BWMA 2024/2025

Author: HULOT Caroline
Matrikelnumber: 1525230
FD-Number: FD0002129
caroline.hulot@informatik.hs-fulda.de

ABSTRACT

The landscape of web development is continuously evolving with new frontend technologies emerging and existing ones being refined. Staying ahead of trends is crucial for developers and businesses alike. In 2025, key advancements in performance, maintainability, and scalability will shape the future of web development.
This paper explores which frameworks and architectures will remain or gain popularity, and why. We delve into popular and emerging frameworks such as React, Vue.js, and Svelte, as well as architectures like Jamstack, Progressive Web Apps (PWAs), and server-side rendering (SSR). Each framework's potential use cases, benefits, and limitations are analyzed, with a specific focus on building robust and scalable e-commerce platforms. Comprehensive documentation and ecosystem maturity are discussed to guide developers in selecting technologies aligned with project requirements.
By understanding these trends and leveraging cutting-edge technologies, developers can make informed decisions to build robust, scalable, and future-proof web applications.

1. INTRODUCTION

The rapid evolution of web technologies demands continual adaptation. For 2025, we aim to build a high-quality e-commerce website focused on speed, responsiveness, safety, and ease of use. We will identify dominant frontend frameworks and architectures to assess their suitability.
Standards like responsive design, Progressive Web Apps (PWAs), and JavaScript frameworks remain essential, while new trends came to reshape the field. This paper aims to recommend a tech stack to address performance issues, boost developer productivity, and ensure high security, delivering excellent user experiences and easy maintenance. Component-based development, modular architecture, and edge computing remain vital, with new trends like meta-frameworks and serverless solutions gaining traction. Our goal is to design a frontend stack that enhances user experience, optimizes performance, and ensures maintainability for long-term business success.

2. APPLICATION

The application we are focusing on is a dynamic e-commerce platform for both desktop and mobile users, designed to offer safe and enjoyable online shopping that meets today's customer needs. This platform is designed to handle high traffic, dynamic content, and secure transactions.

2.1 The key requirements for this application

2.1.1 Performance

Achieve lightning-fast load times and ensure a seamless browsing experience across all devices, especially under heavy traffic.

2.1.2 Scalability

Maintain reliable performance for thousands of concurrent users, even during peak shopping events such as sales or holidays.

2.1.3 Security

Protect user data and transactions against malicious threats like Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and other security threats to ensure safe transactions.

2.1.4 Maintainability

Provid an easy-to-update user interface and backend integration for efficient management and maintenance.

2.1.5 Flexibility

Support dynamic, personalized content to enhance user engagement and improve conversion rates.

2.1.6 Accessibility

Ensure the platform is usable by individuals with diverse abilities and disabilities.

2.2 Clients

Our clients include businesses such as online sellers and marketplaces. These clients require platforms that offer rich user interfaces and integrate a payment systems, inventory management, and analytics tools.

The e-commerce platform will feature real-time updates, interactive interfaces, and flawless user experiences, ensuring that users can navigate and complete transactions efficiently and securely.
This focus on performance, scalability, security, maintainability, flexibility and accessibility ensures that the platform not only meets but exceeds modern consumer expectations, positioning businesses for long-term success.

3. TECHNOLOGY AND ARCHITECTURE

To meet the requirements of a modern e-commerce platform in 2025, we have selected a robust tech stack that ensures performance, scalability, security, and maintainability.

3.1 Frameworks and Libraries

- **React**:
  - Component-based architecture enables reusable UI elements.
  - Extensive ecosystem with support for libraries like Next.js for server-side rendering (SSR) and Tailwind CSS for styling.
- **Next.js (Jamstack)**:
  - Combines static site generation (SSG) and SSR for optimal performance and SEO.
  - Provides dynamic routing, API routes, and advanced features like image optimization.
- **Tailwind CSS**:
  - A utility-first CSS framework to streamline styling and ensure design consistency.
  - Reduces CSS bloat and accelerates UI development.
- **Node.js + Express.js**:
  - Used for building a RESTful backend API to handle server-side logic and data processing.
  - Manages secure communication with the frontend.
- **NPM Libraries**:
  - **RxJS**: For managing asynchronous data streams.
- **State Management**:
  - **Redux Toolkit**: For managing complex global states.
  - **Context API or Zustand**: For localized state management in specific components.

3.2 Architecture

The platform adopts a Jamstack architecture to ensure performance, scalability, and maintainability:

- **Frontend**: Built using React and Next.js to leverage hybrid rendering (SSG + SSR). Designed with a component-based structure for modularity and reusability.
- **Backend**: A Node.js and Express.js backend serves REST APIs, handling CRUD operations and dynamic data processing.
- **Dynamic Interactions**: AJAX and WebSocket APIs provide real-time data updates and interactivity.

4. MAIN

In selecting the technology stack for the modern e-commerce platform described in this paper, we focused on optimizing performance, scalability, security, and maintainability. Below, we break down each technology choice with a critical analysis, highlighting the benefits and potential drawbacks.

4.1 React [1]: Component-Based UI Framework

Pros:

- **Component Reusability:** React's component-based architecture allows us to create highly modular user interfaces. Components can be reused across the platform, minimizing redundancy and making the system easier to maintain.
- **Rich Ecosystem:** React has a massive ecosystem with support for libraries and various testing tools. This gives developers a wide range of options for building scalable applications.
- **Performance Optimization:** React' ensure high performance on dynamic platforms.
- **Strong Community Support:** React's popularity ensures strong community support, which is vital for troubleshooting and continuous learning. Documentation and online resources are abundant.

Cons:

- **Difficulty:** While React is powerful, may be difficult, especially when combined with state management tools like Redux.
- **Extra Code:** For bigger projects, React may need a lot of extra code.

4.2 Next.js [2] (Jamstack)

Pros:

- **Hybrid Rendering (SSG + SSR):** Next.js provides a powerful combination of Static Site Generation and Server-Side Rendering. It makes it possible to generate static pages for faster, while dynamically rendering pages when necessary for content personalization.

Cons:

- **Complexity with Dynamic Content:**Handling dynamic content in a server-side rendered app can add complexity.
- **Longer Build Times for Large Sites:** Pre-generating static pages for large e-commerce websites can result in longer build times, especially if content changes frequently.

4.3 Tailwind CSS [3]

Pros:

- **Utility-First Approach:** Tailwind CSS promotes a utility-first approach to styling, which reduces the need for custom CSS and allows for faster UI development.
- **Customization and Flexibility:** Tailwind offers an extensive range of configuration options.
- **Improved Developer Productivity:** Tailwind reduces CSS bloat by eliminating unused styles, which enhances performance.

Cons:

- **Learning Curve for New Developers:** The utility-first approach may be difficult for developers who are used to traditional CSS methodologies.
- **Large HTML Files:** The use of numerous utility classes in HTML can make the markup bloated and harder to read, especially for larger components.

4.4 Node.js [4] + Express.js [5]

Pros:

- **Scalability:** Node.js is built on a non-blocking, event-driven architecture, making it well-suited for handling a large number of concurrent requests.
- **Asynchronous and Real-Time Communication:** Node.js's asynchronous nature and support for WebSockets enable real-time features, enhancing the user experience.
- **RESTful API Development:** It simplifies the creation of REST APIs, which are used for interacting with the database and providing dynamic content for the frontend..

Cons:

- **Callback :** Despite using async/await, handling complex asynchronous code can still result in issues with callback management.
- **Single-Threaded Limitation:** Node.js's single-threaded nature may become a bottleneck for CPU-intensive operations.

4.5 NPM Libraries: RxJS [6]

Pros of RxJS:

- **Reactive Programming:** RxJS is useful for handling complex asynchronous data streams, making it ideal for managing real-time updates, such as inventory changes, order status updates, or user notifications.
- **Streamlining Asynchronous Operations:** RxJS simplifies error handling and chaining of asynchronous operations, which is especially valuable for building real-time features and complex state management.

Cons of RxJS:

- **Learning RxJS:** RxJS's reactive programming can be difficult for developers unfamiliar with the concept. Complex streams of data can make debugging harder if not carefully managed.

4.6 Security Considerations

For a high-traffic e-commerce website, security is a top priority. Utilizing OWASP [7]recommended libraries for mitigating vulnerabilities like XSS (Cross-Site Scripting) and CSRF (Cross-Site Request Forgery) is essential.

4.7 Exclusion of Alternative Technologies

While evaluating the technology stack for this modern e-commerce platform, several other technologies were considered but ultimately excluded for specific reasons:
Vue.js [8] : Vue.js is an excellent framework for smaller projects or teams that require quick development cycles. It is better suited for projects that prioritize simplicity and smaller teams.

Svelte [9] Svelte is an emerging framework that compiles code to optimized JavaScript at build time, offering outstanding performance with minimal runtime. But its community support is not as extensive as React's.

jQuery jQuery was once a major of web development, but with the advancements in modern JavaScript frameworks like React and Next.js, jQuery has become largely obsolete for building complex, modern web applications.

5. CONCLUSIONS

As we move into 2025, the landscape of web development continues to evolve, with new technologies and architectures driving improvements in performance, scalability, and user experience. For e-commerce platforms, the combination of React, Next.js, Tailwind CSS, Node.js, and Express.js offers a powerful and flexible tech stack capable of meeting the demands of modern users. By adopting a Jamstack architecture, businesses can ensure fast load times, dynamic content delivery, and robust security features, all while maintaining ease of maintenance. Ultimately, the choices made in frontend frameworks and backend technologies will determine the success of e-commerce platforms in delivering seamless, engaging, and secure user experiences that meet the high expectations of today's consumers.

6. REFERENCES

[1] React, "React documentation," https://reactjs.org/docs/getting-started.html, [Accessed: Jan. 19, 2025].

[2] Next.js, "Next.js documentation," https://nextjs.org/docs, [Accessed: Jan. 19, 2025].

[3] T. CSS, "Tailwind css documentation," https://tailwindcss.com, [Accessed: Jan. 19, 2025].

[4] Node.js, "Node.js documentation," https://nodejs.org, [Accessed: Jan. 19, 2025].

[5] Express.js, "Express.js documentation," https://expressjs.com, [Accessed: Jan. 19, 2025].

[6] RxJS, "Rxjs documentation," https://rxjs.dev, [Accessed: Jan. 19, 2025].

[7] OWASP, "Owasp security practices," https://owasp.org, [Accessed: Jan. 19, 2025].

[8] Vue.js, "Vue.js guide," https://vuejs.org/guide/introduction.html, [Accessed: Jan. 19, 2025].

[9] SvelteKit, "Sveltekit documentation," https://kit.svelte.dev/docs, [Accessed: Jan. 19, 2025].