where traditional frameworks often require extensive configuration and multiple dependencies, SvelteKit enabled a complete JWT authentication system to be implemented in merely 47 lines of code. This authentication utility, housed in auth.js, demonstrates the framework's efficiency through its straightforward token extraction and verification process. The implementation required only essential functions for user identification from request headers, standardised response creation, and error handling, eliminating the boilerplate typically associated with authentication systems in other frameworks.

Component reusability emerged as a significant strength during development, with a high reusability rate were achieved by some strategic component design, exemplified by the EntityCardList component that serves as a universal interface for displaying NPCs, players, items, and scenes through a single, parameterised implementation. Similarly, the GameContentForm component functions as a unified creation and editing interface for all entity types, while the ConnectModal component provides relationship management across different entity combinations. This architectural approach substantially reduced code duplication and maintenance overhead while maintaining type safety through integrated TypeScript support.

The framework's intuitive structure became apparent through the natural organisation during development. The logical separation of concerns across directories, with authentication utilities, API clients, database connections, and schemas each occupying dedicated files, reflects SvelteKit's ability to guide developers toward maintainable architecture patterns. The schema system, implemented in entities.js, served as a single source of truth that eliminated inconsistencies across entity management and reduced entity-related code compared to individual entity definitions.

Performance data collection utilised both synthetic and real-world testing methodologies to capture insights into the application's efficiency characteristics. Google Lighthouse testing revealed the tangible benefits of SvelteKit's compilation approach, with the production build achieving perfect performance scores across all Core Web Vitals metrics. The zero layout shift measurement particularly highlighted the framework's stable rendering pipeline, while minimal blocking time correlated directly with the clean component architecture patterns that SvelteKit encourages. Bundle analysis showed a total application size of 1.3MB with optimised JavaScript chunks, demonstrating efficient resource utilisation despite the application's functionality.

Complementary testing using Sitespeed.io provided multi-run validation that confirmed performance consistency across varying conditions. Five-iteration testing cycles revealed minimal variance in key metrics, with average load times of 3.1 seconds and average First Contentful Paint measurements of 0.25 seconds maintaining tight standard deviations. This consistency indicated robust performance characteristics that would translate well to production environments with varying user conditions and network constraints.

The development process also revealed certain ecosystem limitations that required creative solutions and additional implementation effort. Documentation gaps occasionally necessitated experimentation to resolve complex scenarios, particularly regarding server-side rendering configuration and advanced load function implementations. The decision to implement custom authentication rather than utilising the installed @auth/sveltekit package exemplified the trade-offs inherent in working with newer frameworks, where third-party integrations may not yet match the maturity levels found in established ecosystems. Similarly, entity relationship management required manual implementation across 47 lines of model code, representing functionality that more mature frameworks might provide through pre-built solutions.

Throughout the development process, SvelteKit's integration with modern tooling, including TypeScript, Vite, and hot module replacement, provided immediate feedback cycles that accelerated iteration and debugging workflows. The declarative syntax enables focus on business logic rather than framework-specific patterns or boilerplate management. These qualitative observations, supported by quantitative performance measurements and concrete code examples, formed the foundation for evaluation of SvelteKit's practical applicability in full-stack web development contexts.

3. Results

The development process provided insights into the practical developer experience offered by SvelteKit, and he framework demonstrated a low barrier to entry, with its file-based routing and clear project structure enabling rapid progression from setup to feature implementation. A key finding was the remarkable efficiency in code writing. This was exemplified by the implementation of a custom JWT authentication system. the component architecture proved highly effective for maintainability. A high degree of reusability was achieved through design, for instance, a single EntityCardList component was used to display all entity types (NPCs, players, items, scenes), and a unified GameContentForm handled creation and editing for all content. This approach significantly reduced code duplication.

The integration with modern tooling, such as TypeScript, Vite, and hot module replacement, provided an excellent development feedback loop, allowing for immediate visual updates and streamlined debugging. The declarative syntax of Svelte components allowed the focus to remain on application logic rather than framework-specific patterns.

The development process also identified ecosystem limitations. Encounters with incomplete documentation for advanced scenarios, particularly concerning server-side rendering hooks, occasionally required experimentation and consultation of community resources rather than official guides. The decision to implement a custom authentication solution, despite evaluating the @auth/sveltekit package, underscored the relative immaturity of the third-party library ecosystem compared to more established frameworks like Next.js.

3.1 Lighthouse Results

Lighthouse testing of the production build returned an overall Performance Score of 100/100, indicating optimal front-end efficiency. All Core Web Vitals scored perfectly, with minimal rendering delays and zero layout instability:

Metric Result Interpretation

First Contentful Paint (FCP)	0.212 s
Largest Contentful Paint (LCP)	0.212 s
Time To First Byte (TTFB)	0.197s
Speed Index (SI)	0.6 s

Total Blocking Time (TBT)	0.0 s
Cumulative Layout Shift (CLS)	0.0 s

These results confirm that the SvelteKit application delivers a near-instant load experience, with smooth interaction and stable rendering. The optimised build pipeline and minimal JavaScript usage significantly contributed to these results.

3.2 Sitespeed.io Results

The complementary analysis using Sitespeed.io provided multi-run averages that validated the stability and efficiency of the application:

Coach Score	77 / 100
Performance Score	75 / 100
Privacy Score	78 / 100
Best Practice Score	80 / 100
Fully Loaded Time	~3.1 s
First Contentful Paint	~0.25s
Total Requests	~64 (mostly JS)
Total Page Size	~1.3 MB
CPU Long Tasks / TBT	0 ms
Total Blocking Time	0 ms

The multi-run consistency highlights strong backend performance and stable frontend rendering across sessions. While the overall page weight was slightly above 1 MB due to JavaScript dependencies, loading times remained within ideal ranges for both desktop and mobile contexts.

4. Discussion and Conclusion

4.1 Summary

This study aimed to evaluate the SvelteKit framework through the development of a practical, full-stack web application, a Dungeons amd Dragons campaign tracker. The results from Lighthouse and Sitespeed.io showed exceptional front-end performance, with minimal blocking time, fast content rendering, and no layout shifts. The development process demonstrated that SvelteKit offers a clear and accessible structure with modern tooling, contributing to efficient coding and rapid iteration. Overall, SvelteKit provided both technical performance and a positive developer experience.

4.2 Interpretation of Results

The findings indicate that SvelteKit's compiler-based approach effectively reduces browser workload, resulting in superior loading speed and responsiveness. This aligns with the framework's design philosophy of shifting complexity from runtime to build time.

These performance results directly validate the findings of Xu(2021, pp. 34-36), whose comparative benchmarks identified Svelte's compiler-based architecture as the key differentiator for superior performance. The Cumulative Layout Shift (CLS) score of 0 in this project specifically corroborates Xu's finding that Svelte excels at rendering stability, unlike Vue and Angular, which showed significant layout instability in his study.

The zero blocking time (TBT = 0 ms) highlights that SvelteKit applications can achieve excellent real-world performance even on modest hardware. From a development perspective, its declarative syntax and file-based routing system simplified component creation and page management.

The limited availability of third-party libraries and incomplete documentation occasionally slowed progress. This reflects a common characteristic of newer frameworks, while the core functionality is strong, community maturity and ecosystem support are still developing.

4.3 Implications

The results of this study suggest that SvelteKit is a practical choice for future projects that prioritise performance, simplicity, and maintainability. Its fast load times and smooth rendering make it especially suitable for applications with frequent user interactions or real-time updates.For personal or academic projects, SvelteKit's ease of use and minimal boilerplate offer a highly productive workflow. But For large-scale enterprise applications or long-term commercial projects, the relatively smaller ecosystem may currently present integration challenges. These insights will guide future framework selection decisions by emphasising the trade-off between innovation and ecosystem maturity.

4.4 Limitations

While the results were highly positive, several limitations must be noted. The performance tests were conducted in a local environment, which may not reflect real-world network variability or production server conditions. The application was of moderate complexity, meaning that scalability and long-term maintainability under heavy traffic were not assessed. The study also relied on synthetic performance tools (Lighthouse and Sitespeed.io), which, while reliable, do not fully capture user-perceived latency or mobile network differences. Finally, since SvelteKit is still evolving, future framework updates may alter functionality, performance, or tooling behaviour.

4.5 Future Research

Future studies could expand on these findings by deploying SvelteKit applications in production environments to measure real-world performance and stability or by comparing SvelteKit to alternative frameworks under identical project requirements. Investigating scalability and database integration under high load. Conducting user studies to measure developer productivity and learning efficiency across different skill levels. Spouch research would provide broader insights into SvelteKit's potential in enterprise-scale development and its long-term sustainability in the web ecosystem.

4.6 Conclusion

The research question asked about the benefits and disadvantages of SvelteKit as a viable and effective framework for developing full-stack web applications. The answer is that SvelteKit delivered excellent technical performance, intuitive development patterns, and strong architectural flexibility. While still developing in terms of ecosystem maturity, its simplicity and efficiency make it a compelling option for developers seeking modern, high-performance web solutions. Based on both empirical performance data and hands-on development experience, the conclusion is that the benefits outweigh the disadvantages.

4.7 Final Thoughts

This study highlights the growing potential of compiler-based frameworks like SvelteKit in redefining modern web development. By reducing complexity and focusing on speed and maintainability, SvelteKit demonstrates that cutting-edge performance does not have to come at the cost of developer experience. As web technologies continue to evolve, frameworks that prioritise clarity, performance, and simplicity as SvelteKit likely to shape the next generation of front-end and full-stack development practices.

5. References

- Celander, E. & Möllestål, A., 2024. *A Comparative Analysis of Next.js, SvelteKit, and Astro for E-commerce Web Development*.

- Dékány, M., 2025. *Comparative Analysis of React, Vue.js, and Svelte: Technical Evaluation, Performance, and Developer Experience*. Bachelor's Thesis. Brno, pp.31-35.

- Eze, C., 2024. *SvelteKit vs. Next.js: A side-by-side comparison*. Hygraph. Available at: https://hygraph.com/blog/sveltekit-vs-nextjs [Accessed 17 September 2025].

- Jangra, S., 2025. *React vs Svelte: A Performance Benchmarking*. Available at: https://dev.to/im_sonujangra/react-vs-svelte-a-performance-benchmarking-33n4 [Accessed 17 September 2025].

- Bennett, J. 2025. *React vs Vue vs Svelte: The Ultimate 2025 Frontend Performance Comparison*. Medium. Available at: https://medium.com/@jessicajournal/react-vs-vue-vs-svelte-the-ultimate-2025-frontend-performance-comparison-5b5ce68614e2 [Accessed 17 September 2025].

- Kitchenham, B., Pickard, L. & Pfleeger, S.L., 2002. Case studies for method and tool evaluation. *IEEE Software*, 12(4), pp.52–62.
doi.org/10.1109/52.391832

- Next.js, 2025. *Community and Documentation*. Available at: https://nextjs.org/docs/community[Accessed 17 September 2025].

- Putra, F.P.E., Hasbullah, Farhan, M. & Paradina, R., 2025. *Technical Performance Comparison of Modern Frontend Frameworks:* Study on Svelte, React, and Vue. *Brilliance: Research of Artificial Intelligence*, 5(1), pp.355–364.
doi.org/10.47709

- Svelte. 2025a. Introduction — SvelteKit documentation. Available at: https://svelte.dev/docs/kit/introduction [Accessed: 4 October 2025].]

- Svelte, 2025b. *Svelte Overview*. Available at: https://svelte.dev/docs/svelte/overview [Accessed 17 September 2025].

- Svelte, 2022. *Svelte* blog. Available at: https://svelte.dev/blog/announcing-sveltekit-1.0 [Accessed 17 September 2025].

- Sitespeed.io. 2024. *Web performance testing in practice*. Available at: https://www.sitespeed.io/documentation/sitespeed.io/web-performance-testing-in-practice/ [Accessed: 4 October 2025].

- Google Chrome. 2019. Lighthouse performance scoring. Available at: https://developer.chrome.com/docs/lighthouse/performance/performance-scoring [Accessed: 4 October 2025].

- Xu, W. 2021. *Benchmark Comparison of JavaScript Frameworks: React, Vue, Angular and Svelte*. Master's Thesis. University of Dublin, pp.34-36.

6. Appendix

Cedenblad, A., 2025. Individual In-Depth Study: DnD DM Game Tracker [Source code]. GitHub. Available at: https://github.com/CedenbladAlbin/Individual-InDebth-study.git [Accessed 18 September 2025].