Prague University of Economics and Business

Faculty of Informatics and Statistics

COMPARATIVE ANALYSIS OF TRENDING WEB DEVELOPMENT FRAMEWORKS: ANGULAR, REACT, AND VUE.JS

MASTER THESIS

Study programme: FIS

Field of study: Information Systems Management

Author: Amad Ul Hassan

Master Thesis Supervisor: PhDr. Antonín Pavlíček

Prague, May 2024

Acknowledgement

First and foremost, I wish to express my deepest gratitude to Allah Almighty for the strength, wisdom, and endurance bestowed upon me during my academic journey.

I am profoundly grateful to my thesis supervisor, Ing. PhDr. Antonín Pavlíček, Ph.D., for his invaluable guidance, patience, and expert advice throughout the research and writing of this thesis. His mentorship was instrumental in the completion of this work.

Special thanks go to my family—my sister and my parents—who have provided unwavering support and encouragement throughout my studies. Their belief in me and constant motivation regarding my work on the topic of Web Development have been a continuous source of strength.

Abstract

In the rapidly evolving landscape of web development, the choice of a framework plays a pivotal role in shaping the success of a project. This study conducts a comprehensive comparative analysis of three of the most prominent front-end frameworks: Angular, React, and Vue.js. Through an in-depth examination of their architecture, performance, ecosystem, and community support, we aim to provide valuable insights for developers, businesses, and technology enthusiasts.

The study employs a systematic approach, evaluating each framework based on key parameters such as ease of learning, scalability, flexibility, and developer experience. Real-world use cases and industry trends are scrutinized to highlight the practical implications of choosing one framework over another. Additionally, the research delves into the latest updates and features introduced by Angular, React, and Vue.js, shedding light on their adaptability to emerging web development challenges.

The analysis extends beyond technical aspects, exploring the broader developer community's sentiments and preferences. By examining online discussions, forums, and community engagement, the study captures the pulse of the developer ecosystem surrounding each framework. Furthermore, it investigates the impact of these frameworks on job markets and career opportunities.

Through this comparative analysis, our goal is to empower developers, businesses, and decision-makers with a nuanced understanding of Angular, React, and Vue.js. Whether embarking on a new project or considering a migration, the findings of this study offer valuable guidance for making informed decisions in the dynamic realm of web development.

Keywords

Web development frameworks, Angular, React, Vue.js, front-end development, comparative analysis, architecture, performance, ecosystem, community support, ease of learning, scalability, flexibility, developer experience, real-world use cases, industry trends, updates, features.

Table of Contents

List of Figures

List of Abbreviations

AJAX	Asynchronous JavaScript And XML
AOT	Ahead of Time
API	Application Programming Interface
CLI	Command Line Interface
CLS	Cumulative Layout Shift
DOM	Document Object Model
FIS	Faculty of Informatics and Statistics
FCP	First Contentful Paint
CZSO	Czech Statistical Office
HTML	Hypertext Mark-up Language
HTTP	Hypertext Transfer Protocol
KPI	Key Performance Indicator
LCP	Largest Contentful Paint
MVC	Model View Controller
TBT	Total Blocking Time
TTI	Time to Interactive
UI	User Interface
XML	Extensible Markup Language

Chapter 1: Introduction

The narrative of web development unfolds as a tale of metamorphosis, traversing from the early days of ad-hoc coding practices to the present, characterized by a more sophisticated and organized paradigms. (Jaiswal, 2022) aptly characterizes this transformation as more than a mere stylistic shift; it is a response to the imperative need for heightened efficiency and scalability in the face of the escalating complexity of web applications.

During the days of web development professionals practiced a form of artistry where they crafted websites using improvised coding techniques. While this approach allowed for creativity it also resulted in a development environment. Each project presented challenges often forcing developers to reinvent the wheel to meet requirements. As web applications progressed from pages to platforms the limitations of improvised coding practices became evident. The need, for efficiency and scalability grew as developers realized that traditional coding methods were not suitable for managing the complexities of web development (Baida, 2020).

In response to these challenges frameworks emerged as elements in this journey. Developers no longer had to start from scratch with each project; instead, they adopted frameworks as building blocks. These frameworks offered tools, best practices, design patterns and reusable components. They provided a foundation for an organized and systematic development process. Frameworks played a role, in bringing order to what was a chaotic landscape. They offered developers an established structure that encouraged consistency and adherence, to established practices (Baida, 2020). This not improved the readability of the code. Also laid the foundation for a collaborative development environment where teams could seamlessly work within a shared framework.

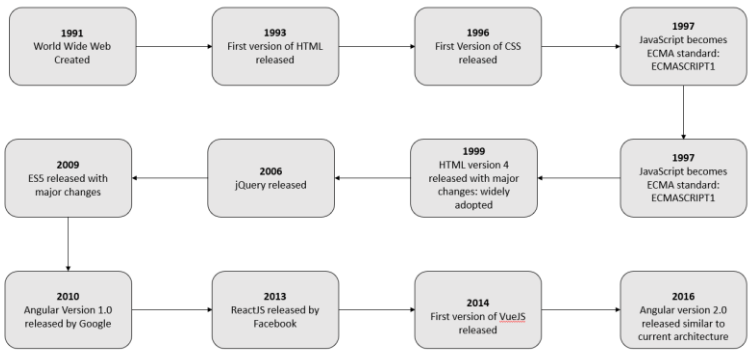

1991
World Wide Web Created
1993
First version of HTML released
1996
First Version of CSS released
1997
JavaScript becomes ECMA standard: ECMASCRIPT1
1997
JavaScript becomes ECMA standard: ECMASCRIPT1
1999
HTML version 4 released with major changes: widely adopted
2006
jQuery released
2009
ES5 released with major changes
2010
Angular Version 1.0 released by Google
2013
ReactJS released by Facebook
2014
First version of VueJS released
2016
Angular version 2.0 released similar to current architecture

Figure 1 History of Web Development

Source: (Rathinam, Analysis and Comparison of Different Frontend, 2022)

The adoption of frameworks also addressed the need, for efficiency and scalability. By encapsulating proven methodologies and design patterns frameworks enabled developers to organize code into modules and encourage reusability (Baida, 2020). This change significantly reduced the time and effort required for development facilitating project lifecycles. Essentially the progress of web development signifies a shift, from the craftsmanship of the past to an organized, cooperative and effective present. Frameworks, often overlooked but important in this story still have a role, in shaping the world of web development. They offer developers the resources and groundwork to navigate the intricacies of the realm.

1.1 Research Aims & Objectives

The overarching goal of this research is to provide nuanced insights and practical recommendations that empower web designers and companies to make informed decisions when selecting a web development framework. The focus is on offering tailored guidance that aligns with the unique requirements and objectives of diverse projects.

1.1.1 Specific Objectives

- **Guidance for Selecting a Web Development Framework:** The main purpose of this aspect is to analyze, provide insights and offer personalized suggestions to web designers and companies when it comes to choosing the right web development framework. We will delve into the strengths and weaknesses of Angular, React and Vue.js considering factors such as scalability, adaptability to project types and industry specific use cases. Our aim is to provide nuanced guidance that takes all these aspects into account.
- **Evaluation of Core Features, Architecture and Capabilities:** This involves conducting an assessment of the core features, architecture and capabilities offered by Angular, React and Vue.js. We will thoroughly explore the principles behind each framework while closely examining their core features and capabilities. Our goal is to provide developers and decision makers with an understanding of how each framework addresses challenges in web development so that they can make well informed choices.
- **Performance Testing and Benchmarking:** The objective here is to evaluate the efficiency, performance and speed of applications developed using Angular, React and Vue.js. We will implement rigorous performance testing procedures along with benchmarking techniques to generate data, for analysis.
- The performance metrics of applications developed using each framework will be compared, taking into account factors such, speed to perform certain task and time taken to load web pages under certain conditions. The goal is to gather data that can objectively evaluate the performance of each framework.

- **Evaluation of Developer Experience:** This entails assessing the developer experience by considering how easy it is to learn each framework the level of community supports the usability and availability of tools. We will look into the learning curves associated with Angular, React and Vue.js. Additionally, we will examine the strength and responsiveness of their communities. The aim is to evaluate the comprehensiveness and effectiveness of development tools and resources while providing insights into the developer experience.
- **Examination of Real-World Case Studies:** The objective is to analyze real world case studies that demonstrate adoption of Angular, React and Vue.js. In depth case studies will be conducted consisting of interviews and surveys with companies and developers who have implemented these frameworks. The focus will be, on exploring challenges faced during implementation solutions applied and overall project outcomes. This objective aims to extract insights and lessons learned from real world implementations of each framework.

1.1.2 Expected Outcomes

The main objective of this research is to provide web designers and companies with a framework that assists in making decisions. It considers project requirements and industry nuances aiming to enhance decision making abilities. The goal is to empower developers with an understanding of Angular, React and Vue.js – their core features, architecture and capabilities. This knowledge will help developers choose the framework, for their project goals.

The research also focuses on providing data on application performance. By comparing Angular, React and Vue.js in terms of efficiency, resource utilization and responsiveness it aims to offer insights for evaluating each frameworks performance more accurately.

In addition to performance analysis, the research strives to shed light on developer user experience. It examines aspects such as the learning curve, community support and availability of tools, for each framework. By extension, this examination aims to enrich the developer experience by offering an understanding of the ecosystem surrounding each framework.

Moreover, this research seeks to compile a repository of real-world case studies that showcase implementations of Angular, React and Vue.js. Through these experiences it aims to extract insights lessons learned from best practices to provide real world perspectives that can be beneficial.

By accomplishing these goals this study aims to offer a finite conclusion that surpasses mere theoretical considerations. It provides insights to improve the decision-making procedures of web designers and companies, in the field of web development frameworks.

1.2 Significance of the study

The significance of this study is underscored by its potential to address critical gaps in knowledge and provide substantial benefits to various stakeholders within the web development landscape.

The following points emphasize the significance of this research:

1. **Empowering Stakeholders' Decision Making:** This study equips web designers and developers, with insights for making decisions when choosing a web development framework. By analysing Angular, React and Vue.js it enables them to align framework choices with project requirements leading to optimal outcomes.
2. **Streamlining Framework Selection:** Companies and organizations can streamline their process of selecting a web development framework by utilizing the framework provided in this study. This is essential for maximizing development efficiency ensuring scalability and aligning choices with term strategic objectives.
3. **Advancing Development Practices:** A comprehensive exploration of core features, architecture and capabilities contributes to the knowledge base of the web development community. This fosters a shared understanding of practices with the aim of keeping the community up to date with technological advancements, in web development.
4. **Optimizing Performance:** Developers and performance engineers benefit from the performance testing and benchmarking aspect of this study as it offers data that allows for an understanding of how each framework performs in real world scenarios. This information is incredibly valuable, in optimizing the performance of applications and making the use of resources.
5. **Enhanced Developer Experience:** Institutions offering web development courses can benefit from understanding the learning curves community support and availability of tools for frameworks. This helps improve the experience by aligning curriculum content with industry needs preparing students for real world situations.
6. **Strategic Adoption of Frameworks:** Real world case studies that showcase implementations serve as references for decision makers in industries. Organizations can learn from these examples identify practices and strategically choose frameworks that align with their business objectives.
7. **Contribution to Academic Knowledge:** This research adds to the existing knowledge by providing a nuanced understanding of the landscape of web development frameworks. It has the potential to stimulate research, discussions and academic exploration into emerging trends and challenges within this field.
8. **Ensuring Long term Technological Sustainability:** Technology leaders and innovators need to understand both the strengths and weaknesses of web development frameworks. The insights gained from this study can guide decisions

to ensure that adopted technologies remain sustainable and relevant, within the changing digital landscape.

In summary this study holds importance not for its immediate practical applications but also, for its broader impact on various fields such as web development, education, industry practices and academic discussions. The results have the potential to shape decision making processes encourage innovation and play a role, in the development of web development practices.

1.3 Methodology

The research methodology, in this study aims to examine three web development frameworks; Angular, React and Vue.js. Inspired by the works of Garcia et al, (2020) our goal is to demystify the complexities associated with choosing a web development framework. By conducting an analysis, we strive to offer practical insights that go beyond personal preferences. In the chapters we will delve into the intricacies of each framework evaluate their efficiency through performance testing and benchmarking and consider developer experience factors such as the learning curve, community support and available tools (Jaiswal, 2022).

It is important to note that selecting a web development framework is not a one-time decision but an on-going strategic process. This research's aim is to provide nuanced insights, into the strengths and weaknesses of Angular, React and Vue.js so that web designers and companies can make decisions based on their project requirements and goals.

In the following chapters we will explore the core architectures of these frameworks in detail. We will also conduct performance evaluations and present real-world case studies to illustrate the impact of these choices.

The primary objective is to provide readers with the knowledge and understanding to effectively navigate the world of web development frameworks. This will help foster innovation and excellence, in the realm.

To begin an in-depth review of existing literature is conducted, which establishes a foundation by exploring the evolution, key concepts and theoretical frameworks relevant to web development and the selected frameworks. After this literature review a comprehensive comparative analysis is carried out to assess the strengths and weaknesses of Angular, React and Vue.js. This analysis delves into principles core features and capabilities with the aim of providing a nuanced understanding of each framework's characteristics.

Thorough performance testing and benchmarking procedures are aspects of this study. They aim to evaluate the efficiency, in the form of performance and speed of applications developed using Angular, React and Vue.js. By collecting and analysing data, it becomes possible to compare performance metrics across applications built with each framework. This approach provides insights into their performance.

Moreover, an evaluation of developer experience is included in this study. It involves analysing factors such as ease of learning, community support availability, and usability of tools for each framework. This examination encompasses investigating learning curves, community dynamics and tool effectiveness that contribute insights into developers' overall experience, with Angular, React and Vue.js.

Detailed case studies are conducted to analyse real life examples of implementation of Angular, React and Vue.js. By conducting interviews and surveys, with companies and developers the study explores the challenges encountered the solutions applied and the overall impact on project success. These case studies provide insights and valuable lessons based on real world scenarios.

The collected data, which includes performance metrics survey responses and findings from case studies, undergoes an analysis. Qualitative data is analysed thematically while quantitative data is statistically examined to identify patterns and make comparisons. The resulting findings are interpreted within the research objectives context.

Throughout the research process ethical considerations play a role. Prior consent is obtained from participants involved in case studies and surveys to ensure confidentiality and privacy. Adherence to standards is maintained at every stage of data collection and analysis while addressing any biases transparently.

This study acknowledges that there are limitations in any research endeavour. Factors such as technologies evolving nature variations in project contexts and potential biases, in responses are carefully considered when interpreting results. This approach contributes to maintaining transparency and integrity throughout the study.

In conclusion, the research methodology unfolds as a multifaceted approach, incorporating literature review, comparative analysis, performance testing, developer experience evaluation, real-world case studies, and rigorous data analysis. This holistic methodology is designed to generate comprehensive insights for the benefit of web designers, developers, companies, and the broader web development community.

Chapter 2: Literature Review

The modern landscape of web development is shaped by the nature of frameworks. These frameworks are platforms that guide the creation of web applications. They consist of libraries, tools and predefined structures providing developers with an approach, to building scalable and maintainable software. Following paradigms like the Model View Controller (MVC) pattern these frameworks greatly contribute to organizing code and ensuring project maintainability. In the world of web development frameworks play a role by offering platforms that streamline application development. They bring together libraries, tools and predefined structures to help developers create software that's both efficient and easy to manage. (Jaiswal, 2022)

In the field of web development choosing the framework has a significant impact on project success. Conducting a comparison between frameworks is essential in understanding their unique features, strengths and weaknesses. This analytical process provides insights into factors such as performance, learning curve, community support, scalability, flexibility and integration capabilities. With this knowledge at hand stakeholders can make decisions tailored to their project requirements.

We have selected Angular, React and Vue.js for this analysis due to their prominence, in the industry well as their distinct attributes. Angular is developed and maintained by Google; it is widely recognized for its robustness and extensive set of features.

React, a product developed by Facebook is known for its proficiency, in constructing user interfaces using a DOM, which helps optimize performance. On the hand Vue.js, although new has garnered recognition for its simplicity and adaptability. This choice allows for an evaluation that takes into account project requirements and developer preferences. (J. Kalinowska, 2019)

This literature review serves a purpose within the framework of analysis. Firstly, it provides context by outlining the milestones in the development of web development frameworks. Secondly it establishes the foundations for conducting the comparative analysis through an exploration of relevant frameworks and concepts. The aim of this review is to enhance our understanding of the strengths and weaknesses, in Angular, React and Vue.js. By providing insights it aims to empower developers and organizations to make informed decisions that align with their specific project needs. (J. Kalinowska, 2019)

2.1 Historical Evolution of Web Development Frameworks

The evolution of web development frameworks is a story, in the changing world of software engineering. In this section we'll explore the context of these frameworks. Understand why they have become such important tools for building web applications. By looking at the development journey of Angular, React and Vue.js we'll gain an appreciation for how these frameworks have evolved into tools. (J. Kalinowska, 2019)

Web development frameworks have become essential in software engineering as they provide developers with an efficient approach, to creating web applications. These frameworks come with defined structures, libraries and tools that promote code reusability, scalability and maintainability. This introduction highlights the role that web development frameworks play in navigating the complexities of today's web application development. (Kozlowki, 2013)

Furthermore, it is safe to surmise that evolution of web development frameworks has revolutionized the way web applications are designed, developed, and delivered. This transformation is anchored in the broader trends of software engineering, emphasizing efficiency, scalability, and user experience. From the static HTML pages of the early '90s to the dynamic, component-based architectures of today, the journey of web development frameworks like Angular, React, and Vue.js reflects a continuous pursuit of innovation and usability.

The foundation of web development in the early 1990s was marked by static HTML pages, offering limited interactivity and functionality. JavaScript emerged as a scripting language to enhance the interactivity of web pages, setting the stage for more complex applications (Flanagan, 2006). The period also saw the rise of CSS for styling, which, alongside HTML and JavaScript, formed the triad of cornerstone technologies for web development.

This was quickly followed by the introduction of Asynchronous JavaScript and XML (AJAX) in the early 2000s represented a paradigm shift, enabling web applications to load data in the background, thereby improving user experience with smoother interactions (Garrett, 2005). jQuery, released in 2006, further streamlined web development by offering a unified framework for animations, event handling, and DOM manipulation, making JavaScript programming more accessible (Resig, 2007).

The development of Angular, React, and Vue.js has had a profound impact on the web development landscape, offering tools that democratize complex development tasks. These frameworks have enabled developers to build sophisticated, scalable, and maintainable web applications more efficiently than ever before. The choice among these frameworks often hinges on specific project requirements, developer preferences, and the distinct features and benefits they offer.

In conclusion, the historical evolution of web development frameworks from static pages to advanced, dynamic web applications highlights the rapid pace of innovation in the field of software engineering. Angular, React, and Vue.js each represent pivotal developments in this timeline, reflecting the industry's move towards more interactive, efficient, and user-centric web applications. The journey of these frameworks is a testament to the continuous evolution of web development practices, driven by the needs of users and developers alike.

2.2 Evolution of Front-End Development Frameworks

The evolution of front-end development frameworks mirrors the broader evolution of web technologies. From the static HTML pages of the early web to the dynamic, interactive interfaces of today, front-end development has undergone paradigm shifts in response to the growing demands for enhanced user experiences. This exploration of the evolution of front-end frameworks illuminates the transition from traditional server-side rendering to client-side rendering, the advent of AJAX, and the rise of single-page applications (SPAs). Understanding these shifts is crucial for contextualizing the development of specific frameworks within the broader historical narrative. (Wohlgethan, 2018)

Frameworks are not mere tools; they are catalysts for innovation within the development community. According to (Kozlowki, 2013), frameworks encapsulate proven methodologies and foster a collaborative development ecosystem. By standardizing development practices, frameworks contribute to the creation of robust, scalable, and maintainable applications. This chapter explores the transformative impact of frameworks on the development process, emphasizing their role in enhancing code quality and contributing to the long-term success of a project.

In the vast and ever-expanding ocean of available frameworks, Angular, React, and Vue.js emerge as lighthouses guiding developers through the complex waters of decision-making. The choice between these frameworks is far from arbitrary; it is a strategic decision influenced by project requirements, team expertise, and the overarching goals of the application under development. As elucidated by (Wohlgethan, 2018) this strategic decision-making process involves a careful consideration of the unique strengths and characteristics of each framework in relation to the project's specific needs.

2.3 Angular: The Opinionated Powerhouse

In 2010, Google released AngularJS, introducing two-way data binding and a model-view-controller (MVC) architecture that simplified the development of single-page applications (SPAs) (Seshadri, 2014).

AngularJS's declarative programming style and dependency injection mechanism offered a robust platform for building large-scale web applications. Angular, which originated from Google’s innovation hub serves as an example of an architecture that firmly follows the Model View Controller (MVC) paradigm.

This section delves into Angular’s complexities drawing inspiration from the insights provided by (Simkhada, 2016). Our exploration aims to grasp how Angular’s comprehensive and structured approach establishes a foundation, beneficial, for building large scale enterprise applications.

Angular has undergone a transformation since its early days as AngularJS to the sophisticated Angular 2 and subsequent versions. During its phase from 2010 to 2012 Angular was born out of efforts by Google engineers Misko Hevery and Adam Abrons. They aimed to tackle challenges faced in developing single page applications (SPAs). The outcome was AngularJS, the predecessor to todays modernized Angular officially launched in 2010. (Shahzad, 2017) AngularJS also brought forth concepts like data binding and modular architecture. It quickly gained traction within the developer community, becoming a catalyst for a paradigm shift in web development. Its distinctive features such, as two-way data binding, directives and dependency injection played roles in its popularity. (Cincovic, 2019)

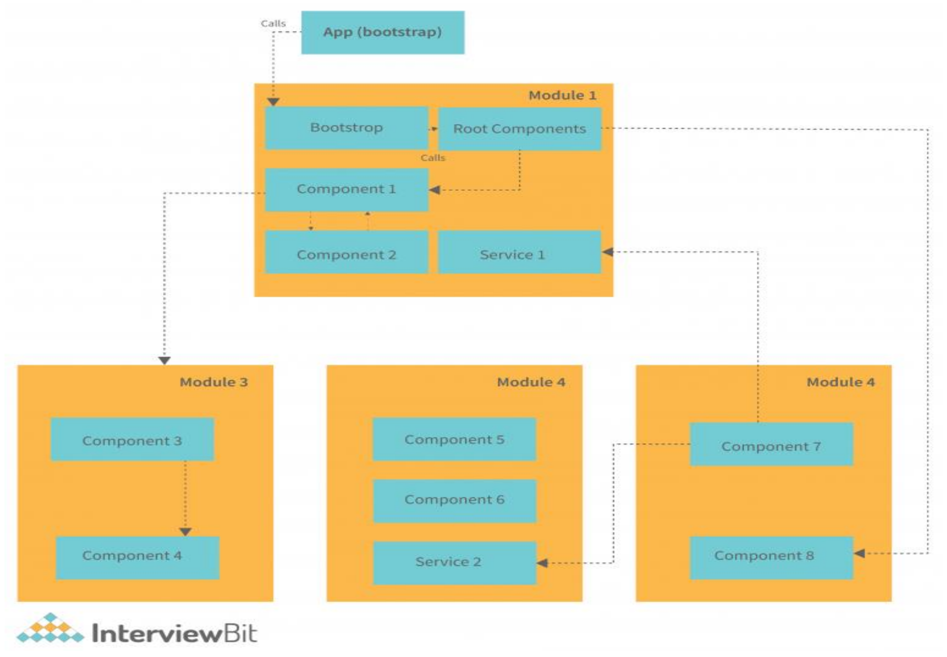


Figure 2: Angular’s Module-inclined structureInterviewBit

Source: (Interview Bit, 2024)

Google's support and the development of an ecosystem, around the framework established AngularJS as an option for developers who wanted to improve the structure and interactivity of their web applications.

However, as AngularJS applications became more complex challenges arose. The bidirectional data binding while powerful sometimes caused performance issues. Realizing the need for an overhaul the Angular team made an announcement in 2014—the introduction of Angular 2. This decision sparked discussions within the community. Demonstrated a commitment to addressing existing challenges while redefining Angular for the future. (Cincovic, 2019)

Angular 2 launched in 2016 was not an update; it involved rewriting the entire framework from scratch. One major change was adopting TypeScript, which's a typed superset of JavaScript. This transition brought benefits such as typing, interfaces and improved tooling that helped developers create structured and maintainable codebases. The Angular team followed a versioning approach that led to updates and enhancements. Subsequent versions like Angular 4 5 6 introduced new features, optimizations and tools while keeping modularity at its core, for building maintainable applications.

A significant advancement arrived with Angular Ivy in 2018 with the release of Angular 8. This update brought improvements, including reducing the size of software packages enhancing ahead of time (AOT) compilation and improving the efficiency of change detection. The ongoing development efforts are focused on refining Ivy addressing performance limitations and keeping up with evolving web standards.

The evolution from AngularJS to Angular 2+ marked a significant milestone, with the latter adopting TypeScript and a more component-based architecture, offering enhanced performance and scalability (Google, 2016). React's ecosystem expanded with the introduction of hooks in React 16.8, enabling state management and side-effects in functional components without classes (Facebook, 2019). Vue.js's continued development led to the release of Vue 3, featuring the Composition API for improved code organization and reusability (You, 2020).

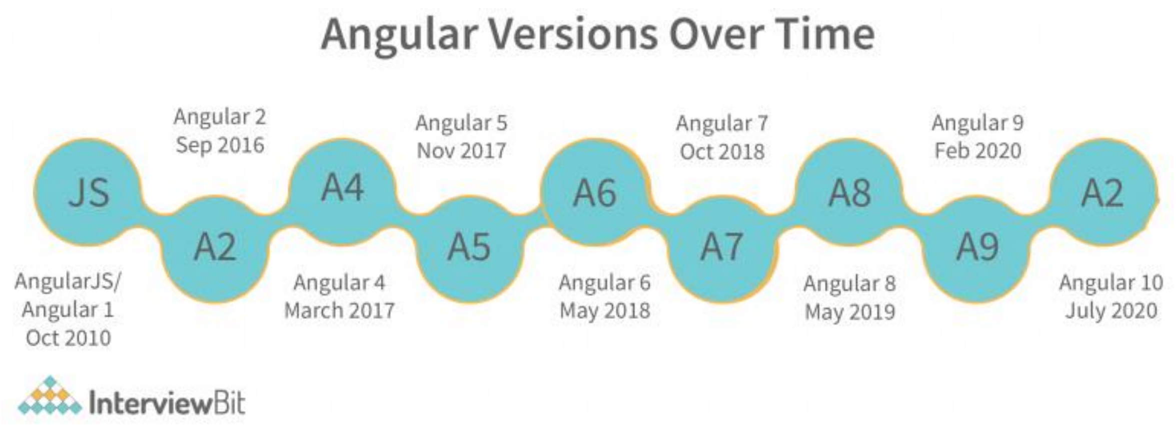


Figure 3. Angular's Version Evolution

Source: (Interview Bit, 2024)

The story of Angular is not, about advancements; it also highlights community and industry acceptance. Its powerful features, combined with the advantages of TypeScript have led to its adoption in both large-scale enterprise applications and individual projects. The active Angular community consisting of developers, contributors and advocates has played a role in shaping the frameworks trajectory.

In today's front end development landscape Angular stands as a player. Its adaptability to web development standards commitment to innovation and dedication to providing a developer experience ensure its continued relevance. The ongoing developments reflect a forward-thinking approach where the Angular team addresses emerging trends and challenges in the changing field of web development. (Cincovic, 2019)

In summary, Angular's history is a testament to continuous evolution, from the pioneering days of AngularJS to the sophisticated Angular 2 and beyond. The journey underscores a commitment to innovation, resilience in addressing challenges, and a community-driven ethos that positions Angular as a foundational framework in the realm of modern web development.

2.3.1 Architecture and Core Features of Angular: A Deep Dive

Angular as a leading framework, in web development sets itself apart with its architecture and a comprehensive range of core features that redefine the landscape of dynamic web application development. In this section we will dive into Angular's architecture and key features unveiling how the framework excels in creating maintainable applications. Its architecture is built on the principles of modularity and component-based development providing developers with a scalable framework for crafting applications. (Cincovic, 2019)

Angular's architectural strength lies in its integration of a module system, a component-based paradigm and a hierarchical dependency injection system. The modular system acts as containers for components, directives, services and pipes—enhancing code maintainability while fostering reusability. This modular structure not improves the organization of the codebase but also facilitates reusability by offering a solid foundation for developing well-structured and scalable applications. (Cincovic, 2019)

At the heart of Angular's philosophy is its emphasis on components, as building blocks. Components encapsulate parts of the user interface while effectively managing presentation logic and styling. This approach goes beyond organizing code; it actively promotes reusability and composability. By leveraging these components developers can create applications more efficiently and flexibly. (Cincovic, 2019)

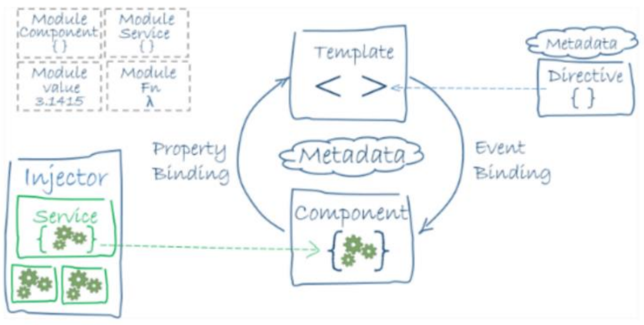


Figure 4 Angular Architecture

Source: (Rathinam, Analysis and Comparison of Different Frontend, 2022)

Angular’s brilliance lies not in its component-based architecture but, in its hierarchical dependency injection system. This system ensures that services and dependencies are injected at the levels within the applications structure promoting modularity and loose coupling of components. As a result, developers can effortlessly manage dependencies across levels of the application enhancing flexibility and maintainability. (Cincovic, 2019)

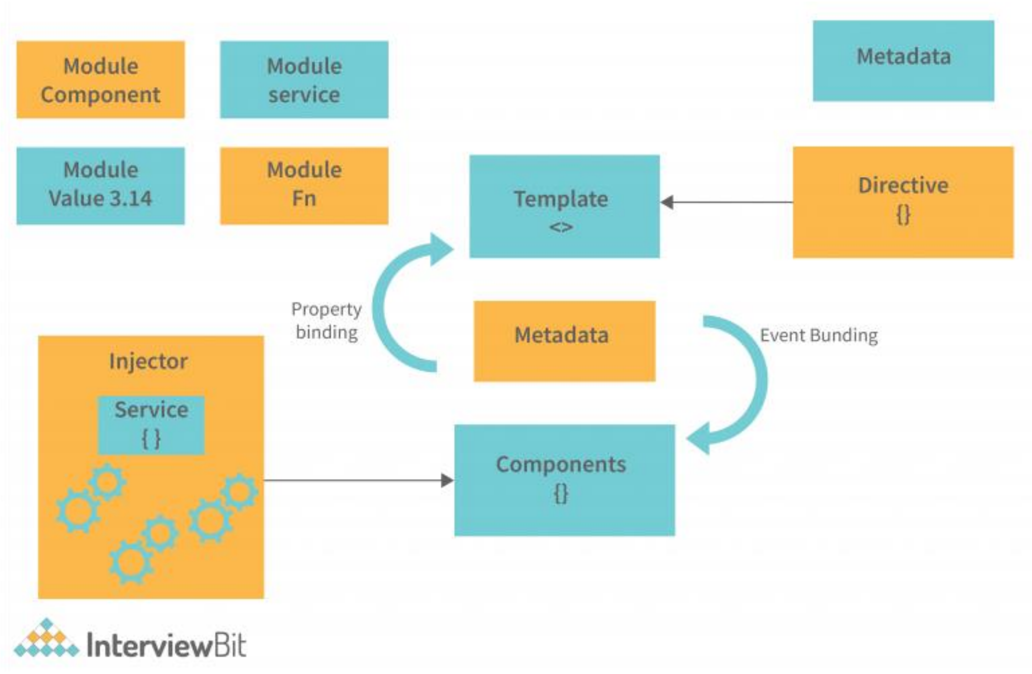


Figure 5: Angular’s Architecture Structure

Source: (Interview Bit, 2024)

The strength of Angular’s architecture stems from the interplay of three elements. The module system establishes foundation components serve as building blocks for UI development and the hierarchical dependency injection system guarantees a flexible and maintainable codebase. Together these aspects form the foundation, for Angular’s ability to create organized, scalable and adaptable web applications.

2.3.2 Fusing Architectural Prowess with Feature-Rich Efficiency

Angular's prowess extends beyond its architectural foundation, encompassing key features like two-way data binding and a robust dependency injection system. This multifaceted approach exemplifies Angular's commitment to providing developers with powerful tools that enhance the efficiency of the development process and contribute to the creation of scalable, maintainable, and reliable web applications. (J. Kalinowska, 2019)

Two-way data binding is a cornerstone feature that underscores Angular's dedication to seamless communication between an application's data model and its user interface. The bidirectional synchronization ensures that changes in the model automatically propagate to the view, and vice versa. This eliminates the need for manual DOM manipulation, simplifying the development process and significantly boosting developer productivity. The automatic synchronization not only streamlines development but also contributes to the creation of responsive and dynamic user interfaces, crucial for modern, interactive web applications. (K. Kowalczyk, 2016)

Simultaneously, Angular's dependency injection system serves as a foundational element for cultivating maintainable, modular, and testable code. By promoting modularity through the creation of loosely coupled components, the system enhances maintainability and simplifies the process of updating or replacing individual components without disrupting the entire application. Testability is prioritized through the ability to substitute dependencies with mock implementations during testing, fostering the creation of robust unit tests. The system's emphasis on decoupling components from their dependencies further enhances code reusability, enabling efficient reuse across different parts of the application or even in distinct projects. (Kozlowki, 2013)

Beyond these core features, Angular emphasizes the pivotal role played by services in its architecture. These services, acting as the backbone, encapsulate business logic and facilitate seamless communication between components. The isolation of business logic within services ensures a clear separation of concerns, enhancing code organization, readability, and overall maintainability. Acting as mediators, Angular services streamline the flow of data and events between different components, contributing to a more organized and comprehensible application architecture. In managing asynchronous operations, such as HTTP requests or data retrieval, Angular services demonstrate mastery, ensuring the application remains responsive and efficient even when dealing with complex, data-intensive tasks.

In summary, Angular's holistic approach, integrating architectural strength, key features like two-way data binding and dependency injection, and the significant role of services, positions it as a powerful framework in the dynamic landscape of web development. This comprehensive understanding sets the stage for exploring Angular's performance characteristics and efficiency in real-world scenarios in the subsequent section.

2.3.3 Performance and Efficiency

The performance and efficiency of Angular as a frontend web development framework are critical factors in its widespread adoption and utilization in creating scalable, robust, and dynamic single-page applications (SPAs). Angular, developed and maintained by Google, has evolved through its versions, with each iteration bringing enhancements aimed at improving performance and making the framework more efficient for developers and the end-user experience. This essay explores the performance characteristics and efficiency of Angular, highlighting its architecture, change detection mechanism, and ahead-of-time (AOT) compilation among other features.

Angular's core architecture, based on TypeScript, offers a structured and scalable foundation for building large-scale applications. The framework's component-based architecture allows developers to create reusable UI components, which can significantly reduce the time and effort required for development tasks (Freeman A. &., 2017)

Moreover, Angular's dependency injection system simplifies the way developers manage dependencies within their applications, leading to cleaner, more maintainable code and, by extension, enhancing overall application efficiency (Freeman A. &., 2017)

One of the critical aspects of Angular's performance is its change detection mechanism. Angular employs a unidirectional data flow and zone.js to detect changes and update the DOM accordingly. This approach ensures that change detection is both predictable and efficient, as Angular only checks the parts of the application that are affected by the state change, minimizing the performance overhead associated with updating the DOM (Kotaru, 2020). This selective rendering process significantly improves application responsiveness and user experience by ensuring that only the necessary parts of the application are redrawn in response to user actions or data changes.

Angular's performance is further enhanced through its support for Ahead-of-Time (AOT) compilation. AOT compilation converts Angular HTML and TypeScript code into efficient JavaScript code during the build phase, before the browser downloads and runs the code. This process reduces the load time and improves the startup speed of Angular applications, as the browser can immediately execute the optimized JavaScript without the need to compile the app first (Freeman A. &., 2017). Moreover, AOT compilation also contributes to improved security by compiling HTML templates and components into JavaScript files long before they are served to the client, reducing the risk of injection attacks.

Angular also offers built-in support for lazy loading, a technique that allows developers to load JavaScript components asynchronously as needed rather than loading the entire application upfront. Lazy loading can significantly reduce the initial load time of Angular applications, making them more responsive and improving the user's perceived performance (Freeman A. &., 2017)

Thus, Angular's performance and efficiency as a frontend web development framework are underpinned by its sophisticated architecture, change detection mechanism, AOT compilation, and support for lazy loading. These features collectively contribute to the creation of fast, efficient, and scalable web applications. Angular continues to evolve, with the Angular team consistently working to introduce optimizations and enhancements that further improve the framework's performance and efficiency, demonstrating its commitment to meeting the demands of modern web application development.

2.3.4 Developer Experience

The developer experience (DX) with Angular as a frontend web framework is a pivotal aspect that significantly influences its adoption and effectiveness in building complex web applications. Angular's comprehensive ecosystem, coupled with its robust tooling and consistent development practices, fosters an environment where developers can thrive, innovate, and efficiently deliver high-quality applications. This essay delves into the various facets of Angular's developer experience, including its CLI, modular architecture, TypeScript integration, and extensive documentation, highlighting how these features contribute to a positive and productive development process.

Angular CLI (Command Line Interface) stands out as one of the most impactful tools in Angular's ecosystem, designed to streamline the development workflow. It provides developers with a powerful and easy-to-use toolset for creating, managing, and deploying Angular applications. The CLI automates tedious tasks such as initialization, configuration, and scaffolding, allowing developers to focus more on coding rather than setup and configuration. Moreover, it supports a wide range of out-of-the-box functionalities, including testing, linting, and building for production, which significantly enhances productivity and ensures consistency across projects (Wilken, 2018)

The modular architecture of Angular, driven by its component-based structure, significantly improves the developer experience by promoting reusability, maintainability, and testability. Developers can encapsulate functionality within components and services, making it easier to manage large applications and collaborate with teams. This modularity also enables lazy loading, allowing for more efficient loading and rendering strategies that can lead to better performance and user experience (Wilken, 2018)

TypeScript, a superset of JavaScript adopted by Angular, offers static typing, classes, and interfaces, which enhance code quality and reliability. Static typing allows developers to catch errors early in the development process, reducing runtime errors and debugging time. Additionally, TypeScript's object-oriented features make Angular applications easier to organize, refactor, and scale, improving the overall development experience (Freeman A. &., 2017).

Angular's extensive documentation and vibrant community play a crucial role in enriching the developer experience. The framework's official documentation is comprehensive, well-organized, and continuously updated, providing a valuable resource for learning and troubleshooting. Furthermore, Angular's community is active and supportive, offering an abundance of resources, including tutorials, articles, and forums, where developers can share knowledge, seek advice, and find solutions to common challenges (Noring, 2016)

In summary, the developer experience with Angular is enhanced by its powerful CLI, modular architecture, integration with TypeScript, and strong community support. These features collectively create a productive development environment that empowers developers to build complex, scalable, and high-performance web applications efficiently. Angular's commitment to improving developer experience is evident in its continuous updates and the introduction of new features aimed at simplifying development workflows and enabling best practices. As a result, Angular not only facilitates the development of cutting-edge web applications but also ensures that the process is enjoyable and rewarding for developers.

2.4 React: The Declarative and Component-Based Paradigm

React, developed by Facebook and released in 2013, introduced a virtual DOM and a component-based architecture, significantly improving the performance of web applications and enhancing the developer's ability to create reusable UI components (Fedosejev, 2015). React's unidirectional data flow and the introduction of JSX syntax provided a novel approach to building interactive user interfaces.

React brought about a change, in the way software is built through its component-based architecture. According to (Paul, 2019) React‘s flexibility empowers developers to create reusable components resulting in a codebase that's scalable and efficient. This section aims to delve into the architecture of React and explore how its design philosophy impacts the development process making it popular among developers.

Initially, React originated as a tool developed by Jordan Walke to address the requirements of Facebooks extensive and dynamic applications. This early stage laid the groundwork for a moment in the history of web development. The decision to open source React in 2013 marked a shift. By adopting a development model React not allowed developers worldwide to utilize the framework but also actively encouraged them to participate in its evolution. This shift fostered an engaged community that accelerated the growth of React. (Komperla, V., Deenadhayalan, P., Ghuli, P., & Pattar, R., 2022)

One of React’s innovations, across the years was the introduction of the virtual DOM concept. This groundbreaking idea, introduced around the time as React became source tackled a crucial performance bottleneck. The virtual DOM enabled React to optimize UI updates by rendering only those components that had undergone changes. Not did it improve performance. It also set the foundation, for Reacts ability to efficiently handle intricate and content heavy user interfaces. (Paul, 2019)

Within the boundaries of Facebook, React quickly became indispensable. Its success in simplifying the development of user interfaces and enhancing performance led to its adoption across projects. The influence of this framework extended beyond Facebooks realm capturing the attention of tech players. This external interest ignited the growth of a community of developers to contribute to Reacts ongoing evolution. Several significant milestones marked Reacts evolution. (Paul, 2019)

The introduction of the Flux architecture in 2014 provided a pattern for managing application states, which complemented Reacts component-based structure perfectly. This architecture played a role in handling complexities encountered in large scale applications. React unfolded its versatility further in 2015 with the launch of React Native—an extension that enabled developers to utilize Reacts paradigm for building platform mobile applications showcasing its adaptability beyond web development.

A significant breakthrough occurred in 2018 with the introduction of Hooks in React 16.8—a feature that empowered developers to utilize state and lifecycle features in components resulting in a concise and readable approach, to managing component logic.

In today’s landscape React remains a force. Regular releases introduce features and optimizations ensuring that this framework stays at the forefront of web development. The React community continues to thrive contributing to an array of third-party libraries and tools. Reacts influence goes beyond its architecture inspiring the industry as a whole to adopt reusable code through component-based structures. (Paul, 2019)

Looking ahead, React can boast of (Angular architecture - detailed explanation. InterviewBit. (2022, 2022) ambitious projects, like Concurrent Mode and React Server whose components highlight the framework’s commitment to pushing performance and functionality boundaries. These initiatives aim to improve concurrent task handling and introduce server-side rendering for components showcasing Reacts dedication to staying at the forefront of technology.

In essence, React‘s history is a tapestry woven with innovation collaboration within the community and an unwavering pursuit of excellence. From its beginnings as a solution, to becoming a global phenomenon today Reacts journey serves as evidence of the power of open-source development and the transformative impact that a dedicated community can have on shaping the future of web development.

2.4.1 Architecture

React, a pioneer, in web development creates a structure that prioritizes simplicity, reusability and efficiency. The heart of React lies in its approach, which simplifies the manipulation of the Document Object Model (DOM). By using a syntax developers can easily describe the desired user interface (UI) state making the code more predictable and maintainable. React takes advantage of a component-based architecture where modular units encapsulate both the logical aspects of UI elements. This not promotes code reuse. Also improves composability. (J. Kalinowska, 2019)

One key aspect that contributes to React‘s efficiency is the Virtual DOM—a representation of DOM elements. This intermediate layer optimizes rendering by updating the parts of the actual DOM that have changed. This approach greatly enhances performance in applications with data.

React’s adoption of data flow ensures an manageable flow of data from parent to child components. This design choice simplifies. Understanding of the applications state. Additionally, the Flux architecture introduces a pattern for managing state in applications through its unidirectional data flow.

JSX is an extension, for JavaScript that allows HTML to be embedded directly within JavaScript code. This streamlines the creation of component structures while also improving security by handling any embedded values to prevent injection attacks.

The introduction of React Hooks, in version 16.8 completely transformed how developers manage state and lifecycle methods in components. This advancement simplifies the organization of component logic making it easier to read and reducing the need, for class components.

React Router, which is a part of the React ecosystem provides a way to handle navigation and routing using a straightforward approach. It allows for navigation without needing to refresh the page resulting in a smoother user experience especially for single page applications (SPAs).

To summarize, React’s architecture combines syntax, a component-based approach the efficiency of the Virtual DOM, one way data flow, expressive JSX syntax, the flexibility of Hooks and the capabilities of React Router. This foundation empowers developers to build web applications that're not only scalable and maintainable but also perform well in real world scenarios.

2.4.2 Performance and Efficiency

The performance and efficiency of React as a frontend framework are paramount to its popularity and widespread adoption for developing dynamic user interfaces. Created by

Facebook, React introduces a reactive and component-based architecture, fundamentally shifting how applications are built and interacted with by users. This essay delves into the core aspects of React's performance and efficiency, including its virtual DOM, component lifecycle methods, and optimization techniques, providing insight into how React maintains high performance while offering a seamless development experience.

React's virtual DOM is a foundational concept that underpins its performance advantages. Unlike traditional approaches that manipulate the browser's DOM directly—often a slow and costly operation—React maintains a lightweight representation of the DOM in memory. When changes occur in the application state, React computes the difference between the current and previous states of the virtual DOM, updating the browser's DOM with only the necessary changes. This selective rendering mechanism minimizes direct DOM manipulation, leading to significant performance improvements and a more efficient update process (Freeman A. , 2020)

The efficiency of React is further enhanced through its component lifecycle methods, which allow developers to control the behavior of components at various stages of their existence. Lifecycle methods such as `componentDidMount`, `shouldComponentUpdate`, and `componentWillUnmount` provide hooks into the rendering process, enabling developers to optimize performance by preventing unnecessary re-renders or executing code at the optimal time (Freeman A. , 2020).

For instance, `shouldComponentUpdate` can be implemented to compare incoming props and state, returning false if a component update is not necessary, thereby avoiding costly DOM updates.

React also promotes performance optimization through techniques such as lazy loading, code splitting, and memoization. Lazy loading defers the loading of non-critical resources at page load time, reducing the initial load time and speeding up application startup. Code splitting, facilitated by tools like Webpack, allows developers to split their application code into smaller chunks that are loaded on demand, further improving load times and resource utilization. Memoization, implemented via React's `React.memo` and `useMemo`, prevents the re-computation of expensive functions or rendering of components when their inputs have not changed, enhancing efficiency (Wieruch, 2018)

For developers and teams looking to visually illustrate the concepts discussed in this essay, placing images that demonstrate the virtual DOM concept, component lifecycle, and optimization techniques can significantly aid in comprehension. These images could be effectively positioned at the beginning of each section to provide a visual summary before diving into the textual explanation.

In summary, React's design and architecture prioritize performance and efficiency, making it an excellent choice for developers building interactive and dynamic web applications.

Through its virtual DOM, component lifecycle methods, and optimization techniques, React ensures applications are both high-performing and efficient, providing a smooth user experience and a productive development environment.

React, developed by Facebook in 2013, offers a standout developer experience (DX) in the realm of web development, distinguished by its declarative programming model, component-based architecture, and a robust ecosystem. This JavaScript library simplifies building user interfaces by emphasizing intuitive development practices, promoting code reusability, and fostering a supportive community environment. At the heart of React's appeal to developers is JSX, a syntax extension that allows for HTML structures to be written within JavaScript code. This integration of markup with logic streamlines the development process, enhancing readability and making the component's structure more transparent (Wieruch, 2018). JSX eliminates the complexity associated with manually creating DOM elements, leading to cleaner and more maintainable code.

The introduction of functional components with hooks in React version 16.8 marked a significant evolution in how developers manage state and lifecycle methods, shifting towards a functional programming paradigm (Banks, 2020).

Hooks enable the use of state and other React features in functional components, eliminating the need for classes and simplifying component logic. This advancement has been widely adopted for its ability to reduce boilerplate, improve code reuse, and enhance the overall maintainability of applications. (Banks, 2020).

React also provides developers with specialized tools designed to optimize the development workflow. The React Developer Tools, a browser extension for Chrome and Firefox, offer deep insights into the component tree, props, state, and context, streamlining debugging and optimization efforts (Flanagan, 2020). These tools allow for real-time inspection and manipulation of React component hierarchies within the browser, significantly enhancing debugging processes and contributing to more efficient development workflows.

Furthermore, React is supported by an extensive and active community, with comprehensive documentation that is regularly updated, including tutorials, guides, and API references (Flanagan, 2020). This wealth of resources, combined with open-source projects, forums, and discussion groups, enriches the React ecosystem, providing developers with libraries, tools, and plugins that extend React's functionality. The community's collaborative spirit accelerates the learning process for newcomers and fosters an environment where knowledge and best practices are freely shared (Flanagan, 2020).

React's developer experience is characterized by features such as JSX, functional components with hooks, extensive developer tools, and a vibrant community. These elements synergize to create a productive and enjoyable environment for building complex, high-quality web applications. React's continuous evolution, driven by feedback from its user base, showcases its commitment to improving the developer experience and maintaining its position as one of the most popular and developer-friendly libraries in the web development landscape. Through its innovative features and supportive ecosystem, React empowers developers to efficiently tackle the challenges of modern web application development, making it a preferred choice for developers looking to deliver compelling user interfaces with optimal performance (Banks, 2020).

2.5 Vue.js: The Balanced Elegance

Vue.js was created by Evan You in 2014, combining the reactivity and two-way data binding of AngularJS with the virtual DOM approach of React. Its lightweight and incrementally adoptable architecture made it particularly suited for integrating into projects requiring faster performance and smaller footprints (You, 2014). Vue.js's simplicity, flexibility, and detailed documentation have contributed to its popularity among developers and companies alike.

Vue.js is widely recognized for its simplicity and user approach offering a blend of Angular’s comprehensiveness and Reacts flexibility. In our exploration of Vue.js we draw insights, from (Macrae, 2018) focusing on its component-based architecture, reactivity and two-way data binding that make it appealing to developers. This section aims to delve into the workings of Vue.js providing a nuanced understanding of its strengths in development scenarios.

Vue.js has gained attention due to its simplicity, flexibility and gradual adoption approach. The key milestones in Vue.jss development reflect its growth and responsiveness to community feedback. The core development team has made enhancements and strategic decisions that have contributed to Vue.jss position in the front-end development landscape. By examining these milestones, we can gain insights into how Vue.js has evolved over time and positioned itself as an adaptable framework for developers. Evan You conceived Vue.js in 2013 as a milestone in the evolution of front end development frameworks. Its initial release as Vue.js 0.8 laid the foundation for a framework that aimed to strike a balance between simplicity and flexibility—a commitment that has defined its journey in years. The release of Vue.js 1.0 in 2014 marked a turning point, for the framework. (Kyriakidis, 2016)

Vue.js gained popularity due, to its focus on stability and comprehensive features. One of its core strengths was its reactivity system, which automatically updated the user interface based on changes in the underlying data. This efficient and user-friendly approach was further enhanced by directives like `v bind` and `v model` making Vue a choice for developers seeking a seamless front-end solution. The release of Vue.js 2.0 in 2016 solidified the frameworks reputation for being forward thinking. It introduced groundbreaking features such as an improved DOM server-side rendering (SSR) and the concept of single file components. The introduction of single file components brought together templates, scripts and styles in a structure streamlining development and promoting organization and maintainability. This version exemplified Vues adaptability by allowing developers to gradually adopt features making it suitable for projects of scales and complexities. (Kyriakidis, 2016)

As Vue.js continued to gain traction, its ecosystem expanded to meet the evolving needs of developers. Key additions included Vue Router, for client-side routing support and Vuex for state management. These updates tackled the challenges faced by developers working on single page applications (SPAs) reinforcing Vue.jss position, as a solution for addressing modern web development obstacles. Vue.js 3.0, which was eagerly anticipated and officially released in 2020 after being announced in 2019 brought a wave of exciting advancements. The Composition API, an approach to organizing and reusing component logic provided developers with flexibility. The reactivity system also underwent refinements resulting in improved performance and making Vue.js more suitable for large scale projects. This iteration exemplifies Vues dedication to staying of the curve and equipping developers with tools that align with emerging industry standards. (Macrae, 2018)

In the landscape Vue.js thrives as a framework supported by tools such as Vue CLI and Vue DevTools greatly enhancing the development experience. Essential components like Vuex for state management and Vue Router for client-side navigation continue to play roles within the Vue ecosystem. The frameworks commitment to enhancement and adaptability ensures its relevance in the ever-evolving field of web development.

Vue.jss journey is a story of unwavering commitment to simplicity, adaptability and a progressive mindset. From its principles, to its innovations Vue.js has evolved to meet the dynamic demands of modern web development. The way Vue.js has evolved and supported developers showcases its dedication to empowering them and meeting the changing needs of the industry. In today's world Vue.js continues to be a symbol of adaptability and innovation inspiring developers globally. (Macrae, 2018)

2.5.1 Vue.js Architecture: A Blueprint for Dynamic Web Development

Vue.js, renowned for its simplicity and versatility, is underpinned by a well-crafted architecture designed to empower developers in building dynamic and responsive web applications. At the core of Vue.js architecture lie two fundamental concepts: reactivity and a component-based structure.

Vue.js excels in its reactivity system, forging a seamless connection between the application's data model and the user interface. This bidirectional data binding ensures that changes in the data model automatically ripple through the UI, cultivating a responsive and dynamic user experience. Complementing this, Vue.js leverages a virtual DOM, a nimble in-memory representation of the actual DO (Baida, 2020). This strategic use optimizes the tracking and updating of changes in the application state, resulting in heightened rendering speed and overall performance.

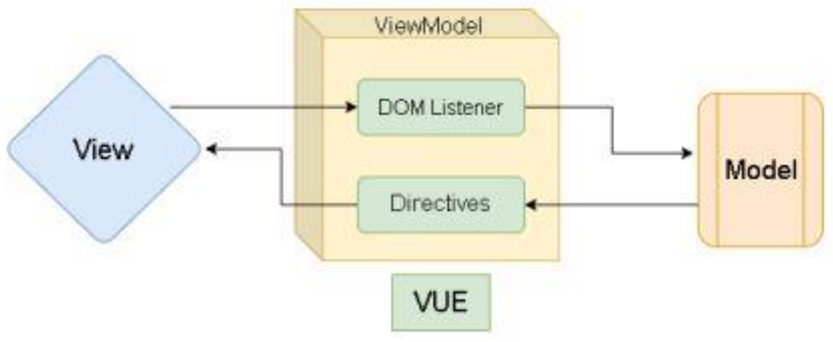


Figure 6 VueJs Architecture

Source: (Rathinam, Analysis and Comparison of Different Frontend, 2022)

The framework embraces a component-based development approach, wherein each component functions as a self-contained and reusable unit. This modular structure becomes the backbone of Vue.js, enhancing code organization while fostering reusability and maintainability. Introducing the innovative concept of single-file components, Vue.js encapsulates templates, scripts, and styles within a unified file. This organizational paradigm simplifies the development process, rendering it more comprehensible and collaborative.

This harmonious marriage of reactivity and a component-based structure forms the bedrock of Vue.js's architectural prowess. Developers operate within a flexible and efficient environment, empowered to craft sophisticated web applications with ease. Vue.js's commitment to simplicity and adaptability shines through in this well-crafted architectural foundation, making it a relevant and powerful force in the ever-evolving landscape of web development. (Baida, 2020)

2.5.2 Vue.js Features: Elevating Web Development to New Heights

Vue.js provides developers with a seamless and enjoyable development experience, supported by a rich set of features that elevate web development to new heights. In the realm of templating and directives, Vue.js adopts a declarative approach, employing a syntax that allows developers to express the structure and behavior of the UI in a clear and concise manner. This approach enhances code readability and simplifies the development process. Dynamic behavior is effortlessly integrated using Vue's directives, such as `v-bind` and `v-model`, which enable dynamic data binding and user input handling. These directives empower developers to incorporate dynamic behavior without the need for extensive JavaScript code. (Macrae, 2018)

Vue Router and Vuex further enhance the development experience. Vue Router facilitates client-side routing, enabling the creation of dynamic routes for single-page applications. This feature ensures a seamless and engaging user experience during navigation. As applications grow in complexity, Vuex steps in as a centralized state management solution for Vue.js. It establishes a predictable state management flow and simplifies debugging and testing processes. Efficient development is streamlined with Vue CLI, a full-featured command-line interface that accelerates project setup and management. It provides predefined build setups and a plugin system, contributing to the swift configuration and maintenance of projects. Enhanced debugging capabilities are offered through Vue DevTools, a browser extension. It enriches the Vue.js development experience by providing tools for inspecting and debugging Vue.js applications. Features include component tree inspection, state and event tracking, and time-travel debugging.

Collectively, these features position Vue.js as a powerful and developer-friendly framework, offering a holistic approach to web development. The combination of a well-architected foundation, characterized by reactivity and a component-based structure, alongside feature-rich elements, ensures that Vue.js continues to thrive in the ever-evolving landscape of modern web development. Vue.js remains a compelling choice for developers seeking an elegant, adaptable, and efficient framework for building sophisticated web applications. (Macrae, 2018)

Chapter 3: Research Methodology

The comparative analysis of trending web development frameworks—Angular, React, and Vue.js—plays a pivotal role in delineating their applicability and utility within the landscape of modern web development. This comprehensive examination, presented in the methodology chapter, is anchored on several pivotal aspects, including a thorough review of framework resources, performance testing, developer surveys, and an exhaustive data analysis aimed at deriving substantial conclusions.

An extensive review conducted in the preceding chapter explored the official documentation, community support forums, and a broad array of development resources available for Angular, React, and Vue.js. This analytical endeavor was paramount in demystifying the core features, inherent benefits, and possible limitations of each framework. A deep dive into developer guides, API documentation, and vibrant community discussions unveiled a comprehensive view of each framework's ecosystem, enriched with libraries, tools, and plugins, thereby augmenting their capabilities significantly. This facilitates the crafting of robust, scalable, and efficient web applications.

A study by (Vyas, 2022) on the comparative analysis of React and Vue.js front-end frameworks echoes the significance of such an ecosystem in supporting web application development. Echoing the sentiments of prior studies, the importance of comprehensive documentation and robust community support cannot be overstated, as they play a decisive role in the framework selection process among developers. These resources are instrumental in not only understanding but also effectively leveraging the frameworks to their fullest potential (Vyas, 2022).

Subsequent to the foundational review of resources, performance testing was identified as a crucial step to empirically assess the efficiency and responsiveness of Angular, React, and Vue.js under varied operational scenarios. The imperative for such testing stems from the understanding that theoretical insights and community feedback, while valuable, are complemented by empirical data for a holistic evaluation. Performance testing, detailed in ensuing sections, encompasses the setup of test environments, adoption of precise metrics and measurement tools, and rigorous data collection processes, paralleling the methodologies espoused by Shahoor et al. (2023) in their work on proactive repairing of memory leaks in single-page web applications.

In our comprehensive analysis of the leading web development frameworks—Angular, React, and Vue.js—our methodology is crafted to encompass a sequence of rigorously structured stages, leveraging an array of test environments and sophisticated testing tools. This approach ensures a robust and nuanced evaluation, grounded in empirical data and reflective of diverse development scenarios. By deploying tests across an Apple MacBook M1 Pro, an Apple MacBook M2 Pro, and a Dell Intel Core i5 7th Gen laptop, our study mirrors the multifaceted nature of modern web development, catering to the intricate demands of varying hardware and software ecosystems.

The core of our methodology unfolds through an elaborate two-stage process, employing Google Lighthouse for performance benchmarking, followed by Puppeteer for simulating real-world user interactions. These stages are designed not only to capture the raw performance metrics but also to reflect the user experience each framework delivers under different conditions.

Our journey into the frameworks' efficiency begins with an in-depth application of Google Lighthouse, a tool renowned for its comprehensive analysis capabilities. Lighthouse offers a prism through which the performance of web applications developed with Angular, React, and Vue.js is dissected across multiple dimensions, including speed, interactivity, and overall user experience. The tool's audits extend beyond mere load times, delving into metrics that signify a framework's capability to deliver content swiftly and interactively. Key performance indicators such as First Contentful Paint (FCP) and Time to Interactive (TTI) are recorded, alongside the overall Performance Score, which aggregates several metrics to provide a holistic view of each application's performance.

This stage is characterized by its rigorous and repeatable testing process. Each test application, representative of its respective framework, is subjected to multiple Lighthouse runs across all three hardware environments. This not only ensures consistency in our findings but also allows us to observe how each framework's performance might scale or vary in response to different computational resources and configurations. The resulting data paints a detailed picture of how Angular, React, and Vue.js stand up to the demands of modern web development, highlighting the efficiency and speed with which they can deliver complex web applications to users.

Following the performance audits, our methodology transitions to the utilization of Puppeteer for an immersive evaluation of user experience. Puppeteer's role in our study extends beyond mere automation; it serves as a bridge to understanding how Angular, React, and Vue.js applications respond to real-world user behaviors and interactions. By scripting a series of typical user actions—ranging from navigating through the application, clicking buttons, to filling out forms—Puppeteer replicates the journey of an end-user, offering insights into the responsiveness and runtime efficiency of each framework.

This stage is crucial for unearthing the practical implications of each framework's performance metrics, translating numbers and scores into a tangible user experience narrative. The simulations conducted via Puppeteer are designed to mirror a variety of user interactions, ensuring a comprehensive assessment of the frameworks' ability to handle dynamic content, manage state transitions, and respond to user inputs efficiently. The outcome of this stage is a rich dataset, capturing not only the speed and fluidity of user interactions but also highlighting any potential bottlenecks or performance issues that could detract from the user experience.

Through the elaborate orchestration of these testing stages, our methodology culminates in a rich tapestry of insights, weaving together the technical prowess and user-centric performance of Angular, React, and Vue.js. The comparative analysis that follows is not merely a juxtaposition of metrics but an interpretation of how these frameworks can serve the diverse needs of the web development community. By documenting and analyzing the data collected, our study aims to equip developers and stakeholders with a comprehensive understanding of each framework's strengths and limitations, thereby guiding informed decision-making in the selection of technology for web development projects.

In essence, our approach transcends traditional benchmarking, offering a holistic view of the frameworks' capabilities within the vibrant and ever-evolving landscape of web development. It is through this detailed and methodical examination that our research endeavors to illuminate the path for developers, navigating the complexities of modern web application development with clarity and confidence.

3.1 Setting up Test Environments

Creating identical test environments for Angular, React, and Vue.js is paramount in conducting a fair and balanced performance evaluation of these frameworks. This objective aligns with the overarching goal of software testing, which is to ensure that applications built using these frameworks can meet desired standards of performance, maintainability, and scalability under varying conditions. The necessity of such an environment has been emphasized by Kitchenham et al. (2002), who argue that the validity of software comparisons hinges on the ability to control and replicate testing conditions. This ensures that any observed differences in outcomes can be attributed to the intrinsic characteristics of the software itself, rather than external factors.

To this end, each framework was employed to develop a prototype application with comparable functionality. This step is crucial, as it mirrors the real-world application of these frameworks, providing a tangible basis for comparison. Basili et al. (1996) support this approach, noting that using prototypes to assess software frameworks helps in understanding not only their performance but also their applicability to different types of projects. The prototypes were designed to reflect common use cases in web development, thereby ensuring the relevance of the performance evaluation to developers and industry stakeholders.

Moreover, the configuration of test environments on identical hardware and software specifications is a critical step in minimizing external variables that might affect performance outcomes. The significance of this standardization is highlighted in the work by Brooks et al. (1995), who state that the consistency of test environments is critical in comparative software studies. Such consistency ensures that the performance metrics obtained are reflective of the frameworks’ capabilities rather than the varying efficiencies of the underlying hardware or software platforms.

The selection of performance metrics and measurement tools is another cornerstone of this methodology. Metrics such as load time, time to interactive, performance, fcp, lcp, and speed index were chosen for their ability to provide a comprehensive overview of each framework's performance.

McCabe (1976) underscores the importance of selecting metrics that can offer insights into the efficiency and responsiveness of software products. These metrics are instrumental in evaluating how well each framework can handle the demands of modern web applications, especially in terms of user experience and resource consumption.

To conduct a fair and balanced performance evaluation of Angular, React, and Vue.js, setting up identical test environments is crucial. This process is carefully designed to ensure that any observed differences in framework performance are attributable solely to the frameworks themselves, rather than to external variables. Here's a detailed breakdown of how we set up the test environments, with a special focus on ensuring consistency and replicability across the testing platforms.

We begin by selecting three distinct hardware platforms to host our test environments: an Apple MacBook M1 Pro, an Apple MacBook M2 Pro, and a Dell Intel Core i5 7th Gen laptop. This diverse selection is aimed at evaluating the frameworks' performance across a range of common computing environments, from cutting-edge ARM architecture to more traditional x86 setups. This diversity ensures that our findings are applicable to a wide range of real-world scenarios and developer setups.

For each hardware platform, we standardize the operating system and browser settings to minimize variations. This includes updating each machine to the latest available OS version and installing the latest version of Google Chrome, ensuring that any performance differences are not due to discrepancies in browser or OS efficiency. This step is vital for creating a controlled environment where the frameworks' performances can be accurately assessed.

With the hardware and software baseline established, we proceed to install Google Lighthouse and Puppeteer on each test machine. These tools are chosen for their robust performance auditing and web interaction simulation capabilities, respectively. Google Lighthouse provides comprehensive performance metrics such as load times and interactivity scores, while Puppeteer allows for the automation of real-world user interactions with the web applications built using Angular, React, and Vue.js. Codes of these applications can be found in online plateform github.com. This online github repository also includes the puppeteer test codes. To access the website this link can be used: https://github.com/AmadHashmi/ComparisonOfFramworks. These code files will be uploaded with this thesis.

To ensure a fair comparison, we develop prototype applications with identical functionality across all three frameworks. These prototypes are designed to incorporate a variety of common web development features and challenges, such as data binding, component rendering, and state management. This step is crucial for evaluating how each framework handles similar tasks under identical conditions, providing a solid basis for comparison.

Once the prototype applications are ready, we configure each test environment to run these applications. With the environments set up, we execute a series of performance tests using Google Lighthouse, measuring key performance indicators (KPIs) such as First Contentful Paint (FCP), Time to Interactive (TTI), and overall Performance Score. These tests are run multiple times to ensure consistency and to account for any potential outliers.

Following the performance tests, we use Puppeteer to simulate real-world user interactions with each prototype application. This involves scripting common user behaviors and measuring the applications' responsiveness and runtime efficiency. This stage is critical for assessing how each framework performs under the kind of interactive scenarios users would encounter in real life.

Throughout the testing process, performance data and user interaction metrics are collected systematically. This data is then analyzed to draw comparisons between the frameworks, focusing on both their performance metrics and their ability to deliver a smooth user experience.

By adhering to this detailed and structured approach to setting up and utilizing test environments, we ensure that our evaluation of Angular, React, and Vue.js is both comprehensive and fair. This methodology allows us to isolate the frameworks' intrinsic characteristics as the variables affecting performance, thereby providing developers and stakeholders with clear, actionable insights into each framework's capabilities and limitations.

3.2 Metrics and Measurement Tools

In the domain of web application development, the evaluation of user-centric performance metrics is essential for understanding the end-user experience. This understanding is critical, as the perceived performance of a web application can significantly influence user engagement and satisfaction. To conduct a rigorous and empirical analysis of such metrics, it is imperative to employ a robust measurement framework. Google Lighthouse stands at the forefront of such frameworks, providing a comprehensive suite of performance metrics. These metrics serve as a quantitative foundation for evaluating the user experience provided by web applications developed using Angular, React, and Vue.js.

This section presents an in-depth examination of the selected performance metrics, which are pivotal to the Google Lighthouse testing environment. Each metric encapsulates a distinct facet of the user experience, from the initial visibility of content to the stability of the user interface during interaction. Through a detailed exposition of First Contentful Paint, Largest Contentful Paint, Total Blocking Time, Cumulative Layout Shift, and Speed Index, we endeavor to furnish a holistic understanding of their implications for web application performance. The academic rigour applied in this analysis ensures that the derived insights are both methodologically sound and relevant to the field of web development. The subsequent discourse will dissect these metrics extensively, providing clarity on their methodological underpinnings and their practical implications within the broader landscape of web application performance assessment.

Google Lighthouse captures a range of data points that collectively paint a detailed picture of a website’s performance. The metrics we focus on are instrumental in quantifying how quickly and smoothly a web application delivers content, becomes interactive, and remains stable as users interact with it. A nuanced analysis of these metrics allows us to identify and understand the implications of design and development choices on user experience.

As we proceed, each performance metric will be discussed in detail, elucidating their individual contributions to the overall performance score and their impact on the perceived quality of a web application. Understanding these metrics is key to discerning the subtleties of web framework performance, providing a comprehensive view of the strengths and limitations of Angular, React, and Vue.js within the context of real-world web usage.

3.2.1 Performance Score

The performance score is a composite metric that reflects the overall performance of a webpage. It's computed by weighting individual metric scores, such as FCP, LCP, and TBT. This score encapsulates a broad spectrum of user-centric performance issues, offering a snapshot of what users experience when they visit a page. It helps to quickly gauge whether a webpage is performing at an optimal level or if there are areas that require improvement.

3.2.2 First Contentful Paint (FCP)

FCP is the time it takes for the browser to render the first piece of DOM content after a user navigates to a page. It could be text, an image, or even a canvas or SVG. This metric is critical as it gives the user a visual confirmation that the page is loading. The goal is to have FCP occur within the first 1.8 seconds of the page starting to load, as this is considered a good user experience. Delays in FCP can lead to user frustration, increased bounce rates, and potentially lower user satisfaction.

3.2.3 Largest Contentful Paint (LCP)

LCP measures the render time of the largest content element visible within the viewport. This could be a large image, block of text, or a video. It's an important user-centric metric for measuring perceived user experience. Unlike FCP, which might render something as minimal as a line of text, LCP provides a better sense of a fully loaded page. A good LCP is considered to be under 2.5 seconds. This metric is important because users typically regard a page as ready when the main content has loaded, so a quicker LCP can lead to a perception of a faster, more responsive site.

3.2.4 Total Blocking Time (TBT)

TBT quantifies the non-interactive time between FCP and Time to Interactive, when task lengths exceed 50ms. Essentially, it measures the total amount of time that a page is unresponsive to user input. TBT is important for understanding a page's interactivity and responsiveness. Long tasks can block the main thread, preventing users from interacting with the page. For a good user experience, the TBT should be less than 300 milliseconds.

3.2.5 Cumulative Layout Shift (CLS)

CLS is the sum of all individual layout shift scores for every unexpected layout shift that occurs during the entire lifespan of the page. A layout shift happens any time a visible element changes its position from one frame to the next. This metric is crucial for understanding the visual stability of a website. High CLS can be frustrating for users, leading to mistaken clicks and difficulty in reading content. To ensure a good user experience, the CLS should be less than 0.1.

3.2.6 Speed Index

The Speed Index measures how quickly the content of a page is visibly populated. It's a general representation of how quickly a user sees a page's content load. This metric is important because it provides an indication of how quickly a website can deliver content in a usable state. It's different from other metrics that may measure the time to first byte or even full page load. Instead, it's about the visual completeness. The lower the Speed Index, the better the perception of the page's performance.

Each of these metrics offers valuable insights into different aspects of the user experience. For instance, while FCP and LCP provide timing on visibility of content, TBT and CLS speak to the interactivity and stability of the page post-load. Collectively, these metrics allow developers to pinpoint both strengths and weaknesses in their applications' performance, guiding targeted improvements and optimizations to enhance user satisfaction. basis for comparison.

3.3 Data Collection

The collection of data through automated tests on prototype applications developed using Angular, React, and Vue.js is a crucial component of the research methodology. This phase is designed to objectively evaluate each framework's performance under a variety of conditions, which are reflective of real-world application usage. The decision to automate these tests stems from the understanding that automation can significantly increase the efficiency, repeatability, and accuracy of the testing process (Fewster & Graham, 1999). Automated testing tools can execute predefined actions on the application's interface, simulate user interactions, and measure performance outcomes such as load time, responsiveness, and resource utilization.

Each test was repeated multiple times to account for the inherent variability in network conditions and server response times, which can significantly influence the performance metrics of web applications. This approach is aligned with the principles outlined by Myers, Sandler, & Badgett (2004), who emphasize the importance of repetition in software testing to ensure the reliability and validity of the results. Repeating tests under varied conditions allows for the collection of a more robust dataset, from which more accurate and generalizable conclusions can be drawn.

The data thus collected were systematically recorded and prepared for analysis. This preparation involved cleaning the data to remove any anomalies or outliers that could skew the results and aggregating the data to facilitate comparative analysis. The significance of data preparation in performance evaluation cannot be overstated, as it directly impacts the validity of the conclusions drawn from the analysis. According to Kitchenham & Pfleeger (2008), proper data preparation is essential for conducting meaningful statistical analyses that can reveal significant differences and trends in the data.

Furthermore, the comprehensive dataset enables a deep dive into the performance characteristics of Angular, React, and Vue.js, offering insights into each framework's efficiency, scalability, and suitability for different types of web development projects. This dataset forms the foundation upon which the subsequent analysis and conclusions are based, highlighting the critical role of systematic data collection and preparation in empirical software engineering research.

Incorporating automated testing and rigorous data preparation practices not only strengthens the methodological approach of this study but also ensures that the evaluation of Angular, React, and Vue.js is grounded in objective, empirical evidence. By adhering to established best practices in software testing and data analysis, this research contributes valuable insights into the comparative performance of these popular web development frameworks, aiding developers and stakeholders in making informed decisions.

3.4 Developer Surveys

Enhancing the section on conducting developer surveys to gather qualitative data on Angular, React, and Vue.js involves a deeper examination of the survey methodology, the importance of qualitative data in understanding developer experiences and preferences, and the specific areas of inquiry relevant to web development frameworks. Incorporating additional insights and referencing scholarly sources will provide a more detailed and comprehensive explanation.

Developer surveys serve as a critical component of the research methodology, offering invaluable insights into the subjective experiences and preferences of developers who have worked extensively with Angular, React, and Vue.js. Recognizing the importance of qualitative data in software engineering research, Storey et al. (2020) emphasize that surveys can uncover perspectives on the usability, functionality, and community aspects of software tools that are not readily apparent through quantitative metrics alone. In this vein, the surveys distributed aimed to capture a wide array of developer experiences, preferences, and evaluations of the frameworks in question.

The surveys were designed with a deliberate focus on key areas that significantly impact the choice and effectiveness of a web development framework: ease of use, scalability, community support, and overall satisfaction. These areas were chosen based on literature suggesting their critical role in the success and adoption of software frameworks (Pretorius & Solms, 2018). Ease of use addresses the learning curve and developer productivity when using the frameworks. Scalability evaluates how well applications built with these frameworks can grow in complexity and user base. Community support looks into the availability of resources, tools, and forums for solving development challenges. Lastly, overall satisfaction measures the developers’ contentment with the framework's capabilities, performance, and development experience.

To ensure a comprehensive understanding, the survey encompassed both closed and open-ended questions. Closed-ended questions aimed to quantify preferences and experiences on a Likert scale, facilitating statistical analysis. Open-ended questions, on the other hand, allowed developers to express their thoughts and experiences in their own words, providing deeper insights into the reasons behind their preferences and the nuanced strengths and weaknesses of each framework. This mixed-methods approach aligns with recommendations by Creswell & Creswell (2017) for enhancing the depth and breadth of research findings.

A diverse group of developers was targeted for the survey to ensure a broad spectrum of experiences and opinions. This diversity is crucial for capturing a representative snapshot of the developer community's views on Angular, React, and Vue.js, as noted by Easterbrook et al. (2008), who highlight the value of incorporating varied perspectives in software engineering research to achieve more generalizable and robust conclusions.

The collected data were then analyzed to identify patterns, themes, and discrepancies in developer experiences and preferences. This qualitative analysis is pivotal in understanding the subjective factors that influence framework selection and use in web development projects.

It is also important to note that a copy of the survey, including both the closed and open-ended questions, has been included in the appendix of this document. This inclusion serves to provide transparency regarding the methodology and allows for replication of the study, as recommended by Shull et al. (2007), who advocate for the openness in empirical software engineering research to facilitate knowledge sharing and validation of findings.

Chapter 4: Performance Testing and Analysis

The emergence of automated testing tools has marked a significant advancement in how developers approach testing and quality assurance. Among these tools, Puppeteer, a Node.js library developed by Google, has garnered attention for its comprehensive capabilities in automating tasks associated with headless Chrome or Chromium browsers via the DevTools Protocol (Google Developers, 2021). Puppeteer's inception was motivated by the need for a robust, high-level API that could simplify interactions with the Chrome DevTools Protocol, offering developers a versatile tool for web scraping, automated testing, and browser automation (Lighthouse Team, 2020).

The functionality of Puppeteer extends beyond basic automation; it is adept at simulating complex user interactions on web pages, such as clicking buttons, filling out forms, and navigating through sites. This capability is critical for developers seeking to test the user experience and operational efficiency of web applications under various conditions (Smith, 2019). Moreover, Puppeteer's ability to capture screenshots and generate PDFs of web pages provides an invaluable resource for developers in documenting and reviewing the visual and functional aspects of web applications.

Automated testing, particularly with tools like Puppeteer, plays a pivotal role in modern web development practices. The capacity to programmatically perform tests that simulate real-user interactions on web applications is a cornerstone of continuous integration and delivery pipelines. This automation is instrumental in identifying performance bottlenecks and functional inconsistencies early in the development cycle, thereby enhancing the overall quality and reliability of web applications (Jenkins, 2018). Furthermore, the use of Puppeteer in automated testing frameworks aligns with the agile methodology, facilitating rapid iterations and deployments (Fowler & Foemmel, 2006).

The application of Puppeteer in the performance testing of web applications is an area of significant interest. By automating the creation, updating, and deletion of content within web applications—for example, note-taking features—researchers can gather precise data on the time taken to complete these tasks across varying data set sizes (10, 50, and 100 notes). Such data are crucial for assessing the scalability and responsiveness of web applications, providing developers with insights into how these applications perform under different load conditions. Repeating tests multiple times, as suggested by Baxter et al. (2021), ensures the accuracy and reliability of performance metrics by accounting for variability in network and server conditions.

The strategic use of Puppeteer in this research methodology underscores the value of automated testing in the empirical evaluation of web application performance. By simulating user interactions and automating repetitive tasks, Puppeteer allows for a more thorough and nuanced understanding of an application's performance landscape. This approach not only mitigates the potential for human error in manual testing but also facilitates a more standardized and reproducible testing environment, essential for objective performance analysis (Patton, 2014).

In summary, Puppeteer represents a paradigm shift in automated web testing, offering developers a powerful tool for ensuring the performance, functionality, and user experience of web applications meets the highest standards. Its integration into performance testing and analysis workflows exemplifies the ongoing evolution of web development practices towards automation and efficiency. As web applications continue to grow in complexity and scope, the role of tools like Puppeteer in maintaining and enhancing the quality of these applications will undoubtedly increase in importance.

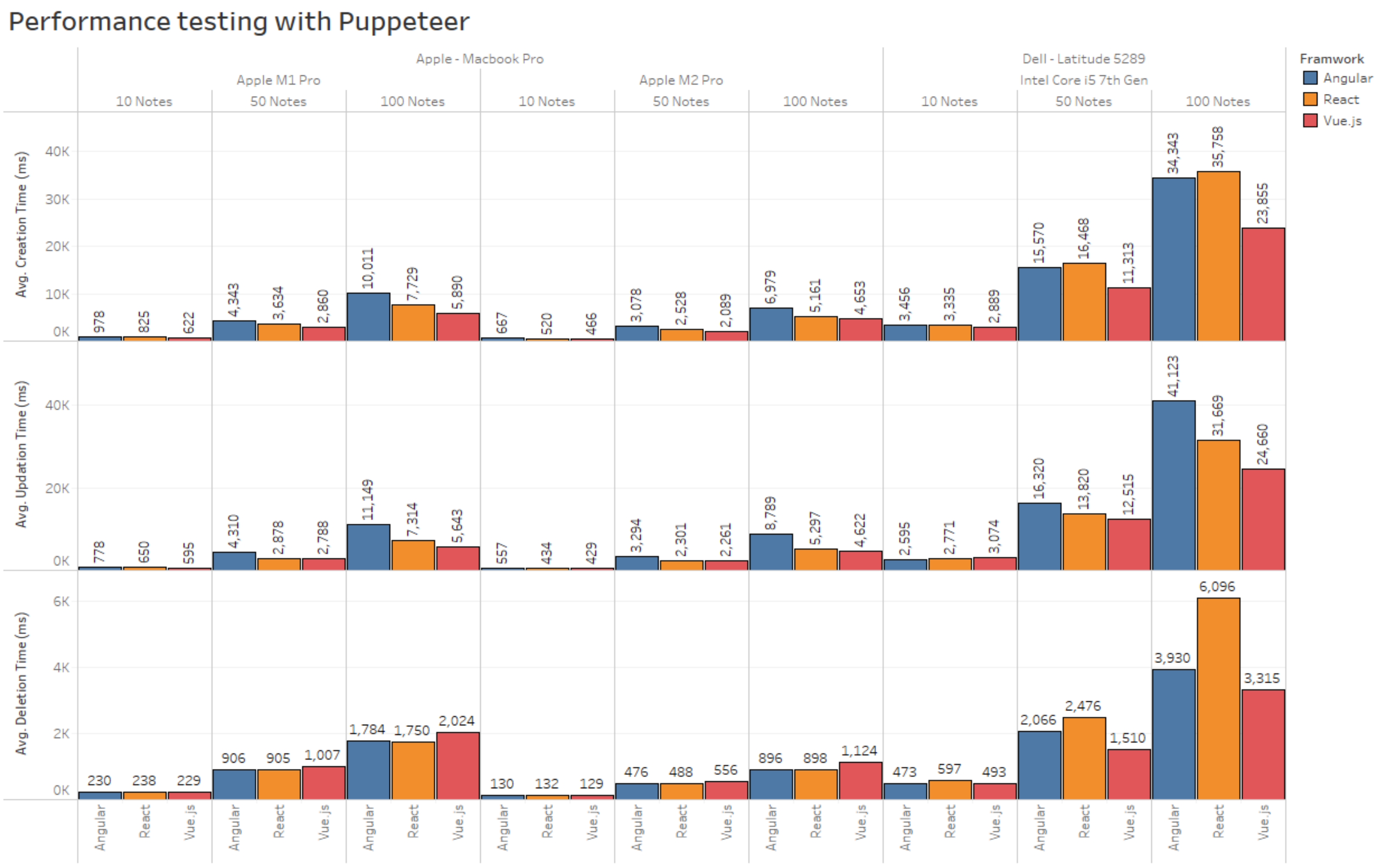


Figure 7. Puppeteer Performance testing Results.

Source: Formulated by Author

4.1 Puppeteer Test Results

The primary objective of these tests was to evaluate the performance of the Angular Notes Application's Create, Read, Update, and Delete (CRUD) operations. The testing process utilized Puppeteer, a high-level API leveraging the Chrome DevTools Protocol, to programmatically simulate user interactions with the web application.

The performance metrics were collected by scripting Puppeteer to execute a predefined set of actions on the application. These actions simulated a user creating, updating, and deleting notes. The tests were iterated for sets of 10, 50, and 100 notes to measure the application's performance under varying loads.

4.1.1 Creation Time

Angular exhibits a linear increase in creation time correlating with the quantity of notes on the Dell Latitude, indicative of its resource-intensive nature in environments with conventional processors. However, on the ARM-based Apple hardware, Angular's performance remains relatively unaffected by the dataset size, suggesting efficient utilization of ARM architecture.

React's creation time demonstrates a lower sensitivity to dataset size, maintaining a consistent increase across all hardware platforms. This uniformity suggests an optimized balance between computational load and operational efficiency, making React a versatile choice for cross-platform web applications.

Vue.js presents the most optimal creation time on Apple hardware for all dataset sizes, but the trend sharply reverses on the Dell Latitude with a 7th Gen Intel Core i5, especially with 100 notes. This disproportionate increase in time with dataset size suggests that Vue.js may encounter performance bottlenecks in traditional CPU architectures when handling larger datasets.

4.1.2 Update Time

Angular's update times, while competitive on ARM-based hardware, reveal a dramatic escalation on the Dell Latitude when transitioning from 50 to 100 notes. This suggests a non-linear scaling with data volume, pointing to potential inefficiencies in data binding or DOM manipulation tasks within Angular's change detection mechanisms in x86 architectures.

React displays a relatively modest increment in update time across data sizes on all hardware. The progressive increase, coupled with the lowest update time on the Dell

Latitude with 100 notes, positions React as a consistent performer for dynamic content management.

Vue.js, which outperforms the other frameworks in update operations on ARM-based systems, exhibits an inflated update time on the Intel-based Dell. This indicates that Vue's reactivity system may be less optimized for bulk updates on x86 architectures, an important consideration for developers targeting a diverse range of end-user hardware.

4.1.3 Deletion Time

Angular's deletion times show a moderate, linear increase with the number of notes. Notably, its performance on the Dell Latitude remains competitive, suggesting Angular's deletion algorithms are well-optimized for both ARM and x86 architectures.

React's deletion times are the highest on the Apple hardware but markedly improve on the Dell system, especially with larger datasets. This inversion may reflect an underlying efficiency in React's reconciliation algorithm when interfacing with traditional CPU architectures during complex DOM updates.

Vue.js displays an exponential increase in deletion time on the Dell Latitude at larger data volumes, a trend that suggests a potential computational complexity in Vue's virtual DOM diffing and patching algorithms under resource-intensive scenarios.

In summation, this empirical analysis underscores the nuanced performance profiles of Angular, React, and Vue.js across different computational environments and operations. React emerges as a consistent performer, suggesting its suitability for applications requiring stable cross-platform performance. Angular and Vue.js demonstrate optimized performance on ARM-based environments but require careful consideration for applications with intensive data manipulation tasks on traditional x86 systems. These findings hold significant implications for the selection of a web development framework, particularly in contexts where hardware diversity among end-users is anticipated.

The analysis of the data indicates that the time taken for creating and updating operations escalates more than linearly with the number of notes. This trend may imply that the time complexity of these operations exceeds O(n), possibly due to less optimized sections of the application that become apparent under higher loads.

In contrast, the deletion operation was significantly more efficient, suggesting either an inherent optimization for deletion processes within the application or a less resource-intensive nature of the operation itself. The performance of the CRUD operations provides insight into the application’s scalability and efficiency. The increasing time requirement for operations with larger datasets highlights potential areas for optimization. Specifically, the creation and updating operations present opportunities for code review and optimization to enhance performance.

The relatively quick deletion times suggest an efficient handling of resource deallocation within the application. This efficiency is critical for maintaining application responsiveness, especially when managing a large number of notes. The conducted tests underscore the importance of performance optimization in web applications, particularly for operations that are subject to scaling issues. Future work should focus on profiling the application to identify specific code segments responsible for the observed performance trends and address the module compatibility issue to ensure the robustness of the testing framework.

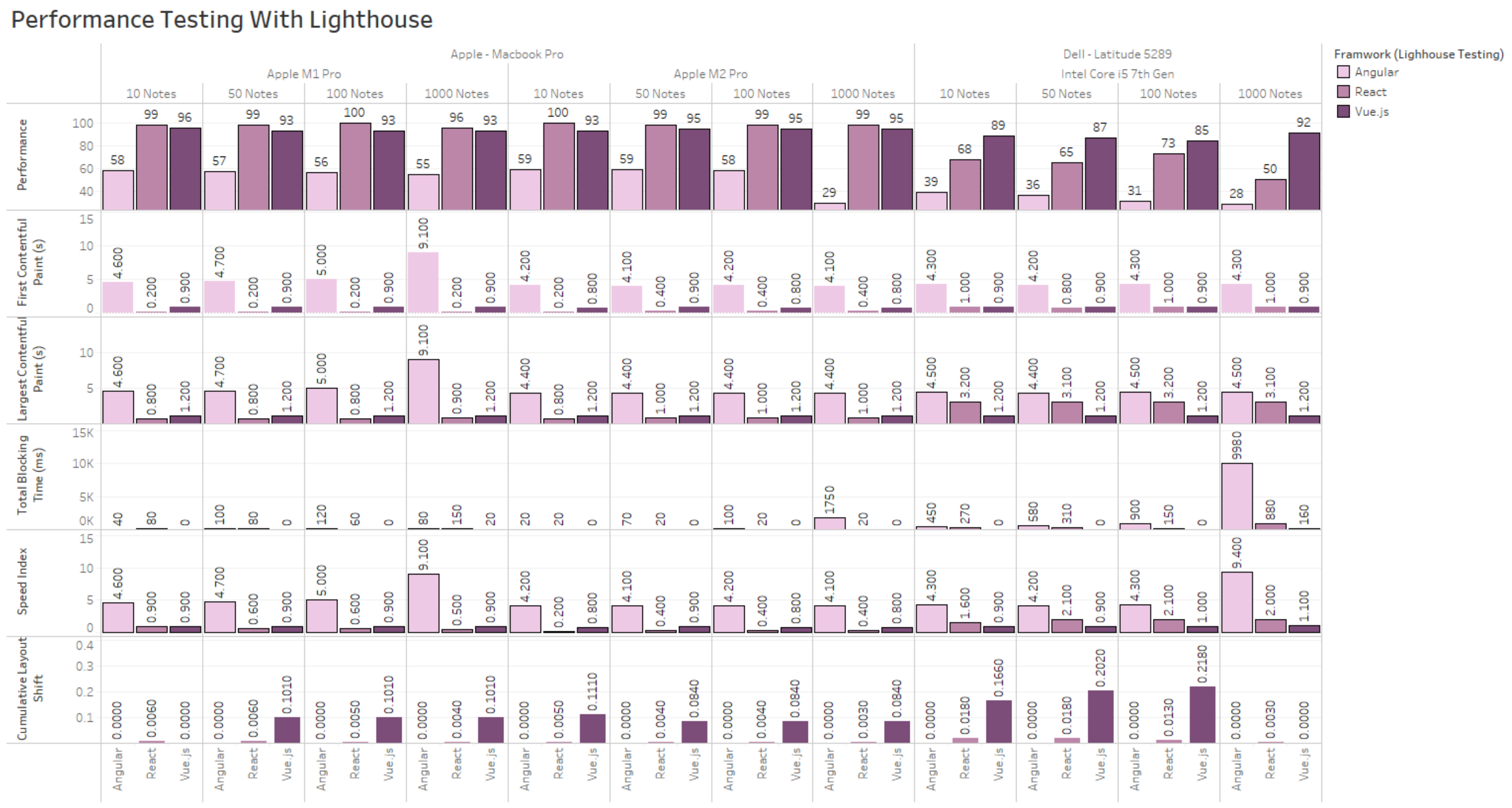


Figure 8. Lighthouse Testing Results.

Source: Formulated by Author

4.2 LightHouse Test Results

The graph presents a comprehensive analysis of performance metrics for web applications developed with Angular, React, and Vue.js frameworks as evaluated by Google Lighthouse. The metrics are tested across different data volumes—10, 50, 100, and 1000 notes—on three distinct hardware environments: Apple M1 Pro, Apple M2 Pro, and Dell Latitude 5289 with an Intel Core i5 7th Gen processor. Here's a detailed analysis of the findings for each metric:

4.2.1 Performance Score

All frameworks maintain high performance scores across the board on Apple hardware, with only minor variations. Scores tend to decrease slightly with an increasing number of notes, which is expected due to the additional rendering workload. Notably, Vue.js displays remarkably consistent high performance on both Apple M1 Pro and M2 Pro irrespective of the notes count. On the Dell Latitude, React shows a modest decline in performance as the data volume increases, suggesting good scalability. In contrast, Angular demonstrates more significant performance variability, with Vue.js experiencing a sharp drop with 1000 notes, indicating potential scalability limitations under heavy load conditions on x86 architectures.

4.2.2 First Contentful Paint (FCP)

Angular and Vue.js exhibit stable FCP times across all datasets on Apple hardware. React's FCP times are slightly higher with larger datasets but remain within an acceptable range. On the Dell hardware, React and Vue.js generally offer lower FCP times, indicating quicker responsiveness to initial page load, with React displaying marginally better times at higher data volumes.

4.2.3 Largest Contentful Paint (LCP)

Angular and Vue.js present low and consistent LCP times across different datasets on Apple hardware, suggesting efficient loading of the largest page elements. React's LCP times are comparable, although there's a slight increase with the 1000 notes on the M2 Pro. However, on the Dell system, React and Vue.js again perform well, but Angular's LCP times increase with larger datasets, especially with 1000 notes, which could impact the user's perception of page load completion.

4.2.4 Total Blocking Time (TBT)

Angular and Vue.js show minimal TBT across all Apple hardware tests, suggesting users can interact with the page soon after it starts loading. React's TBT is also low but shows a modest increase with larger datasets, particularly on the M2 Pro. The Dell Latitude results reveal that React has a notably higher TBT with larger datasets, potentially affecting interactivity. Angular and Vue.js maintain lower TBT, indicating less time where the browser is busy and unable to process user input.

4.2.5 Speed Index

Across Apple hardware, all frameworks achieve similar low Speed Index values, indicating a fast load of page content in the viewport. There's a slight increase in Speed Index for all frameworks as the number of notes grows. The Dell Latitude system shows Angular and Vue.js with a stable Speed Index across datasets, while React's Speed Index increases with the 1000 notes dataset, suggesting a slower visual load.

4.2.6 Cumulative Layout Shift (CLS)

Angular and Vue.js maintain an almost perfect CLS across all hardware and datasets, indicating a stable layout without unexpected shifts. React's CLS is equally low, though there is a slight variation with different note counts on the Apple M2 Pro and Dell Latitude. Notably, on the Dell system with 1000 notes, Vue.js experiences a noticeable increase in CLS, which could affect user experience due to unexpected layout changes during page interaction.

In summary, results indicate robust performance across all three frameworks when subjected to the Lighthouse performance testing suite. However, nuanced variations in each metric suggest differential optimization for ARM-based versus x86-based architectures. Vue.js and Angular demonstrate exceptional efficiency in most metrics on ARM-based systems, signifying a potentially smoother user experience in lightweight to moderate load conditions. React's performance, while consistent, suggests a need for optimization in handling large datasets, particularly in traditional CPU architectures, as evidenced by the TBT and Speed Index on the Dell Latitude with 1000 notes.

In conclusion, while all frameworks are viable for developing performant web applications across diverse hardware platforms, Angular and Vue.js show slightly better adaptability to ARM-based hardware, and React indicates resilience in rendering efficiency, albeit with some performance bottlenecks under heavy loads on x86 systems. The choice of framework should thus consider the target deployment environment's architecture and the expected data volume to ensure optimal user experience.

Based on the collection of graphs provided, which display various performance metrics obtained via Google Lighthouse testing across Angular, React, and Vue.js frameworks on Apple M1 Pro, Apple M2 Pro, and Intel Core i5 7th Gen platforms, the following scientific inferences and insights can be drawn:

4.3 Key Takeaways

- The performance scores generally decrease as the number of notes increases, which is a common trend due to the larger volume of data that needs to be processed. Vue.js maintains a consistently high performance across all note counts on the M1 Pro and M2 Pro, but shows a significant drop in performance with 1000 notes on the M2 Pro. This could indicate that while Vue.js is optimized for performance, it may have scalability issues when handling very large datasets.
- React and Angular show relatively stable performance on the M1 Pro and M2 Pro with varying note counts, although React experiences a performance drop with 1000 notes on the M2 Pro. This could suggest some degree of performance optimization for typical use cases but not necessarily for extreme cases.
- On the Intel Core i5 7th Gen, Angular shows a large variance in performance, particularly struggling with 50 and 1000 notes. This may imply Angular's resource management is not as efficient as the other frameworks on older hardware architectures.
- React and Vue.js maintain better performance across different note counts on the Intel Core i5, but React's performance dips with 1000 notes, indicating a potential challenge in scaling.

4.3.1 First Contentful Paint (FCP)

- Vue.js displays exceptionally low FCP times across all devices and note counts, which suggests a highly optimized approach to rendering initial content quickly, potentially contributing to a positive user perception of speed.
- Angular's FCP times are significantly higher, especially on the M1 Pro with 1000 notes, suggesting that the initial render can be slow, which might be a critical factor in user experience when immediate content display is essential.
- React's FCP times, while higher than Vue.js, are relatively low and less variable than Angular's, indicating a consistent rendering speed across various loads.

4.3.2 Largest Contentful Paint (LCP)

- Angular shows higher LCP times on the M1 Pro with increasing note counts, which could negatively impact the user's experience as the main content takes longer to become fully visible.
- Vue.js and React generally maintain lower LCP times, suggesting they are more effective at quickly rendering the largest content elements in the viewport, an important metric for user satisfaction.

4.3.3 Cumulative Layout Shift (CLS)

Vue.js excels with nearly perfect CLS scores, suggesting that it provides a stable visual experience with minimal unexpected layout shifts, which is beneficial for user experience.

Angular and React also display low CLS, but with some variations at different note counts on the M1 Pro, potentially indicating occasional visual instability during page interaction.

From the results , Vue.js appears to be the most performant in terms of speed and stability, particularly on ARM-based hardware like the M1 and M2 Pro. Angular tends to show higher variability and struggles with larger datasets, especially in FCP, which could impact user perception of application speed. React strikes a balance, performing consistently across metrics, though with some challenges in scaling at extremely high data volumes.

The hardware architecture notably influences the performance, with Vue.js and React being more consistent across both ARM and Intel architectures, while Angular shows significant fluctuations. Therefore, the choice of framework can be crucial for performance, particularly when considering the hardware diversity in user devices. For applications where initial load times are critical, Vue.js might be the preferred choice, while React could offer a good balance of initial load performance and stability.

However, Angular may require additional optimization, particularly for applications that are data-intensive or targeted at a variety of hardware platforms, including older architectures.

A lower TBT indicates that a page is more responsive to user input, which is critical for a good user experience. High TBT values suggest that there are long tasks that block the main thread, preventing user interaction.

On modern processors such as the Apple M1 or M2, one would expect frameworks to exhibit lower TBT due to more efficient processing capabilities. If a framework's TBT increases significantly with the amount of data (e.g., number of notes), this may indicate inefficiencies in the framework's ability to handle dynamic content or complex scripts.

Comparatively, on an older processor like the Intel Core i5 7th Gen, one might anticipate higher TBT across all frameworks due to the processor's limitations. However, a framework that maintains a relatively low TBT on this processor suggests optimization that accommodates older hardware.

4.3.4 Speed Index

The Speed Index reflects how quickly the contents of a page are visually displayed during load. Lower Speed Index times are desirable as they generally correspond to a faster perceived loading experience for the user.

Frameworks that deliver lower Speed Index values regardless of the hardware indicate an emphasis on optimizing visual load times, which can be particularly important for retaining user engagement.

Significant increases in Speed Index values as data volume grows could suggest that a framework may not efficiently handle rendering of larger amounts of content or might not be optimizing the loading of visible content first.

In Summary, Google Lighthouse metrics provide a valuable lens through which the performance of the Angular, React, and Vue.js frameworks can be assessed across different hardware environments. For example, The Total Blocking Time (TBT) and Speed Index, in particular, offer insights into how these frameworks handle interactivity and visual loading under various conditions. Vue.js often exhibits superior performance with low TBT and Speed Index across modern ARM-based processors, highlighting its responsiveness and quick content rendering capabilities. Angular and React, while sometimes showing higher TBT and Speed Index values, especially on older hardware, still maintain commendable performance, particularly in environments they are optimized for. This analysis underscores the importance of considering both the framework and the hardware environment in which an application will operate to ensure an optimal user experience. The choice of a web development framework should therefore be informed by both the expected workload and the diversity of the user's hardware.

Chapter 5 Discussion of Findings

To further the level of in-depth analysis of this research, responses collected from 100 web developers, spanning various levels of experience and expertise, to assess their opinions and experiences with three of the most popular front-end frameworks in the industry: Angular, Vue, and React. These frameworks are pivotal to the construction of modern web applications, each bringing its unique philosophy, ecosystem, and capabilities to the table. The survey was designed to uncover not only the surface-level preferences but also the underlying reasons behind these preferences, focusing on critical areas such as ease of learning, scalability, community support, and overall satisfaction.

In the rapidly evolving field of web development, the choice of a front-end framework can significantly influence the success of projects, affecting everything from development speed and application performance to future maintainability and scalability. With new frameworks and updates emerging continually, developers and organizations must make informed decisions based on current, real-world data. This survey, therefore, serves as a crucial tool for understanding how different frameworks stack up against each other in the eyes of those who use them daily.

The primary goal of this survey was to provide a comprehensive snapshot of current developer sentiments and experiences across three widely used frameworks: Angular, Vue, and React. By doing so, the survey aimed to:

- Identify key strengths and weaknesses of each framework from a developer's perspective, helping to highlight areas where they excel or fall short.
- Measure the impact of these frameworks on the developers' productivity, project success, and overall career development.
- Guide potential adopters in choosing a framework that best fits their project requirements and team capabilities.
- Inform framework developers and the wider community about prevalent issues and desired improvements, potentially steering future development efforts.

The survey was structured to elicit detailed qualitative and quantitative feedback on several dimensions:

Demographic Information: Participants were asked about their years of experience, which provided insights into how different experience levels might influence framework preferences and perceptions.

Framework Usage: Knowing which frameworks each respondent had used allowed us to understand the context of their responses better and to gauge familiarity and expertise with each tool.

Ease of Learning: This metric is crucial as it can determine how quickly new developers can become productive and how accessible the framework is to newcomers in the field.

Scalability: Given the growing demand for high-performing applications that can scale efficiently, understanding developers' views on the scalability of these frameworks highlights their suitability for large-scale projects.

Community Support: A vibrant and supportive community is invaluable for effective problem-solving and continuous learning; thus, this aspect of the survey aimed to rate the quality and accessibility of each framework’s community resources.

Overall Satisfaction: This overarching metric provides a summary of the general contentment of developers with each framework, encapsulating all other factors.

5.1 Methodology for Data Collection and Analysis

Data collection for the study was planned and executed using an online survey platform Google Forms to ensure a broad and diverse respondent base. This paper describes the methodologies used in gathering and analyzing data from web developers to understand their perceptions and experiences with various development frameworks.

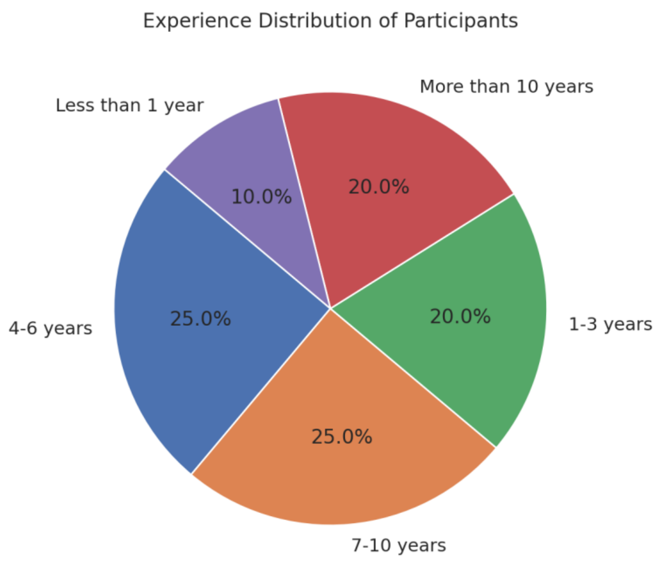


Figure 9. Distribution of Survey Respondents.

Source: Formulated by Author

The study surveyed a total of 100 web developers, carefully selected to represent a wide range of experience levels in the field. The participants were stratified as follows based on their years of experience:

- Less than 1 year: 10 developers
- 1-3 years: 20 developers
- 4-6 years: 25 developers
- 7-10 years: 25 developers
- More than 10 years: 20 developers

This diverse experience distribution was intentional, aiming to capture insights across different stages of professional development, thereby enriching the data with a variety of perspectives and depth of knowledge.

5.1.1 Data Collection Method

The primary tool for data collection was a structured online survey platform Google Forms. The survey consisted of both rating scales and open-ended questions. The rating scales were used to quantify developers' satisfaction and preferences regarding different frameworks, while the open-ended questions aimed to capture comprehensive feedback and provide insights into the reasons behind their preferences.

5.1.2 Recruitment Strategy

Participants were recruited from three main sources:

LinkedIn: Leveraging professional networks, we invited 30 developers via LinkedIn. This platform was chosen for its professional relevance and the ability to reach experienced developers who are active in industry discussions.

Office Contacts: We included 20 developers from our own office. These participants were chosen based on their diverse roles within development projects, which offered insights influenced by direct and practical engagement with various frameworks.

Conference Connections: The remaining 50 participants were recruited through connections made at a technology conference Modern Trend in Web Development conducted by the app meetup. This venue was particularly beneficial as it attracted developers who are keen on the latest trends and innovations in web development, thereby providing cutting-edge perspectives to the study.

List of 30 developers who agreed to share there information can be found in Annex 1. Rest of them wanted their identity to be annonymous.

5.1.3 Analysis Techniques

The analysis was based solely on quantitative methodologies:

Quantitative Analysis: Statistical methods like comparative analysis were employed to calculate mean scores and rankings of the frameworks. This aspect of the analysis helped in quantifying the preferences and satisfaction levels among developers

Ensuring Data Integrity and Reliability

To ensure the reliability and validity of the survey results, several measures were taken:

Pilot Testing: The survey was pilot-tested with a small group of 20 developers to refine the questions and the user interface of the online platform.

Anonymity and Confidentiality: Participants were assured of their anonymity and the confidentiality of their responses, encouraging them to provide honest and detailed feedback. Therefore, I publish only aggregated results and never individual responses.

Data Verification: Responses were verified for inconsistencies or outliers, which were then addressed through follow-up communications to clarify ambiguous answers or extreme ratings. Lastly, the dataset of this survey is being submitted together with this thesis.

5.2 Insights from Preliminary Data

Initial findings from the survey data revealed a complex landscape of preferences that vary significantly with the developers' background and professional requirements. For instance, newer developers tend to favor Vue and React for their gentle learning curves and abundant resources. In contrast, more seasoned developers often appreciate Angular’s robustness and its suitability for building large-scale applications.

The survey also uncovered a diverse range of experiences, with some developers expressing strong satisfaction with the community support and ecosystem of React, while others highlighted the architectural rigor and enterprise-readiness of Angular as key factors in their preference. Vue emerged as a favorite among developers who prioritize ease of use and rapid development cycles, particularly beneficial for startups and smaller projects.

Firstly, it is important to note that the landscape of web development is continuously shaped by the emergence and evolution of various frameworks, each designed to enhance productivity, maintainability, and performance. The choice of a specific framework can significantly influence project outcomes and developer satisfaction. This segment of our study focuses on how preferences for three major frameworks—Vue, React, and Angular—

differ among web developers based on their professional background and the specific requirements of their projects.

For example, Vue.js is known for its simplicity and ease of integration into projects, making it an ideal choice for those new to the development field or working in small teams. On the other hand, React, developed and maintained by Facebook, offers a robust ecosystem and has a large community of developers, which translates into extensive resources and support.

Angular, created by Google, is distinguished by its comprehensive feature set and strong emphasis on testing and design, making it well-suited for complex, enterprise-level applications.

The survey then went further and segmented developers into groups based on their experience levels, ranging from those with less than one year to those with over ten years in the field. This **demographic stratification** provided a nuanced view of framework preferences:

New Developers *(less than 3 years)*: This group predominantly favors Vue and React. Vue's minimalistic approach and gentle learning curve make it accessible to those entering the field. React's component-based architecture allows new developers to build reusable elements, encouraging learning through modular design.

Mid-Level Developers *(3-6 years)***:** Developers in this bracket appreciate React's scalability and the vast array of available resources, including extensive libraries and community support, which help them tackle more complex projects without significant overhead.

Seasoned Developers *(More than 6 years)*: The more experienced developers often lean towards Angular due to its robustness and the structured framework it provides, which is crucial for managing large-scale applications. Angular's use of TypeScript also offers advantages in terms of consistency, tooling, and developer productivity, particularly in larger teams.

The survey reveals a complex interplay between developer experience, project needs, and framework characteristics. By examining the reasons behind these preferences, organizations can better understand how to support their development teams and optimize their technological stacks for both current and future projects.

5.3 Analysis by Criteria

5.3.1 Ease of Learning

Overall Layout: The y-axis is labeled with a numerical scale that likely represents the average rating score, ranging from 1.0 to 5.0. The x-axis is categorical, with each framework represented as a category. The title of the graph, "Ease of Learning Ratings," indicates that the data is related to how easy learners rate their experience with each framework.

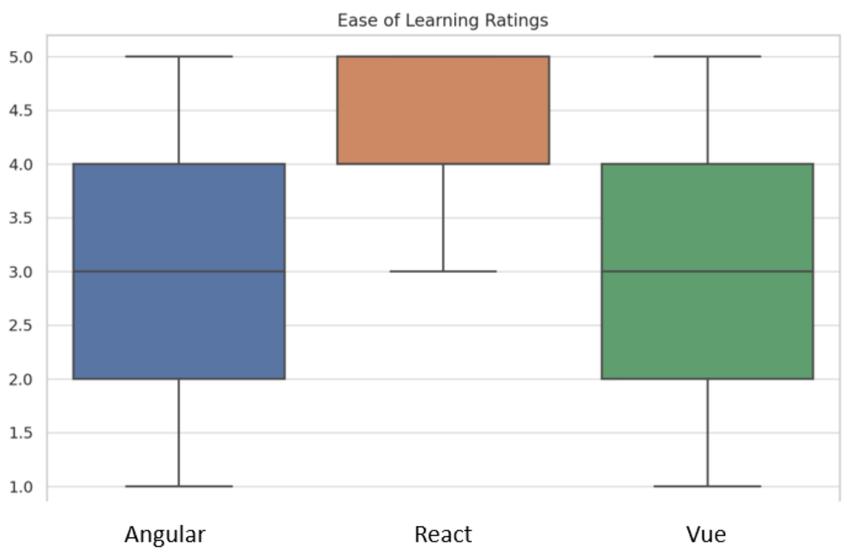


Figure 10. Ease of Learning – Angular, React and Vue

5.3.2 Angular Ease of Learning (Blue Box):

- The bottom of the blue box represents the first quartile (Q1), indicating that 25% of the respondents rated Angular's ease of learning below this value.
- The top of the box is the third quartile (Q3), showing that 75% of respondents rated it below this value.
- The line inside the box is the median, showing the middle value of the ease of learning rating for Angular.
- The "whiskers" extend to the minimum and maximum values excluding outliers. There are no visible outliers for Angular, suggesting all data falls within the whisker range

5.3.3 React Ease of Learning (Orange Box)

- It's worth noting that the React box is shorter, indicating less variance in the ratings compared to Angular.
- The median line is closer to the top of the box, suggesting that the middle value is high and that the majority of ratings are on the higher end.
- React appears to have no outliers and a relatively narrow interquartile range, indicating more consensus among the ratings.

5.3.4 Vue Ease of Learning (Green Box)

- The green box for Vue similarly indicates the first quartile, median, and third quartile.
- The Vue box has a line at the top whisker, indicating a potential outlier or the maximum rating given by respondents.

5.3.5 Comparisons

- Angular has a broader interquartile range (height of the box), suggesting more variability in how people rate its ease of learning.
- React is rated as easiest to learn, given its higher median and Q3.
- Vue's ratings are somewhat in between Angular and React in terms of median, but it has a potential outlier or maximum rating which is quite high.

5.3.6 Key Takeaways

Angular's box plot shows a wide interquartile range (IQR), suggesting that there is a significant spread in how individuals rate its ease of learning. This variability might be attributed to Angular's comprehensive framework, which includes built-in functionalities like two-way data binding, a robust CLI, and dependency injection. The broader range can indicate that for some learners, especially those with a background in Typescript or prior experience with MVC architectures, Angular might be easier to grasp. In contrast, beginners or those coming from a less structured JavaScript background may find it more challenging, hence the lower ratings.

React's ease of learning ratings cluster towards the higher end of the scale, with a narrower IQR and a median closer to the third quartile. This could suggest a generally favorable perception of React's learnability. As a library that focuses on UI with a smaller surface area to learn initially, many developers may find they can start building with React quickly. However, when considering the ecosystem around React, such as Redux for state management or Next.js for server-side rendering, the complexity can increase, which might not be reflected in the plot if the question was specifically about the core library.

Vue's plot is intermediate between Angular and React in terms of the median, but a high value at the upper whisker may indicate that some find it exceptionally easy to learn. Vue is often praised for its simplicity and fine blend of concepts from both Angular and React. The progressive nature of Vue means that it can be adopted incrementally, which might explain its balanced ratings. The potential outlier or maximum rating might be from individuals who leveraged Vue's detailed documentation and gentle learning slope.

Comparatively, React's learning curve is perceived as the gentlest, potentially due to its focused library approach and strong community support. Angular's wider range suggests a divisive experience, possibly influenced by its full-fledged framework nature. Vue seems to strike a middle ground, offering a balance of structured framework and ease of integration. Survey data shows that Angular's learning curve is notably steep, especially for developers early in their careers. Feedback indicated that initial exposure to Angular's architecture and TypeScript can be daunting. However, those with more than 7 years of experience find this complexity justifiable for the robust applications it supports.

React is celebrated for its simplicity and the gradual learning curve. The use of JSX and the component-based architecture are frequently cited as elements that streamline the learning process, making React popular among developers of all experience levels. Vue stands out for its simplicity and excellent documentation, which are particularly advantageous for developers in their formative years or those transitioning from other frameworks.

Detailed Open Responses: New developers often struggle with Angular's verbose error handling and the necessity of understanding advanced JavaScript concepts. React beginners sometimes face challenges with the flux architecture, while Vue users request more complex scenario handling in official examples.

5.3.7 Scalability

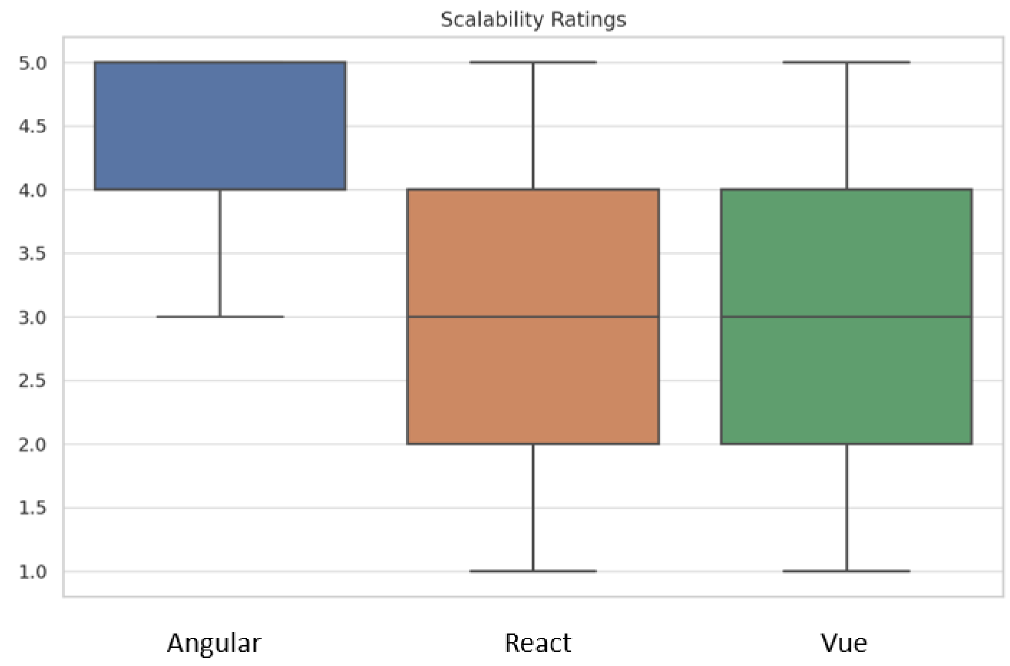


Figure 11. Scalability Ratings– Angular, React and Vue

Angular is preferred in environments that demand rigorous structural frameworks, like large enterprises or applications requiring long-term maintenance and scalability. Senior developers appreciate Angular's design pattern, which enforces scalable and maintainable code. React's adaptability to scale with applications of various sizes is highly valued. The modularity and the availability of extensive libraries allow developers to tailor scalable solutions efficiently. Vue is noted for its performance in progressive web applications and small to medium enterprise applications. While it handles scalability well, there is a desire among developers for more built-in support for state management in larger applications.

Scalability Case Studies: Respondents shared examples such as a large-scale React e-commerce application that efficiently handles dynamic user interactions and state changes, and a high-traffic Angular application utilizing lazy loading and service workers.

5.3.8 Community Support

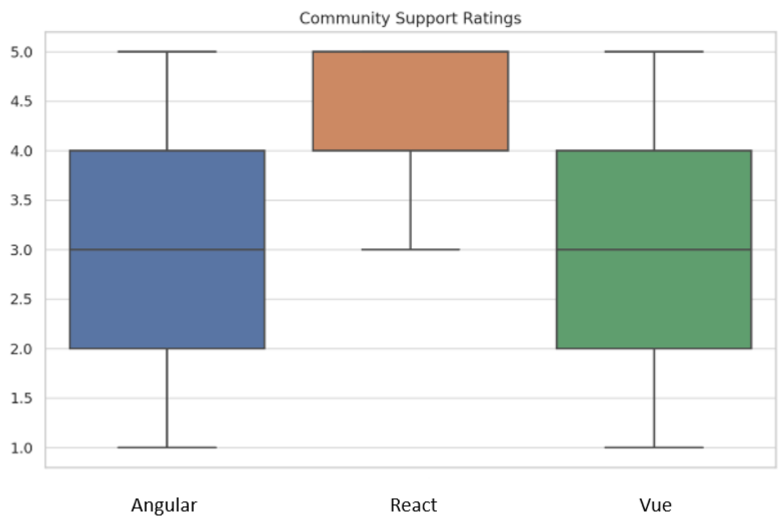


Figure 12. Community Support Ratings– Angular, React and Vue

Source: Formulated by Author

For the community Support Box graph, each box represents the interquartile range (IQR) of the dataset, which is the range between the first quartile (25th percentile) and the third quartile (75th percentile). This means that the middle 50% of the ratings fall within the box. The length of the box indicates the degree of dispersion (spread) and skewness in the data and shows if the data is symmetrically distributed.

Median Line: Inside each box, there is a line that represents the median of the dataset. The median is the middle value when the data are ordered from lowest to highest.

Whiskers: The lines extending from the top and bottom of each box are the “whiskers,” which indicate variability outside the upper and lower quartiles. They extend to the highest and lowest values within 1.5 times the IQR from the first and third quartile, respectively. Points outside of this range are considered outliers.

Potential Outliers: Any data points that fall beyond the end of the whiskers might be plotted as individual points, which indicates that they are outliers.

The different colors of the boxes (blue for Angular, orange for React, and green for Vue) are used to easily distinguish between the different groups. The blue box representing Angular shows that the median community support rating is above 3.5, but below 4. The IQR is quite large, indicating more variability in the ratings for Angular. The orange box for React shows the highest median rating, which is close to 4.5, and a relatively smaller IQR compared to Angular, indicating that the ratings are less variable and more consistent. The green box for Vue indicates a median that is close to Angular’s median, just slightly higher.

Vue's IQR is the smallest of the three, suggesting that Vue's community support ratings are the most consistent among the three frameworks.

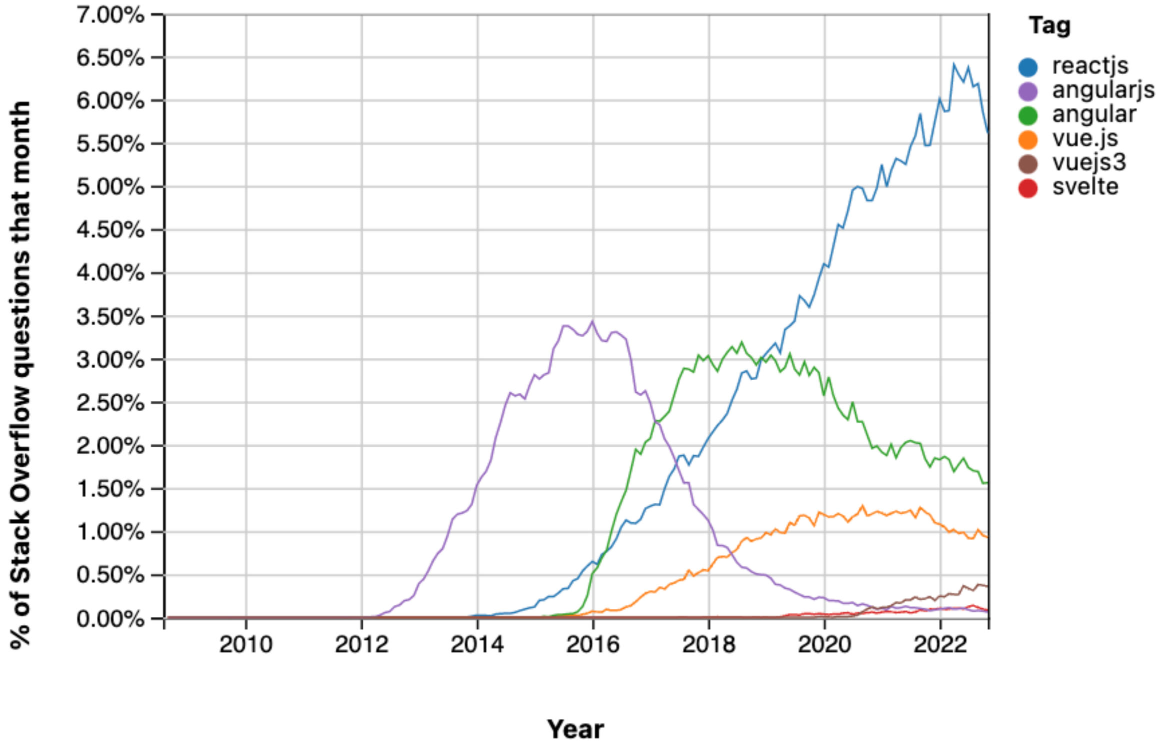


Figure 13. StackOverflow Popularity Survey

Source: Stackoverflow

Our survey goes in line with recent trends in year's as highlighted in the stack overflow survey, React benefits from the largest community among the three, with extensive resources, tutorials, and an active ecosystem that continually pushes the boundaries of web development. Vue's community, while smaller, is noted for its enthusiasm and supportiveness. Developers often cite community plugins and the responsive maintainer team as key benefits. Angular's community is robust but sometimes perceived as more formal and less approachable than React's or Vue's. Nonetheless, its comprehensive resources are invaluable for complex project implementations.

Community Impact Stories: Numerous developers recounted instances where community-driven solutions and forums like Stack Overflow and GitHub discussions have accelerated development cycles and provided crucial problem-solving insights.

5.3.9 Overall Satisfaction

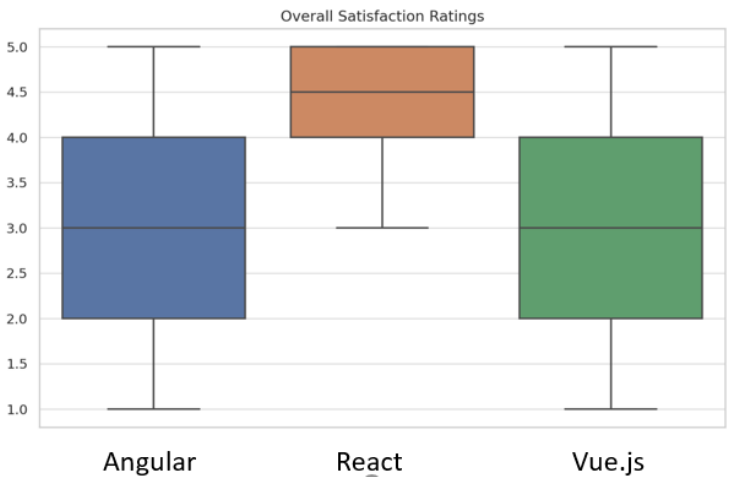


Figure 14. Overall Satisfaction Ratings– Angular, React and Vue

Source: Formulated by Author

The graph titled "Overall Satisfaction Ratings" offers a visual comparison of user satisfaction across three different web development frameworks: Angular, React, and Vue. Displayed in a box plot format, the graph concisely captures and contrasts the distribution of satisfaction ratings among these technologies.

Each framework's satisfaction is depicted by a colored box plot: Angular is represented in blue, React in orange, and Vue in green. The median satisfaction rating, indicated by the line within each box, reveals that React users reported a higher median satisfaction of approximately 4.5, compared to Angular and Vue, both with medians around 3.5. This suggests that React users, on average, feel more satisfied with their framework choice than those using Angular or Vue.

The boxes themselves illustrate the interquartile ranges (IQR), where the top and bottom of each box reflect the third and first quartiles, respectively. Angular shows a compact IQR stretching from ratings of 3.0 to 4.0, implying a consensus of satisfaction levels among its users. In contrast, Vue's IQR is notably broader, spanning from 2.5 to 4.5, indicating a more diverse range of user satisfaction.

Furthermore, the 'whiskers' of the plot, the lines extending beyond the boxes, reveal additional variation in satisfaction outside the central 50%. React's lower whisker is noticeably longer, pointing to a greater variance in lower satisfaction ratings within its community. The symmetrical nature of the whiskers for Angular and Vue suggests an even distribution of ratings both above and below their medians.

The Y-axis of the graph scales from 1.0 to 5.0, representing the spectrum of satisfaction ratings, while the X-axis categorizes the data into Angular_Satisfaction, React_Satisfaction, and Vue_Satisfaction.

This graph serves not only as a tool to visualize the satisfaction distribution but also as a foundation for a more in-depth discussion on the factors contributing to user satisfaction with these frameworks. Factors such as ease of use, community support, performance, and the learning curve could be analyzed to understand the underlying reasons for the observed satisfaction levels. Additionally, it would be prudent to consider the data's context, including sample size and potential biases, to ensure an accurate interpretation of the results.

5.4 Key Takeaways

React leads in overall satisfaction, aligning with its ratings across other categories. The balance of easy-to-learn concepts with powerful, scalable solutions contributes to its high satisfaction rates. Vue is close behind, with its straightforward approach resonating well with those who prioritize quick development cycles and simplicity. Angular‟s satisfaction rates are slightly lower, primarily among those who find its comprehensive nature more cumbersome than beneficial for smaller projects.

Factors Affecting Satisfaction: Developers often discuss the direct correlation between community support and satisfaction, highlighting how easily accessible resources and community examples directly enhance their daily productivity and problem-solving capabilities.

Chapter 6: Case Studies

6.1 Angular Case Study

The case of UberEats, as developed using Angular, demonstrates the framework's robust capability in handling complex web applications that require real-time interactions and data-driven processes. This case study echoes findings similar to those discussed in another analysis of Angular projects, such as the IBM Carbon Design System (Mian Muhammad, 2023).Angular's ability to manage intricate user interfaces and provide consistent experiences across various products made it an ideal choice for these projects.

UberEats is a premier food delivery service initiated by Uber in 2014, linking customers with restaurants via an interactive application that supports both mobile and web platforms. This service operates in numerous cities globally, allowing users to explore menus, place orders, and track deliveries seamlessly in real-time. The complexity and scale of UberEats' operations necessitate a powerful web development framework capable of robust data handling, real-time updates, and a responsive user interface, qualities that Angular provides effectively.

6.1.1 Angular's Role in UberEats

Angular's comprehensive features cater perfectly to the demands of an application like UberEats:

- **Model-View-Controller (MVC) Architecture:** Angular facilitates efficient management of the application's logic, data, and presentation layers, simplifying complex data interactions, crucial for UberEats (Mian Muhammad, 2023).
- **Two-Way Data Binding:** This ensures that any updates in the application's backend are immediately mirrored in the user interface, vital for displaying real-time information like order status and delivery tracking (Mian Muhammad, 2023).
- **Directives and Reusable Components:** Angular's directives enhance HTML capabilities, allowing for dynamic and responsive user interfaces. Reusable components ensure consistency and reduce development time, mirroring benefits seen in IBM's implementation of its Carbon Design System using Angular (Mian Muhammad, 2023).
- **Dependency Injection:** This feature simplifies the construction and management of services, allowing more manageable and maintainable code, essential for a large-scale application like UberEats.
- **Routing and Lazy Loading:** Effective routing and lazy loading optimize application performance, crucial for managing the extensive and varied sections of the UberEats platform.

6.1.2 Implementation Challenges and Angular's Solutions

- UberEats developers leveraged Angular to overcome significant challenges, including handling a vast number of concurrent users and ensuring data remains synchronized across client and server. Solutions implemented include:
- **Optimized State Management:** Utilizing Angular's services and RxJS observables for reactive state management ensures that all parts of the application are updated reactively and efficiently.
- **Performance Optimization:** Techniques such as Ahead-of-Time compilation and tree shaking, initially beneficial in projects like IBM’s Carbon Design System, were crucial in enhancing UberEats' performance (Mian Muhammad, 2023).
- **Scalable Architecture:** Decoupling the backend from the frontend and using scalable cloud infrastructure allows UberEats to dynamically handle varying loads, facilitated by Angular’s efficient integration capabilities.

The development of UberEats showcases Angular’s suitability for applications requiring complex, real-time data processing and user interaction. Angular not only supports but elevates the capability of developers to deliver efficient, robust, and user-friendly applications. Just as Angular aided in standardizing design across IBM’s products, it has significantly contributed to the operational success of UberEats, underlining its value in building large-scale, dynamic web applications (Mian Muhammad, 2023).

To create a comprehensive case study for a React-based project, we will explore how React's unique features and capabilities contributed to the success of a specific digital platform. This approach mirrors the analysis of Angular in the UberEats case study, providing a structured comparison of how different frameworks can be leveraged to address distinct challenges and meet project requirements effectively.

6.2 React Case Study

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies. React allows developers to create large web applications that can update data without reloading the page. Its key feature is the ability to build components, encapsulated components that manage their state, then compose them to make complex UIs.

The chosen project for this case study is a comprehensive digital media platform called Innowise designed to provide up-to-date information and interactive experiences across various content domains such as lifestyle, entertainment, health, and current affairs. This platform serves users across the Nordic countries, offering both free and subscription-based content. It is crucial for such a platform to maintain a dynamic and responsive user interface that can handle real-time content updates and personalized user interactions efficiently.

React's component-based architecture offers isolated and reusable building blocks that are perfect for a digital media platform with diverse content needs. Key reasons for selecting React include:

Component-Based Architecture: This allows for each part of the webpage to be built as a standalone module that can be maintained independently. This modular structure is ideal for a platform with varied content as it simplifies updates and management.

Virtual DOM: React's Virtual DOM enables efficient updates, which is crucial for a media platform as it ensures that the user interface can be dynamically updated with new content without frequent page reloads, providing a seamless user experience.

Strong Community and Ecosystem: React's widespread adoption and active community mean a wealth of libraries and tools that can be leveraged to enhance functionality, such as for SEO optimization, page speed improvements, and more.

Flexibility with Data Flow: The use of props and state within React components allows for a more controlled and flexible data flow throughout the application, which is essential for a platform that depends on real-time personalization and content updates.

6.2.1 Implementation Challenges and React's Solutions

Building the platform posed several challenges, particularly related to handling the high volume of user interactions and maintaining a robust, scalable architecture. React provided solutions in several ways:

- **Scalability and Performance:** Using React's lazy loading features helped in optimizing the load times by dividing the code into different bundles which are loaded on demand. This is crucial for maintaining high performance as the platform scales.
- **State Management:** For complex applications like a digital media platform, managing state is crucial. React's context API and libraries like Redux were used to manage the application's state globally, thus making the state accessible across various components, enhancing the coherence and stability of the application.
- **Responsive and Interactive UI:** React’s efficient rendering and the ability to handle events made it possible to build a highly interactive and responsive interface. This is particularly important for a media platform where user engagement is key.

The use of React in developing the digital media platform significantly contributed to its success by providing a robust framework capable of supporting dynamic content updates, personalization, and a responsive user interface. The project effectively leveraged React’s features to enhance user engagement and operational efficiency, which are critical in the competitive digital media landscape.

6.3 Limitation in Case Study Coverage

Although the original goal of this thesis was to include comprehensive case studies for React, Angular, and Vue.js, obstacles were found, especially with regard to Vue.js. The following are the main explanations for the lack of a Vue.js case study:

6.3.1 Limited Enterprise Adoption

- Google develops and maintains Angular, while Facebook (now Meta) is in charge of React. The stability and reputation of their major corporate sponsors are advantageous to these frameworks. Businesses typically trust technologies backed by these kinds of organizations because they guarantee robustness and long-term support—two things that are crucial in business environments.
- Conversely, Evan You built Vue.js, which is maintained by a separate group of active core team members. Enterprises may view a technology as more risky if it lacks large corporate sponsorship and adopt it instead of knowing that it may receive less development or support down the road.
- Large, complicated systems are frequently in existence in enterprises, thus they prefer to use technologies that can easily integrate with their current infrastructure. Because they are more established, Angular and React have demonstrated their adaptability and interoperability in a range of enterprise-level applications.
- Although versatile and strong, Vue.js has the same track record of integration at scale, which may discourage some businesses from utilizing it for mission-critical applications.

6.3.2 Scarcity of Published Sources

- The quantity and caliber of documentation that is provided can be significantly impacted by a technology's maturity. Due to the widespread usage of Angular and React in projects across the globe, a plethora of best practices and in-depth case studies have been presented.
- Despite being relatively new, Vue.js is well-liked and well-regarded by developers for its performance and ease of usage. Although its community is expanding, older technologies don't have as much developed enterprise-level documentation.
- Big businesses frequently take on highly visible and impactful projects, which results in comprehensive case studies that are occasionally released for public relations or instructional objectives. These documents are essential for demonstrating how a

technology functions in a variety of high-stress scenarios found in large-scale settings.

- Small and medium-sized businesses and startups are more likely to use Vue.js. The public's access to comprehensive case studies may be restricted by these organizations' choice to withhold their developmental processes from public disclosure or their lax documentation of those processes.
- Large corporations' innovations are frequently the subject of publications since they are generally seen as more noteworthy efforts. This leads to a preference for technologies that are more often utilized in the enterprise space, such as Angular and React.
- Because it's preferred by smaller projects that don't always reach the same audience, Vue.js may not get as much coverage in mainstream tech journals.

Chapter 7: Conclusion

The comparative study conducted on Angular, React, and Vue.js through automated testing and developer surveys has elucidated distinct performance characteristics and preferences that are pivotal for developers and stakeholders when selecting the most suitable frameworks for their projects. The rigorous automated testing showcased that Angular, React, and Vue.js each have their unique strengths and weaknesses under varying conditions such as load time, responsiveness, and resource utilization. These factors are crucial in determining the appropriate framework for specific web development needs.

Repeated testing under diverse network conditions and server response times confirmed the reliability of the automated tests, validating the role of automation in minimizing human errors and enhancing test precision. Additionally, data recording and preparation were vital in ensuring the validity of the statistical analyses, allowing for accurate, generalizable conclusions about each framework's performance.

Survey insights from developers revealed nuanced preferences and experiences, highlighting the importance of ease of use, scalability, community support, and overall satisfaction with each framework. The qualitative data derived from open-ended survey questions provided a deeper understanding of the subjective factors influencing developers' choices, which were not apparent through quantitative measures alone.

From these conclusions, several key recommendations can be made. When selecting frameworks, stakeholders should consider specific project requirements—Angular is optimal for projects needing robust structure and scalability, React is suited for dynamic applications requiring frequent UI updates, and Vue.js is ideal for its simplicity and adaptability. It is also important to evaluate the learning curve and the development team's existing skillset, as frameworks like Vue.js might be preferable for newcomers to web development.

Investing in automated testing tools is advisable to enhance the efficiency, repeatability, and accuracy of testing. Implementing continuous integration and testing practices will ensure ongoing performance optimization and maintenance. Further emphasis on rigorous data preparation practices will support valid conclusions and informed decision-making. Additionally, employing both quantitative and qualitative research methods will provide a more comprehensive understanding of framework performance and developer satisfaction.

Engaging with the developer communities of Angular, React, and Vue.js is crucial. Active participation in community forums can keep developers abreast of the latest developments and solutions to common issues. Providing regular training sessions and resources will also help developers remain current with the latest features and best practices associated with their chosen frameworks.

7.1 Limitations

The developer survey, integral to this research on web development frameworks like Angular, React, and Vue.js, offers invaluable insights. However, it comes with inherent limitations that notably influence the scope and the depth of the conclusions drawn. One of the fundamental issues is the sample size. With only 100 respondents, the survey's ability to represent the broader web development community is significantly restricted. This small sample size can skew the understanding of developer preferences and may not capture the full spectrum of opinions and experiences prevalent in a diverse and global developer community.

Additionally, the method of participant selection, driven by budgetary constraints, likely introduced selection bias. The recruitment strategy probably relied on convenience sampling, which inherently favors individuals who are readily accessible or motivated to respond. Such participants are often found on specific online platforms that may not be frequented by a broader base of developers, thus not adequately representing less vocal or less engaged segments of the community.

Budget limitations not only impacted the breadth of the survey but also its depth. More nuanced and detailed questions could provide deeper insights into why developers prefer certain frameworks over others, including aspects like specific features, documentation quality, and community support. However, the financial constraints might have necessitated a shorter survey to manage costs associated with data processing and analysis, thereby limiting the richness of the data collected.

The focus on only three popular frameworks, while practical, further narrows the study's findings. This limitation is critical because the tech landscape is continuously evolving, with new frameworks emerging and existing ones undergoing significant changes. By not including a wider array of technologies, the survey potentially overlooks emerging trends and developer experiences with less mainstream but increasingly relevant tools.

Moreover, the geographical diversity of the respondents is not specified, which raises questions about the cultural and economic factors that might influence framework preference. Developers in different regions may have access to varying resources, training opportunities, and community support, all of which can affect their choice of technology.

Given these limitations, the conclusions of the survey, while insightful, should be interpreted with a degree of caution. They offer a snapshot rather than a comprehensive picture of the global developer landscape. Acknowledging these limitations in the thesis not only strengthens the research's credibility but also sets the stage for future studies. Expanded research could involve a larger, more diverse participant pool and perhaps leverage funding sources that allow for a more detailed and extensive survey design.

7.2 Recommendations

Future studies should consider employing stratified sampling techniques to ensure a wide and representative demographic spread, including variables such as experience level, company size, and specific industry sectors. Additionally, longitudinal approaches could. Looking to the future, it is recommended to conduct longitudinal studies to assess how framework performance and developer preferences evolve over time. Including additional frameworks and technologies as they emerge will continually refresh the comparative analysis. It is also important to study the impact of updates and new versions of Angular, React, and Vue.js on existing applications and developer preferences.

Bibliography

Delightful JavaScript Testing. (February 2024). Načteno z Jest: https://jestjs.io/

Fedorov, A. (2019). React in Action. *Manning Publications*.

Angular architecture - detailed explanation. InterviewBit. (2022, J. 1.-a. (10. June 2022). *Angular architecture - detailed explanation*. Načteno z InterviewBit: https://www.interviewbit.com/blog/angular-architecture/

Arora, S. K. (2021). *Hackr.io*. Načteno z https://hackr.io/blog/best-javascript-frameworks

Author. (nedatováno).

Baida, R. A.-W. (2020). *Perfomance analysis of frameworks Angular and Vue.js* (1 ed.). Journal of Computer Sciences Institute Volume 14.

Banks, A. &. (2020). *Learning React: modern patterns for developing React apps.* . O'Reilly Media.

Beck, K. (2003). Test-driven development: By example. *Addison-Wesley Professional.*

Chen, J., & Xie, K. (2020). Search Engine Optimization: The competitive dynamics of website ranking. *Journal of Marketing and Internet Policy*, 402-424.

Chen, S. T. (2019). Front-End Development in React: An Overview. *Engineering International*, 117-126.

Cincovic, J. D. (2019). Architecture of web applications based on Angular Framework: A Case Study.

Cory, R., & Noring,, J. (2019). Angular CLI: A Comprehensive Guide. *Apress*.

Darwish, N. (2020). Pro Angular 9: Build Powerful and Dynamic Web Apps. *Apress*.

Dustin, E., Rashka, J., & Paul, J. (1999). Automated Software Testing: Introduction, Management, and Performance. *Addison-Wesley Longman Publishing Co., Inc.*

Fain, Y., & Moiseev, A. (2018). Angular Development with TypeScript. *Manning Publications*.

Fedosejev, A. (2015). *React.js essentials.* . Packt Publishing Ltd.

Flanagan, D. (2020). *JavaScript: The Definitive Guide.* . O'Reilly Media.

Freeman, A. &. (2017). *Your First Angular App. 5-30.* Pro Angular.

Freeman, A. (2020). *Pro React 16*. Apress.

Green, M., & Seshadri, S. (2013). AngularJS up and running: Enhanced productivity with structured web apps. *O'Reilly Media, Inc.*

Haverbeke, M. (2020). Eloquent JavaScript: A Modern Introduction to Programming . *No Starch Press*.

Interview Bit. (January 2024). *Angular Architecture – Detailed Explanation*. Načteno z Interview Bit: https://www.interviewbit.com/blog/angular-architecture/

J. Kalinowska, B. P. (2019). Comparison of tools for creating SPA applications using the examples of Angular 2 and React. *Journal of Computer Sciences Institute, Volume 10*, 1-4.

Jaiswal, P. &. (2022). *Competitive analysis of web development frameworks*. Sustainable Communication Networks and Application: Proceedings of ICSCN 2021.

Johnson, R. (2020). Modern JavaScript: Develop and design. *Peachpit Press*.

K. Boczkowski, B. P. (2020). Comparison of the performance of tools for creating a SPA application interface - React and Vue.js. *Journal of Computer Science Institute, Volume 14*, 73-77.

K. Kowalczyk, M. P.-W. (2016). *Angular JS and ReactJS libraries – performance analysis*. https://doi.org/10.35784/jcsi.126: Journal of Computer Sciences Institute, Volume 2.

Karwin, B. (2010). SQL Antipatterns: Avoiding the Pitfalls of Database Programming. *The Pragmatic Bookshelf*.

Kirkpatrick, R. (2021). Accessible design for users with disabilities. *Journal of Web Development*, 34-56.

Koch, M. (2019). Angular for Enterprise-Ready Web Applications. *Packt Publishing*.

Komperla, V., Deenadhayalan, P., Ghuli, P., & Pattar, R. (2022). React: A detailed survey. *Indonesian Journal of Electrical Engineering and Computer Science*, 1710-1717.

Kotaru, V. K. (2020). *Angular for Material Design*. Apress.

Kozlowki, P. (2013). *Mastering Web Application Development with AngularJS*.

Kyriakidis, A. M. (2016). *The majesty of Vue.js*. . Birmingham: Packt Publishing.

Laukkanen, E., Ihantola, E., & Vihavainen, A. (2017). Performance issues and optimizations in JavaScript: An empirical study. In Proceedings of the 39th International Conference on Software Engineering. *IEEE Press*, pp. 61-71.

Learn web development. (February 2024). Načteno z web.dev: https://web.dev/learn/

Levlin, M. (2020). *DOM benchmark comparison of the front-end JavaScript frameworks React, Angular, Vue, and Svelte*. Åbo Akademi.

Lewis, C. (2016). Software Testing and Continuous Quality Improvement. *CRC Press.*

Macrae, C. (2018). *Vue.js: Up and Running.* . O'Reilly Media.

Masharipov, S., Raximova, L., & oliyeva, M. (2023). Comparative Analysis of React JS and Vue JS Front End Frameworks. *International Journal of Advanced Studies*, 123-134. Načteno z https://www.myscience.uz/index.php/issue/article/view/349

Mezzalira, L. (2020). Building Micro-Frontends. *O'Reilly Media.*

Mezzalira, L. (2020). Building Micro-Frontends. *O'Reilly Media.*

Mian Muhammad, A. (Jan 2023). *Case study of Angular projects.* Načteno z Medium: https://medium.com/@miananwar.developer/case-study-of-angular-projects-239fa2462ec6

Noring, J. (2016). *Learning Angular 2.* Packt Publishing.

Overview-Measure. (February 2024). Načteno z Lighthouse: https://developer.chrome.com/docs/lighthouse/overview/

Patel, K., & Jones, R. (2021). Measuring and optimizing web performance: A comprehensive approach. *Journal of Web Engineering*, 1573-1594.

Paul, A. &. (2019). *React Native for Mobile Development.* Apress.

Preece, J. (1994). Human-Computer Interaction. *Addison-Wesley.*

Rathinam, S. (nedatováno).

Rathinam, S. (2022). Analysis and Comparison of Different Frontend. 3.

Seshadri, S. &. (2014). *AngularJS: Up and Running: Enhanced Productivity with Structured Web Apps.* . O'Reilly Media, Inc.

Shahoor, A., Khamit, A., & Yi, J. (2023). LeakPair: Proactive Repairing of Memory Leaks in Single Page Web Applications. *In Proceedings of the 38th IEEE/ACM International Conference on Automated Software Engineering*, pp. 456-467.

Shahzad, F. (2017). Modern and responsive mobile-enabled web applications. *Procedia Computer Science*, 410-415.

Simkhada, K. (2016). *Transitioning Angular 2 User Interface (UI) into React.* Helsinki Metropolia University of Applied Sciences.

Smith , J., Nguyen, T., & I., R. (2018). Improving the performance of single-page applications through intelligent data handling. *Journal of Web Development*, pp. 23-34.

Steinmetz, N., & Nwose Lotanna, V. (2021). React Performance Optimization Techniques. *Packt Publishing.*

Steinmetz, N., & Nwose Lotanna, V. (2021). React Performance Optimization Techniques. *Packt Publishing*.

The library for web and native user interfaces. (February 2024). Načteno z React: https://reactjs.org/

Vyas, R. (2022). Comparative Analysis on Front-End Frameworks for Web Applications. *International Journal for Research in Applied Science and Engineering Technology*, 10(7), 298-307.

Welling, L. (2019). Web performance in action: Building faster web pages. *Manning Publications*.

Wieruch, R. (2018). *The Road to GraphQL: Your journey to master pragmatic GraphQL in JavaScript with React. js and Node. js*. Robin Wieruch.

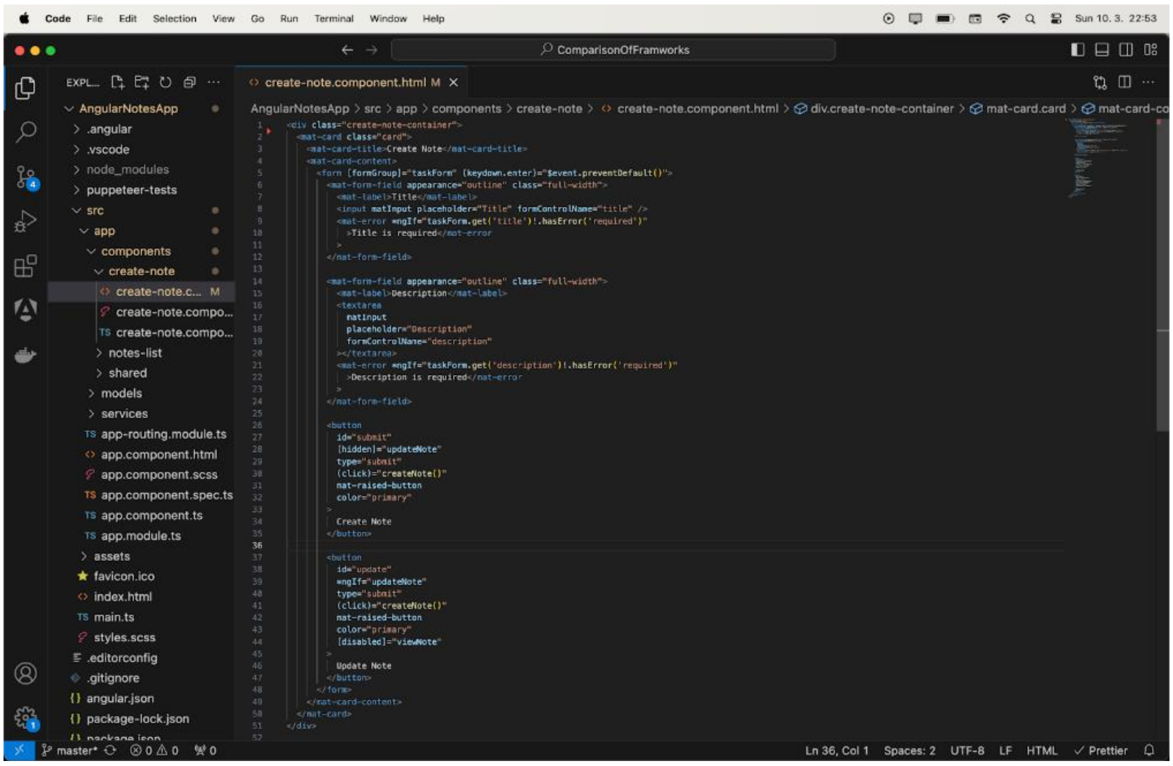
Wilken, J. (2018). *Angular in action*. Simon and Schuster.

Wills, A., Turner, C., & Dean, J. (2022). Frontend performance optimization: Techniques and strategies for improving web performance. *Journal of Frontend Engineering*, 213-229.

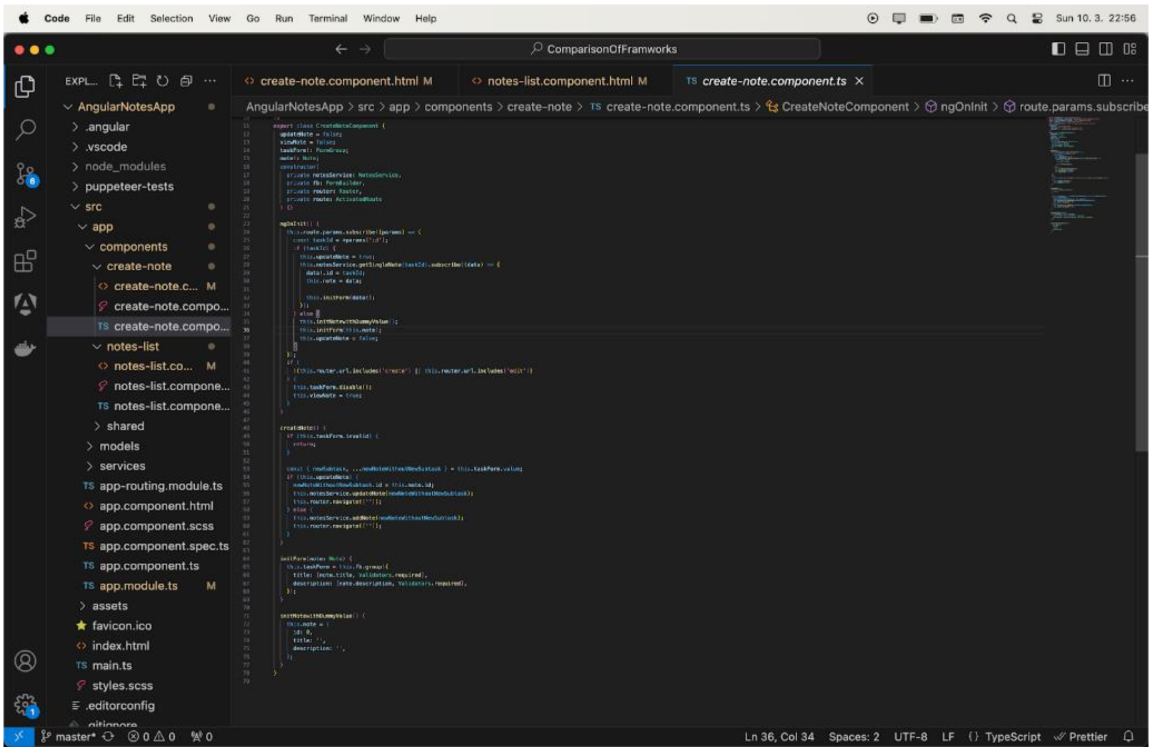
Wohlgethan, E. (2018). *Supporting Web Development Decisions by Comparing Three Major JavaScript Frameworks: Angular, React and Vue.js*. Dissertation, Hochschule für Angewandte Wissenschaften Hamburg.

Appendix A

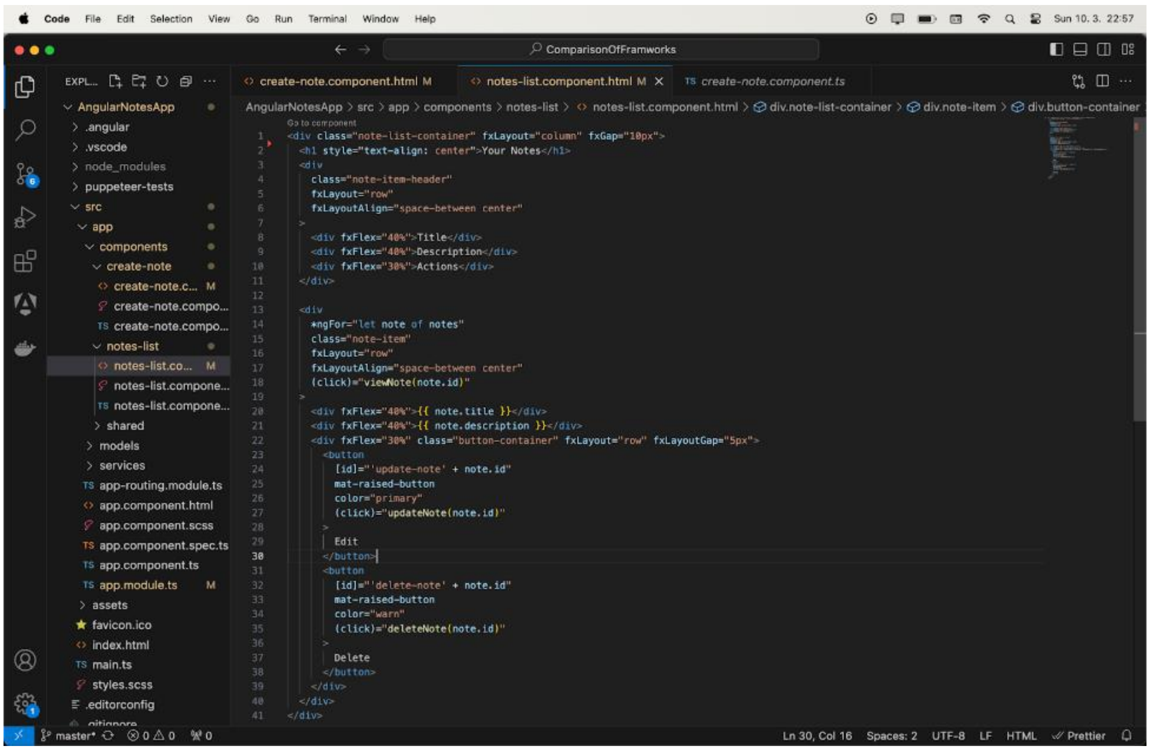
Angular App Important Code


create-note.component.html M
AngularNotesApp > src > app > components > create-note > create-note.component.html > div.create-note-container > mat-card.card > mat-card-co

```html
<div class="create-note-container">
  <mat-card class="card">
    <mat-card-title>Create Note</mat-card-title>
    <mat-card-content>
      <form [formGroup]="taskForm" (keydown.enter)="$event.preventDefault()">
        <mat-form-field appearance="outline" class="full-width">
          <mat-label>Title</mat-label>
          <input matInput placeholder="Title" formControlName="title" />
          <mat-error *ngIf="taskForm.get('title')!.hasError('required')"
            >Title is required</mat-error
          >
        </mat-form-field>

        <mat-form-field appearance="outline" class="full-width">
          <mat-label>Description</mat-label>
          <textarea
            matInput
            placeholder="Description"
            formControlName="description"
          ></textarea>
          <mat-error *ngIf="taskForm.get('description')!.hasError('required')"
            >Description is required</mat-error
          >
        </mat-form-field>

        <button
          id="submit"
          [hidden]="updateNote"
          type="submit"
          (click)="createNote()"
          mat-raised-button
          color="primary"
        >
          Create Note
        </button>

        <button
          id="update"
          *ngIf="updateNote"
          type="submit"
          (click)="createNote()"
          mat-raised-button
          color="primary"
          [disabled]="viewNote"
        >
          Update Note
        </button>
      </form>
    </mat-card-content>
  </mat-card>
</div>
```

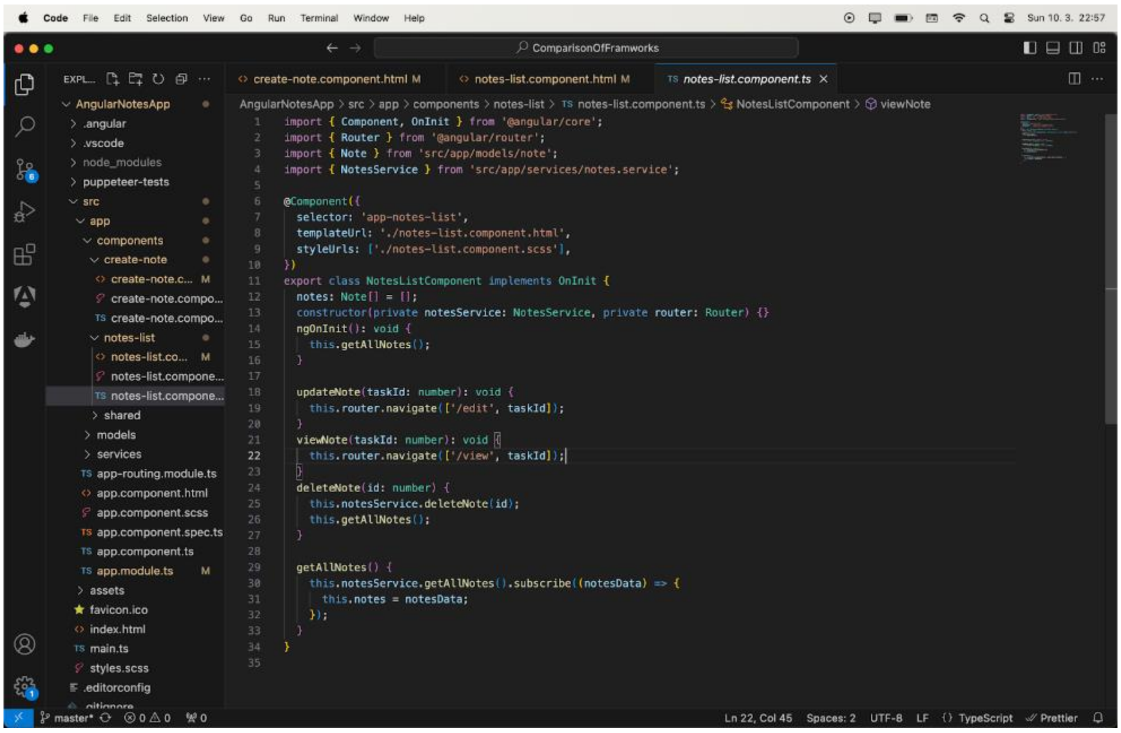


```typescript
export class CreateNoteComponent {
  updateNote = false;
  viewNote = false;
  taskForm!: FormGroup;
  note!: Note;
  constructor(
    private notesService: NotesService,
    private fb: FormBuilder,
    private router: Router,
    private route: ActivatedRoute
  ) {}

  ngOnInit() {
    this.route.params.subscribe((params) => {
      const taskId = +params['id'];
      if (taskId) {
        this.updateNote = true;
        this.notesService.getSingleNote(taskId).subscribe((data) => {
          data!.id = taskId;
          this.note = data;

          this.initForm(data!);
        });
      } else {
        this.initNotewithDummyValue();
        this.initForm(this.note);
        this.updateNote = false;
      }
    });
    if (
      !(this.router.url.includes('create') || this.router.url.includes('edit'))
    ) {
      this.taskForm.disable();
      this.viewNote = true;
    }
  }

  createNote() {
    if (this.taskForm.invalid) {
      return;
    }

    const { newSubtask, ...newNoteWithoutNewSubtask } = this.taskForm.value;
    if (this.updateNote) {
      newNoteWithoutNewSubtask.id = this.note.id;
      this.notesService.updateNote(newNoteWithoutNewSubtask);
      this.router.navigate(['']);
    } else {
      this.notesService.addNote(newNoteWithoutNewSubtask);
      this.router.navigate(['']);
    }
  }

  initForm(note: Note) {
    this.taskForm = this.fb.group({
      title: [note.title, Validators.required],
      description: [note.description, Validators.required],
    });
  }

  initNotewithDummyValue() {
    this.note = {
      id: 0,
      title: '',
      description: '',
    };
  }
}
```



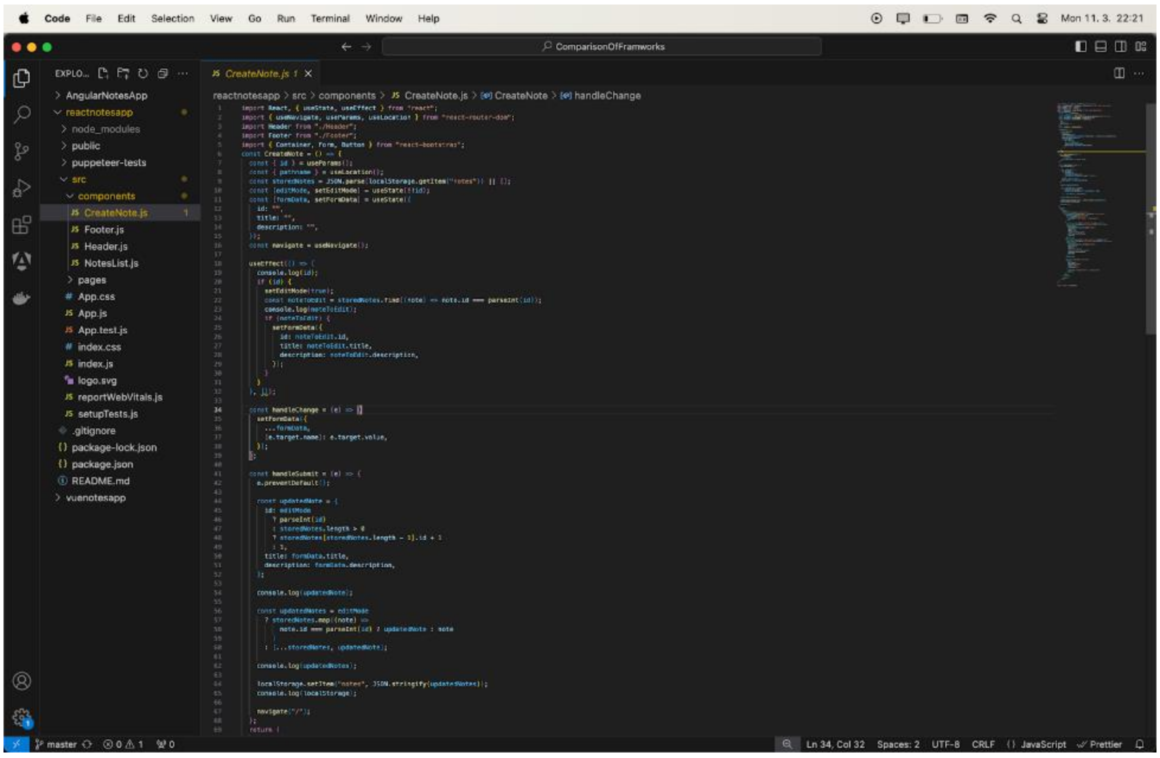
```html
<div class="note-list-container" fxLayout="column" fxGap="10px">
  <h1 style="text-align: center">Your Notes</h1>
  <div
    class="note-item-header"
    fxLayout="row"
    fxLayoutAlign="space-between center"
  >
    <div fxFlex="40%">Title</div>
    <div fxFlex="40%">Description</div>
    <div fxFlex="30%">Actions</div>
  </div>

  <div
    *ngFor="let note of notes"
    class="note-item"
    fxLayout="row"
    fxLayoutAlign="space-between center"
    (click)="viewNote(note.id)"
  >
    <div fxFlex="40%">{{ note.title }}</div>
    <div fxFlex="40%">{{ note.description }}</div>
    <div fxFlex="30%" class="button-container" fxLayout="row" fxLayoutGap="5px">
      <button
        [id]="'update-note' + note.id"
        mat-raised-button
        color="primary"
        (click)="updateNote(note.id)"
      >
        Edit
      </button>
      <button
        [id]="'delete-note' + note.id"
        mat-raised-button
        color="warn"
        (click)="deleteNote(note.id)"
      >
        Delete
      </button>
    </div>
  </div>
</div>
```

II

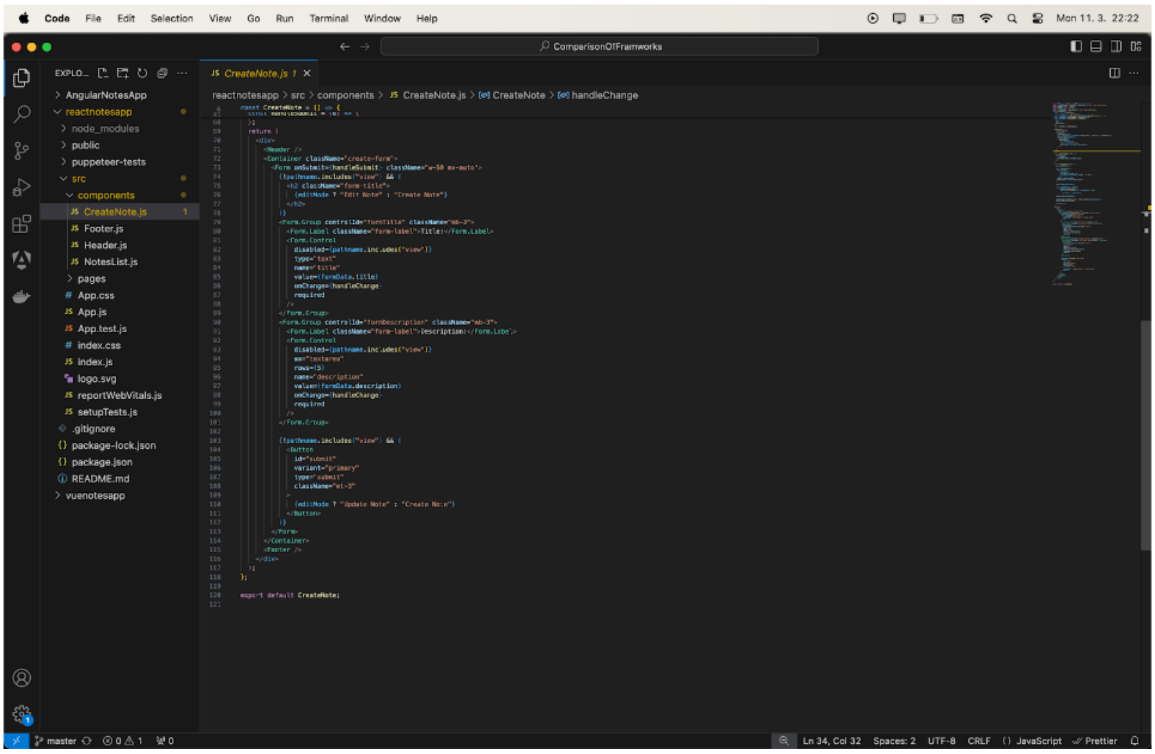


```typescript
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { Note } from 'src/app/models/note';
import { NotesService } from 'src/app/services/notes.service';

@Component({
  selector: 'app-notes-list',
  templateUrl: './notes-list.component.html',
  styleUrls: ['./notes-list.component.scss'],
})
export class NotesListComponent implements OnInit {
  notes: Note[] = [];
  constructor(private notesService: NotesService, private router: Router) {}
  ngOnInit(): void {
    this.getAllNotes();
  }

  updateNote(taskId: number): void {
    this.router.navigate(['/edit', taskId]);
  }
  viewNote(taskId: number): void {
    this.router.navigate(['/view', taskId]);
  }
  deleteNote(id: number) {
    this.notesService.deleteNote(id);
    this.getAllNotes();
  }

  getAllNotes() {
    this.notesService.getAllNotes().subscribe((notesData) => {
      this.notes = notesData;
    });
  }
}
```

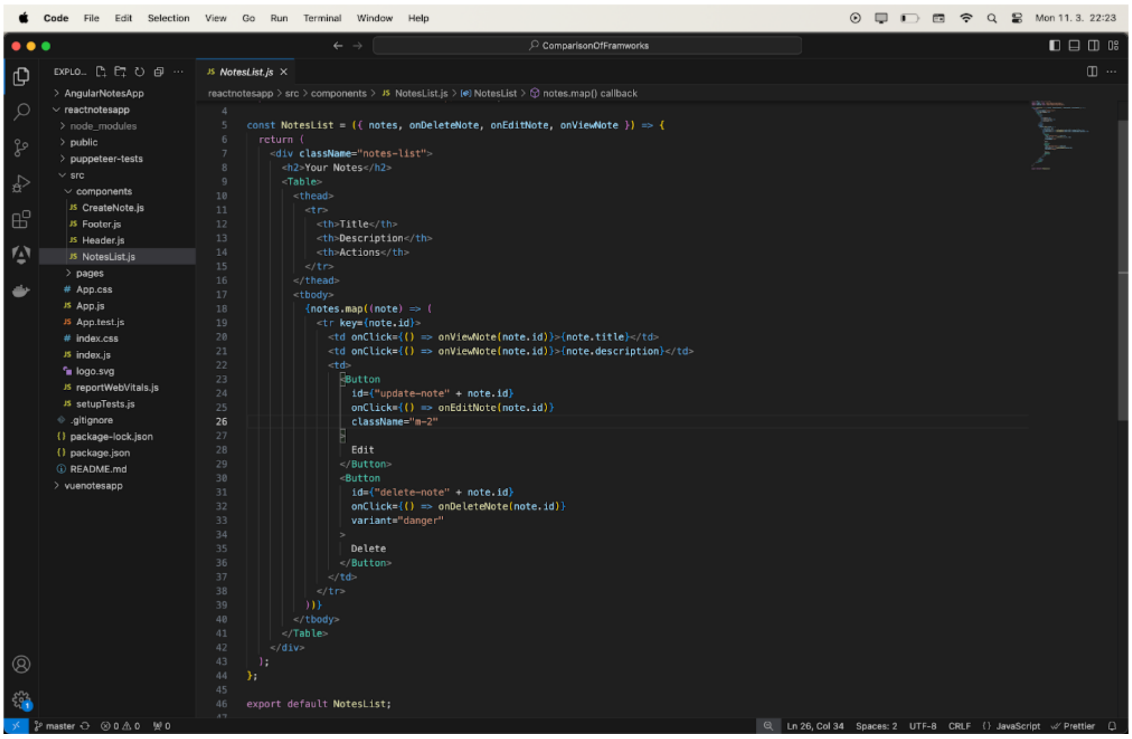
React App Important Code



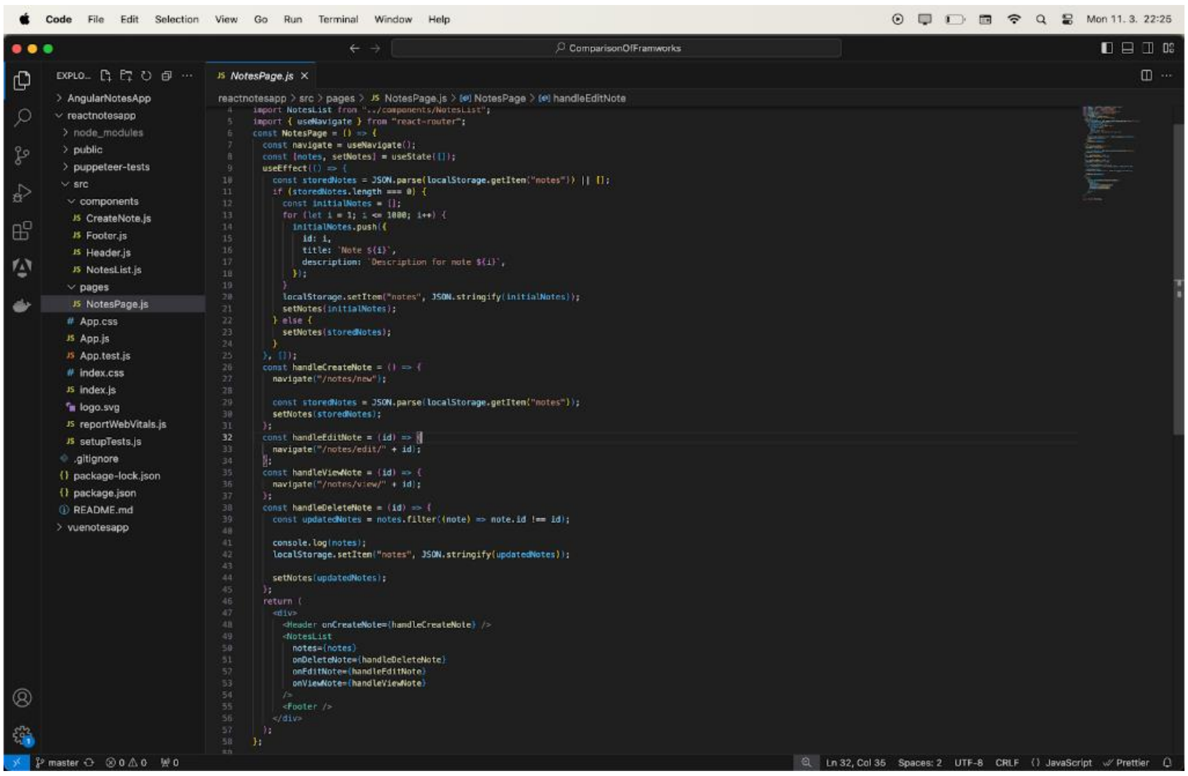
```javascript
import React, { useState, useEffect } from "react";
import { useNavigate, useParams, useLocation } from "react-router-dom";
import Header from "./Header";
import Footer from "./Footer";
import { Container, Form, Button } from "react-bootstrap";
const CreateNote = () => {
  const { id } = useParams();
  const { pathname } = useLocation();
  const storedNotes = JSON.parse(localStorage.getItem("notes")) || [];
  const [editMode, setEditMode] = useState(!!id);
  const [formData, setFormData] = useState({
    id: "",
    title: "",
    description: "",
  });
  const navigate = useNavigate();

  useEffect(() => {
    console.log(id);
    if (id) {
      setEditMode(true);
      const noteToEdit = storedNotes.find((note) => note.id === parseInt(id));
      console.log(noteToEdit);
      if (noteToEdit) {
        setFormData({
          id: noteToEdit.id,
          title: noteToEdit.title,
          description: noteToEdit.description,
        });
      }
    }
  }, []);

  const handleChange = (e) => {
    setFormData({
      ...formData,
      [e.target.name]: e.target.value,
    });
  };

  const handleSubmit = (e) => {
    e.preventDefault();

    const updatedNote = {
      id: editMode
        ? parseInt(id)
        : storedNotes.length > 0
        ? storedNotes[storedNotes.length - 1].id + 1
        : 1,
      title: formData.title,
      description: formData.description,
    };

    console.log(updatedNote);

    const updatedNotes = editMode
      ? storedNotes.map((note) =>
          note.id === parseInt(id) ? updatedNote : note
        )
      : [...storedNotes, updatedNote];

    console.log(updatedNotes);

    localStorage.setItem("notes", JSON.stringify(updatedNotes));
    console.log(localStorage);

    navigate("/");
  };
  return (
```



```
const CreateNote = () => {
  const handleSubmit = (e) => {
  };
  return (
    <div>
      <Header />
      <Container className="create-form">
        <Form onSubmit={handleSubmit} className="w-50 mx-auto">
          {!pathname.includes("view") && (
            <h2 className="form-title">
              {editMode ? "Edit Note" : "Create Note"}
            </h2>
          )}
          <Form.Group controlId="formTitle" className="mb-3">
            <Form.Label className="form-label">Title:</Form.Label>
            <Form.Control
              disabled={pathname.includes("view")}
              type="text"
              name="title"
              value={formData.title}
              onChange={handleChange}
              required
            />
          </Form.Group>
          <Form.Group controlId="formDescription" className="mb-3">
            <Form.Label className="form-label">Description:</Form.Label>
            <Form.Control
              disabled={pathname.includes("view")}
              as="textarea"
              rows={5}
              name="description"
              value={formData.description}
              onChange={handleChange}
              required
            />
          </Form.Group>

          {!pathname.includes("view") && (
            <Button
              id="submit"
              variant="primary"
              type="submit"
              className="mt-3"
            >
              {editMode ? "Update Note" : "Create Note"}
            </Button>
          )}
        </Form>
      </Container>
      <Footer />
    </div>
  );
};

export default CreateNote;
```

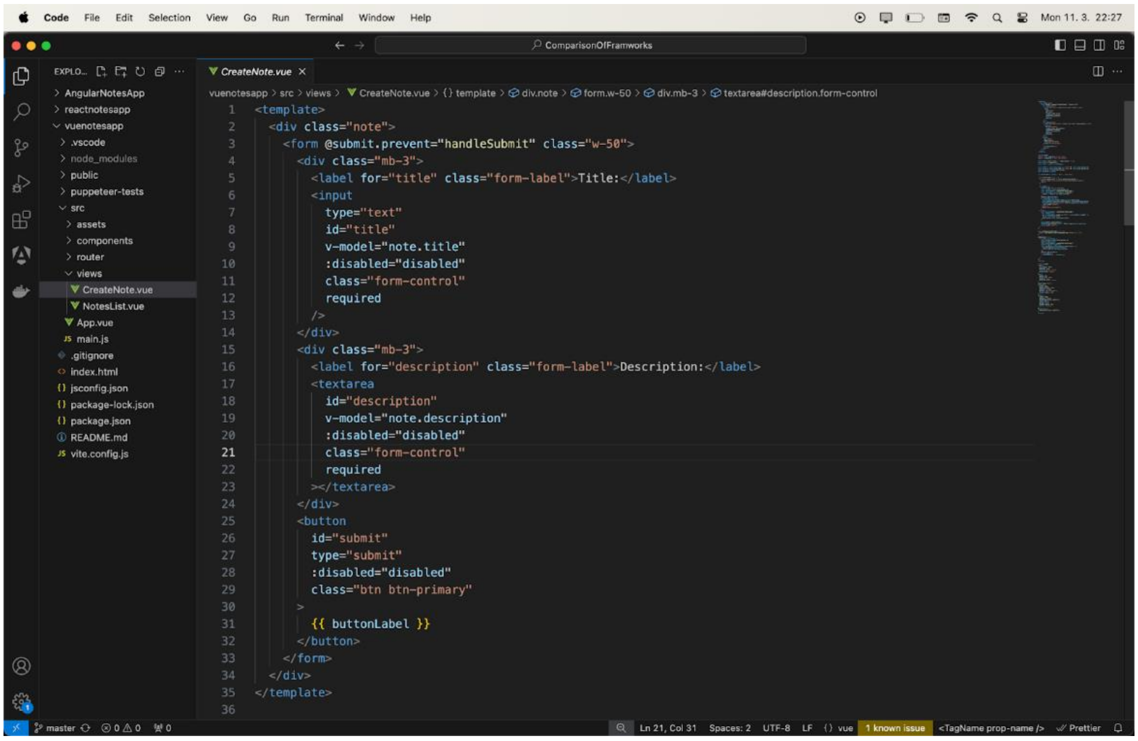


```

const NotesList = ({ notes, onDeleteNote, onEditNote, onViewNote }) => {
  return (
    <div className="notes-list">
      <h2>Your Notes</h2>
      <Table>
        <thead>
          <tr>
            <th>Title</th>
            <th>Description</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {notes.map((note) => (
            <tr key={note.id}>
              <td onClick={() => onViewNote(note.id)}>{note.title}</td>
              <td onClick={() => onViewNote(note.id)}>{note.description}</td>
              <td>
                <Button
                  id={"update-note" + note.id}
                  onClick={() => onEditNote(note.id)}
                  className="m-2"

                >
                  Edit
                </Button>
                <Button
                  id={"delete-note" + note.id}
                  onClick={() => onDeleteNote(note.id)}
                  variant="danger"
                >
                  Delete
                </Button>
              </td>
            </tr>
          ))}
        </tbody>
      </Table>
    </div>
  );
};

export default NotesList;
```

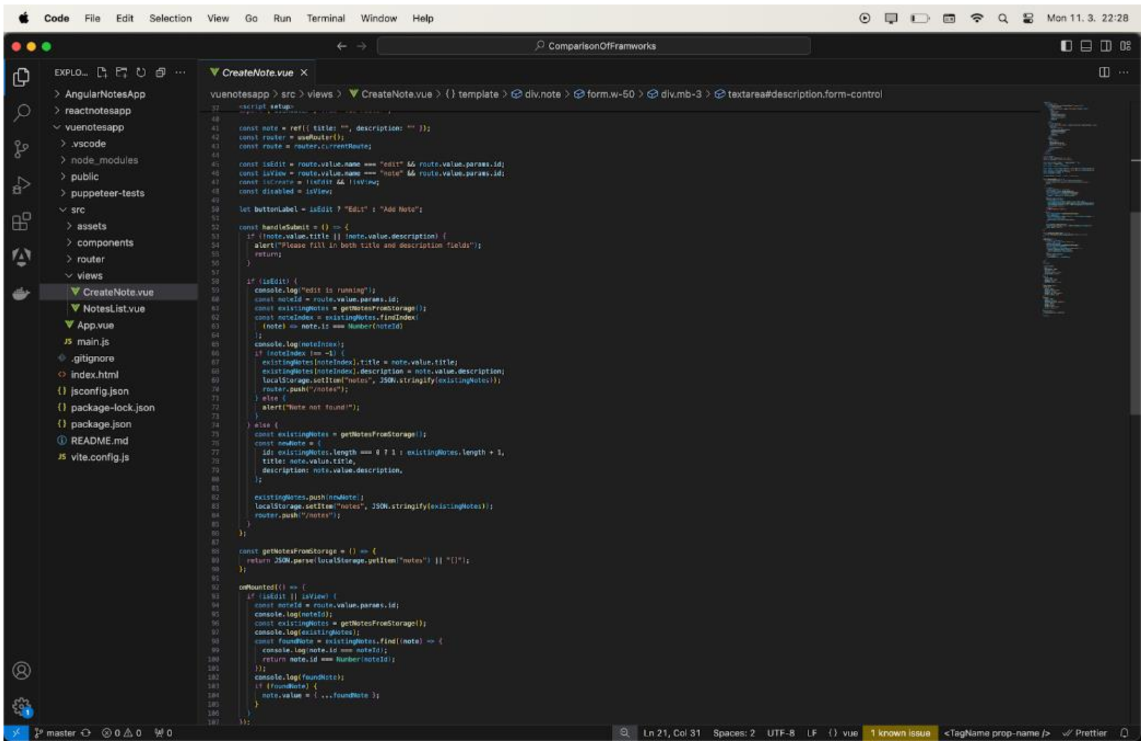


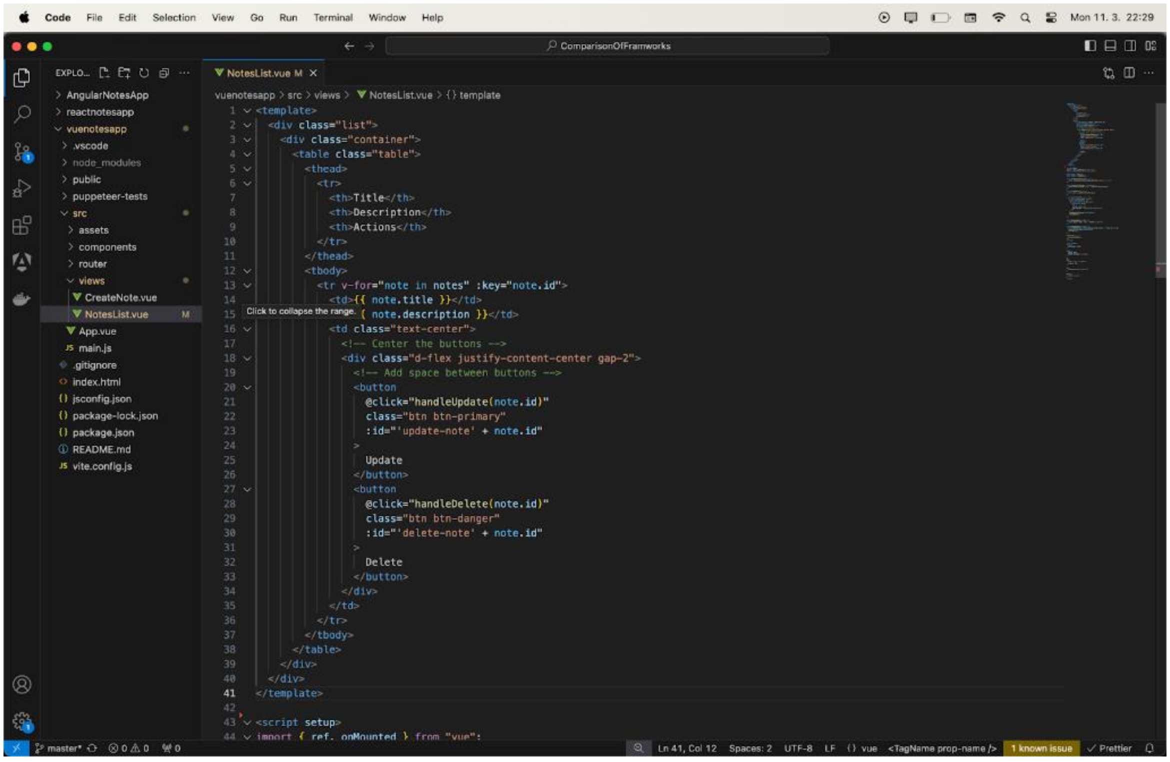
```
import NotesList from "../components/NotesList";
import { useNavigate } from "react-router";
const NotesPage = () => {
  const navigate = useNavigate();
  const [notes, setNotes] = useState([]);
  useEffect(() => {
    const storedNotes = JSON.parse(localStorage.getItem("notes")) || [];
    if (storedNotes.length === 0) {
      const initialNotes = [];
      for (let i = 1; i <= 1000; i++) {
        initialNotes.push({
          id: i,
          title: `Note ${i}`,
          description: `Description for note ${i}`,
        });
      }
      localStorage.setItem("notes", JSON.stringify(initialNotes));
      setNotes(initialNotes);
    } else {
      setNotes(storedNotes);
    }
  }, []);
  const handleCreateNote = () => {
    navigate("/notes/new");

    const storedNotes = JSON.parse(localStorage.getItem("notes"));
    setNotes(storedNotes);
  };
  const handleEditNote = (id) => {
    navigate("/notes/edit/" + id);
  };
  const handleViewNote = (id) => {
    navigate("/notes/view/" + id);
  };
  const handleDeleteNote = (id) => {
    const updatedNotes = notes.filter((note) => note.id !== id);

    console.log(notes);
    localStorage.setItem("notes", JSON.stringify(updatedNotes));

    setNotes(updatedNotes);
  };
  return (
    <div>
      <Header onCreateNote={handleCreateNote} />
      <NotesList
        notes={notes}
        onDeleteNote={handleDeleteNote}
        onEditNote={handleEditNote}
        onViewNote={handleViewNote}
      />
      <Footer />
    </div>
  );
};
```

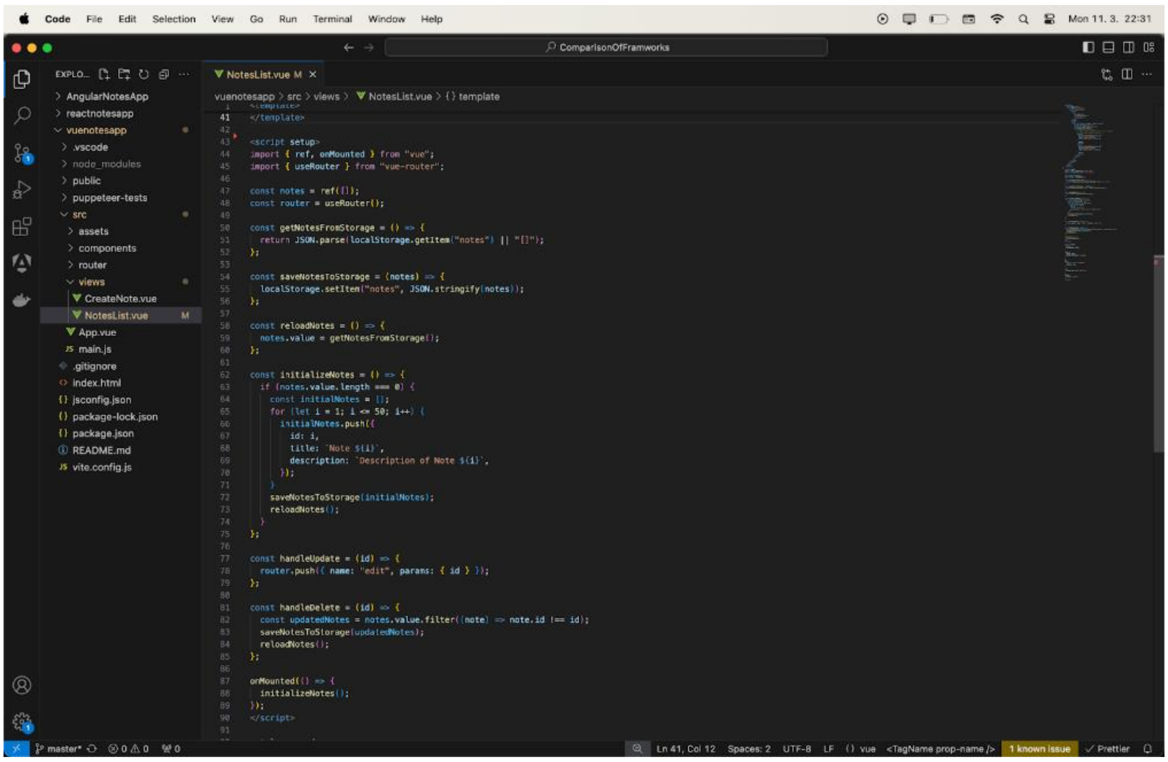
Vue.js App Important Code



```
<template>
  <div class="note">
    <form @submit.prevent="handleSubmit" class="w-50">
      <div class="mb-3">
        <label for="title" class="form-label">Title:</label>
        <input
          type="text"
          id="title"
          v-model="note.title"
          :disabled="disabled"
          class="form-control"
          required
        />
      </div>
      <div class="mb-3">
        <label for="description" class="form-label">Description:</label>
        <textarea
          id="description"
          v-model="note.description"
          :disabled="disabled"
          class="form-control"
          required
        ></textarea>
      </div>
      <button
        id="submit"
        type="submit"
        :disabled="disabled"
        class="btn btn-primary"
      >
        {{ buttonLabel }}
      </button>
    </form>
  </div>
</template>
```


```
<script setup>

const note = ref({ title: "", description: "" });
const router = useRouter();
const route = router.currentRoute;

const isEdit = route.value.name === "edit" && route.value.params.id;
const isView = route.value.name === "note" && route.value.params.id;
const isCreate = !isEdit && !isView;
const disabled = isView;

let buttonLabel = isEdit ? "Edit" : "Add Note";

const handleSubmit = () => {
  if (!note.value.title || !note.value.description) {
    alert("Please fill in both title and description fields");
    return;
  }

  if (isEdit) {
    console.log("edit is running");
    const noteId = route.value.params.id;
    const existingNotes = getNotesFromStorage();
    const noteIndex = existingNotes.findIndex(
      (note) => note.id === Number(noteId)
    );
    console.log(noteIndex);
    if (noteIndex !== -1) {
      existingNotes[noteIndex].title = note.value.title;
      existingNotes[noteIndex].description = note.value.description;
      localStorage.setItem("notes", JSON.stringify(existingNotes));
      router.push("/notes");
    } else {
      alert("Note not found!");
    }
  } else {
    const existingNotes = getNotesFromStorage();
    const newNote = {
      id: existingNotes.length === 0 ? 1 : existingNotes.length + 1,
      title: note.value.title,
      description: note.value.description,
    };

    existingNotes.push(newNote);
    localStorage.setItem("notes", JSON.stringify(existingNotes));
    router.push("/notes");
  }
};

const getNotesFromStorage = () => {
  return JSON.parse(localStorage.getItem("notes") || "[]");
};

onMounted(() => {
  if (isEdit || isView) {
    const noteId = route.value.params.id;
    console.log(noteId);
    const existingNotes = getNotesFromStorage();
    console.log(existingNotes);
    const foundNote = existingNotes.find((note) => {
      console.log(note.id === noteId);
      return note.id === Number(noteId);
    });
    console.log(foundNote);
    if (foundNote) {
      note.value = { ...foundNote };
    }
  }
});
```

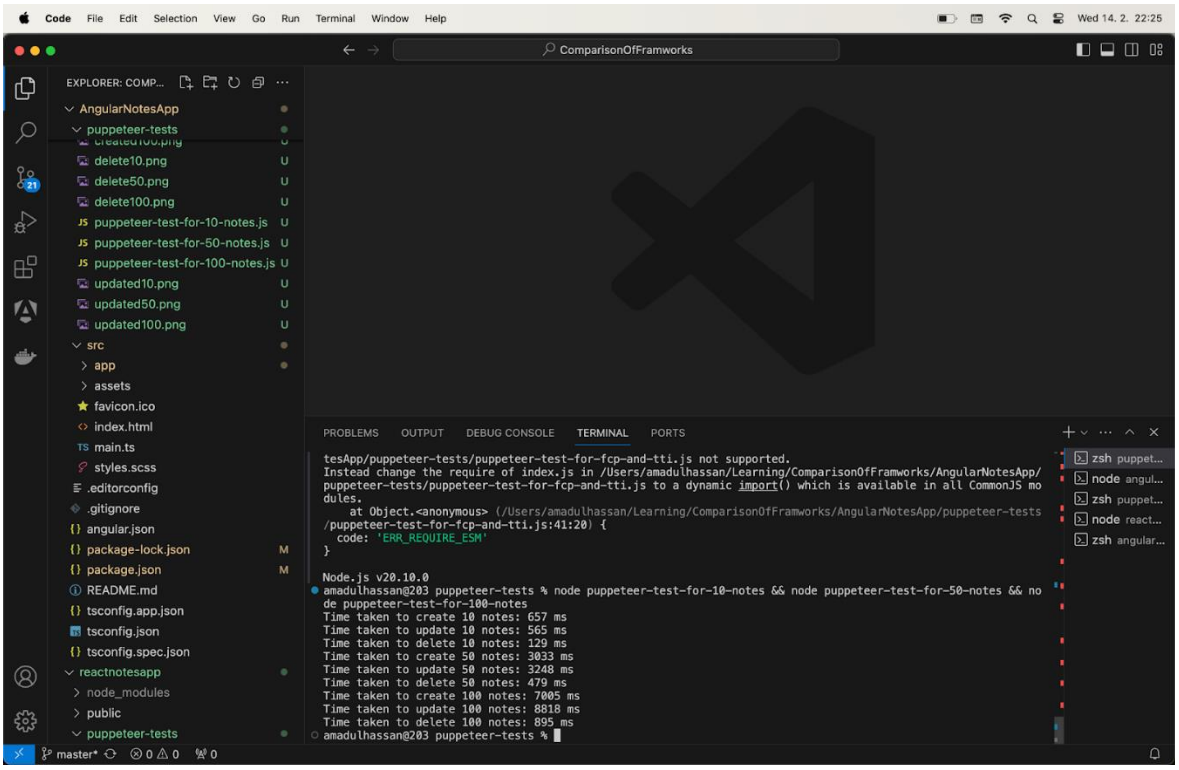

```
<template>
  <div class="list">
    <div class="container">
      <table class="table">
        <thead>
          <tr>
            <th>Title</th>
            <th>Description</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          <tr v-for="note in notes" :key="note.id">
            <td>{{ note.title }}</td>
            <td>{{ note.description }}</td>
            <td class="text-center">
              <!-- Center the buttons -->
              <div class="d-flex justify-content-center gap-2">
                <!-- Add space between buttons -->
                <button
                  @click="handleUpdate(note.id)"
                  class="btn btn-primary"
                  :id="'update-note' + note.id"
                >
                  Update
                </button>
                <button
                  @click="handleDelete(note.id)"
                  class="btn btn-danger"
                  :id="'delete-note' + note.id"
                >
                  Delete
                </button>
              </div>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
  </div>
</template>

<script setup>
import { ref, onMounted } from "vue";
```



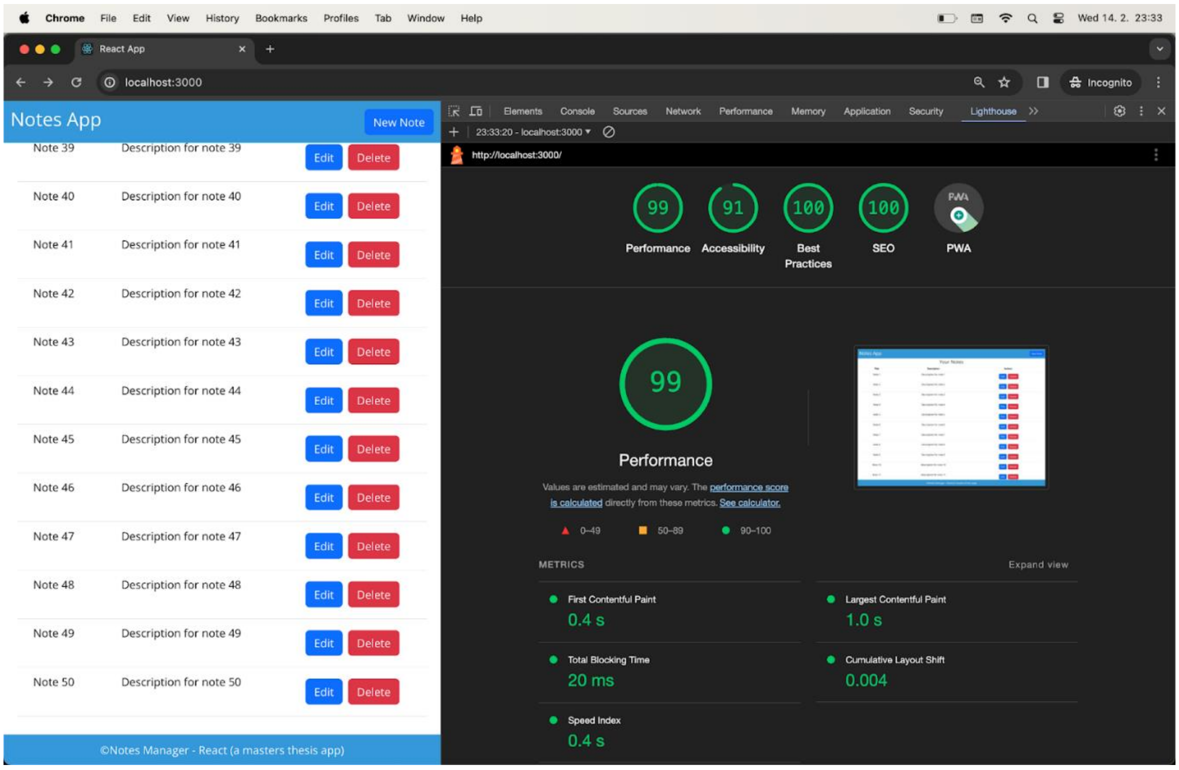
```
</template>

<script setup>
import { ref, onMounted } from "vue";
import { useRouter } from "vue-router";

const notes = ref([]);
const router = useRouter();

const getNotesFromStorage = () => {
  return JSON.parse(localStorage.getItem("notes") || "[]");
};

const saveNotesToStorage = (notes) => {
  localStorage.setItem("notes", JSON.stringify(notes));
};

const reloadNotes = () => {
  notes.value = getNotesFromStorage();
};

const initializeNotes = () => {
  if (notes.value.length === 0) {
    const initialNotes = [];
    for (let i = 1; i <= 50; i++) {
      initialNotes.push({
        id: i,
        title: `Note ${i}`,
        description: `Description of Note ${i}`,
      });
    }
    saveNotesToStorage(initialNotes);
    reloadNotes();
  }
};

const handleUpdate = (id) => {
  router.push({ name: "edit", params: { id } });
};

const handleDelete = (id) => {
  const updatedNotes = notes.value.filter((note) => note.id !== id);
  saveNotesToStorage(updatedNotes);
  reloadNotes();
};

onMounted(() => {
  initializeNotes();
});
</script>
```

Appendix B

Puppeteer and Google Lighthouse Sample Tests



```
tesApp/puppeteer-tests/puppeteer-test-for-fcp-and-tti.js not supported.
Instead change the require of index.js in /Users/amadulhassan/Learning/ComparisonOfFramworks/AngularNotesApp/
puppeteer-tests/puppeteer-test-for-fcp-and-tti.js to a dynamic import() which is available in all CommonJS mo
dules.
    at Object.<anonymous> (/Users/amadulhassan/Learning/ComparisonOfFramworks/AngularNotesApp/puppeteer-tests
/puppeteer-test-for-fcp-and-tti.js:41:20) {
  code: 'ERR_REQUIRE_ESM'
}

Node.js v20.10.0
amadulhassan@203 puppeteer-tests % node puppeteer-test-for-10-notes && node puppeteer-test-for-50-notes && no
de puppeteer-test-for-100-notes
Time taken to create 10 notes: 657 ms
Time taken to update 10 notes: 565 ms
Time taken to delete 10 notes: 129 ms
Time taken to create 50 notes: 3033 ms
Time taken to update 50 notes: 3248 ms
Time taken to delete 50 notes: 479 ms
Time taken to create 100 notes: 7005 ms
Time taken to update 100 notes: 8818 ms
Time taken to delete 100 notes: 895 ms
amadulhassan@203 puppeteer-tests %
```


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Wed 14. 2. 23:33
React App
localhost:3000
Incognito
Elements Console Sources Network Performance Memory Application Security Lighthouse
23:33:20 - localhost:3000
http://localhost:3000/
Notes App
New Note
PWA

Note 39	Description for note 39	Edit	Delete
Note 40	Description for note 40	Edit	Delete
Note 41	Description for note 41	Edit	Delete
Note 42	Description for note 42	Edit	Delete
Note 43	Description for note 43	Edit	Delete
Note 44	Description for note 44	Edit	Delete
Note 45	Description for note 45	Edit	Delete
Note 46	Description for note 46	Edit	Delete
Note 47	Description for note 47	Edit	Delete
Note 48	Description for note 48	Edit	Delete
Note 49	Description for note 49	Edit	Delete
Note 50	Description for note 50	Edit	Delete

©Notes Manager - React (a masters thesis app)

Performance	Accessibility	Best Practices	SEO	PWA
99	91	100	100	

99
Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

● First Contentful Paint 0.4 s	● Largest Contentful Paint 1.0 s
● Total Blocking Time 20 ms	● Cumulative Layout Shift 0.004
● Speed Index 0.4 s	

Appendix C

Web Development Frameworks Survey

Demographic Information

1. How many years of experience do you have in web development?

   - Less than 1 year

   - 1-3 years

   - 4-6 years

   - 7-10 years

   - More than 10 years

2. Which web development frameworks have you used? (Select all that apply)

   - Angular

   - React

   - Vue.js

   - Other (Please specify)

Ease of Use

3. On a scale of 1 to 5, how would you rate the ease of learning for each of the following frameworks? (1 being very difficult and 5 being very easy)

   - Angular

   - React

   - Vue.js

4. Please describe any challenges you faced while learning these frameworks. (Open-ended)

Scalability

5. On a scale of 1 to 5, how would you rate the scalability of the following frameworks for large-scale applications? (1 being very poor and 5 being excellent)

   - Angular

- React

- Vue.js

6. Can you share an example where you successfully scaled an application using one of these frameworks? What was your approach? (Open-ended)

Community Support

7. On a scale of 1 to 5, how would you rate the community support available for each framework? Consider documentation, forums, and third-party tools in your rating. (1 being very poor and 5 being excellent)

- Angular

- React

- Vue.js

8. How has the community support for these frameworks impacted your development work? (Open-ended)

Overall Satisfaction

9. On a scale of 1 to 5, how satisfied are you with the following frameworks based on your overall experience? (1 being very dissatisfied and 5 being very satisfied)

- Angular

- React

- Vue.js

10. What factors contribute to your satisfaction or dissatisfaction with these frameworks? (Open-ended)

Additional Insights

11. In your opinion, what are the strengths of each framework?

- Angular:

- React:

- Vue.js:

12. What improvements would you like to see in each framework?

- Angular:

- React:

- Vue.js:

13. Any additional comments or experiences you would like to share about using Angular, React, or Vue.js?

End of Survey

Thank you for participating in this survey. Your insights are valuable to understanding the current landscape of web development frameworks.

Annex 1

List of developers who agreed to share there information

#	First Name	Last Name	Email	Role	Years of Experience
1.	Ahmadreza Mohammad	Ghorbani	arm.ghorbani@gmail.com	FullStack Web Developer	2
2.	Hector	Alvarado	hectoraaramos@gmail.com	Web Developer	4
3.	Ettiene	Maloun-Chenier	etiennemc@outlook.com	Web Engineer	3
4.	Fatima	Mehak	fatima.mehak257@gmail.com	Junior Frontend Developer	6
5.	Muzamil	Yeseen	yaseen.muzamil.m@gmail.com	Angular Developer	4
6.	Hassan	Zakaria	hz@investdigitally.com	Senior Developer	5
7.	Marek	Mogyorosi	mmo@indigilabs.com	Software Engineer	5
8.	Anton		aki@indigilabs.com	FullStack Engineer	7
9.	Ezzat Batool	Sonia	ezzatbsonia@gmail.com	Junior Developer	2
10.	Sami	Warraich	samiwarraich51@gmail.om	Freelancer	5
11.	Aditya	Kaseri	adityakes@merck.com	FullStack Developer	3
12.	Bardia		bardia.farahmand@yahoo.com	Freelancer	1

13.	Marton	Berton	mbe@indigilabs.com	Web Designer	2
14.	Martin	Demek	martin.demek@fcc-group.cz	Web Application Developer	6
15.	Muhammad	Amir	muhammadaamir045045@gmail.com	FullStack App Developer	6
16.	Martin	Dvorak	martin.dvorak@fcc-group.cz	Freelancer	3
17.	Tomas	Bartos	tb@investdigitally.com	Application Architect	4
18.	Bedrihan	Hoskun	bh@investdigitally.com	Senior Web Designer and Developer	3
19.	Karina	Baranska	kb@indigilabs.com	Web Developer and Tester	6
20.	Osman	Salih	os@indigilabs.com	Testing Engineer	7
21.	Saad	Minhas	saadm.minhas@gmail.com	Web Engineer	7
22.	Serdar	Arslanturk	sa@indigilabs.com	Senior Dev Ops Engineer	1
23.	Alina	Ataniyazova	alina.ataniyazova@gmail.com	Senior Backend Developer	2
24.	Jeyhun	Hasanov	ceyhun.hasanov99@gmail.com	Freelancer	2
25.	Marek	Bartos	mb@indigilabs.com	Backend Developer	4
26.	Petra	Hilde	petra.hildebrandova@fcc-group.cz	Junior Developer	5

27.	Kai	Wong	kwong8339@gmail.com	Software Developer	6
28.	Nijat	Nazarov	nnazarov2020@gmail.com	Freelancer	3
29.	Frankline	Udemmadu	nwike.udemmadu@gmail.com	Angular Developer	4
30.	Paul	Samuel	paulkehindesamuel@gmail.com	Freelancer	4